DASIP’07
WORKSHOP ON DESIGN & ARCHITECTURES
FOR SIGNAL AND IMAGE PROCESSING

The DASIP Workshop results from the success for more than ten years of the French workshop on Algorithm-Architecture Matching. For the 2007 edition, this workshop is extended to the Europe in order to gather our community. The research community in Europe addressing these issues is very active both in academy and industry. The goals of this workshop are to present the latest results in the domain of design and architecture for signal and image processing and to initiate a regular meeting of European researchers addressing this topic.

Introducing new design methodologies is necessary when facing the new emerging applications as, for example advanced mobile communication systems or smart sensors based systems. This forms a driving force for the future evolutions of embedded system designs methodologies.

The DASIP Workshop will give the opportunity for researchers to exchange the ideas and to build the collaboration on emerging topics and technologies. It also aims to strength the links between the European Information Society Technologies (IST) priorities and the researchers in the domain of design and architecture for signal and image processing.

The DASIP Workshop is organized around eight technical oral sessions that cover essential aspects of design and architectures for signal and image processing applications. One industrial session has also been prepared. Besides, poster sessions will favour direct discussion between presenters.

The DASIP Workshop will hold in Grenoble, at the Minatec Innovation Center, one of the european top center for innovation and expertise in micro and nanotechnology.

Stéphane Mancini
Sébastien Roux
DASIP General Co-Chairs

Grenoble
July 2007
DASIP’07 WORKSHOP ORGANIZATION

General Co-chairs
MANCINI Stéphane, GIPSA-Lab, France
ROUX Sébastien, France Telecom R&D, France

Programme Co-chairs
MILOJEVIC Dragomir, University of Bruxelles, Belgium
GOGNIAT Guy, LESTER, France

Programme Committee
AMIRA Abbes, Brunel University, England
ARIDHI Slaheddine, Texas Instruments, France
BECKER Juergen, Karlsruhe University, Germany
BIELECKI Wlodzimierz, Politechnika Szczecinska, Poland
BOBDA Christophe, Kaiserslautern U., Germany
BOURIDANE Ahmed, Queen's U. Belfast, Ireland
DRUTAROVSKY Milos, T. U. of Kosice, Slovakia
ERDOGAN Ahmet, U. of Edinburgh, Scotland
GALAJDA Pavol, T. U. of Kosice, Slovakia
GARDA Patrick, U. Pierre Et Marie Curie, France
HANNIG Frank, University of Erlangen-Nuremberg, Germany
KOCH Peter, CSDR, Denmark
LE MOULLEC Yannick, Aalborg University, Denmark
LEGAT Jean-Didier, Université catholique de Louvain, Belgium
MAMALET Franck, France Télécom, France
MATTAVELLI Marco, Epfl, Switzerland
MEDULLA Giuseppe, Atmel, Italy
MOHAMED Abid, Enis, Tunisia
PAINDAVOINE Michel, Le2i, France
PIGUET Christian, CSEM, Switzerland
PLAKS Toomas, London, England
ROBERT Frederic, University of Bruxelles, Belgium
RUPP Markus, Vienna U. of Technology, Austria
SHAWKY Mohamed M.,Heudiasyc, France
SOARES Indrusiak Leandro, U. of Darmstadt, Germany
SOREL Yves, INRIA, France
WEBER Serge, Lien, France

Organization Committee
ABELLARD Alexandre, Université de Toulon, France
CASSEAU Emmanuel, IRISA, France
GRANADO Bertrand, ETIS-ENSEA, France
LE NOURS Sébastien, Université de Nantes, France
MORAWIEC Adam, ECSI, France
NEZAN Jean-François, IETR, France
SHAWKY Mohammed, Heudiasyc, France
CHILLET Daniel, ENSSAT, France
DASIP’07 CONTENT

INVITED TALK 1
Pr Tughrul Arslan, University of Edinburgh
Design Methodologies and Architectures for a Multi-standard Age ------------------9

INVITED TALK 2
Dr. Mladen Berekovic, IMEC
Reconfigurable Architectures for Low-Power Nomadic DSP Applications ---------11

INVITED TALK 3
Alain Clouard, ST
New benefits of advanced hardware/software codesign and coverification of multimedia System-On-Chip (SOCs) exploiting TLM and SPIRIT standards------------------13

SESSION CAD TOOLS
Multimode architecture design -----------------------------------------------15-22
Emmanuel Casseau, Shafqat Khan, Bertrand Le Gal

A High Level Generic Application Analysis Methodology For Early Design Space Exploration -----------------------------------------------23-30
Muhammad Rashid, Thierry Goubier, Bernard Pottier

Noise model for Accuracy Constraint Determination in Fixedpoint Systems -------31-36
Daniel Menard, Romain Serizel, Romuald

SESSION IMAGE PROCESSING
Efficient memory management for Ray Casting ---------------------------------37-45
Stéphane Mancini, Michel Desvignes

Demosaicing on the TriMedia 3270 ------------------------------------------46-52
Philippeau Harold, Aki Mohamed, Talbot Hugues, Bara Stefan, Fraga-Silva Thiago

Efficient Hardware Implementation of Connected Component Tree Algorithm ----53-60
Nicolas Ngan, Francois Contou-Carrère, Benoit Marcon, Sebastien Guerin,
Eva Dokladalova, Mohamed Akil

Algorithms selection and adaptation in accord with architecture for RBF neural network based face authentication SoC ----------------------------------------61-66
Pierrefeu Lionel, Jay Jacques

- 5 -
SESSION HW/SW CODESIGN

A HW/SW codesign platform for Algorithm-Architecture mapping 67-73
Christophe Lucarz, Marco Mattavelli, Julien Dubois

A SystemC Modular RTOS Model for Embedded Services Exploration 74-81
Emmanuel Huck, Benoît Miramond, François Verdier

A complete methodology using co-simulation and co-verification to embed a
telecommunication application in a System on Chip based on a NoC structure 82-89
Simone Bacles-Min, Christine Hennebert

MicroSpider CAD tool: case study of NoC IP generation for FPGA 90-96
Rachid Dafali, Jean-Philippe Diguet, Emmanuel Juin, Samuel Evain

SESSION 3D AND VIDEO PROCESSING

Algorithm Architecture Adequacy for High Speed 3D Tomography 97-104
Nicolas Gac, Stéphane Mancini, Michel Desvignes, Dominique Houzet

SoPC for 3D Point Rendering 105-111
Lars Middendorf, Felix Mühlbauer, Christophe Bobda, Georg Umlauf

Gaia Video Processing Algorithms: Design Process and Prototyping Activities 112-117
Séverin Provost, Marc Le Roy, Bruno Mamdy, Grégory Flandin

SESSION SENSORS FOR IMAGE PROCESSING

A Multi-Processing 10 000 frames/s CMOS Image 118-123
Jerome Dubois, Dominique Ginhac, Michel Paindavoine

High Performance Embedded Co-Processor Architecture
for CMOS Imaging Systems 124-130
Romuald Mosqueron, Julien Dubois, Marco Mattavelli

SESSION DYNAMIC RECONFIGURATION

Hardware task context management for fine grained dynamically
reconfigurable architecture 131-135
Samuel Garcia, Jean-Christophe Prevotet, Bertrand Granado

A modelling of the flexibility of an RTR-FPGA implementation in comparison to a software
implementation 136-142
Yves Berviller, Oscar Perez, Serge Weber
SESSION TELECOMMUNICATION AND CRYPTOGRAPHY APPLICATIONS

Deploying a telecommunication application on multiprocessor Systems-on-Chip ----143-150
Daniela Genius, Etienne Faure, Nicolas Pouillon

Application of advanced logic synthesis in FPGA-based implementations of digital filters--------------------------151-158
Mariusz Rawski, Pawel Tomaszewicz, Bogdan Falkowski, Tadeusz Luba

Hardware implementation of a multi-mode hash architecture
for MD5, SHA-1 and SHA-2 ---------------------------------------------------------------159-165
Sylvain Ducloyer, Romain Vaslin, Guy Gogniat, Eduardo Wanderley

SESSION EUROPEAN PROJECTS

Evaluation of dynamic partial reconfiguration in professional electronics applications ---------------------------------166-173
Philippe Manet, Jean-Didier Legat, Daniel Maufrond, Leonardo Tosi, Marco Di Ciano,
Olivier Muterre, Pol Cuvelier, Denis Aulagnier, Christian Gamrat, Raffaele Liberati,
Vincenzo La Barba

Developing a context-aware architecture in DySCAS ------------------------------------------174-181
DeJiu Chen, Richard Anthony, Martin Törngren, Gerrit de Boer

Perplexus Project -----------------------------------------------------------------------------182-189
Olivier Brousse, Gilles Sassatelli, Thierry Gil, Lionnel Torres, Michel Robert

SESSION POSTERS

HLS-based Implementation of Real-time face detection parallel architecture -------190-196
N. Farrugia, F. Mamalet, S. Roux, F. Yang, M. Paindavoine

Algorithm-Architecture Affinity: Parallelism Changes the Picture -----------------------197-200
R. Abildgrend, A. Saramentovas, P. Ruzgys, Y. Le Moullec, P. Koch

Analysis and modeling of the influence at distributing an application on a Time Triggered Architecture ---------------------------------201-208
D. Kant, M. Bulmann

O. Sentieys, O. Berder, P. Quémerais, M. Cartron

Self-Organization in Networked Reconfigurable Systems for
Computer Vision Applications ---------------------------------------------------------------214-220
K. Drechsler, J. Schulte, D. Murr, F. Mühlbauer, C. Bobda
Framework for Fast Performance Evaluation of Flexible Models Applied to Interconnect
Delay ------------------------------------------------------221-228
A. Vander Biest, D. Milojevic, A. Richard, F. Robert

Speed-up run-time reconfiguration implementation on FPGAs ---------------229-236
F Berthelot, D Houzet, F Nouvel

An ultra-low voltage high gain operational transconductance amplifier for biomedical
applications -------------------------------------------------------------237-240
F. Bautista, S. O. Martinez, G. Dieck-Assad, O. Rossetto

Interconnected distributed RAM in SynDEx -------------------------------241-245
A. Maccari, J.F. Nezan, F. Urban, M. Raulet

On Modeling the RapidIO communication link using the AAA Methodology ------246-251
S. Moreau, S. Aridhi, M. Raulet, J.F. Nezan

Configurable Instruction Architecture for Q/IQ on the Altera Nios II Processor ------252-255
A. Ben Atitallah, P. Kadionik, N. Masmoudi, H. Levi

A triple path nonlinear phase frequency detector and resistor scalar schemes using in PLLS
for fast pull-in ----------------------------------------------------------256-262
M. Lin, A.T. Erdogan, T. Arslan, A. Stoica

OFDM/OQAM transceiver implementation -------------------------------263-267
C. Sahnine, J.P. Javaudin, G. Degoulet, B. Jahan

Performance estimation of communication processors using design of experiments - 268-274
D. Monnereau, D. Bolot, D. Ragot, N. Julien, E. Senn

ECG Signal Artifacts Suppression in MRI Environment by Means of
LMS filtering -------------------------------------------------------------275-280

Data Path Optimization using Redundant Arithmetic and Pattern Matching --------281-288
S. Belloeal, R. Chotin-Avot, H. Mehrez
INVITED TALK 1

Pr Tughrul Arslan, University of Edinburgh
Design Methodologies and Architectures for a Multi-standard Age.

The talk will focus on the emergence of new standards, specially for wireless communication and imaging targeting mobile hand held devices. The talk will then describe State of Art on the design of architectures targeting multiple standards and satisfying the strict performance criteria in terms of power and throughput required to run such standards on power constrained mobile devices. Examples will be provided for both generic and application specific cases.
INVITED TALK 2

Dr. Mladen Berekovic, IMEC
Reconfigurable Architectures for Low-Power Nomadic DSP Applications.

Nowadays Digital Signal Processors are being used in a variety of nomadic applications where they are increasingly replacing hardwired logic. However, especially in applications with stringent requirements on the power consumption, the move to programmable DSP architectures still poses major challenges. This talk will present how coarse-grain reconfigurable architecture (CGRAs) techniques can be used to combine high performance signal processing capabilities and power efficiency for typical nomadic DSP applications. We will show that CGRAs can beat state-of-the art (SoA) VLIW DSPs by a factor of 4-10. The talk will give an outlook on characteristic problems that digital designs will face in future CMOS process technologies.
INVITED TALK 3

Alain Clouard, ST
New benefits of advanced hardware/software codesign and verification of multimedia System-On-Chip (SOCs) exploiting TLM and SPIRIT standards.

High complexity and short time-to-market of new generation multimedia SOC's are driving industry leaders to more and more early modelling at abstraction levels way above VHDL/Verilog RTL. The talk will describe tremendous benefits that are observed using TLM modeling in SystemC for bittrue, cycle-less modeling across the teams contributing to SOC architecture, design, verification, firmware and software development. Simultaneous usage of XML SPIRIT standard further increases the flow efficiency.
Multimode architecture design

Emmanuel Casseau  
IRISA Lab./R2D2  
CNRS UMR 6074  
Lannion, France  
Emmanuel.casseau@irisa.fr

Shafqat Khan  
IRISA Lab./R2D2  
CNRS UMR 6074  
Lannion, France  
Shafqat.khan@irisa.fr

Bertrand Le Gal  
IMS Lab.  
CNRS UMR 5218  
Talence, France  
bertrand.legal@ims-bordeaux.fr

Willy Aubry  
IMS Lab.  
CNRS UMR 5218  
Talence, France

Abstract: In a mobile society, more and more devices need to continuously adapt to changing environments. Such modeswitches can be smoothly done in software using a general purpose or digital signal processor. However hardware components can cope with throughput and power constraints. Reconfigurable hardware technologies offer partial reconfiguration at runtime but require too long reconfiguration times to rapidly changing applications and they are usually not power efficient for mobile devices. In this paper we propose a methodology to implement multiple configuration (or mode) and multi-constraint systems into a single circuit using conventional hardware technologies.

Keywords: Digital signal processing, embedded systems, flexible devices, high-level synthesis, reconfigurable systems, system on chip.

I. INTRODUCTION

In many applications, such as personal computers or high-performance computing, the main focus of processor design is to increase the performance of the processor when running a wide set of applications. Fast growing increase in the development of DSP, multimedia and communication applications causes the widespread research on efficient platforms for integrating complex systems on a chip. When the specifications for any application are given, it can be executed either in a general purpose processor or in ASIC or in FPGA. A general purpose processor’s hardware contains all the basic blocks needed to build any logic of mathematical function imaginable but the limitations are memory latencies, limits in the parallelism available in the program, i.e. performance, and power consumption. ASIC allows designers to optimize the hardware for one or more parameters. However due to lack of flexibility, ASIC can not cope with the evolving standards and applications of today’s world. FPGA provides flexibility but at the cost of large performance, area, power and reconfiguration time penalties. So the search for flexibility and performance at the same time is the requirement in digital design.

Keeping in view the requirement of digital design, multimode systems are proposed to realize a set of selected configurations into a single system. The main intention of multimode systems is to implement multiple configurations or modes using conventional hardware technologies. Although the reconfigurability of multimode systems is not as much high as of FPGA but on the fly it can be very efficiently reconfigured to a set of configurations for which it is designed for. Multimode systems provide both reconfigurability and efficiency in terms of area, performance, power consumption and reconfiguration time.

One of the goals of multimode system design is to minimize area by reusing resources effectively among different configurations. High-level synthesis (HLS) is an automated process that generates a register transfer level (RTL) architecture from an algorithmic specification and user defined constraints [1,2]. Conventional scheduling and binding algorithms used in HLS can accomplish resource sharing efficiently. The main idea of this paper is to make use of such algorithms in the synthesis of multimode systems. In [3], multi-throughput architecture design has been investigated. In this paper configurations with different kinds of constraints (timing, resources) are targeted.

The paper is organized as follow. Section II presents HLS basics. In section III related work on the implementation of multimode architectures is discussed. Section IV focuses on multimode system features. Our technique for the implementation of multi-constraint multimode system is presented in section V and section VI. Section VII presents first experimental results obtained applying the proposed methodology. Finally conclusions are presented in section VIII.

II. HIGH LEVEL SYNTHESIS

The increasing DSP application complexity coupled to the time to market constraint urge designers to increase the abstraction level of their work using design tools to implement their applications. High-level synthesis is analogous to software compilation transposed to the hardware domain. The source specification that models the algorithmic behavior of the application to be implemented is written in a high-level language (Matlab, C, SystemC, etc.). An automatic refinement process allows the mapping of the described behavior onto a targeted technology and depending on user defined constraints. A design flow including high-level synthesis thus allows fast algorithm implementations. It especially claims to speed up design time versus register transfer level hand coding.

HLS is a constraint-based synthesis flow (Figure 1): hardware resources are selected from libraries of components designed and characterized for a targeted technology. HLS can also be constrained to limit the hardware complexity (i.e. the number of allocated
In order to get better performance of a system, the best option is hardware implementation rather than software implementation. For hardware implementation system on chip (SOC) is becoming more and more popular. However in several implementation of SOC the component utilization is low and a lot of people are working to increase the component utilization in SOC designs. One of the options to increase component utilization is multimode architecture i.e., a single architecture to execute several applications (so called configurations or modes). One way to obtain multimode architectures is through designer knowledge and experience to identify similar patterns in different configurations and design the circuit in such a way to utilize the similar patterns again and again for the different configurations. In order to automate the design of multimode architectures, designers have tried to formalize methodologies for incorporating additional steps at different levels in conventional HLS processes.

An approach for the development of multimode systems was proposed by Kumar and al. [4]. This technique makes use of small scale reconfigurability along with high-level synthesis of multimode systems. In small scale reconfigurability the chip area is divided into fixed logic and reconfigurable logic areas. High performance along with reconfigurability is achieved by implementing most of the circuitry in the fixed logic and only small portion in the reconfigurable area. Although the design will not be fully reconfigurable like FPGA, this will incorporate the reconfigurability up to the desired extend. In the proposed process for the synthesis of multimode systems, data flow graphs of each configuration are first traversed for critical path and slake analysis (slake is the difference between latency constraint and critical path of a particular DFG). Each DFG is then scheduled in the order of increasing slake. Control step matching is done to increase the resource sharing within control steps. Then resource binding is done in the order of decreasing resource usage. Finally the controller is designed to drive the system smoothly.

To cope with the reconfigurability required by the previous approach, another approach proposed by Chiu and al.[5] for obtaining multimode architectures consists of representing different configurations with data flow graphs (DFGs) and then schedule each DFG separately under given resource/timing constraints. These scheduled DFGs are then concatenated using dummy nodes to obtain a unified DFG. In the next step resource binding of the concatenated DFG is performed using maximal weighted bipartite matching algorithm. This technique ensures that resource utilization will be high in the final design. However this technology does not take care of the complexity of the controller. Both controller cost and interconnect increase have been taken into account recently in [3]. DFGs are first scheduled individually. DFG scheduling is sorted such as the DFG with the larger number of compatible operations is scheduled first and so on. The approach aims at maximizing the similarities within each control step to minimize the controller complexity. Resource binding is completed after the scheduling of every mode based on a bipartite weighted matching algorithm which aims at maximizing similarities with previously bound DFGs to minimize interconnect overhead. This datapath and control step similarity-based approach provides promising results and is carried on in

\[ Y = A \times (X + B) \]

Fig. 1 High-level synthesis design flow

II. RELATED WORK:

In order to get better performance of a system, the best option is hardware implementation rather than software implementation. For hardware implementation system on chip (SOC) is becoming more and more popular. However in several implementation of SOC the component utilization is low and a lot of people are working to increase

1 Compatible operations are operations which can be executed by a same arithmetic operator.
this paper. Instead of handling DFGs in a sequential way, we propose to process the scheduling of the DFGs all together whatever the configuration.

IV. MULTIMODE SYSTEMS

IV.1. INTRODUCTION

The input specification for a multimode system consists of a set of data flow graphs corresponding to the set of configurations for which the multimode system is to be designed. In a DFG, nodes represent operations and edges define the data dependencies between nodes. We can represent a single mode system by only one DFG and a multimode system by a set of DFGs, each one corresponding to a particular configuration of the system.

Area minimization in a multimode architecture is obtained through resource sharing between different configurations. But sometimes penalties which occur due to the sharing of resources may overshadow the advantages of multimode systems. This extra cost involves a controller complexity increase and extra steering logic in the datapath. Resource in general means computational, storage or interconnect elements. In order to explain how resource sharing is increased in a multimode architecture, let us take a system that can handle two modes. Each mode can be represented by its DFG as shown in Figure 2. We assume that the applications corresponding to DFG 1 and DFG 2 are timewise mutually exclusive so the datapath and the controller of the multimode system should be such that it can implement either mode.

![DFGs of each mode](image)

Fig. 2 DFGs of each mode

We first assume that both DFG1 and DFG2 are implemented separately that is to say in a single mode way. If maximum performance is targeted then DFG1 would require two multipliers (MULT) and one adder (ADD) and DFG2 would require one MULT and one ADD. So the total number of arithmetic components required to implement these two DFGs in a single mode architecture is three MULTs and two ADDs. Let us schedule DFG 1 and DFG 2 using list scheduling [1] under throughput constraints. Assuming that ADD latency is 1 cycle and MULT latency is 2 cycles then latency constraint on DFG 1 and DFG 2 are five control steps each. Scheduling of DFG 1 and DFG 2 are shown in Figure 3.

As the tasks are timewise mutually exclusive, they can be implemented onto a multimode architecture in which components may be shared by both tasks. Such an implementation would require only two MULT and one ADD, and the proper mode would be invoked based on the task to be executed. For the time being if we ignore the extra cost of the steering logic and the controller complexity due to the sharing of components between the two configurations it is clear that the number of arithmetic components required is reduced which leads to a reduction in area and power consumption. However in current digital signal processing designs the extra cost of the steering logic and the controller complexity due to resource sharing reduces the benefits of such architectures. Therefore while designing multimode systems further steps are necessary to reduce these extra costs.

![DFG scheduling](image)

Fig. 3 DFG scheduling

Similarly storage elements can be shared between the different configurations of the multimode system. In Figure 3 let us assume that inputs to DFG1 and DFG2 are registered inputs. So a, b, c and d are registered inputs to node V1 and V2 of DFG1 and similarly e, f, g and h are registered inputs to node N1 and N2 of DFG 2. If both DFG1 and DFG2 are implemented separately then eight registers are required to store these inputs. However with a multimode architecture only four input registers are necessary because of the sharing of registers between ‘a’ and ‘e’, ‘b’ and ‘f’, ‘c’ and ‘g’, ‘d’ and ‘h’. In fact, register sharing can be applied between each mode whatever the kind of data to be stored (input, output, variable, constant). Thus the number of storage elements can be reduced. However the effect of the increase of the controller complexity due to the loading signals of the merged registers as well as the extra interconnect elements have to

- 17 -
be taken into account during the multimode architecture synthesis.

IV.2. INTERCONNECT COMPLEXITY

Every data transfer needs an interconnection path from its source to its sink. Interconnection may includes multiplexors, buses and wire links etc. Two data transfers can share all or part of an interconnection path if they do not take place simultaneously. During the interconnection binding, the main objective is to maximize the sharing of interconnection elements and thus minimize the interconnection cost, while still supporting the conflict free data transfers required by the application. In the designing of multimode systems, whenever the resources are shared between different configurations it is obvious that extra multiplexer/buses in front of datapath can be required to support this sharing. For example, in the case of the shared input registers for a, b, c, d and e, f, g, h, of DFG1 and DFG2, since they are not connected to the same arithmetic resources (a and b are linked to a multiplier whereas e and f are linked to an adder), additional interconnect elements are required. These extra elements or steering logic increase area, consume power and cause extra delay in the design. It is thus necessary to take into account this extra cost.

There are several methods to reduce the steering logic in multimode systems. One of them is subgraph or pattern identification and matching, in which recurring patterns are identified in the DFGs of different configurations. Then these patterns along with standard RTL components are used as library components and used in the design whenever required. The main advantage of this scheme is that instead of designing steering logic for basic operators again and again it is easy to design the optimized pattern once and use it whenever need arises. Although this method gives promising results in terms of steering logic reduction, it requires costly extra effort for pattern identification and matching [6]. Therefore, additional care has to be taken during the high-level synthesis process of multimode systems to reduce the impact of extra interconnect cost. That is to say specific scheduling and binding algorithms are to be performed.

IV.3. CONTROLLER DESIGN

The controller is usually composed of a finite state machine (FSM) and a decoder which steers the multiplexers and loads the registers depending upon the tasks (transfers, storages) to be executed. The complexity of a controller for a multimode system can be relatively high compared to a single mode system. As discussed earlier due to the sharing of components (arithmetic components and registers), multimode systems use more muxes than single mode systems. As the number of muxes increases the steering signals generated by the controller also increase. This will increase the complexity of the controller and therefore its cost.

Another reason for the increased complexity of the controller is the register sharing. When the registers are shared between modes, the merged register load commands is the OR logic function of the original single mode register load commands. Due to the lack of similarities within the control steps, extra logic may be required for the merged register load commands, which will increase the complexity of the decoder part of the controller. However this extra control logic can be overcome by making a suitable scheduling and binding. For example if the scheduling is done in such a way to maximize the similarities between operations of the different configurations within each control step then no extra logic gate may be required for the merged register load commands.

V. MULTI-CONSTRAINT MULTIMODE SYSTEM DESIGN FLOW

V.1. OVERVIEW

In this section we propose an approach for the generation of multimode architectures that can support different kinds of constraints corresponding to the different configurations. For example, if we have two configurations, one can be optimized for area and the other for performance. Figure 4 shows the proposed design flow.

![Multimode system design flow](image)

The design flow consists of three major steps:

1) Analysis of the applications: In this step the analysis of each mode which need to be integrated is done independently. The constraints for each application are also analyzed independently. The algorithmic description of each application is transformed into a formal model of representation such as a data flow graph DFG or a Control data flow graph CDFG [7]. The
set of constraints (timing, resources, ...) imposed by the
designer for each application is added to the flow graph
model. From this model the mobility of each node
corresponding to the different applications is found out
in an independent way by calculating the as soon as
possible (ASAP) and, if a performance constraint is
targeted, as late as possible (ALAP) execution times.

2) Merging of the models of representation: In this step,
all the annotated flow graphs are merged in a single
flow graph.
3) Unified behavioural synthesis: In this step, the high-
level synthesis process is applied to the unified model of
representation. HLS not only meets all the constrains
corresponding to the different applications but also try to
minimize the total cost of the multimode architecture
(area, power consumption).

V.2. DESIGN FLOW

The objective of the approach suggested in this paper
is to implement multimode systems using high-level
synthesis algorithms. In this section we explain the design
flow assuming the implementation of two configurations is
considered. We assume the behaviour of configuration 1
can be depicted by equation (1) and the behaviour of
configuration 2 by equation (2).

\[
E = (A \times B) + (C \times D) - 2
\]
\[
J = 3 \times ((F \times G) + (H - I))
\]

V.2.1 Graph modelling

To carry out the process of multimode architecture
synthesis the applications are modelled by their data flow
graph DFG (Figure 5).

![Data flow graph for configurations 1 and 2](image)

Fig. 5 Data flow graph for configurations 1 and 2

The configurations may have different constraints (i.e. timing, resources). The management of these constraints
will be done during the scheduling and binding steps of the
HLS process. In this example, we assume the main
emphasis during the synthesis of these two configurations is
to meet a timing constraint for configuration 1 and to
implement configuration 2 with limited resources and with
minimum latency.

The synthesis of the multimode system begins with the
selection of the operators required to implement the set of
operations of the two modes. This selection is based on the
library depending upon different optimization (speed, area
every). For simplicity we assume that for each operation there
is only one type of operator available. Sometimes, in order
to optimize the architectures, multifunction operators whose
area, power consumption).

The synthesis of the multimode system begins with the
selection of the operators required to implement the set of
operations of the two modes. This selection is based on the
library depending upon different optimization (speed, area
every). For simplicity we assume that for each operation there
is only one type of operator available. Sometimes, in order
to optimize the architectures, multifunction operators whose
area, power consumption).

V.2. DESIGN FLOW

The objective of the approach suggested in this paper
is to implement multimode systems using high-level
synthesis algorithms. In this section we explain the design
flow assuming the implementation of two configurations is
considered. We assume the behaviour of configuration 1
can be depicted by equation (1) and the behaviour of
configuration 2 by equation (2).

\[
E = (A \times B) + (C \times D) - 2
\]
\[
J = 3 \times ((F \times G) + (H - I))
\]

V.2.1 Graph modelling

To carry out the process of multimode architecture
synthesis the applications are modelled by their data flow
graph DFG (Figure 5).

![Data flow graph for configurations 1 and 2](image)

Fig. 5 Data flow graph for configurations 1 and 2

The configurations may have different constraints (i.e. timing, resources). The management of these constraints
will be done during the scheduling and binding steps of the
HLS process. In this example, we assume the main
emphasis during the synthesis of these two configurations is
to meet a timing constraint for configuration 1 and to
implement configuration 2 with limited resources and with
minimum latency.

The synthesis of the multimode system begins with the
selection of the operators required to implement the set of
operations of the two modes. This selection is based on the
library depending upon different optimization (speed, area
every). For simplicity we assume that for each operation there
is only one type of operator available. Sometimes, in order
to optimize the architectures, multifunction operators whose
area, power consumption).

V.2. DESIGN FLOW

The objective of the approach suggested in this paper
is to implement multimode systems using high-level
synthesis algorithms. In this section we explain the design
flow assuming the implementation of two configurations is
considered. We assume the behaviour of configuration 1
can be depicted by equation (1) and the behaviour of
configuration 2 by equation (2).

\[
E = (A \times B) + (C \times D) - 2
\]
\[
J = 3 \times ((F \times G) + (H - I))
\]

V.2.1 Graph modelling

To carry out the process of multimode architecture
synthesis the applications are modelled by their data flow
graph DFG (Figure 5).

![Data flow graph for configurations 1 and 2](image)

Fig. 5 Data flow graph for configurations 1 and 2

The configurations may have different constraints (i.e. timing, resources). The management of these constraints
will be done during the scheduling and binding steps of the
HLS process. In this example, we assume the main
emphasis during the synthesis of these two configurations is
to meet a timing constraint for configuration 1 and to
implement configuration 2 with limited resources and with
minimum latency.

The synthesis of the multimode system begins with the
selection of the operators required to implement the set of
operations of the two modes. This selection is based on the
type of components available in the library. It is the task of
designer to provide the library of resources which contains
the operators for performing the different arithmetic
operations, as well as storage components and interconnect
components. In general for each type of arithmetic
operation different types of operators are available in the
library depending upon different optimization (speed, area
every). For simplicity we assume that for each operation there
is only one type of operator available. Sometimes, in order
to optimize the architectures, multifunction operators whose
area, power consumption).

V.2. DESIGN FLOW

The objective of the approach suggested in this paper
is to implement multimode systems using high-level
synthesis algorithms. In this section we explain the design
flow assuming the implementation of two configurations is
considered. We assume the behaviour of configuration 1
can be depicted by equation (1) and the behaviour of
configuration 2 by equation (2).

\[
E = (A \times B) + (C \times D) - 2
\]
\[
J = 3 \times ((F \times G) + (H - I))
\]

V.2.1 Graph modelling

To carry out the process of multimode architecture
synthesis the applications are modelled by their data flow
graph DFG (Figure 5).

![Data flow graph for configurations 1 and 2](image)

Fig. 5 Data flow graph for configurations 1 and 2

The configurations may have different constraints (i.e. timing, resources). The management of these constraints
will be done during the scheduling and binding steps of the
HLS process. In this example, we assume the main
emphasis during the synthesis of these two configurations is
to meet a timing constraint for configuration 1 and to
implement configuration 2 with limited resources and with
minimum latency.

The synthesis of the multimode system begins with the
selection of the operators required to implement the set of
operations of the two modes. This selection is based on the
type of components available in the library. It is the task of
designer to provide the library of resources which contains
the operators for performing the different arithmetic
operations, as well as storage components and interconnect
components. In general for each type of arithmetic
operation different types of operators are available in the
library depending upon different optimization (speed, area
every). For simplicity we assume that for each operation there
is only one type of operator available. Sometimes, in order
to optimize the architectures, multifunction operators whose
area, power consumption).
model keeps its ASAP and ALAP (for application having timing constraint) execution times which were calculated previously. After this step, a model of representation which contains all the operations to be performed for all the possible scenarios as well as their mobility is thus available.

![Fig. 7 Unified model of representation](image)

V.2.2 Synthesis process

A. Resource allocation

The main objective of the resource allocation is to calculate the number of arithmetic resources required to implement the final architecture while meeting the constraints imposed on each mode. In order to respect the constraints corresponding to each configuration, allocation is first done individually for each configuration, that is to say allocation is done for each mutually exclusive branch of the unified model. In a second step, the allocated resources are combined to obtain the number of resources needed for the implementation of the set of applications. A list of arithmetic resources available for the scheduling and binding step is then provided. Since the configuration are timewise mutually exclusive, all the resources are then shared without distinction of configuration (equation 3).

\[
\text{Number(resource\_type\_i)=Max\{number(resource\_type\_i) for each mutually exclusive branch\}}
\]  

(3)

In the case of a configuration to be synthesised under a resource constraint, it is the task of the designer to specify the number of each type of arithmetic resources. On the contrary, in the case of a configuration to be synthesised under a timing constraint, the allocation is performed by the synthesis process. There are different techniques available in the literature for the allocation of resources under a timing constraint. Usually allocation of resources is done after applying a force directed scheduling [8]. In our case, resources are shared between the different modes thus allocation is to be done before the scheduling. The allocation thus consists of allocating operators according to the average number of operations per cycle. This estimation of the number of resources is either exact or lower than the actual requirements. It is thus assumed that during the scheduling of a particular configuration the number of allocated hardware resources can increase in order to meet the timing constraints. In that case, no re-scheduling is realized in order to obtain a solution in a short run-time. The set of resources is increased and the scheduling of the next operations benefits of this update. For each configuration, the complexity of the allocation phase is \(O(n)\) where \(n\) represents the number of operations of the configuration. If the parallelism of the application is uniformly distributed all along the application then the number of operators allocated is optimal for this application.

B. Scheduling and binding

In a multimode architecture, similarities in operations among the modes within the control step and datapath similarities play a very important role in the reduction of the controller and datapath overheads. One of the objectives of the scheduling of the proposed multimode architecture synthesis flow is to manage the parallelism of the flow graph while respecting the synthesis constraints of the different modes. Another objective is to optimize the use of operators while minimizing the interconnect overhead. The scheduling algorithm used is based on list scheduling [1]. Main priority function is based on mobility. In [9], to reduce the latency of HLS generated mono-mode architectures which are constrained by the number of resources, successor and predecessor information are used for ordering contesting operations that have equal priority value. In the proposed synthesis process, such kind of information is also used but in order to limit the extra cost due to inter-mode resource sharing. Binding is performed at the same time to maintain this feature.

The basics of the scheduling algorithm are the following. For timing constrained configurations, the priority function is based on the mobility and the number of immediate successors. For resource constrained configurations, the priority function is based on the number of immediate successors that can be fired and the distance from sink. For the scheduling of each control step, first of all we search for the operation which has the highest priority among the list of all eligible operations of the unified graph whatever the configuration. This operation is scheduled. Then we search in each different configuration the eligible operations which can be scheduled in this control step. For this purpose, the list of eligible operations is processed to find out whether there is any operation which is compatible with the operation which has just been scheduled in this control step. First aim is to maximize the similarities within each control step to minimize the controller complexity. For example let us assume we have already scheduled one addition operation of any configuration on an ADD operator in a particular control step. Then another add operation which can be scheduled in this control step belonging to one other configuration (timewise mutually exclusive) will be search. Then, in order to minimize the interconnect cost, the intention is to schedule, for each configuration, that compatible operation...
whose cost of merging is smaller to share the associated resource compare to another compatible operation. In that case, the priority function is based on the mobility and successor and predecessor information.

As said before, the number of resources is increased on the fly if necessary for the scheduling of zero-mobility operations. When every zero-mobility operation has been scheduled and if they are available resources left, then eligible operations are scheduled according to the principle above.

Then next control step is considered for the scheduling of new eligible operations.

\[ s(n) = x(0) - \sum_{i=1}^{N-1} x(i) \] (6)

We have decided to implement a 256-tap FIR filter, a 128-tap LMS filter and a tree composed of 64 data to subtract. Resources (arithmetic operators, register and interconnect components) were implemented with 32 bits. IEEE754 floating-point representation is used for arithmetic operators.

A profiling step of the functional complexity of the three algorithms has been done based on simulations to estimate the a priori fusion interest of these applications. Results are presented in table 1. The FIR and the LMS filter complexities are similar in term of number of computations whereas the subtraction tree is quite different. It should be mentioned that from an implementation point of view, during their execution FIR and LMS filter timing repartitions are different. It means resource sharing will be different.

<table>
<thead>
<tr>
<th>Application</th>
<th>ADD</th>
<th>MULT</th>
<th>SUB</th>
</tr>
</thead>
<tbody>
<tr>
<td>256 - FIR</td>
<td>255</td>
<td>256</td>
<td>0</td>
</tr>
<tr>
<td>128 - LMS</td>
<td>255</td>
<td>256</td>
<td>1</td>
</tr>
<tr>
<td>64 - Tree</td>
<td>0</td>
<td>0</td>
<td>63</td>
</tr>
</tbody>
</table>

Table 1. Mono-mode complexity

Three different kinds of constraints have been applied:
- the FIR filter synthesis is area constrained with 3 adders and 3 multipliers,
- the LMS synthesis is time constrained under 70 clock cycles,
- the subtraction tree synthesis is “unconstrained”, that is to say an “as fast as possible” implementation using resources of other configurations is targeted.

The synthesis of each application with a mono-mode HLS design flow using the GraphLab tool has been first performed. These syntheses have been performed respecting strictly the designer constraints presented above.

For the LMS filter, 4 adders, 4 multipliers and 1 subtractor are necessary to satisfy the timing constraint. Only one subtractor is used for the subtraction tree (the subtractor that will be allocated for the LMS synthesis). Then the logical synthesis has been performed. Area results and implementation latencies are presented in table 2. Area results correspond to the complete architecture, i.e. the data-path, its controller, the memory unit and the memory banks. Latency corresponds to the number of clock cycles.

We have then applied our multi-mode design flow to implement theses applications in a single architecture. The synthesis was performed on a single graph model.
corresponding to the 3 applications and their own constraints. The design area and its comparison to the mono-mode components are presented in table 3.

<table>
<thead>
<tr>
<th>Resources</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Area (slices)</td>
</tr>
<tr>
<td>FIR</td>
<td>2933</td>
</tr>
<tr>
<td>LMS</td>
<td>3914</td>
</tr>
<tr>
<td>Tree</td>
<td>495</td>
</tr>
</tbody>
</table>

Table 2. Synthesis results for mono-mode components

<table>
<thead>
<tr>
<th>Resources</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Add</td>
</tr>
<tr>
<td>Multi-modes</td>
<td>4</td>
</tr>
<tr>
<td>3 IP</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 3. Synthesis results

The multi-mode architecture is 27% smaller than the sum of the mono-mode architectures. In the same time, it is important to underline that the FIR filter has been also speed-up compared to its mono-mode implementation: the latency falls down from 89 clock cycles to 67 cycles. This speed-up is issued from the number of previously allocated arithmetic resources for the LMS filter which are shared between all the applications integrated in the multi-mode component (inter-mode component sharing).

If we now consider the same performance for both mono-mode and multimode applications we obtain the results presented in table 4. We can notice that the FIR area has increased. Using these results, the final size for the 3 components having the same timing performance as the multimode component is about 8100 slices without including their input and output ports. The same solution using a multi-mode component is 5375 slices providing a 34% area saving.

<table>
<thead>
<tr>
<th>Resources</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Area (slices)</td>
</tr>
<tr>
<td>FIR</td>
<td>3722</td>
</tr>
<tr>
<td>LMS</td>
<td>3914</td>
</tr>
<tr>
<td>Tree</td>
<td>495</td>
</tr>
</tbody>
</table>

Table 4. Synthesis results for mono-mode components targeting multimode architecture performance

These first experiments with simple DSP applications show that the multi-mode design flow permit to save area versus mono-mode design flow but also permit to speed up some applications with hardware sharing.

VI. CONCLUSION

In this paper a methodology for implementing different applications into a single architecture is described. The methodology is based on high-level synthesis. Due to the sharing of resources among different configurations this methodology provides an efficient performance/area/reconfigurability trade-off. The different configurations of the multimode architecture can be optimized for heterogeneous constraints (latency, number of resources, ..) which further increases the interest of the proposed design flow. The design of multimode systems implies extra costs mainly due to the increase of the complexity of the controller and extra steering logic in the datapath. They can be reduced by introducing some additional steps in conventional algorithms used in high-level synthesis. Work in progress focuses on the synthesis of more complex DSP applications. Binding improvements are also investigated to reduce as much as possible area and power consumption.

REFERENCES

A High Level Generic Application Analysis Methodology For Early Design Space Exploration

Muhammad Rashid
Thomson Silicon Components
Rennes, France

&
Université de Bretagne Occidentale
Brest, France
Email: muhamad.rashid@thomson.net

Thierry Goubier
Université de Bretagne Occidentale
Brest, France
Email: thierry.goubier@gmail.com

Bernard Pottier
Université de Bretagne Occidentale
Brest, France
Email: pottier@univ-brest.fr

Abstract—The software implementations of sophisticated multimedia applications/algorithms are often huge and it is virtually impossible to analyze these applications without generic automated tools and appropriate methodologies. Architectural implementation choices for these applications based on merely designer experience without objective measures can lead to costly re-design loops. Application analysis at algorithmic level can produce a variety of useful information providing valuable design space exploration indications.

We present a generic application analysis approach based on instrumentation based profiling for early design space exploration. First we transform the source specification of application implemented in a high level language (Smalltalk in this paper) into an internal trace tree representation by dynamic analysis. The trace tree of the source specification is then characterized/explorated at the algorithmic level. The results of characterization provide guidelines to the designer to select target architecture(s) for the application. These guidelines include memory, control and processing orientations as well as the inherited spatial parallelism in the specification. The aim is to improve application architecture matching by bridging the gap between application specification and target architecture.

As a case study, we have taken MPEG-2 decoder implemented in Smalltalk. Experimental results show the applicability of the the proposed methodology for early design space exploration.

I. INTRODUCTION

The exponential growth of VLSI technology is yielding more and more powerful multicore architectures enabling massive integration of processing units and fast communication channels on a single chip [1] [2]. A leading example of this industrial trend is the Cell processor from IBM/Toshiba/Sony that has 9 cores [3]. Cisco has described a next-generation network processor containing 192 Tensilica Xtensa cores [4]. The performance gains in these multicore architectures depend on effective application parallelization across the cores. At the same time, the continuous growing effort of developing multimedia standards with higher compression efficiency, better quality of service and more functionalities have resulted in an extremely high level of algorithmic complexity and sophistication [5]. As a result, architectural implementation choices for complex multimedia algorithms based on designer experience without objective measures become extremely difficult or impossible tasks [6]. It may leads to late realization of sub-optimal (or even wrong) solution in the design cycle resulting in costly design iteration. The term algorithmic complexity itself is not well defined [7]. In this paper we consider it in terms of number of arithmetic operations, memory bandwidth, regularity of the algorithm and possibility of parallel processing. Here the term complexity is not used in its strict mathematical definition only considering the size of algorithm minimal descriptions. In a broader sense, we are mainly interested in run-time aspects of algorithm complexity metrics.

In order to make appropriate choices about architectural implementation of the given application at the beginning of the design cycle, it is desirable to measure and understand the algorithmic characteristics/complexity of an application before starting the design of the target architecture [6]. The objective of application exploration is to obtain approximate measures that can identify classes of candidate architectures for the actual implementation. Consequently, an automated and generic application exploration tool in the early design space exploration is required.

In this paper, we propose a high level generic application analysis framework that characterizes the application at an higher abstraction level without any architectural directives. The proposed framework provides desirable characterization results and can be used as a first step in design space exploration. It bridges the gap between specification of a system and the definition of a target (or a set of target) architecture(s) for that system. This characterization information of the application is obtained very early in the design process and can be used in three different ways.

- When neither the specification nor the architecture is fixed, the designer can refine both aspects by using

This work was carried out to address an industrial problem relevant to the early design space exploration for multimedia applications.
characterization results.

In this paper we focus on the second point by presenting a high-level generic framework for application analysis. The basic idea is to take a standard code as an input (without performing any additional effort) and to analyze it for early design space exploration. The starting point is an executable software specification written in Smalltalk [8] which is a high level language. This input specification is then automatically transformed into internal trace tree representation by dynamic analysis of the specification. By dynamic analysis, we obtain valuable information about the run time behavior of the specification in a form of trace tree that represents implementation-independent specification characteristics and provide information about the inherent characteristics of the application. We then present a generic analysis framework to analyze and explore the trace tree (representation of the source specification). By using the information provided by the analysis results, the designer is guided in his architectural choices since he gets an insight of the application behavior.

The exploration of the design space for embedded systems may have different meanings. The proposed analysis framework is generic in a sense that it may extract multiple characteristics/features of the application depending on a particular analysis requirement by simply defining new analysis operations on the trace tree representation of the application. In this paper, we focus on the features at system level (where the target architecture is not yet defined) including exploitation of spatial parallelism and application orientation of the application in terms of processing, control and memory accesses.

This paper is organized as follows: Section 2 describes state of the art in the domain of application characterization for early design space exploration targeting the multimedia/video field. It also summarizes the innovative points of our framework. Section 3 describes proposed generic analysis framework to extract the important characteristics of the application. Section 4 describes some usage scenarios of analysis results and their significance in design space exploration. Section 5 partially evaluates MPEG2 decoder algorithm [9] (2D IDCT, Huffman decoding) to illustrate the proposed analysis methodology. It provides some examples of the analysis results in terms of computational complexity, data flow and inherent spatial parallelism. Finally, section 6 concludes the paper.

II. REVIEW OF RELATED WORK AND CONTRIBUTION

A. Related Work

In [6], some state-of-the-art high-level application analysis approaches for multimedia system design have been comprehensively reviewed including the academic and commercial frameworks. Two main axes are typically recognized: the orientation of the application in terms of processing operations (algorithmic complexity) and amount of inherited spatial parallelism present in the application. So we consider related work in application analysis techniques, application characterization and spatial parallelism.

1) Application Analysis Techniques: It may be static or dynamic. Static analysis techniques yields bounds on run-time best and worst cases [10] [11]. The main drawback of these techniques is that the processing complexity of multimedia algorithms heavily depends on the input data statistics while static analysis can only detect upper and lower bounds [12]. In dynamic analysis [13], alternative solutions are available for tracing a program behavior [14]. It includes source code modification, byte code modification, instrumenting the virtual machine and method wrappers [15] [16] [17]. Instruction level profiling is also a form of dynamic providing information at a relatively high level of abstraction (at function level). However it does not provide the statistics about the processing operations executed by those functions. The information gathered with profilers strictly depends on the underlying machine and on the compiler optimization. This is against the requirement of high level system design in which complexity evaluation depends only on the algorithm itself [18].

2) Application Characterization: A characterization approach to find the algorithmic complexity is presented in [19]. The source code is instrumented and simulated to collect execution counts that capture the dynamic behavior of the application. Specification characteristics are then computed by statically analyzing the code together with the collected dynamic information. The idea is to explore the design space with the results that are accurate enough to prune out infeasible design alternatives. In [18], authors introduce an integrated tool for the complexity analysis of C reference descriptions. The tool is capable of measuring all C language operators during the execution of algorithms. The tool capabilities also include the simulation of virtual memory architectures extending it to data transfer and storage analysis. Simulator can be configured to provide measurements and performances of the algorithm under study on user configured memory architectures. In [20], application specified in systemC is statically analyzed to get some analysis results as well as simulated to get some dynamic information. By combining the static analysis and simulation results, work load is estimated in form of metrics. Then a HW/SW partitioning tool is driven based on these analysis results. The purpose is to characterize application functions according to three kinds of targets: GPP, DSP and ASIC. In [21], a high level exploration methodology based on C language is presented. Applications are characterized using a hierarchical graph-based representation of the application, resulting in a set of metrics. These metrics characterize the application in terms of memory bandwidth, processing parallelism and relative control/processing/data transfers. In [22], a way of measuring arithmetic complexity of multimedia applications is described. However it only measures the number of arithmetic and control operations and ignores other aspects of complexity like number of memory accesses, possibility of parallel operations and so on.

3) Spatial Parallelism: In [23], a profile based technique is presented to extract parallelization from a sequential application. It transforms the source specification to a graph based
representation to identify parallelizable code. As it measures the memory dependencies between different functions of the application so granularity of the extracted parallelism is larger and is not well suited to extract fine grain parallelism. SPRINT [24] tool automatically generates an executable concurrent model in SystemC starting from sequential C code and user defined directives. First, it transforms C code to control flow graph which is further transformed to model the different concurrent tasks. Again this tool only extracts functional parallelism and leaves the extraction of data parallelism. Commit research group from MIT presents a framework [25] that exposes task, data and pipeline parallelism present in an application written in StreamIt [26] (an streaming programming language). It further explains that not all parallelism has equal benefits so it is critical to leverage the right combination of task, data and pipeline parallelism.

B. Contribution

The main innovations of our work versus the state of the art tool technology can be summarized as follows.

1) Source Specification: Almost all the application analysis methodologies cited in this section start with application specifications written in C language. We propose an application analysis framework in which application is specified in Smalltalk. It is a dynamic implicitly-typed language where objects, not variables, carry type information, freeing the programmer from declaring variable types. The combination of a polymorphic, pure object-oriented language which is simple yet powerful with an incremental programming environment and a large robust class library make Smalltalk an attractive choice for building complex systems that must adapt to the changing needs of embedded system design [8].

2) Instrumentation Techniques: All the instrumentation based application analysis approaches for design space exploration instrument the source code but in the proposed framework, parse tree of the source code is instrumented due to its simplicity, control, generality and preservation of original source code semantics. Smalltalk is a reflective programming language, whereby the objects that define the language are themselves built with the language. It allows the programmer to extend the language and environment in a way that is unparalleled in conventional programming environments like C. Consequently, the proposed framework advocates to instrument on a parse tree rather than any other form.

3) Generic Application Analysis: In most of the cases , analysis results are summarized/restricted to only some special design metrics [21] [18] [23]. According to the fact that design space exploration of the embedded systems may have different requirements like extracting inherited spatial parallelism, analyzing application orientation, code optimizations etc, a generic analysis framework is more attractive choice keeping in mind the increasing heterogeneity of applications and architectures [4]. Our analysis framework is generic and can be extended to fulfill multiple requirements of design space exploration by simply defining new operations on the trace tree representation of the source specification.

III. PROPOSED ANALYSIS FRAMEWORK

Figure 1 show the proposed framework which is divided into two parts. The first part is related to the transformation of source specification (written in Smalltalk) into an intermediate trace tree representation. The second part is related to analyze or explore the trace tree of the source specification to get desirable analysis results. We describe the two parts of the proposed framework in the following paragraphs without going into implementation details as it is not the main focus of this paper. We refer interested readers to [27] for the implementation details of the proposed framework.

![Fig. 1. The Skeleton Of Analysis Framework](image)

### A. Source Specification Transformation into Trace Tree

The first step is to write the source specification in Smalltalk enabling the designer to specify and program algorithms at a high level of abstraction (executable specifications) very early in the design cycle. Executable specifications in Smalltalk are high level programs, far simpler than their equivalent implementation on heterogeneous SoCs. In order to transform source specification into a trace tree (intermediate internal representation) that contains information about the execution of an application at run time, dynamic analysis is used as
the code instrumentation mechanism that allows insertion of code to monitor and track the runtime behavior of the source specification of the application. It enables to extract valuable information about how the specifications works at runtime [14]. The run time information is recorded in a form of trace tree representing implementation independent specification characteristics. Each function in the application is transformed into trace tree representation. Individual traces for each function are then combined to get the final trace tree of the application that characterizes the complete application [27].

1) Steps Of Transformation Process: First part of figure 1 summarizes the required steps of source specification transformation into trace tree. These sub-steps are:

- **Instrumentors** generate probing messages for recorders in four sub-steps. The first sub-step is to parse source code and generate an abstract syntax tree (parse tree). The second sub-step is to instrument (alter) the parse tree to generate a new parse tree with additional nodes. The third sub-step is to compile the instrumented syntax tree. The output of this sub-step is a compiled method. The fourth and last sub-step is to replace the original source code with the compiled source code. The implementation details of these sub-steps can be seen in [27].
- The output of Instrumentors is in the form of probing messages. The **Recorder** answers these messages and create events on reception of probes activation messages. These events are **RecordBlock**, **RecordItem**, **RecordMethod**, **RecordVariable** and **RecordAssignment** as shown in figure 2.
- **Visualizers** are responsible to bind each event to the original source code.

2) Trace Tree: The output of the first part of the framework is a trace tree which represents the sequence of recorded events, in a tree-like form. A typical use of the trace tree is to hierarchically show the structure of function calls. The basic entity of the trace tree is an **Abstract Record** which in turn has its sub entities (children) as shown in figure 2.

![Fig. 2. Record Hierarchy in Trace Tree](image)

- The **AbstractRecord** is the super class of all the nodes in a trace tree. It is an abstract class and does not have any instances. However, all the entities in a trace tree are the subclasses of **AbstractRecord**. It contains the common behavior of all the entities in a trace tree.
- Every operation (computation, memory transfer, control) in a trace tree is recorded as **RecordItem**.
- The top level of every function in a trace tree is recorded as **RecordMethod**.
- Every variable in a trace tree is recorded as **RecordVariable**.
- Every assignment to the variable in a trace tree is recorded as **RecordAssignment**.

The detail description of each entity like **RecordBlock**, **RecordItem**, **RecordMethod** and **RecordVariable** in a trace tree can be found in [27].

B. Trace Tree Analysis

Once source specification is transformed into trace tree representation, we perform operations on trace tree for different types of analysis. These analysis operations may be, for example, checking the value assigned to each variable in each step of the program execution. In the context of code rewriting, one may perform operations for type-checking, code optimization, flow analysis and so on. These operations are performed on basic entities like **RecordBlock**, **RecordItem**, **RecordMethod** and **RecordVariable** of the trace tree as shown in figure 2.

We keep basic entities of the trace tree and their subclasses independent of the analysis operations that apply to them by packaging related operations from each class in a separate object named as **visitor** and passing it to elements of the trace tree. There are different analysis purposes and for each analysis purpose, there is a separate visitor.

The proposed analysis framework is generic as it is not restricted to a particular set of analysis operations. It allows the designer to extend the framework by defining new analysis operations (visitors) to fulfill different requirements of design space exploration such that for each analysis operation, there is a corresponding visitor for the trace tree making a visitor hierarchy similar to visitors on a parse tree [28]. The root and all elements of a parse tree are **ProgramNode(s)**, while the basic element of a trace tree is **Abstract Record** as shown in figure 2.

IV. Usage Scenarios of Analysis Results in Design Space Exploration

We have mentioned in the introductory part of this paper that design space exploration for embedded systems may have different requirements due to the growing heterogeneity of applications and architectures [3]. In section III, we have explained that we can perform multiple analysis operations on trace tree representation of the source specification to build a generic analysis framework. Each analysis operation in our proposed framework depends on a particular design requirement of design space exploration. These analysis operations may be for extracting spatial parallelism, analyzing application orientation, type-checking, code optimization and for many other requirements. To illustrate these concepts, we perform analysis operations on trace tree representation of the
source specification to extract application characteristics in terms of application orientation [19] [18] [20] and inherited spatial parallelism [21] [23] [25]. In this section, we present a number of the usage scenarios of our analysis results and their significance in the design space exploration.

A. Application Orientation

The orientation of an application gives guidelines about architecture selection. An application may have three types of operations: computations, memory and control. Our analysis results describe the operations in terms of percentages of these three basic types of operations. These types are:

1) Computation Oriented Operations: It includes the arithmetical operations (Addition, Multiplication, Subtraction etc) as well as logical operations (And, Or etc). The higher percentage of this type of operations tells the designer that how much a particular function is computationally intensive. Consequently, designers should more concentrate on computation optimization. There are different types of computation operations. The analysis results show the percentage of each type of computation operations in a function. For example, if most of the operations are multiplications, then target architecture should have dedicated hardware multipliers, hence guiding the designer towards architecture selection.

2) Memory Oriented Operations: The percentage of this type of operations indicates the frequency of memory accesses in a trace tree. The higher percentage of this type of operations tells the designer that a particular function is data access dominated and is most likely to require a high data bandwidth. It indicates that the computations are not performed on previously computed data (reside in local memories) but performed on the input data (data entering to trace tree). Therefore in the case of real time constraints, some efficient mechanism of data movement and high performance memories are required.

3) Control Oriented Operations: The percentage of this type of operations indicates the frequency of control operations in a trace tree. This percentage guides the designer to evaluate the need for complex control structures to implement a function. The functions with high percentage of this types of operations are good candidate for a GPP processor implementation rather than a DSP processor implementation, since the latter is not well suited for control dominated functions. In addition to this, a hardware implementation of these control dominated functions would require large state machines.

B. Spatial Parallelism

Trace tree of a particular function or the complete application shows the existing parallelism among the operations of the function or application. It implies the possibility of mapping different operations/functions to different PEs (processing elements) of the target architecture for concurrent execution. In other words, we can exploit the inherited spatial parallelism present in the application. We represent the amount of average inherited spatial parallelism for every function in the source specification by $P$ [21] such that functions with higher $P$ values are considered as appropriate to architectures with large explicit parallelisms. Functions with lower $P$ value are rather sequential, so the acceleration can only be obtained by exploiting temporal parallelism. $P$ enables the classification of application functions according to their criticality or in other words their capability to exploit the inherited spatial parallelism.

The value of $P$ at any hierarchical level of a trace tree is computed by dividing the total number of operations (RecordItems in a trace tree) by its Critical Path [21]. The Critical Path at any hierarchical level of a trace tree is the number of longest sequential chain of operations (processing, control, memory). It is computed for each hierarchical level.

When we compute the value of $P$ for a hierarchical level in a trace tree, we assume that the parallel execution of sub hierarchical levels is possible and the value of $P$ is given as the ratio between the sum of all operations in the sub hierarchical levels of the node and the longest of all the critical paths. For example, if a node $A$ has three sequential sub nodes $B$, $C$ and $D$ containing 10, 20 and 30 sequential operations respectively. Now the value of $P$ at each sub node $B$, $C$ and $D$ is 1 (as they contain only sequential operations and hence no spatial parallelism) but the value of $P$ at node $A$ is 2 (60 divided by 30) assuming that all the sub nodes can be executed in parallel on different execution units. Functions with highest $P$ (spatial parallelism) value can be first considered since they show the most important optimization potential regarding the acceleration.

C. Guidelines for mapping

The mapping process requires application model (in form of different functions) as well as architecture model (in form of processing elements, interconnections etc) to map the application behavior on the architecture model. Our analysis results enable the designer to identify the most complex functions in terms of computations in an application, which may be the best candidates for mapping to the fastest PEs. The designers also prefer to map the functions which communicate heavily (identified by analysis results) to the same PE or to the PEs connected by dedicated busses.

D. Estimation

The performance estimation of different functions of the application on multiple processing elements (PEs) of the architecture is another important issue in the design space exploration. For example, assuming a function $F_1$ is mapped to processing element PE1. If $F_1$ contains $X$ integer-type multiplication operations (revealed by analysis results), and executing such an operation on PE1 requires $Y$ clock cycles (Known to designer from architecture model), then the execution time of function $F_1$ on processing element PE1 will be $X * Y = XY$ clock cycles.

V. Evaluation Results

A. MPEG-2 Decoder

In this section we give analysis results of some parts of MPEG2 decoder application [9] implemented in Smalltalk [29]
to illustrate our tracing based analysis methodology. MPEG-2 is a well known encoding and decoding standard for digital video. The basic principle is to remove redundant information prior to transformation and re-inserting it at the decoder. There are two types of redundancies: Spatial Redundancy to remove correlation of pixels with their neighboring pixels within the same frame and Temporal Redundancy to remove the correlation of pixels with neighboring pixels across the frames. The MPEG-2 decoder can be summarized in the following three points [9]:

- **Parser** is responsible for parsing the MPEG-2 bit stream and performing Huffman and Variable run-length decoding (VLD). The input to the parser is MPEG-2 bit stream. The output of the parser is an interleaved stream of quantized macro blocks encoded in the frequency-domain, and offset encoded motion vectors. In the following steps, the quantized macro blocks are inverse transformed while motion compensation is performed to decode offset encoded motion vectors.

- **Inverse Transformations** step is due to the spatial redundancy reduction at the MPEG-2 encoder. The inverse transformations map each 8x8 block from the frequency domain back to the spatial domain. Each block is reordered, inversely quantized and then followed by an inverse DCT. Similarly, encoded motion vectors are decoded.

- **Motion Compensation** step is due to the temporal redundancy reduction at the MPEG-2 encoder. It performs the motion compensation to recover predictively coded macro blocks. The motion compensation uses the motion vectors to find a corresponding macro block in a previously decoded reference picture. The reference macro-block is added to the current macro-block to recover the original picture data.

For simplicity, we experiment with 2D Inverse Discrete Cosine Transform and Huffman Decoding to illustrate our analysis approach.

### B. Trace Tree Representation

The figure 3 shows the trace tree representation of a single function and its corresponding source specification.

It consists of two parts. The right hand side shows the original source specification (written in Smalltalk) while the left hand side shows its corresponding trace tree representation showing all the operations/events according to the execution order. The right hand side of the figure 3 highlights the portion of the original source code according to the selected operation in the trace tree on the left hand side. It means one can easily go through the trace tree step by step and visualize all the entries (RecordBlock, RecordItem, RecordVariable, RecordMethod and RecordAssignment) of a trace tree in each step of the source code execution. The symbol \# shows a RecordItem. However, the symbol \# at the top of trace tree shows a RecordMethod. Similarly symbol \{ \} shows a RecordBlock and so on.

An **Actions** menu regroup three commands:

- **Update** refreshes the trace tree display with the current contents of the trace.
- **Reset** resets the trace recorder, removing all recorded events.
- **Uninstall** uninstalls instrumentation from the instrumented methods.

A trace tree is generated for each function in the source specification of the application using the flow in figure 1 and individual trace trees of each function are combined to get the final trace tree of the application that characterizes the complete application [27].

### C. 2D Inverse Discrete Cosine Transform

2D IDCT (for 8x8 image blocks) source specification is first transformed into trace tree using the flow in figure 1. We perform analysis operations on trace tree representation to get analysis results for 2D IDCT. Table I shows the analysis results for different functions in 2D IDCT.

1) **Orientation**: From a structural point of view, 2D IDCT is composed of two identical and sequential 1D-DCT sub-blocks (operating on rows and columns), so the corresponding trace
trees have the same orientation values for both methods as seen in table I.

The first observation is that the percentage of control operations is zero for all the methods, since it is composed of deterministic loops and does not contain any test. Secondly we observe that computation percentage for 2D IDCT functional blocks are higher so it is computation oriented. The results also show a good percentage of memory operations.

Figure 4 shows the percentage of each type of computation in the 2D IDCT. It does not contain any floating point but only integer type operations, i.e. processors with dedicated floating point units are not necessary and processor selection should focus on integer performance instead. Furthermore, 27% operations are multiplications, so selected processors may have dedicated hardware multipliers.

The fact that there is no need for complex control structures, the high data-accesses requirements and the coarse grain parallelism mean that optimizations can be obtained with a pipelined architecture with possible coarse grain dedicated hardware modules providing a large bandwidth. So if high performances are required, an ASIP or a programmable dedicated hardware can be introduced within the SOC.

2) Spatial Parallelism: We can notice that at the lowest level of granularity (1D-DCT sub-blocks operating on rows and columns), the value of $P$ is 1 indicating no fine grain spatial parallelism. It shows that these sub blocks (methods) are sequential in nature and does not contain any inherited parallelism. However the level of parallelism increases at the higher level of granularity (2D IDCT). The value of $P$ at this level is 24.14 indicating that a coarse grain parallelism is available.

D. Huffman Decoding

We first generate the trace tree for each function of Huffman decoding and then combine the individual trace trees to get the complete trace tree of Huffman decoding using the flow in figure 1. We perform analysis operations on trace tree representation to get analysis results. Table II shows the analysis results for representative functions of Huffman Decoding.

1) Orientation: It can be noticed that these functions have relatively high percentages of control operations denoting heavily conditioned data-flows. The percentage of computation operations also indicates an important computation frequency. There are less number of memory operations as compared to computations and control operations. It indicates that these methods are control and computation oriented. Figure 4 shows the percentage of each type of computation in the Huffman decoding. There are no floating-point but only integer-type operations. Furthermore, there are no multiplications, so selected processors have no need for dedicated hardware multipliers. The results show that 45% of the computations are logical operations.

2) Spatial Parallelism: We have not shown the value of $P$ in table II because the value of $P$ remains 1 at all hierarchical levels of trace tree. It reveals that suitable target architecture for Huffman decoding algorithm may be a GPP (General Purpose Processor). There is no need for a DSP and for a complex data path structure, since the parallelism cannot be exploited at any level.

VI. CONCLUSIONS

In this paper we have proposed a high-level application analysis methodology which aims at guiding the embedded systems design process for early design space exploration targeting the multimedia domain. Application is specified in a higher level object oriented language (Smalltalk) and then transformed into a trace tree representation by dynamic analysis. Unlike conventional dynamic analysis techniques, instrumentation is done on abstract syntax tree rather than source code. Instrumented application is then executed to
get trace tree representation. Due to the diverse requirements of embedded system design space exploration, the proposed framework enables the characterization of applications trace tree by a generic analysis approach which is not restricted to only a set of metrics. Designer can extend the framework by defining new operations on the trace tree of the source specification. To illustrate the methodology, we discuss the significance of some analysis results in the design space exploration of embedded systems and perform analysis operations on trace tree representation of the application. The outcome is a set of guidelines characterizing the application in terms of processing, control and memory orientation as well as in terms of potential spatial parallelism. Experiments with 2D IDCT and Huffman decoding algorithms shows that our approach not only helps designers to intensively comprehend the application but also provides valuable indications to highlight architectural opportunities and directions to improve application architecture matching.

The proposed framework is a starting point of the complete design flow and is still in its early phase of development. Using the proposed methodology, we are developing a framework for automatic conversion of sequential programs to parallel programs dedicated to streaming applications. We are working to integrate our analysis framework with the available synthesis tools by scheduling the trace tree to a family of coarse grained reconfigurable architectures.

REFERENCES


Noise model for Accuracy Constraint Determination in Fixed-point Systems

D. Menard, R. Rocher, O. Sentieys
IRISA/INRIA, University of Rennes,
6 rue de Kerampont
F-22300 Lannion
daniel.menard@irisa.fr

R. Serizel
K.U.Leuven, ESAT/SISTA
Kasteelpark Arenberg 10
B-3001 Leuven-Heverlee
romain.serizel@esat.kuleuven.be

Abstract— Most of DSP applications are specified and design with floating-point arithmetic but they are finally implemented into fixed-point architectures. Thus, the design flow requires to integrate a floating-point to fixed-point conversion stage. This conversion process aim is to optimize the implementation cost under an accuracy constraint. This accuracy constraint is linked to the application performances. The determination of this constraint is one of the key point of the conversion process. In this paper, a method is proposed to determine the accuracy constraint from the application performances. The fixed-point system is modelled with a infinite precision version of the system and a unique noise source located at the system output. After, the presentation of our approach, the noise model is especially detailed. To validate our noise model, different DSP application benchmarks have been tested and the adequation between our model and real noises have been measured.

I. INTRODUCTION

Computing oriented applications, stemming from digital image and signal processing domains, are widespread in embedded systems. To satisfy the cost and power consumption challenges, fixed-point arithmetic is favoured compared to floating-point arithmetic. In fixed-point architectures, memory and bus widths are smaller, leading to a definitively lower cost and power consumption. Moreover, floating-point operators are more complex to process the exponent and the mantissa, and thus, their area and latency are greater than those of fixed-point operators. Nevertheless, DSP algorithms are usually specified and designed with floating-point data types. Therefore, prior to the implementation flow in embedded systems, a fixed-point conversion of the application specification is required.

Fixed-point arithmetic introduces an unalterable quantization error which modifies the application functionalities and degrades the desired performances. Minimal computation accuracy must be guaranteed to maintain the application performances. In the fixed-point conversion process, the fixed-point specification is optimized such as the implementation cost is minimized as long as the application performances are fulfilled. Nevertheless, the performance degradations are not analyzed directly in the conversion process and an intermediate metric is used to measure the computation accuracy. Indeed, the exploration of the fixed-point search space is more complex if the application performances are managed directly. The fixed-point computation accuracy can be evaluated with two kinds of methods corresponding to analytical and simulation-based approaches. In the simulation-based method, fixed-point simulations which evaluate the application performance metrics take definitively more time than for the accuracy metric. For example, in a wireless communication receiver, the bit-error rate measurement requires to use a great number of samples and thus requires very long simulation time. In the case of analytical approaches, a mathematical expression of the metric is determined. Then, the expression is evaluated much more quickly than in the case of simulation-based approach, and gives an estimation of the application quality. However, linking the analytical metric with performance or quality measurement for every kind of application is generally an issue.

The global conversion method is decomposed in two main steps. Firstly, a computation accuracy constraint is determined according to the application performances, and secondly the architecture cost is minimized under this accuracy constraint during the fixed-point conversion process. This computation accuracy metric can be the quantization error bounds or the Signal to Quantization Noise Ratio (SQNR). The minimal value determination for the computation accuracy metric is a difficult problem and cannot be defined directly. This accuracy constraint has to be linked to the quality evaluation and performances of the application.

In this paper, a method is proposed to determine the accuracy constraint from the application performances. In our approach, the metric used to evaluated the computation accuracy is the SQNR. The SQNR constraint is determined by modelling the fixed-point system behavior with infinite precision version of the system and a unique noise source located at the system output. The accuracy constraint is determined as the maximal value of the noise power which keeps the desired application quality. The proposed noise model is especially detailed in this paper. To our knowledge, no output quantization noise model is available to determine the computation accuracy constraint in the literature.

The paper is organized as follows. The fixed-point conversion process and the determination of the accuracy constraint are presented in Section II. The noise model is detailed and
justified in Section III. This noise model is validated in Section IV and the adequation between our noise model and real quantization noises is shown through examples.

II. ACCURACY CONSTRAINT

A. Fixed-point conversion process

A fixed-point data is composed of an integer part and a fractional part. The number of bits associated with each part does not evolve during the processing leading to a fixed binary position. The fixed-point conversion aim is to determine the number of bits for each part. Thus, as illustrated in figure 1, this process can be divided in two main modules [1], [2].

![Fixed-point conversion process](image)

The first module corresponds to the determination of the integer part word-length. The number of bits for this integer part must allow to represent all the values taken by the data, and is obtained from the data bound values. Thus, firstly the dynamic range is evaluated for each data. Then, these results are used to determine, for each data, the binary-point position which minimizes the integer part word-length and which avoids overflow. Moreover, scaling operations are inserted in the application to adapt the fixed-point format of a data to its dynamic range or to align the binary-point of the addition inputs.

The second module corresponds to the determination of the fractional part word-length. The number of bits for this fractional part defines the computation accuracy. Thus, the data word-lengths are optimized such as the implementation cost is minimized under accuracy constraint. The fractional part word-length determination corresponds to an optimization problem where the implementation cost and the application accuracy must be evaluated. In our approach, the computation accuracy is evaluated through the Signal to Quantization Noise Ratio. The analytical approaches have been favoured to evaluate the computation accuracy. Indeed, compared to the simulation-based techniques, they allow to obtained reasonable optimization time during the fixed-point space exploration. In the case of simulation-based approaches, a new fixed-point simulation is required when a fixed-point data format is modified.

The accuracy constraint corresponds to the minimal value for the SQNR which allows to respect the desired application performances or quality. The approach used to obtain the minimal value of the SQNR is presented in the next paragraph.

B. Accuracy constraint determination

The accuracy constraint corresponding to the minimal value of the SQNR \((SQNR_{\text{min}})\) is defined according to the system performance constraints. In our approach, the fixed-point system is modelled by the system infinite precision version and a unique noise source \(b_y\) located at the system output. The accuracy constraint is determined from the maximal value of the noise power which allows to keep the desired application performances. The performances are measured by simulation. The system floating-point version is used and the noise \(b_y\) is added to the output. The noise model used for \(b_y\) is presented in Section III. The power of \(b_y\) is increased as long as the measured performances are acceptable. Most of the time, the floating-point simulation has already been developed during the application design step, and the application output samples can be directly used. Therefore, the time required for exploring the noise power values is significantly reduced, and becomes negligible with regards to the global implementation flow.

![Model for accuracy constraint determination](image)

Figure 3 shows the global process of accuracy constraint determination followed by the fixed-point design process. After the accuracy constraint determination, fixed-point conversion is achieved. The fixed-point specification is determined in order to optimize the implementation and to fulfil the accuracy constraint. This optimized fixed-point specification is simulated to measure the real performance obtained with this specification and to verify if the application performance constraints are still achieved. In the opposite case, the minimal value of the SQNR is adjusted and the fixed-point process is repeated. With this approach, few fixed-point simulations are required.
III. NOISE MODEL

A. Quantization noise model

The use of fixed-point arithmetic introduces an unavoidable quantization error when a signal is quantified. A common used model for signal quantization has been proposed by Widrow in [3] and refined in [4]. The quantization of a signal $x$ is modelled by the sum of this signal and a random variable $b$, which represents the quantization noise. This additive noise $b$ is a uniformly distributed white noise that is uncorrelated with the signal $x$, and independent from the other quantization noises. In this study, the round-off method is used rather than truncation. Quantization by rounding process leads to an error with a zero mean. If $q$ is the quantization step (accuracy), the noise values are in the interval $[-q/2; q/2]$.

\begin{equation}
\eta \sim \text{Uniform}[-q/2; q/2]
\end{equation}

B. Quantization noise model for fixed-point system

The output quantization noise is the contribution of the different noise sources. Each noise source is due to the elimination of some bits during a cast operation which follows an arithmetic operation. These different noise sources are independent of each other [4]. These noise sources are propagated through the different operations. A propagation model for each arithmetic operator is proposed in [5]. The operator output noise is a weighted sum of the input noises associated with each operation input. The weights of the sum do not include noise term, because the product between the noise terms can be neglected. Thus, it can be demonstrated that the output quantization noise is a weighted sum of the different noise sources as shown in figure 4. The contribution of each noise in terms of statistical parameters depends on the fixed-point format after quantization, and the gain between the output and the noise source.

\begin{equation}
\eta = \sum_{i=1}^{N} \beta_i \eta_i
\end{equation}

In this context, two extreme cases can be distinguished. In the first case, a quantization noise source predominates in terms of variance compared to the other noise sources. A typical example is an extensive reduction of the number of bits at the system output compared to the other fixed-point format. In this case, the level of this output quantization noise exceeds the other noise source level. Thus, the probability density function of the output quantization noise is very closed to the one of the predominant noise source and can be assimilated to a uniform distribution. In the second case, an important number of independent noise sources have similar statistical parameters and no noise source predominates. All the noise source are uniformly distributed and independent of each other. By using the central limit theorem, the sum of the different noise sources can be modeled by a centered normally distributed noise.

From these two extreme cases, an intuitive way to modelize the output quantization noises of a complex systems is to use a noise $b$ which is the weighted sum of a Gaussian and an uniform noise. Let $f_b$ be the probability density function of the noise $b$. Let $b_n$ be a normally distributed noise with a mean and variance equal respectively to 0 and 1. Let $b_u$ be a uniformly distributed noise in the interval [-1;1]. The noise $b$ is defined with the following expression:

\begin{equation}
b = \nu (\beta \times b_n + (1 - \beta) \times b_u)\end{equation}

The weight $\beta$ is set in the interval $[0; 1]$ and allows to represent the different intermediate cases between the two extreme cases presented above. The weight $\nu$ fixes the global noise variance.

IV. VALIDATION OF THE PROPOSED MODEL

A. Validation methodology

The aim of this validation section is to analyze the adequation between our model and real quantization noises. Our model is valid if a balance weight $\beta$ can be found to model the noise probability density function with equation 1. The adequation between the real noise and our model is analyzed with the $\chi^2$ goodness-of-fit test. This test is a statistical tool which can be used to know if an observed noise $b_y$ follows a chosen probability density function $f_b$ [6]. The test is based on the distance between the two probability density functions.
If \( y_s \) is the observed frequency for bin \( s \), \( E_s \) is the expected frequency for bin \( s \) and \( k \) the number of bin \( s \), the statistical test is:

\[
\chi^2 = \sum_{s=1}^{k} \frac{(b_{ys} - E_s)^2}{E_s}
\]  

(2)

This statistical test follows a \( \chi^2 \) distribution with \( k - 1 \) degrees of freedom. Therefore, if the distance is higher than a certain value, then the hypothesis \( H \) (the distance is higher than the expected value) is rejected. The significance level of the test is the probability to reject \( H \) when the hypothesis is true. Choosing a certain value for this level will set the threshold distance for the test. According to [7], the significance level \( \alpha \) should be in \( [0.001, 0.05] \).

Concerning the observed noise, there is no \textit{a priori} knowledge of the balance coefficient \( \beta \). Thus the \( \chi^2 \) test has to be used collectively with a searching algorithm. The idea is, for a given observable noise, to find the \( \beta \) coefficient for which the \( f_b \) fits the best to the noise. The method is to compute the \( \chi^2 \) goodness-of-fit test for different values of \( \beta \). In the interval \( [0; 1] \), different values of \( \beta \) are tested, and, progressively, the search interval is reduced. When the test succeeds, the balance coefficient is found otherwise the noise cannot be modeled with our approach. This procedure, called the \( \beta \)-searching algorithm, allows to check the validity of the model on several examples.

\subsection{B. FIR filter example}

The first system on which the before-mentioned test has been performed is a well-known simple algorithm: a 32-tap FIR filter. The signal flow graph of one cell \( i \) is presented in figure 5. To simplify the presentation, the integer part word-length for the multiplication output and the input and output addition are set to be equal. Thus, no scaling operation is necessary to align the binary point position at the adder input.

Two kind of quantization noise sources can be located in the filter. A noise source \( b_m \) can be located at each multiplier output if bits are eliminated between the multiplication and the addition. The number of eliminated bits \( k_m \) is obtained with the following expression:

\[
k_m = \text{wl}_{\text{mult}} - \text{wl}_{\text{add}} = \text{wl}_{x} + \text{wl}_{h} - \text{wl}_{\text{add}}
\]  

(3)

A noise source \( b_o \) is located at the filter output if bits are eliminated when the addition output is stored in memory. The number of eliminated bits \( k_o \) is obtained with the following expression:

\[
k_o = \text{wl}_{\text{add}} - \text{wl}_{o}
\]  

(4)

The adder word-length \( \text{wl}_{\text{add}} \) is varying between 16 bits and 32 bits, while the output of the system is always quantized on 16 bits.

\begin{figure}[h]
  \centering
  \includegraphics[width=0.5\textwidth]{fig5.png}
  \caption{Signal Flow Graph for one FIR filter tap}
  \label{fig5}
\end{figure}

The word-length of the input signal (\( \text{wl}_{x} \)) and the coefficient (\( \text{wl}_{h} \)) are equal to 16 bits. If no bit is eliminated during the multiplication, the multiplier output word-length \( \text{wl}_{\text{mult}} \) is equal to 32 bits. The adder input and output word-length are equal to \( \text{wl}_{\text{add}} \). At the filter output, the data is stored in memory with a word-length \( \text{wl}_{o} \) equal to 16 bits.

The probability density function of the filter output quantization noise is presented in figure 6 for different values of \( \text{wl}_{\text{add}} \). The noise is uniform when one source is prevailing (the adder is on 32 bits). As the influence of the sources at the output of the multiplier is increasing (the length is decreasing), the distribution of the output noise tends to become gaussian. These simple visual observations can be confirmed using the \( \beta \)-searching algorithm. When the output of the multiplier is on 16 or 17 bits, \( \beta = 0 \), the sources are numerous. Their influence on the system output is a gaussian noise. While the length of the multiplier is increasing, \( \beta \) also grows and eventually tends to 1. When \( \text{wl}_{\text{add}} \) is greater than 26 bits, the variance of the noise sources \( b_m \) located at each multiplier output is insignificant compared to the variance of the noise source \( b_o \) located at the filter output. Thus, this latter is prevailing. Its influence on the output signal is an uniform white noise. In this case, the boundary values of the noise are \([-7.4768e^{-006}, 7.4768e^{-006}]\), so the range is actually equal to
$2^{-16}$. Figure 7 depicts the evolution of $\beta$ for different adder word-lengths, varying from 16 to 32 bits.

![Graph showing the evolution of $\beta$ for different adder word-lengths](image)

Fig. 7. Balance weight $\beta$ found for different adder word-lengths. This weight is obtained with the $\beta$-search algorithm presented in section IV-A

C. Benchmarks

To validate our noise model, different DSP application benchmarks have been tested and the adequation between our model and real noises have been measured. The real noises are obtained through simulations. The output quantization noise is obtained from the difference between the system outputs obtained with a fixed-point and a floating-point simulation. The floating-point simulation which uses double-precision types is considered to be the reference. Indeed, in this case, the error due to the floating-point arithmetic is definitely weaker than the error due to the fixed-point arithmetic. Thus, the floating-point arithmetic errors can be neglected.

For each application, different output noises have been obtained by evaluating several fixed-point specifications and different application parameters. The number of output noises analyzed for one application is defined through the term $N_t$. For these different applications based on arithmetic operations, the input and the output word-length are fixed to 16 bits. The different fixed-point specifications are obtained by modifying the adder input and output word-length. Eight values are tested for the adder: (16, 17, 18, 19, 20, 22, 24, 32).

The results are presented in Table I for two significance level $\alpha$ corresponding to the boundary values (0.05 and 0.001). For each application, the number $N_s$ of real noise which can be modelled with our noise model are measured. The adequation between our model and real noises is measure with the metric $\Gamma$ defined with the following expression:

$$\Gamma = \frac{N_s}{N_t}$$

This metric corresponds to the ratio of output noises for which a weight $\beta$ can be found to model the noise probability density function with equation 1.

The different applications used to test our approach are presented in this paragraph. Fast Fourier Transform (FFT) has been achieved on vectors made-up of 16 or 32 samples. Linear time-invariant recursive systems have been tested through an eight-order infinite impulse filter (IIR). This filter is implemented with a cascaded form based on four cascaded second order cells. For this cascaded eight-order IIR filter, 24 permutations of the second-order cells can be tested leading to very different output noise characteristics [8]. Three forms have been tested corresponding to Direct-form I, Direct-Form II and Transposed-Form. An adaptive filter based on the APA algorithm [9] has been tested. This filter is made-up of eight taps and the observation vector length is equal to five. A non-linear non-recursive filter has been tested through a second-order Volterra filter.

More complex applications have been studied through a WCDMA receiver and a MP3 coder. For the third generation systems based on the WCDMA technique, the receiver is mainly made up of a FIR receiving filter and a rake receiver with synchronization mechanisms [10]. The rake-receiver is made-up of three parts corresponding to the transmission channel estimation, synchronization and symbol decoding. The synchronization of the code and the received signal is realized with a Delay-Locked Loop (DLL). The noises are observed at the output of the symbol decoding part.

A MP3 coder has been studied. This application can be divided in two signal flows. The one composed from the polyphase filter and the MDCT. The other is made-up of the FFT and the psychoacoustic model. The results are presented only for the first part. The observed noises are those which result from fixed point conversion on the MDCT block of an MP3 coder. MDCT stands for Modified Discrete Cosine Transform, it is basically a DCT with dynamic windowing. The test has been run on 28800 noise signals taken out from the output of the MDCT. It is important to check that the power of the signal is high enough in relation with the power of the noise. The lowest signal to noise ratio is $16dB$ which is rather high and allows using the noise model.

The results show that our noise model can be applied to most of the real noises obtained for different applications. For some applications, like FFT, FIR, WCDMA receiver and Volterra filter, a balance coefficient $\beta$ can be found. These four applications are non-recursive and the three first applications are LTI systems.

For the eight-order infinite impulse filter, almost all the noises (97%-100%) can be modelled with our approach. For these filters, 90% of the output quantization noise are modelled with a balance coefficient $\beta$ equal to 0. Thus, the output noise is a purely normally distributed noise. In linear time-invariant system, the output noise $b_y'$ due to the noise $b_y$ corresponds to the convolution of the noise $b_y'$ with $h_g$. This term $h_g$ is the
impulse response of the transfer function between the noise source and the output. Thus, the output noise is the weighted sum of the delayed version of the noise $b_g$. The noise $b_g$ is a uniformly distributed white noise, thus the delayed versions of the noise $b_g$ are uncorrelated. Even if only one noise source is located in the filter, the output noise is a sum of non-correlated noises and this output noise tends to have a gaussian distribution.

For the MP3 coder, when the level is 0.001 the test is successful about 87% of the time (78% when $\alpha$ is 0.05). The fairly high percentages tend to show that it is relevant to use this model. The second observation is that none of the found $\beta$ values is higher than 0.5. The noises are mostly gaussian, and no source is prevailing.

V. CONCLUSION

In embedded systems, the fixed-point arithmetic is favoured but application performances can be reduced due to finite precision computation. In the fixed-point conversion process, the fixed-point specification is optimized such as the architecture cost is minimized as long as the application performances are fulfilled. Nevertheless, the performance degradations are not analyzed directly in the conversion process, and an intermediate metric is used to measure the computation accuracy. In this paper, a noise model has been proposed to determine the accuracy constraint in case of complex application. A system output noise is modelled from a gaussian noise and a uniform noise. The different experiments show that this model is adequate in most of the case for different DSP applications. Now, different experiments will be conducted to show the efficiency of our approach to determine the computation accuracy constraint and the adequation between the desired

### REFERENCES


Efficient memory management for Ray Casting

Stéphane Mancini, Michel Desvignes
GIPSA-lab, CNRS/INPG, Grenoble, France

Abstract—This paper presents the usefulness of the n-Dimensional Adaptive and Predictive Cache (nD-AP Cache) memory hierarchy to improve the performances of the Ray Casting algorithm. The nD-AP Cache is aimed to exploit n-dimensional and temporal locality which is a main characteristic of multimedia applications. To have the best cache efficiency, a “phase-locked” parallel Ray Casting algorithm is also proposed. Some new features are added to the nD-AP Cache to take into account the geometrical and temporal characteristics of the fetch sequence issued by such a control-oriented algorithm. The measures acquired from a prototyping board show that the nD-AP Cache allows to reach a high hit ratio at a low hardware cost, and this for two applications of Ray Casting. As a conclusion, this work shows the versatility of the nD-AP cache architecture.

I. INTRODUCTION

The management of high quantities of data is a challenge for many digital systems. This problem is getting more and more complex with the increase of the quantity of memory embedded in digital integrated systems such as System on Chip (SoC). As an example, the International Technology Road-map for Semiconductors Consortium plans that memory will occupy 90% of a circuit in the next years. Then, it is mandatory to provide efficient memory hierarchies that are optimized together with a cache friendly optimisation of applications.

This paper demonstrates the usefulness of the nD-AP Cache architecture (n-Dimensional Adaptive and Predictive Cache) [1] and the associated methodology for a Ray Casting application. Some improvements of the cache features are proposed to match the Ray Casting algorithm as any algorithm with similar patterns of memory accesses. The measures acquired from a prototyping board validate the proposed memory hierarchy. These results demonstrate the versatility of the nD-AP cache that has been successfully validated with other applications.

Ray Casting is an algorithm used in medical imaging applications such as visualisation or tomographic reconstruction. The main tasks of Ray Casting are to process the traversal of a 3D grid by a ray and to apply a shading function at each cell crossing the ray. For a data set visualisation application the Ray Casting simulates the propagation of light in the volume to produce an understandable image [2]. At each cell traversed by the ray, the algorithm computes the lighting effects. Tomographic reconstruction algorithms use Ray Casting to simulate the acquisition of a volume in tomographic (X Rays) or nuclear PET imaging modalities [3], [4]. With the PET modality, the algorithm simulates the acquisition of data, called a sinogram, that is a 4D structure. The Ray Casting algorithm is used to compute the sinogram data. Each bin, or datum of the sinogram, is the sum of the voxels along a Line Of Response (LOR) between two detectors of the cylindrical camera.

The Ray Casting algorithm is a very CPU consuming and control oriented algorithm: so, it is a nice algorithm to benchmark a memory system. Hierarchical grid traversal is mostly useful for visualisation but in this work we focus on a regular grid traversal algorithm, which is still

\(^{1}\) Positron Emission Tomography
used for tomographic applications.

The next section describes the Ray Casting algorithm and a state of the art of hardware and software architectures provided in literature to accelerate this algorithm. Data management appears to be the main bottleneck and we briefly describe some general results about this problem. From this, we propose together the phase-locked Ray Casting and some new features of the 3D-AP cache memory hierarchy to optimize the RCPG\textsuperscript{2} system. Follows a discussion from the analysis of the cache efficiency measured on a prototyping board. Finally, we conclude and present future work.

II. State of the art

A. Grid traversal

Several algorithms are used to project a 3D data set to produce an image [2] or a sinogram. They have different trade-offs between quality, accuracy and speed. In this paper we focus on the RCPG (Ray Casting in Projective Geometry) [5] grid traversal algorithm which computes the exact intersection between a ray and all the cells. It is an algorithm similar to the family of Bresenham-like algorithms [6], [7] but it can be implemented with fixed point numbers. Other algorithms are grid sampling and shear-warp rendering [8] but they do not provide enough accuracy for tomographic reconstruction.

The RCPG algorithm is an iterative algorithm: from a current cell crossed by the ray, the next cell is defined by a minimization of a cost function which is iteratively updated. At each step, the ray is propagated along the direction $x$, $y$ or $z$ which depends on the face where the ray and the current cell intersect. The parameters of intersection between the ray and each face are stored and updated at each step.

The main characteristic of this algorithm is that it is control-oriented and the sequence of memory accesses depends on the parameters of the ray. The iterative process is straightforward to implement but the memory management is a bottleneck to speed-up the process because it is difficult to prefetch the next data from the current one.

B. Memory management and caching for Ray Casting

As memory accesses are the main bottleneck of the Ray Casting algorithm, software and hardware architecture can be optimized thanks to optimal caching strategies, the reorganisation of computations, approximations of the algorithm and the reduction of the number of crossed cells.

The main idea underlying the different cache strategies is the “ray coherency”, which states that a set of coherent rays will traverse the same zone of space. A cell, or voxel, will be crossed by different rays and an efficient memory strategy has to keep it’s data in cache memory as long as needed. The main difficulty is that the data set does not fit in a cache as today it is common to have grids of size $512^3$ and even $1024^3$. Furthermore, standard processor caches are not efficient when managing random accesses in a 3D structure.

The cache efficiency of software systems can be optimized thanks to multi-threaded software [9], [10]: each thread deals with a small set of rays and an efficient memory strategy has to keep it’s data in cache memory as long as needed. The main difficulty is that the data set does not fit in a cache as today it is common to have grids of size $512^3$ and even $1024^3$. Furthermore, standard processor caches are not efficient when managing random accesses in a 3D structure.

The cache efficiency of software systems can be optimized thanks to multi-threaded software [9], [10]: each thread deals with a small set of rays and the speed-up comes from the fact that some grid data used by each thread are in the cache memory. This technique is limited by task context switch, cache trashing and by the available computing power.

To reduce the memory accesses, object order processing splits the volume on small blocs of voxels that are loaded in memory once, one after the other. Each bloc processes the set of rays it crosses. When a ray exits a bloc, it is transmitted to the next bloc and the algorithm ends when no more blocs have rays to process. It can be implemented on dedicated architectures [11], general distributed memory parallel architectures [12] and even on a multi-threaded multi-core software [10]. Object order processing has the advantage to be memory efficient, because a computing unit stores a bloc only once,
but it needs a complex communication scheme and a complex load balancing strategy between the nodes.

The memory bottleneck becomes crucial as hardware architectures allow very high computing power. The design of an hardware system allows to match the pipeline and the memory subsystem as proposed by the VoxelCache [13], [14]. It is a full-associative cache that manages blocs of the volume data. A prefetch mechanism loads the blocs of volume contiguous to the fetched voxel, depending on the ray direction. It happens that the cache size limits the size of the volume because of cache trashing.

Hardware systems benefit of uniform memory accesses and a solution is to allow only parallel rays but with a loss of accuracy and quality. Such a strategy is implemented on dedicated hardware [15], [16], [17] and underlies volume rendering on commodity PC graphics hardware such as GPU [18], [19]. Some parallel rays are sampled at the same interval to provide “plane parallelism”. Perspective volume rendering is then simulated by a perspective transformation of the resulting image. This solution is very popular for visualisation because it is fast but it suffers of too low accuracy for tomographic image reconstruction.

An efficient solution to overcome the memory bottleneck is to reduce the number of traversed cells by pre-computing an intermediate data structure which contains only valuable data and some information on the space zones to skip. The most famous of such algorithms are the hierarchical grids and space-leaping algorithms [20]. In a hierarchical grid such as an octree or a k-d tree, a cell contains either a volume data or a pointer to a lower resolution grid. Space-leaping acceleration relies on the transform distance of the volume to compute the number of cells that can be skipped to the next sample from the current one without loss of information. The hardware speed-up of both strategies is difficult because the intermediate data computation may slow down the whole process. Also, their irregularity counteracts the prefetch mechanisms of caches. Hierarchical grid data caching is under work and the present paper gives some results for the uniform grid traversal and caching problems.

C. Optimisations of structured data management

Caching of multi-dimensional structures is a well known problem in the field of multimedia and scientific computer science. The best performances are reached when the cache architecture, the data structures and the application match.

1) Cache architectures for structured data management: Cache architectures for general purpose processors have been optimized to deal with structured data and especially for multimedia applications. The challenge is to estimate the cache lines to prefetch from an analysis of the past fetches, without the knowledge of the initial data structure. The simplest optimization is One Bloc Look-a-head (OBL) and Stride Prediction Table (SPT) [21] is one of the most efficient. With OBL, the line following the current fetch is always prefetched. SPT associates to each load instruction the previous address fetched and the address stride that was previously computed. SPT prefetches the line a stride ahead the next load issue. As an example, SPT is implemented in the Intel-Core processor as the “Smart Cache” feature [22]. Although efficient for software applications, SPT is too complex for specific hardware, FPGA targets and embedded systems because it needs an associative memory to store the load instructions and the associated tags.

Specific caching hardware can be implemented, more or less tighten to the application. The most obvious strategy is to pipeline computations and memory accesses. But it makes little use of the fetch coherency and parallelisation is difficult. Similar to pipelining, deterministic caching [23] analyses a part of the fetch sequence to compute the needed data. It may be of low overhead but some memory is necessary to store the fetch sequence and the corresponding intermediate internal variables. On-line cache accesses with a prefetch mechanism is the most
efficient way to reach a high throughput with a low pipeline latency. The nD-AP Cache [1] allows this with a low hardware cost.

2) Optimizations of applications for cache efficiency: Applications have to be transformed in such a way that they produce fetches in a cache friendly way: the next iteration of a loop has to produce a fetch at an address close to the previous one. The main results we can find in the literature are about the transformation of nested loop when data indices are affine functions of loop indices [24]. Tiling is another popular optimisation which decomposes a loop on a higher level loop to produce tiles and an inner loop in each tile.

Optimization of data dependant and control oriented applications is inspired of loop transformations strategies: for example the tiling can be applied even on applications such as 3D rasterisation [25]. The "Ray coherency" optimization is a particular case of tiling applied to visualisation and sinogram computing. In the case of visualisation, a tile is a part of the resulting image: the set of rays passing through a tile is a set of coherent rays. In the same way, the processing of a sinogram can be split on tiles of the 4D structures of LORs.

III. System architecture

This section describes the system architecture and the caching strategy together with a modified Ray Casting algorithm. After a description of the new features added to the 3D-AP Cache to get higher performances, a single pipeline architecture is proposed. Then follows the proposal of different strategies to parallelise the pipeline.

A. Pipeline overview

The RCPG system is implemented on a SoPC (System on Programmable Chip) and contains the following units :

- A RCPG pipeline connected to a 3D-AP Cache
- A 64 bit PLB CoreConnect system bus
- A PPC 405 microprocessor core
- External DDR-SDRAM memory

As shown figure 1 the RCPG pipeline performs grid traversal and usually fetches the volume data from a 3D-AP Cache. The volume data is stored in the external memory and some data are duplicated in the cache. The 3D-AP Cache has a DMA engine and it grabs data from the external memory when it performs prefetches or manages the cache misses. The PPC processor is in charge of the main control and performs some initialisations.

B. 3D-AP Cache

The 3D-AP Cache is an extension of the 2D-AP Cache [1], improved by some new features and tracking algorithms. The purpose of this cache is both to provide a virtual access to the volume data and to prefetch a geometrical zone of data, as shown figure 2. Low cost signal processing algorithms are used to perform a coarse estimation of the next grid index that may be issued by the processing unit (see [1] for more details). The estimated geometrical zone is then prefetched by the cache. The tracking mechanisms exploit nD locality and temporal locality and are more efficient than standard cache mechanisms for several applications.

The 3D-AP cache is aimed at the following characteristics :

- it provides a semi-general purpose mechanism
- it is a low cost architecture
- The target technology may be ASIC or FPGA

![Fig. 1. RCPG pipeline and cache interface](image-url)
It is semi-general purpose because many algorithms share its assumptions about the probability distribution of memory accesses. Furthermore, the tracking mechanisms and the index to cache memory direct mapping are space-saving and prevent the use of associative memory.

C. 3D-AP Cache optimization

1) “Phase locked” iterative propagation: We propose the “phase locked” propagation of rays to optimize the RCPG algorithm together with the nD-AP Cache. The “phase locked” propagation enables a virtual loop over the traversed cells. Indeed, the RCPG algorithm is synchronized over a set of rays to propagate them together along a main direction, orthogonal to a phase plane. For each ray, a step of the RCPG algorithm returns the next cell to cross. This is iterated while the resulting cell remains in the phase plane. When all the rays of the set are out of the phase plane, then the phase is updated to the next one. The process loops until all the rays exit the volume. The phase acts as a virtual loop, the innermost one being the ray index. To be efficient, the cache have to contain a slice of data about the phase plane.

2) 3D-AP Cache improvements: The experiments on Ray Casting have raised the need of more efficient tracking mechanisms and new cache behaviours. To manage different classes of fetch sequences, multi-mode trackers implement together several tracking mechanisms that may be dynamically selected. The mapping of fetch indices to the internal cache addresses can also be dynamically chosen to fit different sizes of cache. The trackers are joined together to allow more complex cache zone displacements: a displacement request from a tracker makes the other trackers to center on their estimated center. Furthermore, the user module can now select the priority of misses over cache updates.

The dynamic selection of the priority of misses is efficient especially when the misses are faster than the uploading of new zones into the cache. At a first sight, updates of the cached zone have a higher priority than misses because it prevents the user module to always request misses if the trackers are too slow. But fetches can evolve differently on each dimension and misses can have different priorities depending on the cache geometry. At the time the user module requests a high priority miss, the cache update is interrupted, the miss is served and the update continues over.

The phase locked propagation of rays benefits from these improvements. The multi-mode tracking is necessary because the virtual loop on the propagation direction can be efficiently tracked by a linear tracker while the other dimensions are tracked by statistical trackers. The misses have high priority along directions perpendicular to the phase direction and have low priority along the later. Indeed, the phase increases (or decreases) uniformly faster than the other directions in the average case. The worst case is when the rays have a 45-degree angle with all or some of the main axes.

D. Parallelisation

Parallelisation of ray-casting is necessary to reach high performance and our architecture allows several types of parallelisation. The main idea is to parallelise the computations of coherent rays over several pipelines, each one getting its data from a cache. Different pipelines can get their data from a shared cache, what is called “Group-level parallelisation”. The “Cluster-level” parallelisation is a parallelisation of groups that get their data from a higher level cache.
enable 3D locality different scenarios of synchronisation between the pipelines are proposed.

1) Group-level parallelisation: At the group level, several pipelines share a common cache with a time-slot policy and are tightly synchronized along a joint phase. Indeed, it is possible for several pipelines to share a cache because a pipeline processes a voxel each 4 to 5 clock cycles. To maintain 3D locality of fetches, a synchronizer allows the pipelines to update their phase only when all of them are ready to do so.

2) Cluster-level parallelisation: At the cluster level, several groups and associated caches get their data from a higher level cache to form a cluster. The groups are softly synchronised together. A cluster of rays is distributed along the different pipelines in order to minimize the cache size. A soft synchronisation tolerates a small deviation between the phases of the groups to allow them to perform update transactions from the main cache at different instants.

IV. Results

The whole system is designed in VHDL and some metrics are extracted from a prototype that is implemented on a AVNET prototyping board with a Virtex II Pro FPGA. The board runs at 30 MHz but this is not an issue as logic synthesis of the VHDL code on the Virtex 4 technology reports a clock frequency up to 200 MHz. A memory simulator allows to measure the cache performance for different memory configurations. In this section we present results of the system’s complexity and measures of the cache efficiency for different applications of the Ray Casting algorithm.

A. Hardware complexity

Table I gives the complexities of the RCPG pipeline and of the cache. These results are obtained in fixed point arithmetic, with the bit widths set to reach the accuracy needed by tomographic reconstruction. One can notice that the cache uses standard technology and its complexity is very low.

<table>
<thead>
<tr>
<th></th>
<th>FG</th>
<th>DFF</th>
<th>DSP48</th>
<th>BRAM (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCPG pipeline</td>
<td>1739 (3%)</td>
<td>996 (2%)</td>
<td>6 (4%)</td>
<td>9 (0.7%)</td>
</tr>
<tr>
<td>3D-AP Cache</td>
<td>1299 (2.5%)</td>
<td>365 (0.5%)</td>
<td>0</td>
<td>8 (0.7%)</td>
</tr>
</tbody>
</table>

TABLE I

RCPG synthesis results of the different units for a Xilinx Virtex IV technology. Percentages are relative to the V4FX60 device capacity.

B. Cache efficiency

In the next sections, the cache efficiency is measured as the ratio between the number of fetches divided by the number of clock cycles to get the data. The prototype is instrumented to measure the cache timing and available measures are, for example, the cache hit ratio, the number of clock cycles to initiate the cache, the number of data loaded into the cache and the system bus occupancy.

1) System settings: The memory simulator is dynamically configured with a latency \( l \) and a throughput \( d \), which are respectively the number of clock cycles from the burst request to the arrival of the first data and the number of clock cycles between two data: the higher \( d \) is, the slower the memory bus is. The system bus has a 8 Bytes data bus and a voxel data is 1 Byte.

Measures are performed for different architectures. The group-parallel pipeline is made of 4 RCPG pipelines sharing a 3D-AP Cache of 8 KByte. The cluster-parallel pipeline is made of 2 group-parallel pipelines.

2) Visualisation: The cache efficiency is measured for a visualisation application which is a typical example. Experiments show that the cache is efficient for different viewpoints and for outspreading rays. For visualisation, each ray comes from a viewpoint, crosses over a pixel of the camera and traverses the volume. As the cache behaviour depends on the geometry of the rays, the most difficult case is the oblique viewpoint, when rays are near to the diagonal, and the best one is the front view point, when rays are about parallel to a main axis.
The figure 3 plots the average cache efficiency for different bus performances: the horizontal axis is the memory latency and a curve has a constant bus throughput $d$. The volume has a resolution of $256^3$ voxels and the image is $288^2$. These sizes are representative and are the limit case: the density of rays is about 1 and there is little data re-use. When rays are distant of more than one volume cell, it is likely that a cell is only needed once. At the opposite, a cell data is used more often if rays are of higher density, for example in the case of a zoom on a specific region of the volume.

Results from figure 3 are good: the group level pipeline has a cache efficiency about 80% for a memory system with a typical latency of 4 clock cycles. The average efficiency is lowered by the special cases at the borders of the image, as shown figure 4. Most of the tiles are processed with a cache efficiency higher than 90%. Indeed, the set of rays numbers lower on the border of the image than the typical case for which the cache is tuned. Figure 4 also shows that the cache efficiency is a little bit lower at the very center of the image for the front viewpoint.

At these locations, the rays evolve slower than elsewhere because they are almost parallel to an axis and the speed parameter of the tracker is then overestimated.

Measures on the cluster-level parallel architecture for the two viewpoints are obtained from simulation because our development board is of too low complexity. Simulations show that two groups of pipelines sharing a higher level cache gives a speed-up of 1.5.

C. Sinogram computing

The results from figure 5 show that it is more efficient to compute a sinogram than to visualise a volume. Measures are performed to compute a sinogram corresponding to the geometry of an “ECAT EXACT HR+” PET camera: it is a cylinder of 32 rings and each ring has 576 detectors. The Ray Casting is used to compute the integral of a line blending two detectors. Measures are performed for ray’s endpoints belonging to a single ring (horizontal segment) and in different rings (oblique segment). The whole sinogram is a 4D structure and a set of 256 rays is issued from a tile of it.

The pipeline has an efficiency higher than 90% with a typical memory. Simulations show that the cluster level architecture still have an efficiency about 85% and provides a speed-up of 1.9 compared to a single group-level pipeline. These very good results are due to the fact that the rays of a 4D tile are crossing together in a “tube” and their density is higher than in the case of visualisation.
D. Discussion

Comparison with other solutions is not straightforward because existing architectures implement a grid sampling algorithm, whereas we propose to implement an exact grid traversal rays casting. Furthermore, most of these algorithms implement early-ray termination, which is useful for visualisation because unseen voxels do not need to be rendered, but our algorithm processes the whole grid to compute the estimated sinogram.

[14] reports simulation results of a cache designed for a volume rendering hardware architecture. For each sample, the pipeline fetches 8 data from the VoxelCache which is a full associative cache that contains blocks of voxels. The VoxelCache holds 512 lines each of 64 voxels to make a 32 KByte memory. The simulation results of [14] show a 90% pipeline utilisation but the size of the rendered volume is limited to $128^3$ cells because the VoxelCache is trashing when it cannot hold all the voxels along a line.

Our solution reaches an equivalent throughput with 4 times less memory and without associative memory: it is space-saving and has a better usage of FPGA resources. Also, a high level of parallelism is allowed, depending on the geometry and the density of rays. At last, the 3D-AP Cache and the “phase locked” RCPG allow any size of grid and the system is scalable.

E. Improvements

Experiments and measures show the necessity of various improvements to get higher performances. We need an increase of the bandwidth between the higher level cache and lower level caches and a new geometry of cached zones. The first improvement is mandatory to increase the level of parallelism because the data transmission between the high level cache and the low level caches have to overlap with computations. New geometry of cached zone should give better results for the processing of the sets of rays that make a 45-degree angle with all or some of the main axes. Furthermore, the cache should optimize the use of the burst capacity of memories.

The automatic setting of the cache parameters is also a major challenge. Some of the parameters have to be set dynamically to get higher performances, especially to manage special cases such as small set of rays at the border of the image. Also, the cache parameters are actually constant. The adaptive setting of the cache speed seems to be necessary but it is more challenging than the adaptive setting of the cache size, as already implemented.

V. Conclusion

In this paper we exposed the joint optimisation of a Ray Casting algorithm and a 3D-AP Cache memory hierarchy. The iterative Ray Casting algorithm have to be re-ordered to get a better cache efficiency. 3D spatial and temporal locality is maintained thanks to the phase locked propagation of rays at different level of parallelism: a single pipeline processes a set of coherent rays, several parallel pipelines are synchronized to form a group and a set of these groups are also synchronized to form a cluster. The resulting architecture is of memory saving and scalable to any size of grid volume. Measures on a prototyping board show the efficiency of this strategy for different kinds of ray geometry (visualisation and simulation of sinogram).

The results obtained for uniform grid traversal make us confident to extend this strategy to the processing of hierarchical grids. We are currently working on a multi-level cache that should prefetch parts of the tree structure of an octree.

This work demonstrates the versatility of the nD-AP Cache. This cache is validated for different applications with fetch sequences that are data dependent, non linear and, in this paper, control oriented. The analysis of the different target applications leads to the proposal of new features of the nD-AP Cache. The nD-AP Cache scheme is developed together with a methodology, this later being on the way to be automatized.
REFERENCES

Demosaicing on the TriMedia 3270

H. Phelippeau¹,², M. Akil², T. Fraga-Silva¹,², S. Bara¹, H. Talbot²

Abstract—In recent years, digital cameras have become the dominant image capturing devices. They are now commonly included in other digital devices such as mobile phones and PDAs, and have become steadily cheaper, more complex and powerful. In such devices, in order to keep cost down, incomplete colour information is recorded at each pixel location. Consequently, one of the most important operations in the imaging processing chain associated with digital cameras is the Colour Filter Array (CFA) interpolation. Clearly the quality of this reconstruction is of crucial importance to the overall picture quality experienced by the end-user.

This paper first compares the image interpolation quality of the state-of-the-art demosaicing algorithms with both subjective and objective criteria. It clearly appears that the best image quality is obtained with the Hirakawa algorithm [4]. However it is associated with a high computational complexity, unsuitable for a current mobile platform.

In a second part, we propose an optimized implementation of the Hirakawa algorithm for a mobile DSP platform. Our reference is the NXP TriMedia TM3270 [9]. We used the TriMedia Compilation and Simulation System (TCS) for optimization and profiling, and we made used of the following techniques: LUT, custom TriMedia operations, separability, loop unrolling, restricted pointers and bit shifting. Using such techniques, we decrease the number of cycles/pixel by 95%.

Index Terms—Bayer pattern, computational complexity, demosaicing, digital camera, interpolation, TriMedia, TM3270, VLIW

1. INTRODUCTION

Still digital imaging and video are constantly evolving among handled devices like digital cameras, camera phones, PDAs, etc. When a digital camera shot or video is recorded, the camera needs to perform a significant amount of data processing to provide the user with a viewable image. This implies that digital cameras require the use of a high-performance Digital Signal Processor (DSP) and hard-logic (ASIC). A generic NXP block diagram of a digital camera system is presented in Fig. 1. Our reference DSP platform is the TriMedia TM3270, an advanced digital signal processor that was designed mainly for executing multimedia tasks. In the digital still camera front end, either a CCD or a CMOS image sensor converts light into electrons at the photosites. The analog signal from the image sensor is filtered, amplified, and digitized by the high-speed analog to digital converter. An auto-focus system adapts the position of the lens, and an auto exposure loop adapts the sensor gain and the exposure time. The TriMedia DSP provides processing power to handle various industry-standard, computationally intensive imaging, audio, and video algorithms. An LCD interface can receives digital video or image for display and a memory stores embedded software and image data. An important step of the processing chain is the Bayer color filter array (CFA) interpolation also called demosaicing. Indeed, most recent digital cameras are using a Bayer color filter array placed in front of a single sensor (Fig. 1). The sensor samples a single chromatic value per spatial position, and an interpolation algorithm is needed to build a full RGB color image. The visual image quality depends greatly of this process. Many different reconstruction algorithms exist, with variables visual quality interpolation and variables computational complexity [1, 2, 3, 4, 5, 6], which are studied in Section II. Among these, we observe that the best visual image quality interpolation is obtained using the Hirakawa algorithm. However this algorithm has a high computational complexity. Section III is dedicated to the TriMedia/CPU64 architecture, Hirakawa demosaicing algorithm implementation and optimization on TriMedia TM3270 and the results. We conclude in Section IV.

¹H. Phelippeau, T. Fraga-Silva and S. Bara are with the NXP SEMICONDUCTORS, 2, Esplanade Anton Philips Campus Effiscience Colombelles, BP 20000 14906, Caen Cedex 9 (e-mail: {name.surname}@nxp.com).
²H. Phelippeau, T. Fraga-Silva, M. Akil and H. Talbot are with the Institut Gaspard-Monge, CNRS-UMLVEESIEE UMR 8049, A2SI Laboratory, Groupe ESIEE, BP 99, 93162 Noisy-le-Grand, France (e-mail: {name.first_letter.surname}@esiee.fr).
II. DEMOSAICING ALGORITHMS: RELATED LITERATURE

In this section we propose a comparison of the image visual quality and of the computational complexity between the several state-of-the-art algorithms [1, 2, 3, 4, 5, 6]. For the illustration, we used one sample coming from a Kodak database, commonly called “lighthouse”, obtained from the PCD format, extracted at the 512x768 resolution. This image was scanned at 3 colour samples per pixel from film original following the Kokak professional PhotoCD procedure. The image was spatially sub sampled to obtain an image consisting of only one color per pixel, in agreement with the Bayer filter color arrangement shown in Fig. 2.

A. Different demosaicing approaches

Gunturk proposed in [7] that demosaicing algorithms can be classified in three main groups. The first group is composed by algorithms with heuristics approach, in which we can include [1, 2, 3, 4, 5, 6]. The second group concerns the algorithms that treat the demosaicing problem as a restoration problem [7]. Finally the third group considers the demosaicing problem as an image formation inversion problem. This latter class is composed of particularly compute-intensive methods and is therefore not included in this study.

B. Demosaicing algorithms quality measurements

To evaluate the reconstruction quality we can use two classes of criteria, objective fidelity criteria and subjective fidelity criteria.

1) Objective criterion
An objective criterion can be obtained by calculating the mean square error (MSE) in each R, G and B plan between the original image and the interpolated image. Fig. 3 shows the mean square error obtained for the studied algorithms; it is measured over all 24 images of the Kodak database.

Fig. 2. Bayer pattern beginning sequence for the simulation

2) Subjective criterion
Image quality measurement with the subjective evaluation of human observers remains one of the most effective criteria. Fig. 4 shows the images resulting from the studied algorithms on a typically problematic pattern of the “lighthouse” image.

C. Computational complexity

In this part we are interested in the computational complexity of the demosaicing algorithms under study. As a reference we can look in [4] at how Hirakawa operates. The computational algorithm complexity can be obtained by counting the number of variable multiplications (Var Mults) and fixed multiplication (Fixed Mults), additions (Adds), comparisons (Compares) and absolute values (Abs Values). In which fixed multiplications means multiplications with a fixed coefficient. This operation can be hard coded with cumulative shift and additions. Table I shows the results obtained for the Hirakawa demosaicing method. Indeed the algorithm can be separated in five main stages as follows: the first step consists in the construction of a vertically interpolated full colour image and a horizontally interpolated full colour image. Given the method of images interpolation it is straightforward to count 15 additions and 5 multiplications per pixel. The second step consists of transforming these 2 interpolated images in the CIELAB color space. With the same method, we easily count 16 additions and 18 multiplications per pixel. The next step consists in the calculation of a homogeneity map with adaptive parameters. It will be used for the interpolation direction choice (vertical or horizontal). We count 24 multiplications,
48 additions, 49 compares and 12 absolute values for the homogeneity and 6 compares and 2 absolute values for the adaptive parameters. The last step consists to apply a median filter to reduce the interpolation artefacts. We count 1 fixed multiplication, 9 additions and 16 compares per pixel and per iteration. The same method is used to calculate the computational complexity of each other studied algorithms. Table II lists these results.

### Table I

<table>
<thead>
<tr>
<th></th>
<th>Var Mults</th>
<th>Fixed Mults</th>
<th>Adds</th>
<th>Compares</th>
<th>Abs Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpolation</td>
<td>0</td>
<td>5</td>
<td>15</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Color space conversion</td>
<td>0</td>
<td>18</td>
<td>16</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Homogeneity</td>
<td>24</td>
<td>0</td>
<td>48</td>
<td>49</td>
<td>12</td>
</tr>
<tr>
<td>Adaptive parameters</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Median filtering (per iteration)</td>
<td>0</td>
<td>1</td>
<td>9</td>
<td>16</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table II

<table>
<thead>
<tr>
<th></th>
<th>Var Mults</th>
<th>Fixed Mults</th>
<th>Adds</th>
<th>Compares</th>
<th>Abs Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bilinear</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Hamilton</td>
<td>0</td>
<td>5</td>
<td>14</td>
<td>1/2</td>
<td>15</td>
</tr>
<tr>
<td>Malvar</td>
<td>0</td>
<td>19</td>
<td>21</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Hibbart</td>
<td>0</td>
<td>6</td>
<td>12</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Gunturk</td>
<td>0</td>
<td>480</td>
<td>480</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Kimmel</td>
<td>0</td>
<td>120</td>
<td>180</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Hirakawa</td>
<td>24</td>
<td>24</td>
<td>88</td>
<td>71</td>
<td>12</td>
</tr>
</tbody>
</table>

III. TRI MEDIA/CPU64 ARCHITECTURE

### A. TM3270 VLIW Media Processor and programming environment

The TriMedia processors are supported by a TriMedia Compilation and Simulation (TCS) system (Fig. 5) that helps the software designer to simulate and optimize multimedia applications entirely in the C and C++ programming languages.

1) TM3270 VLIW Media Processor Core

The TM3270 is a 32-bits VLIW coprocessor of the TriMedia family [9] (Figure. 6). It operates at up to 350 MHz and executes 5 instructions per clock cycle, providing an extensive set of multimedia instructions for better fitting advanced video processing functions. It uses 128 32-bits registers (unified register-file), a 128kB data cache (4-way set associative) and a 64kB instructions cache (8-way set associative). A TM3270 has 35 execution units and supports SIMD multimedia and IEEE754 FP operations.

2) Programming environment: TriMedia Compilation and Simulation (TCS) tools

The TCS system speeds up creation of highly optimized multimedia applications. It provides a suite of system software tools to compile and debug multimedia applications, analyze and optimize performance, and simulate execution on a TriMedia processor. TriMedia TCS releases feature the following:

- ANSI-compliant C/C++ compilation system
- Source level debugging
- Performance analysis and enhancement tools
- Cycle-close machine-level simulator
- TriMedia core device libraries
- Example programs
- Comprehensive online documentation
- Includes pSOS operating systems from Wind River formerly Integrated Systems, Inc. (ISI)

![TriMedia Compilation and Simulation and system components](image-url)
B. Hirakawa demosaicing algorithm implementation using TCS tools

1) The Hirakawa demosaicing algorithm: steps description

The algorithm steps are as follows:

a) Bayer pattern interpolation
The first step consists to build a vertically interpolated and a horizontally RGB color image. We build first a vertically and a horizontally interpolated green image respectively by convoluting the Bayer pattern with the vertical element [-1; 2; 2; 2; -1] and the horizontal element [-1; 2; 2; 2; -1]. Red and blue color channels are then constructed with the known hue constant method. This consists in the bilinear interpolation of red and green, and blue and green, plane difference.

b) Color space conversion
This step consists to convert the vertical and the horizontal RGB color interpolated images into the CIELAB color space. We apply the known RGB to CIELAB transformation.

c) Adaptive parameters calculations
This step consists of calculating adaptive parameters in the vertical and in the horizontal interpolated image; this step establishes adaptive selection parameters; this parameters depend on pixel position, luminance and chrominance.

d) Homogeneity map calculation
The previous step permits to evaluate the neighbor’s homogeneity of a pixel both in the vertically and in the horizontally interpolated image. The final output pixel belongs to the image with the most homogeneous neighbors. Regions homogeneities are estimated by comparing neighbor pixels luminance and chrominance Euclidian distances to the adaptive parameters. The final step consists of convoluting the obtained homogeneity maps with a 3x3 unit filter to take into account the neighbor’s direction.

e) Interpolation artifact reduction
The goal of this step is to reduce the interpolations artifacts residues by apply a median filter in the green and red difference plane and in the blue and green difference plane.

2) Algorithm optimization steps

Various optimization methods supported by the TriMedia Compilation System as well as techniques for exploiting the fine-grain parallelism of the TriMedia architecture can be used. In this part each step of the algorithm is optimized separately. The language implementation supports the standard C library, as defined in the ANSI/ISO C Standard. No other libraries are supported. We used the following techniques and tools to improve the program execution times: loop unrolling, code alignment with –tmXXX_opt, restricted pointers, operations separability, look-up table, and custom TriMedia functions. (Fig. 7). The measure used to determine the performance of the following examples is the number of clock cycles required to perform a certain function. A general optimization goal is to minimize this number. This can be achieved by increasing the so-called instruction level parallelism of the code. The TriMedia processor is capable of executing up to five operations concurrently. We illustrate and explain these techniques for the optimizations of the steps a) to e) and we show their results in Table III. For the optimizations, we used the “lighthouse” Kodak sample extracted at 256x384 pixels and the TCS for TM3270 core processor running at 350MHz clock.

a) Loop unrolling
The naive version of the Bayer pattern reconstruction algorithm has different interpolation treatments according to the pixel color. These operations induce conditional choices (if conditions). By looking at Fig. 8 we realize that the Bayer grid is the repetition of a 2x2 color square. It is possible to eliminate the charge of the choice conditions by processing the four pixels of a 2x2 square at each loop step. Moreover, taking advantage of the VLIW parallelism, this treatment will improve the performance by scheduling the operations over the four pixels simultaneously. A code example of this method is showed below:

```c
for (j=0; j<height-1; j+=2)
{
    for (i=0; i<width-1; i+=2)
    {
        Img [(j+i)*width] = Red_pixel (i);
        Img [(j+i+1)*width] = GreenR_pixel (i);
        Img [(j+i)*width+1] = GreenB_pixel (i);
        Img [(j+i+1)*width+1] = Blue_pixel (i);
    }
}
```

Fig. 6. TM3270 organization
The loop unrolling technique was applied over all algorithm steps.

b) Look-up tables

The color space conversion uses functions calls for calculating the cubic root. In order to decrease the computational complexity of this transformation, we can replace these functions calls by a simple memory fetch using a look-up table.

c) Custom TriMedia operations

The TM3270 has a set of custom operations such as absolute value calculation, maximum or minimum function that enable the programmer to use the processor optimally and can be executed really faster than usual C/C++ operations. Some examples of these operations are listed on Fig. 7.

TriMedia architecture allows working with 8-, 16- or 32-bits data types using custom operations. This means that it is possible to represent 4, 2 or 1 pixel(s) in each 32-bit register depending of their data representation (8-, 16- or 32-bits). In this optimization we used 32-bits operands like IABS, IMAX, IMIN and INONZERO for the adaptive parameter calculation and homogeneity map calculation.

d) Separability

The three techniques presented above are known techniques for algorithm optimization in architecture level. Algorithmic optimizations were also performed in homogeneity map and interpolation artifact reduction calculation.

The homogeneity map calculation has a convolution filter with a 3x3 mask for average calculation. Its optimization can be operating by separating the convolution filter application, first vertically and after horizontally. The convolution separable matrix can be written as follows:

$$
\begin{pmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{pmatrix} =
\begin{pmatrix}
1 \\
1 \\
1 \\
\end{pmatrix} \oslash
\begin{pmatrix}
1 & 1 & 1 \\
\end{pmatrix} (1)
$$

However this method has the disadvantage of using a temporary image between vertical and horizontal convolutions, enable to pass from 8 additions per pixel to 4 additions per pixel for the convolution of a mask 3x3, reducing to 50% the convolution complexity.

As it is said before the interpolation artifacts reduction consists to apply a 3x3 median filter on the difference image between the green and red channel and between the green and blue channel. In order to decrease the computational complexity of the median filter we can apply an approximated median filter describe as follows. Let us consider \( m(.) \) as a median function, and \( A \) is a 3x3 scalar image square. If \( A \) is expressed as follows:

$$
A = \begin{pmatrix}
a & b & c \\
d & e & f \\
g & h & i \\
\end{pmatrix} (2)
$$

A good approximation \( \hat{m}(.) \) of \( m(.) \) can be expressed as:

$$
\hat{m}(A) = m\{m(a,b,c), m(d,e,f), m(g,h,i)\} (3)
$$

This approximation enables to pass from 36 comparisons to 12 comparisons, and then reduce of 67% the computational complexity.

C. Experimental results

In this part we will comment the experimental results obtained for the Hirakawa algorithm optimization. Table III presents the results of the optimization with the TCS system on the TM3270 core.

1) Naive implementation of the Hirakawa algorithm
The first row of the Table III presents the results obtained with the naive Hirakawa algorithm implementation, naïve means without any optimization. This version uses external function calls, if and else conditions, complex mathematical operations (division, power, cubic root) and none loop optimization. In this version, the interpolation artifact reduction has already been implemented with an approximated median filter described in section B.2.d.

We can note that the homogeneity map calculation is the most complex step of the algorithm, taking up to 77.1% of the total complexity. This step needs to calculate 13 absolute values and 39 powers per each pixel, complexity increase with functions calls. The same effect also occurs on the color space conversion (6 powers per pixel) and the adaptive parameters calculations (4 absolute values and 8 powers per pixel). Color space conversion also works with floating points, increasing its complexity calculation. Bayer pattern interpolation and artifacts reduction calculations have their complexity increased due to if and else conditions.

At each step of the algorithm, we applied appropriate techniques to decrease calculation complexity; results are shown on Table III.

The last column of Table III presents real time processing capabilities. We consider an image with a standard VGA resolution (640x480 pixels) and a processor with a 350MHz clock ratio. The algorithm spends up to 20 seconds for process an image on the TM3270.

2) Performance improvements of the Hirakawa algorithm

Appropriate optimization techniques described in section B are applied at each step of the algorithm.

In the color space conversion calculation, we replaced the cubic root operation by a look-up table; we remove also function calls and complex mathematics calculations. For this step we reduced the number of cycles per pixel from 2788.8 to 136.2 (95% of improvement). Considering the overall number of cycles, we calculated an improvement of 10.9%.

The second row of Table III shows the optimization results using the custom TriMedia operations and replacing powers with multiplications. We removed function calls, complex mathematics' calculations and if and else conditions for 3 algorithm's steps (b, d and e). This increases significantly algorithm performances, passing from 21,6 thousands cycles per pixels to 1,4 thousands (improvement of 94% reporting from naïve algorithm).

We can decrease the homogeneity construction map complexity by separating the convolution mask in two masks, it allows to reduce the number of cycles from 393.7 to 382 for this step (3% of improvement).

Loop unrolling was applied at all algorithm's steps, reducing the algorithm complexity from 1433.7 to 893.7 cycles per pixel (37.7% of improvement). The main contribution was observed over the homogeneity map calculation step (from 382 to 121 cycles/pixel, 68% of improvement), where we evaluate in parallel the homogeneity of a pixel according its neighbors. This technique doesn’t increase the interpolation artifact reduction performances. This algorithm's step uses a median filter, it implies vectors sorting and if and else conditions, hindering in its performance.

The last two rows of Table III show two known optimization techniques: restricted pointers and bit shifting. The use of restricted pointers tells the compiler that no overlapping is possible, so it doesn't need to wait a store for the pointed value before loading another. It is also possible to replace the multiplication and division operations by a bit shifting, since these operations are realized in base 2 over integer operands. Using these techniques, we decrease the number of cycles per pixel from 893.7 to 828.3 (7% of improvement).

We observe in Table III that we can increase the performance of 96.6% reporting to the naïve implementation by using all optimizations. However, the interpolation artifact reduction step doesn’t have a good improvement reporting to naïve implementation, porting high complexity calculation to the optimized version. This step affects the algorithm in 78% of its complexity.

IV. Conclusions and future work

In this paper we initially showed using the objective Mean Square Error measure and the subjective measure of visual image quality evaluation by human observers that the Hirakawa demosaicing algorithm gives the best interpolated image quality. After the presentation of the TriMedia/CPU64 architecture and particularly the TM3270 VLIW processor core, we presented the TriMedia Compilation and Simulation (TCS) tools features and capabilities. In a third part we showed how we could optimize the algorithm implementation using various basic optimization methods supported by the TCS as well as techniques for exploiting the fine-grain parallelism of the TriMedia architecture (LUT, loop unrolling, separability and custom TriMedia operations). We finally showed that the Hirakawa demosaicing algorithm could be optimized to reduce computational cost by over 96% compared to its naïve version. Still better results could be obtained using instruction level parallelism optimizations, cache access optimizations, fixed point representation, and advanced custom TriMedia operations. A particular attention has to be attributed to the IAR step that represents 78.11% of the final optimized algorithm; a solution to decrease the complexity may be found by study the existing quick sort methods for the median function computation.
# References


[3] H. S. Malvar, Li-Wei He, R. Culter, “High-Quality Linear Interpolation For Demosaicing of Bayer-Patterned Color Images”, Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA.


<table>
<thead>
<tr>
<th>Table III</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIRAKAWA DEMOSAICING ALGORITHM OPTIMIZATION ON THE TriMedia Compilation System (CORE TM3270)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Naive</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>LUT</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Custom operations and multiplications</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Separability</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Loop unrolling</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Restrict pointers</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Bit shifting</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Efficient Hardware Implementation of Connected Component Tree Algorithm

N. Ngan, F. Contou-Carrère, B. Marcon, S. Guérin, E. Dokladalova and M. Akil

Abstract—The graph-based image processing algorithms are very promising, as they allow bridging the gap between low- and high-level processing implementations. The algorithms using connected components tree have been used (under several variations) in numerous applications going from image filtering, video segmentation, image registration, image compression to data visualization. However, the hardware implementation remains a very challenging problem due to the used complex data structures.

The paper presents an efficient hardware implementation of the connected component tree algorithm. After the discussion of existing algorithms the attention is paid to the efficient connected component tree construction on an FPGA platform. The experimental results are presented and compared to the execution on VLIW processor and Pentium.

Index Terms—Image processing, component tree, connected filters, graph, hardware implementation, FPGA, VLIW, embedded architecture.

I. INTRODUCTION

At present, our society lives significant mutation caused by the democratization of the vision sensor - in terms of low cost and quality progress. Since several years, the camera system applications spread in everyday life. One can distinguish three major application groups: public and private security, automotive applications (such as blind spot detection, collision avoidance, parking assistance) and interactive urban equipment. We assist to a real boom of the well-known security applications (public space or home protection, traffic control). However, most used cameras include image sensors capable only of picture or video acquisition. Obviously, a centralized transfer and processing of images acquired by thousands of cameras is not feasible today. For that reason, the research work focuses on the redevelopment of vision-based systems situated just behind the sensors, making cameras “intelligent” and transferring only pertinent information.

Most vision systems based on “intelligent” cameras already available on the market are based on a DSP or an FPGA where the DSP/FPGA is programmed/configured to fit the application. Both solutions, however, find only a partial answer in the terms of the performance/flexibility trade-off. They are mainly penalized by i) the variety of existing mathematical and algorithmic domains and the necessity to simultaneously execute different algorithms; and ii) by the development time and cost of every dedicated solution for each particular application.

Given the enormous investment in the equipment, the users start to ask for flexibility of such systems, especially in the public domain. To achieve the system flexibility in the variety of existing or possible future vision applications, ranging from the low- to the high level, one needs to find a common denominator or its approximation fitting the entire chain of image processing. This is the “never-ending” problem of the hardware design space exploration [3], [12], [14], [25] or [27] (see [2] for a list of reconfigurable architectures providing various flexibility degrees, i.e. [6]). To overcome the algorithm disparity, making the parallelization and data access optimization effort boundless, our research focuses on the formalism using graph theory.

We can cite numerous applications formulated by graph approach: image filtering [13] and segmentation [8], video segmentation, image registration [16], image compression [23] and data visualization [7]. Note that many other important applications are organized around graphs (e.g. data and knowledge bases, CAD optimization, numerical problems, simulations).

The main advantage is a possibly unified formalism using the tools from the computational geometry: connectivity, data adjacency and proximity, graph coloration and transformation, tree construction and merging etc. However, the hardware implementation is very challenging problem due to the used complex data structures (see Fig. 1). The graph structures can

Fig. 1. Examples of graph structures: a) regular grid, b) general graph c) component tree.
be very large and, for each operation, the applications need to access to a large, data-dependent portion of the graph.

Among the graph-based algorithms, we are interested in the image processing operators using connected components tree ([17], [20], [23]). Although it has been used (under several variations) in numerous applications going from low-level (filtering, image to high-level (topological watershed) processing, it remains challenging problem for the efficient embedded hardware implementation. From the three phases of the component tree applications (see Fig. 2), we concentrate on the implementation bottleneck - the component tree creation.

The remainder of the paper is organized as follows. Section II discusses the general state of the art of the connected components extraction. We give some examples of known hardware implementations. Section III details the efficient FPGA implementation of connected component tree. The reader can find the performance comparison in the Section IV. Finally, the Conclusions recall our research work and present its extensions.

II. STATE OF THE ART

The computation of connected components of an image is the first step of many applications. A number of existing publications prove the importance of availability of an efficient parallel implementation. Generally speaking, three implementation groups exist in the literature:

- **Level set based** – each gray level is processed separately which makes the parallelization very easy. This approach appears efficient for some filter implementation but can penalize high-level operators by increasing the number of processed pixels [1].

- **Queue based (region growing)** – a system of hierarchical queues (FIFOs) is used to realize the given order in the pixel processing. The major inconvenient of those methods is the need of a specialized memory system of FIFOs, often exceeding in size the original image [15]. In this framework, the main parallelization effort consists of a parallel access to the given neighborhood. A dedicated memory organization has been proposed in [21] permitting to retrieve the 4- or 8-connected neighborhood in only one clock cycle.

- **Component tree based** – the connected components of the level sets can be organized in a tree structure, called component tree. Only several references show how the tree-based algorithms can be implemented in a chip multithreading way (CMT) [18].

As special cases we can mention Associative net [19] and GraphStep Architecture [11] where authors present a distributed computing on a general graph. Nevertheless, these solutions are not yet ready for mobile embedded applications.

1) **Connected component tree algorithms**

Our study considers two reference algorithms: the often-cited Salembier algorithm [23] and the Najman-Couprie algorithm [20]. While the Salembier algorithm is generally faster, its worst-case complexity is quadratic; the complexity of the Najman-Couprie algorithm is linear.

a) **Algorithm description**

In both referenced algorithms, a connected component is a set of points verifying the following: i) for any pair of points from the set, there is a sequence of adjacent points connecting them, itself included in the component, ii) all points are above a given threshold. The inclusion operation allows organizing the connected components in a tree structure (i.e. Fig. 3 and Fig. 4).

In order to define the parent-child relation (Fig. 4), two principles are used: flooding (Salembier) or emerging (Najman-Couprie). In both cases the consequence is the need of an order of processing the pixels (sorting the pixels). The flooding starts from the image minima and progressively floods the processed levels (image relief). The emerging principle will start with the image local maxima and it will decrease the processed levels.

Then the component tree construction can be outlined by two generalized operations:

- Process a point according to the given order, provided that the point is not already processed.
- Look for and assign the father.
- Compress paths in the graph (according to the Tarjan’s Union-Find algorithm [24]).

Often, during the tree construction are collected the attributes used for the filtering. Hence the filtering is performed immediately after all pixels have been processed.

In the following section, we describe the hardware implementation specificity of these operations. We do not detail the initial algorithms here as it overpasses the scope of this paper. One can find exact description in [20] and [23].

III. HARDWARE IMPLEMENTATION

A. **Choosing an adapted tree structure**

Before choosing an algorithm to implement in the FPGA, we have to determine how to store our tree in memory. The idea is to find a structure that can contain a deterministic number of nodes in the tree. Moreover, we need to define exactly the size of each node in order to map efficiently the tree in memory and ease usual computations on the tree.

There are two main types of tree structure: the component tree structure and the point-tree structure [4]. See (Fig.3) as an example of the differences between a component tree and a point-tree (Fig.4).
For a component tree, each node represents a set of points (pixels), which have the same level in the picture. The point-tree is the burst version of the component tree. Each node of a point-tree represents a pixel in the picture. This tree is consequently much deeper than the component tree but the number of nodes in the final tree is known (it is the number of pixels in the picture).

The component tree direct structure is obviously not adapted for a memory mapping because neither the number of nodes nor the size of each node is known a priori. This number depends on the different levels and the positions of the pixels.

The first advantage of a point-tree is about the fixed number of nodes. The second advantage is that each node has a fixed size that is an important feature; we can therefore specify the size of the memory knowing the number of nodes.

Besides, the “point-tree” is an interesting solution compared to others solutions such as the binary tree or the canonical tree [5] which have also a fixed number of nodes. The binary tree and the canonical tree are more complex in their structure and the number of leaves in this type of tree is also more important. These characteristics would be critical in terms of processing time when dealing with the tree pruning.

**B. Tree memory mapping**

The main issue of mapping the tree in memory is how to model the connections between the nodes. These relations between parent and child nodes are easily made by using pointers in standard languages like C++. Moreover, we do not know a priori the number of child nodes of each node in the tree. One of the main features of a tree structure is that each node can have no more than a single parent node. The most relevant strategy is thus to reverse the pointing between the parent and the child: each child node has only to store the address of the parent node in order to create the connection. Note that the root node of the tree stores its own address.

These connections are established by using a RAM block which words are as wide as the size of the address of the pixels. This size is determined by the size of the picture. For instance, a 160x120 picture needs 15 bits words (8 bits for the coordinate X and 7 bits for the coordinate Y). The tree is simply a “parent address table” (Fig. 5). We call it “Parent Table”.

It is important to notice that this implementation strategy has an impact on the reading direction of the tree: the reading must start from the leaves to the root. This direction is particularly adapted for computing some criteria such as the height.

In order to filter the image, some attributes (height, area, volume, etc.) can indeed be computed for each node after the tree construction (Fig. 6.a). They quantify the relative importance of each node in the “point-tree”.

Additional control bits (Fig. 6.b) are also needed for the tree construction. Three control bits are used: “is_processed”, “parent_ok” and “son_ok”. The first bit “is_processed” is necessary for the algorithm and the two last bits are useful for determining if a pixel node is a leaf (son_ok = ‘0’) or a root (parent_ok = ‘0’) of the tree. Particularly, we would have to find the root of the tree during the computation to link the nodes.

The same address is used for the three tables: Parent Table, Attribute Table and Control Table. These three tables (Parent Table, Attributes Table and Control Table) covered our tree mapping concept in memory.

**C. Global Architecture**

The global architecture (Fig. 7) can be divided in three main blocks: a block to sort the pixels (pixel sorter block), a block to compute the tree (tree and criteria computing block) and a block to filter the tree (tree pruning block).

This architecture describes an algorithm inspired from Tarjan’s Union-Find. See [4] and [20] for a detailed description.

![Diagram](image-url)
Note that the tree computation block (Block 2) does not need any pixel values. It only needs the pixel addresses considering that the pixels are already sorted by the previous block. The whole tree construction is based on address manipulations with the use of the control bits. We denote by N+1 the total level number in the picture.

Figure 7. Global architecture

1) *Block 1: The pixel sorting block*

The first step of the algorithm is the pixels sorting from the highest to the lowest level. In others words, the pixels are processed by decreasing order.

A first approach is to use a “hierarchical queue” (Fig. 8). Each pixel address is stored in a specific FIFO representing a level creating thus an “address histogram” of the picture. The main advantage of this strategy is to sort the pixels “on-the-fly”. However, it consumes an important quantity of memory blocks because it needs as many FIFOs as N+1 levels in the picture, and the said FIFOs need to have a length equal to the number of pixels in the picture.

Figure 8. Hierarchical queue

The good solution to avoid the high consumption of memory blocks is to use the *counting sort method*. The objective is to build directly the queue in a memory block (Fig. 9) which size is equal to the number of pixels in the image. This technique is divided in two image reading passes.

The first reading pass is dedicated to the histogram creation. This pass divides correctly the queue for each level and index pointers are generated in each created division to prepare the next pass.

In the second reading pass, each index pointer increases according to the pixel value and the queue can be filled. The different pixels addresses are thus progressively distributed in the queue.

The direct hardware transcription of this method is not optimized. Indeed, the first pass would require as many counters as the number of levels in the picture. What is more, the index pointer start values are generated by cascading adders at the direct counter outputs and this structure could be critical for the maximum frequency allowed in the architecture. Others counters are also needed for the second pass to increase the index pointer values.

A good strategy to overcome this difficulty is to focus on the index pointer start values generation. A cumulative histogram implementation (Fig. 10) is the right solution to avoid the cascading adders during the first pass. The cumulative counting can be realized with OR gates for each counter input commands: the index counter is increased either by its direct command or a command from an upper level counter. A counting command is activated according to the pixel value. This counting command can therefore be transmitted to lower level counters and the index start values are gradually generated during this pass.

Note that the counting commands are shifted by one level to generate the right index positions for the first pass. During the second pass, the input counting commands switch to the direct level counting commands so that the queue can be filled correctly according to the pixel value.

Figure 10. Counting sort architecture
This architecture does not need any FIFOs and is very economic in terms of memory resources. However, this technique is a bit longer compared to the previous one (the hierarchical queue). Although the first pass could be realised “on-the-fly” during the picture storage, the second pass brings additional time for the second image reading.

Nevertheless, the counting sort technique is the right choice to save memory blocks.

2) Block 2: The tree computing block

The tree computing block processes sequentially the pixel addresses in the queue to compute the parent address table (Tab Parent). The pixel address in the queue is considered as the origin pixel address for the structuring element and its neighbors are processed sequentially respecting the algorithmic procedure. That is why a 4-connected adjacency relation (Fig. 11) is used for each origin pixel to reduce the processing time. The neighbor processing order is not important and a clockwise order can be chosen for instance.

![Fig. 11. 4-connected adjacency relation](image)

a) Direct architecture implementation

The tree computation block FSM (Finite State Machine) starts by reading the origin pixel address queue and then processes sequentially its neighbour pixels (Fig. 12). It ends with the origin pixel address table updates. In this last state, the pixel control bit “is_processed” is turned on indicating that the pixel has already been processed and has been written in the different tree tables. This pixel control bit has to be checked in priority for all the pixel neighbours before beginning to find the tree local root. If this bit is turned off, the FSM jumps to the following neighbour to be processed. Otherwise, the FSM starts the finding root procedure. This procedure corresponds to a local memory reading loop realised by consecutive address checking. During the finding root procedure, the child pixel checks the parent pixel control bits until the ‘parent_ok” control bit is turned off. This means that the parent pixel is the local tree root and it can be linked to the origin pixel (Link state).

The tree computation ends when the pixel origin address queue is empty meaning that all the pixels in the image have been processed.

To prepare the tree pruning and the attribute computations, the tree leafs need to be stored in a FIFO called “Leaf FIFO” along the tree construction. Notice that the tree construction begins with leafs. We can affirm that if an origin pixel has not got any child pixels at the end of its processing, it means that it is a tree leaf.

The criteria computing sub-block can start when the tree is complete. This block consists on the Parent Table reading (starting from the tree leafs) and the progressive attributes computation. According to the criteria, these computations may need the pixel value of each node.

![Fig. 12. Tree computation block FSM](image)

b) The shortcut technique implementation

Obviously, the deeper the tree branch is, the longer the finding root procedure takes. This difficulty can be resolved by simulating a virtual path compression. This technique is one of the two key heuristics to reduce the union-find algorithm complexity. See M.Couprie and L.Najman’s paper [20]. This compression can be done by adding another table containing the local root of each pixel during the tree computation. This table virtually simulates a smaller tree and the finding root procedure is accelerated.

![Fig. 13. Tree computation block interactions](image)

This shortcut technique can be implemented by the combination of a FIFO storing the pixels in the tree branch that
need to be updated called “Root FIFO” and a table called “Root Table” identical to the “Parent Table” (Fig.13). During the finding root procedure, each recursion stores the current parent pixel address in the “Root FIFO”. Another state called “Write root” is added right after the Link state to update the pixels in the FIFO by the origin pixel address in the “Root Table”. Thus, the origin pixel address becomes virtually the direct parent of those updated pixels.

The Root Table has the same size as the Parent Table and the Root FIFO has to be as large as the number of pixels in the picture (in the ideal case).

3) Block 3 : The tree pruning block

This block (Fig. 14) is the final step of the image segmentation. Considering the point-tree structure, the tree pruning block has to deal with the image stored in memory because all pixel values are needed to characterize a component (a set of pixels with the same value) in the tree.

The “Leaf FIFO” gives the reading starting points of the tree and according to the parent pixel values and their attributes, the system decides to store the child or parent pixel address in a FIFO called “Zone FIFO”. This FIFO aims at grouping the pixels in the tree that need to be modified. The modifying value corresponds to the last parent pixel value that does not respect the criteria. The tree pruning procedure ends when all the leafs have been processed by the block.

The output image (or segmented image) has to be stored in another memory block to avoid overwriting the pixel value in the original image.

![Fig. 14. Tree pruning block interactions](image)

The tree pruning block FSM is quite simple. The leaf pixel address is taken from the “Leaf FIFO”, in order to pick up its value in the original image memory, its parent pixel address in the “Parent Table” and its attributes in the “Attribute Table”. If the pixel attribute respects the criterion, its address is then stored in the “Zone FIFO” to overwrite all the concerned pixels with the last parent pixel value whose attribute does not respect the desired criteria.

IV. EXPERIMENTAL RESULTS

The architecture previously presented, has been simplified to be implemented on an FPGA Stratix II 2S60 (Altera board). The input image resolution is 120x160 coded with 8 bits pixels and the different address tables (Parent Table, Root Table) must have 15 bits words. As mentioned above, the Control Table has 3 bits words (“is _processed”, “parent_ok” and “child_ok”). All tables can be synthesized into on-chip memory blocks especially M4K and M-RAM.

These on-chip memory blocks can be divided into the “Parent Table”, the “Root Table”, the “Control Table”, the origin pixel address queue and the two images (one for keeping the original picture and the other for storing the segmented picture).

Note that the Attribute Table is not necessary to build or to prune the tree. The pruning can simply be tested based on the pixel values and specified by arbitrary threshold. We want to focus on the tree processing time and validate our architecture conception.

We can also decrease the number of level counters by only taking the four most significant bits of the input pixel value. The FIFOs (Root, Leaf and Zone) size are defined according to the free memory space left. Smaller FIFOs decrease the overall performance. Note that it is possible to keep the same FIFO for the Root FIFO and the Zone FIFO because they are not used at the same time.

A. Implementation results: area

Table I and Table II contain the results in terms of memory and resource utilization obtained after the synthesis on the StratixII FPGA.

<table>
<thead>
<tr>
<th>Table I</th>
<th>MEMORY BLOCK OCCUPATION STRATIX II 2S60</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>~Memory Division</td>
</tr>
<tr>
<td>Original Image (120x160)</td>
<td>262144</td>
</tr>
<tr>
<td>Segmented Image (120x160)</td>
<td>262144</td>
</tr>
<tr>
<td>Origin Pixel address queue</td>
<td>288000</td>
</tr>
<tr>
<td>Parent Table</td>
<td>491520</td>
</tr>
<tr>
<td>Root Table</td>
<td>491520</td>
</tr>
<tr>
<td>Control Table</td>
<td>98304</td>
</tr>
<tr>
<td>Root FIFO</td>
<td>30720</td>
</tr>
<tr>
<td>Leaf FIFO</td>
<td>30720</td>
</tr>
<tr>
<td>Zone FIFO</td>
<td>30720</td>
</tr>
<tr>
<td>Total Memory</td>
<td>1985792</td>
</tr>
<tr>
<td>Occupation 2S60</td>
<td>78%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table II</th>
<th>SYNTHESIS REPORT STRATIX II 2S60</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resources</td>
<td>Quantity</td>
</tr>
<tr>
<td>Combinational ALUTs</td>
<td>2680</td>
</tr>
<tr>
<td>Logic Registers</td>
<td>3564</td>
</tr>
</tbody>
</table>
B. Implementation results: timings

We use two types of image: a GradientTest picture and a Man picture taken from an infrared camera (Fig. 15). The Table III illustrates the complexity of the tree construction in terms of the number of connected components. The resulting execution time for both images is resumed in the Table IV.

Since the tree building execution time depends on the image complexity (its contents) the execution time is not known a priori. This complexity is not only based on the number of gray levels and the connected components but also on their mutual relations (adjacency).

Note that in our examples (Fig. 15), even if the number of connected components for the Man picture is higher than the GradientTest picture (Table III), the execution time is lower (Table IV).

The shortcut technique allows to reduce the dependency on the image complexity. This shortcut technique is less efficient for the GradientTest because the pixels are already spatially ordered.

C. Comparison

In order to evaluate the computing performance of our implementation, we have measured the Salembier and Najman-Couprie algorithms on a desktop PC and on a configurable VLIW processor [26].

The choice of desktop PC is straightforward as an accessible and common development environment. The VLIW architectures are characterized by their capabilities to issue and complete more than one operation at a time with a reduced hardware complexity. The concurrent operations are explicitly encoded in the instruction word. This allows exploiting fine-grain parallelism including pipelining, multiple computing resources, while specifying independent operations per instruction. These features make the VLIW architecture interesting for embedded applications. Among the VLIW devices already available on the market of mobile devices one can cite Trimedia [22], CT3616 [9] and C62xx/C64xx from TI [10].

At the same time, the fine-grain parallelism extraction is important for the component tree construction algorithms where the execution is strongly dependent on the image contents. Table V shows the two VLIW core configurations.

Generally speaking, the 4-issue VLIW configuration multiplies by two the 2-issue VLIW hardware resources. Consequently, the 2-issue VLIW should allow exploiting higher instruction parallelism. The VLIW processors have been simulated with the Trimaran software and the results for Salembier and Najman-Couprie algorithms are presented in Tables VI and VII.

Assume the following definition of Instruction Level Parallelism (ILP):

\[
ILP = \frac{Average\_Issue\_Operations}{Total\_Cycles} \quad (1)
\]

We can see the low ILP almost independent on the VLIW issue width. It illustrates the complexity of the algorithm parallelization. The doubling of computing resources is useless because of the data access bottleneck.
Table VIII contains the theoretical estimated execution time (at 50MHz and 200MHz) for both configurations. These results are compared with Pentium (3GHz and 2GB RAM).

V. CONCLUSIONS AND FUTURE WORK

The paper presents an efficient hardware implementation of connected component tree algorithms. After the introduction of the state of the art, the involved implementation challenges are discussed. The results of the FPGA implementation are presented for each phase of the component tree construction.

The achieved hardwired performances are compared to the results on a Pentium processor (at 3GHz) and configurable VLIW cores (with 2 and 4 issues). The resulting minimal speedup of our implementation at FPGA is 2. The poor instruction level parallelism, obtained in the VLIW cores, illustrates well that the bottleneck of the studied algorithms resides in the access to the data.

Assuming that this study is motivated by the search for a new approach to the flexibility of embedded systems, it demonstrates the FPGA implementation feasibility of relatively complex algorithms. In the next, we will concentrate on the generalization of the proposed architecture and power consumption characteristics.

Concerning the design challenges, the main objective is to explore and improve the interconnections between computing and memory resources.

ACKNOWLEDGMENT

The authors would like to thank M. Couprie from A2SI laboratory at ESIEE and B. Deloison from SAGEM for the very useful discussions on the component tree algorithm implementation.

REFERENCES


Algorithms selection and adaptation in accord with architecture for RBF neural network based face authentication SoC.

Lionel Pierrefeu
Laboratory Hubert Curien - IMAGe
University Jean Monet
18, rue B. Lauras
FRANCE
lionel.pierrefeu@univ-st-etienne.fr

Jacques Jay
Laboratory Hubert Curien - IMAGe
University Jean Monet
18, rue B. Lauras
FRANCE
jay@univ-st-etienne.fr

Abstract: This paper describes the algorithms applied to a Radial Basis Function (RBF) neural network. This neural network is used as a classifier to design a human face authentication system. The aim of this project is to obtain a low cost system on chip (SoC) to replace password identification for one person on mobile devices. Several parts of the neural network need to be modified to obtain good performances for this application with cost limited hardware resources. For the system design, the algorithms are selected and adapted in accord with architecture implantation on hardware platform (methodology AAA). Also we present different adaptations made for a full integration of RBF neural network (learning and recognition steps).

I. INTRODUCTION:

Authentication and recognition systems based on biometric measurements are in full development because of their potential applications in many research areas, such as surveillance or security for access control or personal identification. Some systems rely on stable characteristics of the body such as fingerprints, hands or iris geometry, other exploit dynamic features such as voice. Human face contains a lot of details which can be used to recognize a person, and the acquisition is very easy with a simple image sensor.

Our research focuses on the development of an industrial system on chip (SoC) for face based authentication. It would replace password identification in electronic consumer’s products such as cellular phones or computers. This is a challenging problem since the system must work under uncontrolled conditions (light, pose) [15]. Moreover, it has to fulfill both practical and industrial requirements. Indeed, from the user point of view, the system must be easy to use and to configure. From the industrial point of view, it has to be a real time and a low cost mono chip system.

As a result, the design conception must follow the AAA methodology (Adequation Algorithm Architecture). The chosen algorithm for face authentication is a radial basis function (RBF) neural network. This classifier has been selected for its good ratio between performances and complexity [1][2].

This neural network offers good performances and presents a very suitable architecture for process parallelization [3].

The RBF neural network configuration depends on several parameters which must be adjusted to obtain satisfying classification performances. Some of them are set during the system design. The others are determined during the learning phase and are user-dependant. So we present some solutions to perform a good parameter configuration for a limited cost. The first configuration step is partially performed by the user. It consists of acquiring several images of the user as reference images. Some tests are proposed to verify the quality of image set. The second step is the kernel configuration. We proposed an automatic solution to compute it and, at the same time, save hardware resources. Then we present simplification made for the neural network weighting configuration.

The article is organized as follows: first a general description of the RBF neural network is proposed. Secondly neural network parameters are introduced with their respective calculation solutions. Finally we present results (times, surfaces) obtained with proposed algorithms.

II. AUTHENTICATION ALGORITHM:

A. RBF neural network presentation:

In this project, the aim of an authenticator like a RBF neural network is to perform a classification between different classes exploiting their universal approximation properties. Indeed, RBF algorithm is a kernel-based network composed of several radial activation functions for interpolating [5][6][7][8] in high dimension. It allows to compute the complex decision regions by overlapping the radial functions. A training phase with a reference data set (images face of the person to identify) allows to determine the parameters of the functions. The parameters are defined with a goal of convergence towards the clustering function that leads to optimal classification results. After a short overview of the neural network principle, the RBF parameters and training method will be detailed.
B. RBF presentation:

The RBF structure \([2][5][9][10]\) is composed of three layers (Fig. 1). Like multi-layer perceptron, each one is fully connected to the following one. The first layer of the network corresponds to the characteristic input vector \((p_i)\). In our case, data of characteristic vectors are obtained from pixels intensity of the normalized face image. Then the input layer dimension is equal to the input data size. The second layer consists of the hearth of the network intelligence. It is used to cluster the input data. Each hidden unit performs the kernel function on the input vector. The output layer performs a weighted sum (6) of all hidden nodes.

\[
f_j(c, p) = \frac{d(c, p)}{\sigma}
\]

As we see, each Gaussian function permits the classification cluster interpolation. The interpolation function is adjusted with learning steps. It composes of different algorithms which determine and fix each kernel parameters (Gaussian width, centroid). So we will present in following parts each steps for the learning procedure and present all respective algorithms developed to automate this treatment.

III. ALGORITHMS OPTIMISATION:

To obtain an implantable version of the neural network, some modifications are needed. At the same time, algorithms need to be added for the performances improvement. In the following parts, we will present algorithms developed for parameters configuration.

A. Reference vectors determination:

The references vectors of kernels function are proposed by the user. It’s correspond to the center of each radial function and it is used to compute the distance to the input vector. This is a crucial step which determines the performances of the system. Even with a very good classifier a good set of reference images is needed. This is the base to obtain good performances. So we propose an automatic method to help user for the reference images acquisition. Before that, we need to determine what a good set of reference images is.

Image reference set need to be acquired in condition close to usual condition of system utilization. On top of that, images need to present enough details of the object (human face). For the reference images acquisition procedure, it asks to the user to present his face in different orientations to the image sensor (see example figure 2). To obtain a good system configuration we need a lot of details of user face, it leads to have images with enough difference from one to another. It also has an importance for the definition of following parameters. So to sum up: images need to be representative and enough different to present different details.

![Fig. 2. Example of a set of face images.](image)

To help the user in this procedure, we have developed an algorithm which determines the quality of images set, and rejects images too similar or too far from the others images. This algorithm is simple and
does not need a lot of hardware resources and offers good results. The treatment is not based on the images contents whereas it only evaluates the distances between all images. Accepted images must respond to the test, and must have mean distance contained between two thresholds.

C. Method description:

The first step is to compute a distance between all centroids pairs (DBCP) to form the inter-distance matrix (3).

\[
M = \begin{bmatrix}
1 & x_{12} & x_{13} & \cdots & x_{1n} \\
x_{12} & 1 & & & \\
x_{13} & & \ddots & & \\
\vdots & & & \ddots & \\
x_{1n} & & & & 1
\end{bmatrix}
\]

For each image we cumulate all distances from others images. The result obtained is compared with two thresholds in order to determine whether the images are too close or too far from the others images. The first threshold is used to reject images too similar to others. The second one rejects images too different to others, it may detect images acquired in bad conditions. If an image is rejected, the user needs to acquire a new image.

B. Gaussian width:

As we see, the radial function selected is the Gaussian function. This function is defined with two parameters, the center of the Gaussian describes in the previous section (reference vector) and the width which corresponds to the kernel influence. The role of the kernel is to filter the distance calculated between reference vector and input vector. The Gaussian width determines the selectivity of the kernel.

For the sake of clarity, using a two dimensional representation (Fig. 3) allows to easily understand explanation of the width importance. When the Gaussian width changes it results in a variation of selectivity for the neural network. If the width is too small (Fig. 3.1), the system influence area falls and the no recognition cases increase. In inverse proportion if it is too important (Fig. 3.3), false detections appear.

The selectivity of system depends on the position of each centroid, if the distance between two reference vectors is too important, the width need to be more important. Then distances between references vectors (DBCP) could change the selectivity of system from employment to other and the quality replying of the authentication could be affected. So a good width calculation needs to depend on centroids position. The more a centroid is similar to others, the more the width of this kernel needs to be influential.

Generally for RBF neural network, classical clustering algorithms are employed for the calculation, like K-mean [9], other error minimization [10] and heuristic methods [11][12]. Actually all these algorithms are based on error minimization between clusters. Our neural network is a one class classifier; it has to recognize only one person face and to be able to reject each other face, so only one cluster is available. Then a solution is needed to adapt the width to an optimal value.

A solution could be the creation of a second cluster [14] with unknown person’s images. It would permit to apply traditional algorithms, to calculate error minimization between clusters. From a hardware point of view, this solution is not optimal, because of important memory consumption and higher design complexity. Another solution is to keep only one cluster, and with a statistical study to develop a heuristic method based on the distance between the centroids (DBCP).

The Gaussian calculation is an exponential based function. It costs a lot of resources and it is a real problem to compute it in real-time function. The solution is using look up table (LUT) which contains a precalculated (learning step) Gaussian function. At the same time, we decide to limit hardware cost (number of random access memory blocs) by using only one Gaussian function for all neural network kernels. So we need to find a method to calculate a unique Gaussian width which maximise performances for the entire network.

C. Proposed method:

The aim of this study is to determine an automatic method to define an optimal Gaussian width which maximizes the difference between true and false detections. With classical methods, the classification is done by interpolating a clustering function which
separates different classes. The optimization of the clustering function is generally performed by an error minimization between classes. As it is previously explained, in our case only one class exists. The optimal width search is performed by replacing this lack of information by using a database of face images. Testing a set of stranger faces with the neural network allows evaluating the system response and situates the false detection rate. So it replaces the error minimization function which separates the reference cluster to the stranger one, by maximizing the difference between good and wrong face authentications. Testing a lot of stranger faces permits the system to automatically converge to the optimal width.

D. Method description:

Referring to the relation established between the DBCP and the system selectivity, the adaptation of the width value must be based on the DBCP. If the centroids are too distant from one to another, the width value needs to be increased and reciprocally. Some tests show that referencing the width calculation with the Mean Distance Between Centroids (MDBC) gives some good results and seems to be well adapted. In fact mean distance is not the optimal value for the width. The best result is given with approximately 20% less than the mean value (4). The method proposed is simple:

- First, calculate the MDBC (4).
- Secondly, calculate the width (5).

\[
MDBC = \left[ \frac{2}{Nb(Nb-1)} \sum_{i=1}^{n} \sum_{j=1}^{n} ||c_i - c_j|| \right] \quad (4)
\]

\(Nb\): number of ref. vectors
\(c\): reference vectors
\(n\): input data size

\[
Width = \frac{MDBC}{\text{Cst_Div}} \quad (5)
\]

Method evaluation is done with the Olivetti Research Laboratory database (ORL) [13]. This database is composed of 10 different images of each of 40 distinct subjects. The choice of this database of face images is representative of target application. Indeed, for some subjects, the images were taken at different times, varying the lighting, facial expressions (open/closed eyes, smiling/not smiling) and facial details (glasses/no glasses). All images were taken against a dark homogeneous background with the subjects in frontal position (with tolerance for some side movement).

For the test, one series (face images of one subject) is chosen and randomly 5 images of this person are selected as reference images. These 5 references images are going to be used for the system learning and the 5 other images of this person (different from training images) are available for the test plus the 390 faces of unknown persons. This test series allows calculating detection rates (good and false). Fives tests images for the good visage are sufficient because of the low dispersion of the result and for wrong visages 395 images are enough to converge to a good result. For different values of the Cst_Div, the calculation is performed with values taken from 3.6 to 0.05 with a decrement of 0.05. The test is done with all series of the ORL database (each 40 subject becomes the reference person). Figure 4 presents statistical results obtained after calculation: the authentication results given at the output layers of the neural network. It permits to have a better visualization of the system performance. To obtain the true and false detection rates, it only requires to compared the score to a fixed threshold.

\[
S(f, w) = \sum_{j=1}^{m} f_j w_j \quad (6)
\]

\(f\): hidden layer results
\(w\): synaptic weights

The graphic presented in the figure 4 shows the results. There are two results presented as function of Cst_Div variations: mean results for good faces images (first curve) and mean results for wrong faces images (second curve). The third curves represents the difference between the two results curves. It shows that an optimal value for Cst_Div maximizes the difference between good and false detections. The value which gives the best result is 1.2. Automatically computing Gaussian width value by referencing the width with the mean distance between centroids gives good results and optimizes the system performances. The Gaussian width calculation is automatic and adapted to images references variation.

E. Synaptic weights calculate simplification:

The output of the RBF neural network is computing with linear combination. The neural output consists of a weighted sum (6) as following:

\[
S(f, w) = \sum_{j=1}^{m} f_j w_j \quad (6)
\]
Synaptic weights $w_j$ are the weighting connections between hidden and output layers. It can be computed by searching for the linear combination which normalizes the output. The synaptic weight configurations are performed by testing an input vector and adapt weights to obtain the desired output result. The method used to achieve is a mean square minimization. To perform this operation, we need to resolve the following equation (7).

$$W = \left( Y^T Y \right)^{-1} Y^T T$$

$W$: Output weight matrix.
$Y$: Interdistance matrix $M$ filter by kernels.
$T$: Desired output matrix.

If the inter distance matrix is not square, the algorithm normally used for this inversion is based on Eigen vector calculation: the pseudo matrix inversion. But, the decision witch use only one cluster for the face classification allows to obtain a square matrix for inter distance matrix. So it is possible to use traditional inversion algorithm (pivot of Gauss Jordan method).

F. System output and recognition decision:

Classical RBF network bases their classification results on a threshold. The configuration of the threshold is very simple (binary result), but it presents a lack of flexibility. Regarding to the proposed method, varying the selectivity of the system is very easy (little variation on the width...). It could offer interesting possibility (table 1 & 2) to the system performance control (flexible to strict).

IV. RESULTS:

A. Authentication step:

This optimized version of RBF neural network was implemented on FPGA development board (Altera StratixII – EP2S60). The design development has been done in the respect of the AAA methodology, exploiting all parallel treatments possibilities.

Input vector is reading from an internal RAM block of FPGA. The different coefficients, reference vectors, Gaussian LUT, synaptic weights, are stack in static memory, they are preloaded from an EEPROM (SD-CARD). At the output, the input vector score is directly sent to the human machine interface (HMI). The following figure present the different parts of the network implantation.

![Diagram of neural network](image)

Fig. 7. Diagram of neural network

The neural network implantation (without the learning parts) uses approximately:
- 700 LUTs
- 500 registers
- 400 000 bits of memory
- one test duration time: 90µs

To resume, the implementation is very cost limited, it only use 5% of CLB and 30% internal memory resources.

B. Learning step:

As explained in this paper, learning step consists of compute all neural network parameters. Algorithms are writing in ANSI-C language for a soft processing on a processor IP. It does not increase the system complexity; reuse of the micro processor normally uses the communication between the system and the user. Even if it is possible to integrate it directly in VHDL IP because of their simplicity; it’s a good compromise: limited in development time.

All this algorithms are simple and don’t cost much time of process because of presented optimizations. Algorithms have been tested on RISC 32 bits processor Axis ETRAX 100LX (100 Mips). Treatments on this platform take less than 100ms.
With a 40 MHz processor (like Nios or Microblaze) we could estimate a time processing less than 1 second which is compatible with a standard utilization.

V. CONCLUSION:

All optimization and algorithms adaptation permits a good comprise between authentication performances, hardware resources and processing times. The proposed architecture with a soft processor allows a quick implementation of learning algorithms.

Now we still need to test the totality of this authentication system with a standard user, to verify system stability and autonomy.

VI. ACKNOWLEDGMENTS:

This work was supported by a region Rhône-Alpes research grants.

VIII. REFERENCES:


A HW/SW codesign platform for Algorithm-Architecture Adaptation

Christophe Lucarz, Marco Mattavelli
Ecole Polytechnique Fédérale
de Lausanne (EPFL)
CH 1015 Lausanne, Switzerland

Julien Dubois
Université de Bourgogne,
Laboratoire LE2I
21000 Dijon, France

Abstract—The increasing complexity of signal processing algorithms has lead to the need of developing the algorithms specifications using generic software implementations that become in practice the reference implementation. This fact can be particularly observed in the field of video and multimedia processing where reference software is the main normative reference. Adapting the algorithms specified by such software models into architectures composed by processors and dedicated HW elements becomes a very resource consuming task for the complexity of the models and for the large choice of possible partitioning options. This paper describes a new platform aiming at supporting the adaptation of algorithms specified by generic non optimized software specifications into mixed SW and HW implementations. The platform is supported by profiling capabilities specifically developed to study data transfers between the SW and the HW modules. Such profiling and optimization capabilities can be used to achieve different objectives in the algorithm architecture adaptation process such as optimization of memory architectures or low power designs by the minimization of data transfers.

I. INTRODUCTION

Algorithm-Architecture Adaptation is a very difficult task when dealing with very complex signal processing algorithms such as the one faced today in many application fields. It consists in finding the best architecture match for an algorithm that is usually written using a sequential program such as C or C++. The architecture can be composed of HW components as well as other heterogeneous components such as DSP processors and FPGAs. Mapping large software specifications onto a heterogeneous hardware platform is a complex and difficult process that cannot be achieved in one single shot. Usually the program is partitioned into smaller components in order to master the hardware design by dealing with components of manageable size. The problem arises when the hardware blocks need to be validated as single elements or when they are put together. The interfaces are sometime critical to be designed and appropriate test vectors need to be generated for conformance and performance testing. Furthermore, in the process of transforming the reference software into a real implementation (i.e. adapting the algorithm to architectures) the possibility of exploring different architectural solutions for specific modules and study the resulting data exchanges between components for defining optimal memory architectures is a very attractive approach.

The paper presents a platform that supports the profiling and testing of hardware modules as direct ”plug-ins” of the original reference software algorithm. The paper presents also the features of the profiling tool which enable the designer to measure the data transfers needed for the interface of the hardware component so that the designer can investigate different memory architectures optimizing data exchanges and the bandwidth between the different hardware modules.

The paper is organized as follows: section 2 presents a brief state of the art on integrated HW/SW platforms. Section 3 provides a general view of the platform introducing the innovative elements. Section 4 describes the details of the platform that enables HW/SW support. Section 5 presents the capabilities of the profiling tool and explain how it can be used to study and optimize data transfers satisfying different criteria. Section 6 proposes an example of integration of the Motion Estimation module of MPEG-4 part 2.

II. STATE OF THE ART

Testing the implementation in HW of sections of a reference software is not a trivial task. It requires a platform which enables the designer to seamless ”call” the hardware component directly from the (reference) software. This is possible only if the hardware component is closely linked to the (reference) software environment. Some HW/SW co-design platforms can be used to support the algorithm-architecture adaptation methodology, but all of them suffer from the fact that there exist no simple procedure capable to seamlessly plug hardware modules described in HDL to a pure software algorithm. Either the memory management is a burdensome task or the call of the hardware module is done by an embedded processor on the platform.

Environments which support both hardware and software implementations are generally based on a platform containing an embedded processor and some dedicated hardware logic like FPGA as described in the work of Andreas Koch [2]. The control program lies in the embedded processor. However, data on the host are available easily thanks to virtual serial ports. But the plugging of hardware modules inside the reference software running on the host remains the most difficult task.

The work of Martyn Edwards and Benjamin Fozard [3] is interesting in the way a FPGA-based algorithm can be activated from the host processor. This platform is based on the Celoxica RC1000-PP board and communicates with the
host by using the PCI bus. The control program is on the host processor, sends control information to the FPGA and transfers data in small shared memory which is part of the hardware platform. In this case, the designer must explicitly specify the data transfer between the host and the local memory. Many other works about coprocessors have been reported in literature. Some examples are given in [4] [5]. However, the problem of seamless plug-in of HDL modules is still existing, the specification of the data transfers that remains in charge of the designer might be a very burdensome task when dealing with complex data-dominated video or multimedia algorithms.

In some works on coprocessors, data transfers can be generated automatically by the host like for instance is found in [6]. However, data are copied in the local memory at a predefined location. Thus, the HDL module must be aware of the physical addresses of the data in the local memory. Again the management of the addresses can be a non-trivial and resource consuming task when dealing with complex algorithms.

The Virtual Socket concept implemented in a support platform has been presented in [10] [9] [7] and has been developed to support the mixed specification of MPEG-4 Part2 and Part 10 (AVC/H.264) specifications in terms of reference SW including the plug-in of HDL modules. The platform is constituted by a standard PC where the SW is executed and by a PCMCIA card that contains a FPGA and a local memory. Also for this platform the data transfers between the host memory and the local memory on the FPGA must be explicitly specified by the designer/programmer.

Specifying explicitly the data transfers would not constitute a serious burden when dealing with simple deterministic algorithms for which the data required by the HDL module are known exactly. Unfortunately for very complex design cases, where design trade-offs are much more convenient, and often are the only viable solutions, than worst case designs, data transfers cannot be explicitly specified in advance by the designer.

The work described in this paper is based on the Virtual Socket platform extended by adding the virtual memory capability to allow automatic data transfers from the host, running the SW part, to the local HW memory. The goal of such platform implementation is to provide a "direct map" of any SW portion to a corresponding HDL specification without the need of specifying any data transfer explicitly. In other words, to extend the concept of Virtual Socket for plugging HDL modules to SW partition with the concept of virtual memory. HDL modules and software algorithm share a unified virtual memory space. Having a shared memory - enforced by a cache-coherence protocol - between the CPU running the SW sections and the platform supporting HW avoids the need of specifying explicitly all the data transfers. The clear advantage of such solution is that the data transfer needed to feed the HDL module can be directly profiled so as to explore different memory architecture solutions. Another advantage of such direct map is that conformance with the original SW specification is guaranteed at any stage and the generation of test vectors is naturally provided by the way the HDL module is plugged to the SW section.

III. DESCRIPTION OF THE VIRTUAL SOCKET PLATFORM

The Virtual Socket platform is composed of a PC and a PCMCIA card that includes a FPGA and a local memory. The Virtual Socket handles the communications between the host (the PC environment) and the HDL modules (in the FPGA inside the PCMCIA).

![Fig. 1. The Virtual Socket platform overview](image)

Given that the HDL modules are implemented on the FPGA, they have only a physical access to the local memory (see figure 1). This was the case of the first implementation of the Virtual Socket platform, with the consequence that all the data transfers from the host to the local memory had to be specifically specified in advance by the designer himself. Such operation beside being error prone or be implemented transferring more data than necessary is not straightforward and may become difficult to be handled when the volume of data is comparable with the size of the (small) local memory. Therefore, an extension has been conceived and implemented so as to handle these data transfers automatically.

The Virtual Memory Extension (VME) is implemented by two components: the hardware extension to the Virtual Socket platform (Window Manager Unit) and a Virtual Manager Window (VMW) library on the host PC. The cache-coherence protocol is implemented in the Window Manager unit (WMU) using a TLB (Translation Lookaside Buffer) and is handled by the software support (VMW). The HDL module is designed simply generating virtual addresses relative to the user virtual memory space (on the host) to request data and execute the processing tasks. The processing of the data on the platform using the virtual memory feature proceed as follows. The algorithm starts the execution on the PC and associated host memory. The Virtual Socket environment allows the HDL module to have a seamless direct access to the host memory thanks to the Virtual Memory Extension and allows the HDL module to be started easily from the software algorithm thanks to the VMW Library. Figure 2 shows what are the interactions between the
unified virtual memory, the reference software algorithm, and the HDL module.

Given a reference software composed of several functions A, B, C, D and E. In order to test the HDL modules separately, the designer needs to execute some parts of the reference algorithm using the host processor and to test the hardware module. The Virtual Socket platform is the support for the hardware module for testing. To deal with mixed HW/SW algorithms, it is very convenient if the HDL and C functions have access to the same user memory space, named unified virtual memory on figure 2. This memory is part of the host hardware and contains the data to process. This host memory space is trivially available by the processor which executes the reference software, but it is much less evident for the Virtual Socket platform which is on the FPGA and is the support for the HDL modules.

For instance the designer wants to run function C on the reference software and functions D and E together using one HDL module which merges the two functions. The section of the reference code the designer intends to execute in hardware is replaced by the following piece of code which is called the HDL module calling procedure:

```c
int main(int argc,char *argv[]) {
    /* [. . .] Reference Software Algorithm stops here */
    /* Beginning of the HDL module calling procedure */
    /******** OPEN / CONFIGURING THE PLATFORM ********/
    Platform_Init(); // Virtual Socket
    VMW_Init();  // Virtual Memory Extension

    /******** PARAMETERS SETTINGS ********/
    Module_Param.nb_param = 4 ; // number of parameters
    Module_Param.Param[0] = A ; // parameter 1

    /******** HDL MODULE START ********/
    Start_module(1, &Module_Param) ;

    /******** CLOSING THE PLATFORM ********/
    VMW_Stop();   // Virtual Memory Extension
    Platform_Stop(); // Virtual Socket

    /* End of the HDL module calling procedure */
    /* [. . .] the Reference Software Algorithm continues*/
}
```

The HDL module calling procedure is composed of the following steps:

1) the designer must configure the platform by using the "Platform_Init( )" and "VMW_Init( )" functions from the Virtual Socket API and VMW API

2) The designer must set a given number of parameters needed for the configuration of the HDL module. This can be done thanks to the data structure "Module_Param". Sixteen parameters are available for each HDL module.

3) the HDL call function is started. This function writes the parameters in the register memory of the Virtual Socket platform (see figure 1). "Start_module( )" drives the Virtual Socket platform and the VME to activate the HDL module. The function "Start_module( )" is from the VMW API

4) when the entire job is finished, the platform is closed.

The VMW library manages all the data transfers between the main memory (unified virtual memory) and the local memory of the platform because as the HDL module is in a FPGA, it has access only this local memory. Thanks to the VME, the HDL module has access to the host memory without intervention of the designer. Data are sent to the HDL module and results are updated in the main memory automatically thanks to the software library support. When the HDL module finishes its work, the hardware call function is terminated by closing the platform and the reference software algorithm can be continued on the host PC.

![](image)

**Fig. 2.** interactions between the C function, the HDL module and the shared memory space

IV. DETAILS ON HW IMPLEMENTATION AND SW SUPPORT

The following section describes in more details how the Virtual Socket platform supporting the Virtual Memory Extension is implemented. The first part explains how virtual memory accesses are possible from the HDL modules. Then, the Virtual Memory Window library, i.e. the software support is described in details to show how virtual memory accesses are handled. The final part explains how HDL modules can be integrated in the platform using a well-defined protocol.
A. **HDL modules virtual memory accesses**

The HDL modules are implemented on the FPGA, so that they have access only to the local memory of the Virtual Socket platform. With the implementation of the Virtual Memory Extension, the HDL modules have a direct access to the software virtual memory space located on the host PC.

The left part of figure 1 shows how the connections between a HDL module, the Virtual Socket platform and the Virtual Memory Extension are implemented. The virtual addresses generated by the HDL modules are handled by the Virtual Memory Controller (VMC) and the Window Memory Unit (WMU). The WMU is a component from the work of Vuletić and al. [8]. The WMU translates virtual addresses into physical addresses. The VMC is in charge of intercepting precise signals at right time from the interface between the HDL module and the platform in order to send information to the WMU which executes the translation. Among the signals intercepted by the VMC, can be mentioned the address signal, the count signal (number of data requested by the HDL module) and the strobe signal. The virtual addresses refer to the unified virtual memory space and the physical addresses refer to the local memory on the card. A physical address is composed of an offset and a page number. The local memory (on the current PCMCIA card platform) is composed of 32 pages of 2 kB. The offset corresponds to the location of the data in the page. The software support library (on the host PC) fills the pages of the local memory with the requested data coming from the virtual memory. When the WMU receives an unknown virtual address, it raises an interrupt through the interrupt controller of the card. The interrupt is taken in charge by the software support (on the host PC) and the requested data are written from the host memory to the local memory.

From the designer point of view using the Virtual Memory Extension, the whole process of data transfers is completely transparent. The only issue the designer has to care of is to generate the virtual addresses accordingly to the data contained in the host memory space. The whole task of transferring data to local memory is done by the platform and its software support.

B. **The software support: the Virtual Memory Window library**

The Virtual Memory Window (VMW) library is built on the FPGA card driver (Wildcard II API), the Virtual Socket API developed by Yifeng Qiu and Wael Badawy bases on the works [9] [10] and the WIN32 API.

The Virtual Socket platform can be used with or without the Virtual Memory Extension. The designer is free to choose if the data transfers between the main memory on the host and the local memory on the card are done automatically (virtual mode) or manually (explicit mode).

C. **The integration of the HDL modules in the platform**

The HDL module is linked to the Virtual Socket platform thanks to a well-defined interface and a precise communication protocol.

![Diagram](image_url)

**Fig. 3. the communication protocol between a HDL module and the Virtual Socket Platform**

Figure 3 illustrates the protocol used by the HDL modules to communicate with the Virtual Socket platform. A HDL module can issue two types of requests: read or write data (in main or local memory, it depends on the operating mode: virtual or explicit). There is a great similarity between the read and write protocols. Figure 3 is an illustration of the communication protocol. The following section describes the steps of the read protocol. The write protocol works exactly with the same steps.

1) The HDL module asks to read data, it issues a read request for reading the memory.
2) The platform accepts the read request and in the case the data are available in the local memory, the platform generates an acknowledgement signal to the user HDL module. In the other case, the Virtual Memory Extension copies the requested data of the host memory into the local memory and then generates the acknowledgement.
3) Once the user HDL module receives the acknowledgement signal, it asks for reading some data directly
from the memory space. This request is performed by asserting a strobe signal together with setting up some other parameters signals (identification number of the HDL module used, the virtual address and how much data must be read).

4) The platform accepts those signals and reads data from the memory space. When the platform finishes each reading, it asserts a strobe signal and the data are ready to input of the user HDL module.

5) The user HDL module receives the data from the interface.

6) The user HDL module asserts a request to ask for releasing the reading operations when finished.

7) The platform generates an acknowledgment signal to release the reading operations.

In the Virtual mode, the read and write addresses contain the addresses of the data in the unified virtual memory space. It was like the HDL modules see the host memory.

V. PROFILING TOOLS: TESTING AND OPTIMIZING DATA TRANSFERS

State of the art signal processing algorithms are essentially data dominated systems and the data flow between the modules must be carefully optimized so that to reach low power design, necessary for any embedded systems implementation. Data transfers provide a relevant contribution to the overall power dissipations and need to be optimized to achieve low power designs. The profiling tools supported by the platform allow the designer to receive feedback information on the data exchanges with the HDL module.

Figure 4 shows the methodology to develop an optimized hardware function (HDL module) versus its data exchanges. The first step is constituted by the validation of the design. Using the Virtual Memory Extension, the equivalency of the C and HDL functions are verified. Virtual memory feature allows the designer to focus only on the HDL module conformance checking. The designer can forget everything about the memory management during this phase. The second phase consists in understanding and having a global overview of the data transfers exchanged between the platform and the HDL module. The way the data are accessed, the re-organization of data can be the object of accurate optimization. When the data exchanged by the HDL module are profiled, the designer enters the last phase in which data transfers are optimized between HDL module and cache memory.

VI. EXAMPLE OF SW-HW MODULE INTEGRATION: MPEG-4 MOTION ESTIMATION

A. Description of the HW module

The HW module performs the macroblock based motion estimation stage required by a frame based MPEG encoder [11]. A motion vector is obtained by selecting the best match between the reference macro-block and any position within a specified search window. The motion estimation algorithm is based on a reduced search strategy that reduces of up to two orders of magnitude the number of possible matchings within the search window, but requires flexibility in term of access latency to any position in the search window. In standard full-search implementations, an exhaustive search procedure is implemented calculating the matching function for all search window position. The approach is resource consuming, but the data access is perfectly regular. In a reduced search strategy configuration [11], [12], the block matching is only processed for a non-deterministic sub-set of block positions in the search window, therefore the access of any area in a search window is necessary. The set of positions is determined during runtime in function of the intermediate results and is calculated by a software processor (implemented by a MicroBlaze embedded in the Virtex IV FPGA). On the next generation of Virtual Socket platforms based on a WildCard IV board, the MicroBlaze could be replaced by a hardwired processor (PowerPC) embedded in the Virtex IV FPGA. In this example of module implementation, the search window width can be set up to 256 pixels, and the window height is fixed to 40 pixels. An external memory enables to store the full search window and the reference block. The internal memory permits to reduce the number of accesses to the external memory and any block in the internal memory can be accessible without any additional latency time. Any matching can be performed in less than 180 ns (with an overall clock frequency at 100 MHz) when accessing data in the internal memory. For instance, for a full-search configuration, the designed architecture processes at 14 frames per second at CIF video resolution format (i.e. 352x288 pixels frame size) with 41x25 pixels search range. As in other classical designs, the matching metric is based on the Mean Absolute Difference (MAD) evaluation. So as to obtain higher performances, the number of components that performs the MAD processing has been multiplied by a factor of 4 and a specific optimization of the data transfer architecture is needed to provide the required input-data bandwidth. The problem of optimizing such architecture consist on the complexity of the possible operating mode defined during runtime and on the requested module flexibility versus the variable image size and search window size.

![Fig. 5. an architecture of the MPEG Motion Estimation IP block](image)

B. HW module design and integration with the reference SW

The interfacing of the HW motion estimation module results very simple thanks to the Virtual Memory feature and to the wrapper module (User_IP_Block). The designer just needs to specify the following parameters: two pointers respectively on the beginning of the two consecutive images, the image size (height and width), the search window and the block
sizes (height and width). All of the parameters are stored into the IP_PARAM array. The address generation is implicitly calculated inside of the wrapper with the different parameters. The motion estimation process is executed for each reference block and the associated search window. The first step is the validation of the module functionality. For this phase, one input memory has been implemented. The conformance tests with the reference SW have permitted the validation of the motion estimation process.

The second step is the optimization of the memory architecture. The goal is to reduce as much as possible the size of the internal memory without affecting the performance of the module. The different levels of cache memory have indeed an important influence on the system’s performances and their behavior is not easily predictable for algorithms that change during runtime. The profiling information extracted during the execution of the motion estimation process can be extremely useful to improve the architecture performance, cost and power dissipated by reducing the number of accesses to an external memory and by minimizing the internal memory size. Another possible optimization is the suppression of the latency between the two MAD evaluations. To obtain optimized matching task in parallel with a data-transfer task, the input-data memory can be split into two memory banks. A portion of the search window and the associated block can be stored in each memory bank. The processing runs on one bank while input data can be transferred simultaneously into the other one. The virtual memory extension provides the data transfer information that can be used for this optimization task.

VII. CONCLUSION

This paper describes the implementation of a platform capable of supporting the designer in the different steps aiming at achieving the algorithm-architecture adaptation of a processing system described by a reference software. The platform provides a seamless environment for testing hardware modules which have been transformed from the reference software into HDL hardware modules. On one side conformance of the HDL modules with the reference SW is guaranteed at any stage of the design, on the other side the designer can focus on different aspects of the design. First design efforts can be focused on the module functionality without worrying about data transfers, then using the profiled data transfer on design of appropriate memory architectures or any other design optimization that matches the specific criteria of the design.

REFERENCES


A Modular SystemC RTOS Model for Embedded Services Exploration

Emmanuel Huck
Benoit Miramond
François Verdier

ENSÉA, University of Cergy-Pontoise, France
Email: [firstname.name]@ensea.fr

Abstract—System level modeling has been adopted for few years as a way to face the growing design complexity of embedded systems. In this systems the control of embedded applications is more and more often devoted to a Real-Time Operating System (RTOS). This RTOS can either be deployed in software or hardware, partially or completely, depending on the non-functional constraints of the global system. Consequently, these new design decisions concerning the implementation of the control must be taken early in the design flow for software and hardware parts.

In this paper we present the structure of our high level RTOS model, built to rapidly explore and evaluate different operating system services strategies (as scheduling policies) and implementation choices. Indeed, modeling an RTOS in SystemC implies to take into account its dynamical mechanisms and to explicitly control the scheduling of the simulated processes. We give some experimental simulation results of the RTOS model corresponding to the exact behaviour of an embedded RTOS at the first stages of the design.

I. INTRODUCTION

The design of new embedded systems on a chip (SoC) now faces two main issues. Firstly, algorithm complexity continuously increases, implying more and more computing power. Secondly, the increasing complexity and heterogeneity of targeted hardware architectures make the corresponding systems and their control harder to design. Moreover, the software integration phase follows the same growing complexity.

In order to face these difficulties and the design constraints such as cost, time-to-market, or tools (un)availability, engineers have to take decisions very early in the design flow. Among their choices concerning the target architecture, they have to find a valid partitioning between software and hardware that respects real-time and dynamical constraints of the attended application. Automatic partitioning methods and tools have been shown to work theoretically. But it is not used practically because there is actually no way to quickly verify non-functional constraints such as performance, power consumption and so far in a predictable and error-prone manner before the prototype is realized. This often leads to overdimensioning the global architecture, leading to over costs, surface, and power consumption.

Moreover, hardware and software designs are made independently and it is then difficult to come back on fundamental design decisions for design time reasons. Indeed, a full evaluation of a hardware design with classical synthesis tools at RT level is very time consuming. Moreover, as for the software part, one can develop the application on a desktop station, but specific problems appear only when integrating code on the real platform, where timing constraints may largely differ and debugging is much more difficult. Thus, in spite of emergent System Level Design Languages (SLDL), it lacks efficient co-design tools to develop and integrate both parts together.

The problem has been amplified with the growing presence of specific RTOS for embedded applications. RTOS are used in particular for dynamical applications, where computations (timing and behavior) mainly depend on the environment. In this context it is necessary to take into account the final system’s dynamics in order to simulate and evaluate its unpredictable behavior. Once again it lacks some tools for evaluating (even by simulation) the whole system, including the dedicated OS which controls the custom platform. It is the reason why specific methodologies are absolutely needed to explore OS implementations at different abstraction levels in order to gradually estimate the way they impact the system behavior, the application execution time or the required memory.

In this paper, we propose the use of high level executable and generic models, which permit to explore, validate and refine a real SoC application for both hardware, software and RTOS parts. We adopt SystemC [11] as the support language for joint modeling of both hardware and software, and define a framework allowing exploration and evaluation of various RTOS services managing such complex platforms.

This work falls under the OveRSoC project [2], which consists in developing a methodology for the design and evaluation of reconfigurable system-on-chip (RSoC). In this paper the management of the dynamic reconfiguration aspect is not addressed and constitutes one of the future developments of this work.

The structure of this paper is the following: in the next section, some related works on modeling OS are described. Then we present in Section III. the concepts of our RTOS model, discussing its implementation in SystemC. Its generivity and modularity are also presented. Some experimental results are provided in Section IV. Finally, we conclude and discuss future works.
II. STATE OF THE ART

A. Modeling languages and tools

The trend in embedded system design enhances an increasing need for high level design languages and verification tools for co-design.

A lot of synchronous languages exists to design real time applications (Esterel, Lustre, Signal or SyncCharts etc.). However they do not include the behavior of the hardware part. That explains why system level design languages have been developed, such as HandelC, SpecC, SystemVerilog, AADL, YML, ImpulseC or SystemC [11]. SLDLs allow to focus on the hardware design at a higher level than the transistor level simultaneously with the embedded software simulation. Finally, SystemC become now the standard language largely adopted by the community in the joint modeling field.

Based on these languages, a lot of EDA actors and academicians now begin to provide tools allowing high level simulation and codesign, like Coware, Synopsis, SPACE codesign [4], Scicos/SyLab or Ptolemy [3], that unfortunately exclude RTOS part.

B. Methods

To face the increasing complexity of SoC, new design methods appear to ease the design phase and avoid mistakes, detected only during integration, when hardware and software are developed separately.

Le Moigne et al. [10] propose the MCSE method for codesign. It allows to refine a design model from a high level view with CPUs to a view with communications details. The MARTES project provides an other methodology inspired from MARTE [14], SysML and UML4SoC. Recently, a trade-off has been made with SLDL languages between simulation speed and accuracy. This is possible thanks to a high level modeling called Transactional Level Modeling (TLM). The SystemC modeling language comes with different levels of abstraction and we try to follow its refinement methodology called TLM 2.0 [11].

C. Existing Real-time system models

Many actors tried to produce frameworks for software/hardware codesign and modeling.

Yu, Gerstlauer and Gajski [15] developed a new language in this goal : SpecC. They propose an RTOS model implemented in SpecC, with a refinement method which allows an easy and automatic implementation of its services.

SoCOS [5] is an other proposal of SLDL framework to model SoC including the RTOS. It was also developed when SystemC was not as evolved as nowadays. This model can handle multiple clocks and has the ability to create processes dynamically. It proposes three computational models : asynchronous, reactive (interruption), and synchronous. It allows an automatic refinement after adding timing data to the code.

In [7], authors describe a SystemC configurable and modular model that can reproduce the behaviour of many OSes like Linux or DSP/BiOS. Their tasks model allows to deal with interruptions if tasks are decomposed in elementary time slots inferior to the nearest interruption event. This works of course only if the interruption arrival time is predictable. They also provide a generic model of driver to wrap any real driver, in order to include the environment (and use input/output data) to the simulation.

Hastono [6] worked on extending SystemC with aspects like dynamic process creation, process control, preemption, process prioritizing. The application is first modelled by a task graph then translated into SystemC TLM by refining communications and finally tested with different ordering policies. This model also takes into account the Worst-Case Execution Time (WCET) by modeling it as a Gumbel stochastic density.

Posadas et al.[13] develop a SystemC 2.0 library to model an RTOS conform with the POSIX Application Programming Interfaces (API), called PERFidy. This tool allows to automatically evaluate execution time of any segment of code between two system calls, which is necessary for a timed SystemC simulation.

Madsen [9] presents a SystemC-based framework modeling MPSoC controlled by multiple RTOS. The architecture corresponds to an OS per processor composed of a scheduler and a resource allocator. All RTOS are connected to a synchronizer module, based on a single clock, which allows tasks communications and handles intra and inter processors dependencies. The scheduler is implemented with the SytemC Master-Slave Library in order to attempt multiple messages at a time and is able to run a different kind of ordering policy. A resource allocator handles resources contention and can change task priority following the Priority Inheritance Protocol to avoid dead-locks.

The approach proposed by Hessel [8] is focused on the design and refinement methodology. Author describes the application in SystemC TLM and refines it to the possible RTL or IP implementation based on libraries. The first step consists in partitioning the application in Sw/Hw pieces. Then software modules are clustered and abstract channels are added to establish communication between clusters. These clusters are assigned to processors. At this time, using the libraries with a manual treatment, first simulation could be done and different scheduling policies could be examined for optimisation. Some power consumption estimations are provided also to help this selection.

Houzet [12] also proposes a design flow intended to automatically refine and generate code for both embedded RTOS and application, in order to avoid manual hardwork and possible mistakes. It is based on the VCI interface, which allows to wrap IPs for the communication between hardware accelerator, and the MPI-2 (Message Passing Interface) library. The customised RTOS is based on RTEMS and the model uses the SPIRIT format to describe the architecture.

We can see it exists a lot of OS models. But it takes some real time behavior modeling completed to a specific model structure (generic and modular) allowing fast exploration by customizing and refining the model.
III. OUR RTOS MODEL

This section presents our framework proposal allowing a rapid embedded system evaluation. We explain how to implement a simple RTOS model with SystemC, able to preempt tasks. We then present how to decompose this model into multiple modules in order to ease the exploration of different RTOS architectures.

A. OS model definitions

An operating system acts as a system software layer which provides an interface between application programs and the hardware support (including peripherals). Applications are divided into processes or tasks. We do not consider in this work POSIX threads particularity. In hard real-time systems, each task is annotated with time constraints (like deadline or periodicity) depending on the desired application behaviour.

An operating system can also be seen as a services provider that gives processes access to hardware resources and controls their effective and correct sharing out. In our work, we mainly focus on the following basic services:

- Scheduling : The scheduler is responsible for ordering the task execution in the execution units. The scheduling policy it employs will have a deep impact on the whole system’s performance. There are many strategies to order application’s tasks. These strategies may be based or not on time and process priority, and may be imposed before the execution (static scheduling, decided at compile time) or dynamically calculated, depending on the application evolutions (react to new inputs or interrupts). The last strategy being the most flexible but relies on complex scheduling algorithms and could produce unpredictable behavior.

- Preemption : As there is only one process running on a CPU at a given time, a RTOS must be able to stop the running task at any time and start an Interrupt Request (IRQ) treatment. This ability is called preemption and involves a context switch : load/restore CPU registers and memory contexts, from/to the TCB (Task Control Blocks, which contain all task parameters needed by the OS).

- Process synchronisation : To allow tasks to synchronise together, or to protect from illegal accesses on shared resources (memory or input port), we also consider synchronisation mechanisms such as a basic semaphore service.

The scheduling policy is one of the major factor of performance, but the context switch overhead, the application partition into processes and the memory and buses performances must also be taken into account to evaluate the system behaviour. We then argue that a very early modeling and simulation of an application simultaneously with its custom RTOS could help the evaluation of the future system’s performance. This will ease to define, among other things, which scheduling policy best fits some given real time requirements and verify early partitioning choices.

In order to allow exploration of RTOS design choices, we propose a generic RTOS model with the followings constraints:

- Genericity : Our model is generic in the sense that its structure and its behavior allows modelling of most implementation strategies. In this sense, genericity eases services exploration. This as been made possible thanks to a modular and extensible design effort (these properties are developped in section III-D).

- Transparent from user point of view This is posisible by a clear separation between the service interfaces (use standard interface like POSIX) and their implementation (heterogeneity).

- Refinement of the model : The model allows multiple levels of implementation of the same service thanks to common API.

Our approach consists in modeling only the behaviour of the software part (OS plus application) with total abstraction of the underlying platform, that is to say a top-down approach, from the OS point of view.

B. SystemC RTOS model

The model is based on SystemC, itself based on a C++ class library. SystemC is devoted to hardware simulation and design. It allows to model the concurrent execution of modules’s (called sc_modules) processes, which execute C++/SystemC code.

The execution time modeled in the SystemC kernel is devoted to concurrent (hardware) modules but not to the time-sharing principle as in classical micro-processors. Indeed, the SystemC kernel carries out a functional process code of a module in a null time and then runs out the simulated time reference according to the value explicitly given by the directive \texttt{wait()} into the code. This implies knowing in advance, or to be able to estimate, the execution time and add the directive \texttt{wait()} after each portion of code. If not, the whole application runs in zero simulation time. SystemC also provides mechanisms to synchronize modules execution with events (\texttt{sc_event}). Each module can communicate outside through ports (\texttt{sc_port}) connected to signals, as simple as wires (\texttt{sc_signal}) or more complex as for buses (\texttt{sc_channel}). These are in fact modules that provide a particular API and modules can call the corresponding method through their ports, as shown on Figure 1.

![Fig. 1. SystemC simulating block in parallel](image-url)
As for other design languages, it is possible to have modules composed of sub-modules, allowing different level of abstraction, the lowest being the transistor level. We widely use the abstraction and hierarchical modeling mechanisms offered by this modeling language.

Because the SystemC kernel executes C or C++ code, it allows us to also simulate software parts of a design. To adapt a software application in our SystemC model, each task function is executed in a distinct sc_module’s process. We choose to implement the RTOS as a SystemC hierarchical channel, namely a SC_channel, which can then provides system services to software tasks through an interface offering the standard system API. In this way, system calls in task code are no more calls to standard syscall functions but calls to the corresponding RTOS object methods (Figure 2). Porting an application onto our model only needs to adapt the syntax of the system call towards the virtual (modeled) RTOS API. This code transformation can easily be done thanks to the use of macroinstructions masking all OS calls.

Our OS model should be able to dynamically create a task, but the SystemC kernel does not allow dynamic creation of modules. We use the SystemC sc_spawn() function which allows to create a process after the beginning of the simulation. As the SystemC kernel does not allow dynamic creation of modules, we thus model task by pure C or C++ functions. Each task call can an OS service by calling the corresponding model method. Then dynamically created task are in fact SystemC processes (sc_thread) belonging to the RTOS module itself, as represented in Figure 3.

In order to perform a timed simulation of the software application along with the operating system, this model implies to annotate each code portion between two system calls with wait() statements. The execution time depends actually on the given target architecture (memory hierarchy, CPU frequency, number and kind of instruction etc.), but we can easily be satisfied with rapid estimates for our modeling level.

This model allows to take a real application and simulates it without changing any instruction except adding the approximated simulation execution times.

### C. Time management and scheduling

As discussed previously, it is necessary to introduce in the RTOS model a mechanism making it possible to circumvent artificially the execution engine of SystemC so that only one task occupies the execution unit at a given time. To cope with this, we encapsulate the wait() statements by a (pseudo) system call towards our model (the os_wait() call) which will manage this suitable behaviour. Each task is associated with an internal data structure describing its characteristics and OS parameters (as in a real OS). A specific internal SystemC event is included in each task’s TCB. This event is used by the scheduling algorithm for synchronising tasks executions. Each task in our model is a SystemC thread having this event in its sensitivity list. In this manner, we can explicitly force nonparallelism, by making all tasks wait for their own event.

For modeling task preemption, OS listens for a periodic tick timer, or an interrupt event, which starts the OS scheduling algorithm. This scheduler can thus take the decision to awake the adequate process: it triggers the corresponding task event and then, the task starts. This task runs its functional code and calls os_wait() to simulate its execution time. The os_wait() waits for both the time given as argument and for a specific event called stop. If the os_wait() function loop is interrupted, then it calculates the remaining time and wait again, but now indefinitely for the task event. When the scheduler decide to await a task, it trigger the corresponding task event, as for launching any task; and then the task can continue its os_wait() execution, for the remaining duration. Thus, we are able to accurately model the sequential execution of tasks and possible preemption at any time as shown in Figure 4.

An obvious limitation of task execution accuracy in our model consists in the fact that even if preemption could occur at any time, task blocks are already executed (in zero time) before the os_wait(). As long as a timed functional validation is concerned, this limitation has no consequences on the simulation results: the global application behaviour can be verified and potential deadlocks identified. If a more precise simulation is expected, it is possible to reduce the task block granularity until an instruction accurate model of the processor (Instruction Set Simulator), but is out of the scope of our work.

---

Fig. 2. Task running on a SystemC RTOS

Fig. 3. RTOS model able to create tasks dynamically
Fig. 4. A task runs its functional code in 0 time, and then performs an OS_wait() with a specific duration (100ns in this example). The preemption applies in fact on the os_wait() method, allowing an internal computation of the preemption duration (caused here by an interruption -ISR-).

D. Modularity and genericity

In order to build a generic model of a RTOS, the main services have been grouped into several categories. Locality of shared internal data has been the most important criterion for defining these categories. The actual services categories in the model presented in this paper are the following:

1) Process management and CPU time share (scheduling) : task create, suspend, ...
2) Synchronous and asynchronous communications: by ports, shared memory or FIFO
3) Synchronization : semaphore/Mutex, conditional variables and process joining
4) Memory management : allocate / free
5) Storage management : file read / write
6) Interruptions management
7) Distributed OS management
8) Timing : explicit simulation time wait statement

The last group (timing) has been added only for simulation and execution time modeling purpose and do not represents any real OS service.

In order to offer these services to the application’s tasks and render their implementation relatively independent one from the other, we choose to implement each group of service in the form of a hierarchical SystemC sub-channel (see Figure 5). Sub-channel inherits from the sc_interface class and thus offers the services defined in its interface. To offer the whole API, the RTOS exports each sub-channel interface, by using the sc_export mechanism.

The interface has been defined in a way that it is independent from the particular implementation of the services. This clear separation between the services interface (i.e. the definition of the system calls) and their implementations is an important feature of our model. It guarantees and eases the exploration of the services implementations by replacing a block by another one assuming it implements the same interface.

The main drawback of decomposing the RTOS functions into separate blocks consists in insuring correct interoperability between separate SystemC objects. For example, the semaphore manager module obviously needs to communicate with the scheduling module in order to inform which task has to be put in sleeping state or has to be awakened. To solve this, we have implemented a second distinct interface in each service block. It consists in an internal interface through which other service groups can ask for services. The same mechanism (sc_interface inheritance) is used for the internal services. The re-schedule() method is a good example of an internal service offered by the scheduling block to other services. Each service block is thus added with inter-module communication ports and inter-module communications are realised by point-to-point connections (see Figure 6). We will show in the next section that this modular structure of the RTOS model really eases both customisation and exploration of several implementations of basic or dedicated services. Several implementations of the same group of services can be explored and validated by replacing a service block by another compliant with its two (internal and external) interfaces.

One of the major feature that could be explored in this way is the hardware/software partitioning of an embedded RTOS.

Fig. 5. Modular OS model: The RTOS is modeled by a hierarchical sc_channel containing several sub-channels which represent and model a service group. The main channel provides all the services to the application through a single API (the interface of the OS channel).

Fig. 6. Internal interface of inter module services
IV. EXPERIMENTS

A. The RTOS model implementation

We have implemented a model of RTOS inspired from μC-OS-II[1]. As described in the previous section, services of the OS are encapsulated into SystemC modules. As each module is associated to a part of the system API, system calls toward modules follow those of μC-OS-II. In μC-OS-II, the scheduler is only based on user defined tasks priorities without round-robin since each task is associated a single priority. Inversely, our model can accept a large choice of scheduling strategies. The model obviously contains a real-time clock that allows the modeling and the simulation of fundamental RTOS behaviours such as periodic tasks, sleeping tasks and OS tick sensitivity. The tick period of the clock has been set in the following experiments to 500 µs (Figure 8). The OS tick periodically awakes the scheduler to simulate time quantums. We applied our method to a small synthetic application (see Figure 7) to demonstrate the correct behaviour of this RTOS model.

This artificial application is composed of three tasks and an interruption handler. The main task only executes once with the highest priority, initialises global resources like a mutex and a "Rendez-Vous", launches task 2 and task 3 and does a little job after. The mutex (semaphore initialized to 1) protects a shared resource (like access to a video picture on memory) between task 2 and task 3. The rendez-vous (semaphore initialized to 0) semaphore is unlocked by the interruption handler when an interruption occurs. As shown in Figure 7, the OS model contains an interrupt handler able to be executed each time the corresponding interrupt is raised (virtually triggered by the simulation engine). Interrupts are modeled as particular processes with the highest task priority, considering the priority is not really relevant for interruptions. Task 2 has a period of 1 ms and has a lower priority than task 3. It periodically runs the same job, which consists in taking the mutex, doing some computation on the shared data, releasing the mutex, doing computation, and then sleeping until next period (figure 7). The third task runs each time an interrupt occurs except for the initialization phase where some initial treatments are done (initialising variables). It thus always waits for the rendez-vous semaphore (blocking call) in a loop, then it takes the mutex to perform a job with the shared resource, releases the mutex, and waits again for any new interrupt (via the rendez-vous semaphore).

B. RTOS behaviour simulation

We illustrate with Figure 8, the simulated RTOS activity. This Gantt diagram shows four fundamental RTOS mechanisms: tasks preemption, OS tick sensitivity, interruptions and tasks synchronization.

The scheduling obviously begins with the task PID (Process IDentifier) 0, which is the RTOS initialisation (and after, the idle task). As we can see in area A, when the OS finishes its initialization, it launches the scheduler (in A0) which elects the main task (PID 1 because it is the first task being created). We can see that the scheduler is launched each time a task is created (A1) or when a task ends (A2) or is blocked(A3). In A4, the OS Tick appears and relaunches the scheduler, and continues to run the only one ready task 2. In area B, an interrupt occurs, which launches the scheduler and runs the corresponding IRQ handler (PID -1). This one releases the RDV semaphore and stops. The scheduler then elects task 3 now ready, and not the interrupted task 2, because task 3 have a higher priority. Then when task 3 ends, task 2 can continue its job. In area C, we only focus her on the fact that the timer awakes correctly task 2 each millisecond as it is a periodic task. The area C will be more described in the following.

C. RTOS services exploration

Thanks to the modular nature of our OS kernel, by replacing a module by another, we are able to explore different kind of implementation strategies for a given service. Figure 9 focuses on two different strategies for managing semaphores: awaking (Figure 9.B) or not (9.A) the scheduler upon semaphore release. This simple change can lead to different behaviours as now explained on the same application, where task 2 already
 owns the mutex when an interrupt occurs. The precedent

<table>
<thead>
<tr>
<th>Time (ms)</th>
<th>PID</th>
<th>Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 2</td>
<td>-1 3 2 0</td>
</tr>
</tbody>
</table>

Fig. 9. New semaphore implementation: when released it restarts the scheduler

D. Current work and simulation costs

In practice, all these mechanisms are currently used on a more realistic application in the field of image processing for robotic vision. The application is used to learn object views or landscapes and extracts local visual features from the neighborhood. In this context we profiled this application on a hardware architecture composed of multiple Nios-II processors (MPSoC) prototyped onto an Altera Cyclone-II FPGA circuit. The timing data measured from this implementation are used to back-annotate the high-level model in order to simulate and explore different implementation strategies. The application runs without any change either on our abstract RTOS model or on μC-OS-II on a real embedded platform (except the necessary timing annotations).

<table>
<thead>
<tr>
<th>PID</th>
<th>Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 2</td>
<td>-1 3 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time (ms)</th>
<th>PID</th>
<th>Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 2</td>
<td>-1 3 2 0</td>
</tr>
</tbody>
</table>

Fig.9. New semaphore implementation: when released it restarts the scheduler

We used a Intel Dualcore at 1.66GHz under Linux to perform simulation tests, on a sequence of 1000 images. The simulation does only use one core because SystemC does not work on multiprocessor. The application is fragmented into 31 tasks. As shown on Table I, we can conclude that our RTOS model does not dramatically impact the simulation time (only 9.8% more than the application running with the SystemC kernel primitives without our RTOS model) and is still faster than a real implementation execution (about 32 seconds to treat one image on the NIOS platform with μC-OS II). It allows to rapidly explore the application partitioning and the RTOS behaviour, without needing a real expensive platform.

V. Conclusion

We have presented in this paper the basic concepts of an abstract RTOS model in SystemC for system level design. The high level description of the model allows the designer to early simulate the dynamic behaviour of complex real-time applications. The model can be parameterized in order to represent the exact timing of the OS services on the future architecture. The designer can thus evaluate the impact of different service implementations on the real-time performances of the system. Moreover this model is built in an object oriented manner and each service of the OS can be added or deleted according to the application/architecture needs, updating automatically the operating system API. This modular structure will also permit to explore the deployment of the OS services on multiple processing elements (processors and hardware reconfigurable units). In this direction, we are currently working on inter-OS communication mechanisms based on TLM 2.0 allowing the exploration of distributed OS implementations.

REFERENCES


A complete methodology using co-simulation and co-verification to embed a telecommunication application in a System on Chip based on a NoC structure

Simone Bacles-Min, Christine Hennebert
CEA-LETI, MINATEC, 17 rue des Martyrs, 38054 Grenoble cedex 9, FRANCE
{simone.bacles-min, christine.hennebert}@cea.fr

Abstract— This paper presents a complete methodology to map and test all the development stages of an OFDM communication chain in an emergent System on Chip (SoC) based on a Network on Chip (NoC) structure and protocol. In order to efficiently exploit the various tools, from high level abstraction to instrumentation on the given hardware SoC, a complete methodology is deployed and its purpose is argued. As example, the full integration of a Hybrid ARQ (HARQ) communication chain into a SoC (composed of a full compatible On-Chip (Asic) and Off-Chip (FPGA) components) is considered to point out the relevance of the method.

I. INTRODUCTION

Nowadays, complex SoC device are implemented in Micro/Nano technologies. They incorporate several components such as processor cores, memory, dedicated IP, communication bus, interfaces and configurable functionalities. They integrate a low consumption powerful computer within an area of a few square millimeters. In parallel, the innovative applications need a high algorithmic complexity and manage high rate and complex data flow.

As a generally accepted fact, the design and the integration of an application for such a SoC device is complex and requires a well-structured development methodology with several verification stages. As the hardware platform becomes smaller and smaller, its virtual model needs a higher and higher level of abstraction. The point is to embed complete reliable applications in a SoC in a scaled down time.

The purpose of this article is to outline a complete methodology to implant a telecom application in a given SoC, an ASIC named Faust developed by CEA-LETI and based on a NoC structure and protocol. The paper takes into account the whole steps of the integration, from the specifications of the application to the instrumentation on the platform. At each step, the methodological choices are discussed.

As a shining example, the HARQ protocol is studied and embedded in the Faust ASIC. Given that over wireless channel, the HARQ protocol increases the downlink data rate, the 3GPP standards retained it to equip the 3.5G mobile generation [1]. Faust chip should be an interesting device for such an implementation thanks to its NoC structure highly reconfigurable and because it can handle multi telecommunication applications based on OFDM techniques. It is integrated into an open and flexible SoC platform including Xilinx FPGA and allowing the deployment of our OFDM systems.

The proposed methodology is summarised on Figure 1. In the first step, the application is studied using a high level abstraction model and is qualified in terms of data flow, perfor-
formance and complexity. The specifications are established, especially the hardware/software partition and the data flow progress. The second step may consist on the dedicated IP development in VHDL language, their integration in the network interface and their instantiation in a powerful co-simulation environment. Each new functional IP is individually tested before the final FPGA synthesis and the application integration. The next step is the pre-integration in a complete co-simulation environment, describing in C or VHDL languages, the whole platform (except the CPU replaced by a parser). The co-simulation uses a TLM platform and allows testing rapidly the IPs configuration, the DMA engine computing and the data flow progress. The co-verification employs the true network and CPU, indicates the real timings, the global latency, and validates the interaction between the MAC layer (CPU) and the PHY layer (IP units). In the last step, the experimentation on the real SoC platform demonstrates the cogency of the method.

To understand the following of the paper, the SoC platform, its architecture and its protocol are depicted in Chapter 2. Our methodology flow is described in Chapter 3. In Chapter 4, as example, the HARQ protocol is integrated to validate our methodology and open on future improvements before conclusion.

II. HARDWARE SOC PLATFORM DESCRIPTION

The SoC device depicted in this chapter has been done by CEA-LETI. It allows lots of realistic investigations on the NoC architecture. Our paper focuses on the methodology to develop an application running in such an ASIC device.

A. NoC architecture

A NoC is an innovative communication structure based on a fully distributed system with a specific layered protocol [8]. It is composed of 2D-mesh nodes which form the base structure of communication. This NoC architecture can handle complex data flow progress. The IPs connected to the nodes can be heterogeneous (Figure 2). During a communication scheme, the configuration of each IP can be modified.

1) NoC protocol

The basic piece of information transferred on the NoC is called a flit as flow control unit. It corresponds to a 32-bits data word [9]. A packet is composed of several consecutive flits taken the same way, which minimizes the switching task of the nodes. Data and configuration packets, interruption or dump requests can progress in the NoC. The data are exchanged thanks to a credit mechanism. In the communication chain, each block sends data credits to the previous one. When a block receives credits, it answers by sending the available computed data. As a result, the data flow progress from one block to the next one by a local mechanism, without the CPU intervention.

2) IP unit architecture

Each IP is connected to the network using a specific network interface (NI), which interfaces transparently the functional inputs and outputs of the unit core with the NoC [7]. The NI manages the data flow synchronization, the reconfiguration of the units, the interruption emission, the debug and the test of the unit on the NoC. Inputs and outputs of the hardware IPs are normalized in order to dialog with the NI according to the NoC protocol.

B. Hardware platform description

1) Faust device

FAUST or Flexible Architecture of Unified System for Telecom, a first prototype of ASIC dedicated to 4G telecom applications, has been designed in order to validate the NoC structure [8]. This chip integrates 23 IP units connected to a 20 asynchronous nodes network for a total complexity of 8 millions gates (Figure 3). It is implemented in a 0.13µm CMOS technology from STMicroelectronics. This first integration demonstrates our ability to handle various kinds of IP units on the same NoC. In particular, this chip embeds an ARM core associated to a standard AHB protocol.
2) **SoC platform**

A board (Figure 5) embedding two FAUST chips and two FPGA Xilinx Virtex 4 has been designed. The FPGA are fully reconfigurable. The association of a FPGA and of a FAUST chip forms one channel, the Soc platform handles two channels. Therefore, a MIMO transmitter or a MIMO receiver or a SISO communication system (transmitter + receiver) can be either implemented in the same platform. A host PC is linked to the Ethernet module of the FPGA via a socket mechanism. It is used to load all the code and configuration flits for the application.

![Figure 4: SoC platform](image)

An ARM processor is embedded in the ASIC. The ARM code is developed with the ARM tools and the elf binary words are organized to build the flits for the NoC protocol. A full VHDL simulation of the design allows the validation of the whole application in the whole system with the ARM CPU.

This realistic simulation is very useful to handle the real timings of the hardware and to validate the interactions between the software and the physical layer.

![Figure 5: Instrumentation board](image)

---

III. **PROPOSED METHODOLOGY TO EMBED AN APPLICATION IN THIS SOC PLATFORM**

Considering the FAUST device and its architecture, our goal is to propose a methodology and numerous tools in order to efficiently implant a telecommunication application in this SoC. Next, the methodology will be tested with an example.

A. **Specification aims**

A fine-spun study of the application should lead to define a trade-off between the performance in term of FER and complexity. In particular, the algorithm must be described in fixed point in order to be implemented in hardware. A special attention is held to the partition between hardware and software. Every rules must be applied to design an efficient system:

- to guarantee the data flow fluidity in the NoC,
- to restrict the number of interruption to the CPU,
- to limit the frequency of the IP reconfiguration,
- to favour the local exchanges between the IPs by the credit mechanism,
- to integrate an appropriate intelligence inside each IP in order to give autonomy to the IPs.

B. **Unitary IP test**

At this step, the functional IPs are specified and developed in VHDL language. In order to support the credit mechanism, the IP core must be able to compute as many data as available in the input fifo, and wait in standby when the fifo is empty and the data flow not complete. So, the core is integrated in the network interface that manages the control signals, the configuration and assures the coherence with the NoC protocol. The components of the network interface are chosen in a library.

Because the system is big, complex and new, we have chosen to develop a dedicated co-simulation framework for the unitary IP test. The environment uses only one FPGA. The ASIC NoC structure and protocol is extended into the FPGA and forms a 6 synchronous nodes network. Different resources are connected to the nodes: Two internal memories and an external memory controller, an interface with the RF stage and an Ethernet module to dialog with a host PC (Figure 6). A scenario interpreter is also linked to a node. Many empty locations (in grey colour) allow the instantiation of IP blocks (Figure 6).

In order to accelerate the simulation, all the network components are described in C or C++ language. Only the core of the components is described in VHDL to preserve the real precision after the variable quantification and to allow the debug of the core. The Transaction Level Modelling (TLM) platform is used to ensure the interface and the coherence between HDL and C description languages [9]. TLM to HDL and HDL to TLM translators are employed for co-simulation and SystemC configuration classes have been developed to parameterize the IP core.
A host PC is connected to the FPGA via the Ethernet link. The transfer between the SoC and the host PC is achieved with a socket mechanism. Using these connections, all the IP units are accessible from the host PC. The host PC can also receive interruption, dump or data from any units of the SoC. A generic scenario has been written to test any IP unit whatever its location on the NoC and using the credit mechanism (Figure 7). The test deployment is done in order to validate the embedded characteristics of the tested IP unit: the unit does not dialog directly with the host PC.

To emulate the CPU, a scenario interpreter has been developed. Its purpose is to interpret high level commands that describe finely the scenario unrolling. The IP configuration, the DMA engine programs and the data flow description are launched using these commands. Finally, the interpreter generates all the configuration flits for the low level.

When the IP block generates an interruption, a co-verification can be lead with the true CPU model before the instrumentation on the FPGA. Once all the IP developed and tested, the final FPGA can be synthesized and the IP code is add in a library.

C. Pre-integration

Another design framework is developed to co-simulate the whole SoC represented in Figure 4. A scenario interpreter is added. This framework supports PHY layer application development.

This tool is very useful to elaborate the scenario and the data flow progress from block to block in the device. It is mandatory to generate the configuration flits of all the blocks for the application. But the succession of tasks is sequential. The interactions with the ARM CPU are not represented. So, the scheduling of the whole application will be access during the integration stage. The parts of scenario between two interruptions to the CPU are built at this step. Next, the integration will consist to start the ARM CPU and to order these parts of scenario according to the interruption content.

Future works are contemplated to increase the performance of the co-simulation framework. A graphical interface should be used to easily build an application in consequence of the 2D-mesh network organization of the SoC architecture with LEGO cores linked on the nodes. The flits for data flow description and the IP unit configuration should be automatically generated. In the context of the spirit consortium, the development of such an interface is in progress. It is based on the XML language, designed to store, carry and exchange data between multiple vendors systems and support various languages.

D. Software development / Integration

The software is developed according to the specifications for the embedded processor core in assembler and C/C++ languages. It uses high level tools to rapidly develop and debug the code. Elf binary files, directly used on the platform, are generated. Nevertheless, when the interactions between the processor and the hardware are numerous, the test becomes very long. A test bench environment should be deployed to emulate the hardware behaviour and to fully test the application.

For most applications, the hardware/software co-verification may be essential. It validates all the timings of the embedded system and the interactions between the hardware (PHY layer) and the software (MAC layer). For future works, it should be interesting to integrate a CPU emulator in an abstract high level environment and in the co-simulation framework.

The co-verification is done by a complete VHDL/Verilog description of the SoC. The co-verification shows all the details of the SoC, validates the hardware/software interactions and presents the real tasks succession and the true timings. An

Figure 6 : Co-Simulation design for unitary tests

Figure 7 : Scenario for unitary tests
IP of the processor core is used to handle the real behaviour of the CPU. The co-verification is essential when the data rate is near the SoC limitations. The co-verification must be reserved to final stage of the application development, especially when:

- the data flow progress in the platform is not coherent with the co-simulation observances or the data are erroneous;
- the interactions between the hardware (PHY layer) and the software (MAC layer) must be validated before the instrumentation;
- several CPUs are used.

E. Instrumentation

The instrumentation implements one or two host PC linked to the hardware platform via an Ethernet connexion. Only the configuration files, sent by sockets, allows the use of the functional blocks in Faust device. These files are carefully generated thanks to the pre-integration stage. Some specific blocks can be designed and added in the FPGAs and accessed with ChipScope. The scheduling of the whole application is considered thanks to the integration stage.

IV. Example: Integration of the HARQ Protocol in a SoC with a NoC Structure

In order to improve this integration method and the relevance of this architecture, the HARQ algorithm is studied and its adequacy with the SoC Faust is depicted. The HARQ process is a packet acknowledge technique which combines both Automatic Repeat reQuest and Forward Error Correction mechanisms [2]. It has been chosen by the 3GPP consortium to increase the quality of the downlink data flow for the 3.5G mobile generation. For these reasons, this study case is well-adapted to test the exposed methodology and to fit with the proposed SoC architecture:

- It leads to design new IPs dedicated to the HARQ mechanism and to integrate these IPs in the NoC according to the SoC architecture.
- It needs IP reconfiguration during the data flow progress to handle the packet retransmissions.
- It generates interruption to the CPU in order to inform about the packet error status.
- It can handle complex data flow for the combination of the retransmitted packets.

In the next section, the HARQ algorithm is detailed and an implementation of the algorithm is adapted to the NoC architecture in accordance with the methodology flow.

A. The HARQ Process

Three basic ARQ schemes are described in [3]:

- The Stop and Wait (SW) mechanism is a sequential packet transmission. The transmitter sends a new packet when the previous one has been acknowledged by the receiver.
- In order to avoid latency, the “Go Back N” (GBN) mechanism holds a buffer at the transmitter side. Several packets are sent consecutively. When one of them is erroneous, all the packets from the erroneous one are retransmitted.
- By adding a buffer at the receiver side, only the erroneous packets are retransmitted. The acknowledged packets are reordered at the receiver side. This mechanism is called Selective Repeat (SR).

The experimentation shows that SR algorithm requires two buffers and needs to reorder the packets. SW is easier to implement and can be advantageously parallelized with an N-channel SW algorithm: N independent and parallel SW mechanisms are launched in concurrency. For the first hardware implementation, 1-channel Stop & Wait algorithm is retained before proceeding with 2-channel SW.

1) The combination

At the receiver side, the combination of the retransmitted packets depends on the puncturing patterns [1] [6].

- With the Chase Combining method (CC) [4] the same puncturing pattern is used for all the retransmissions. The combination consists of a summation.
- With the Incremental Redundancy (IR) method, the global rate is incrementally increased at each retransmission by adding parity bits. So each retransmitted packet is not self decodable, and the size of the retransmitted packets depends on the number of parity bits added.
- Partial IR is a particular IR case. The puncturing patterns may be different but the packet size is the same. It leads to a simpler MAC layer specification.

In the sequel of this paper only the CC and the partial IR combination methods are considered in order to handle a constant packet size.

2) System C simulation chain

The full HARQ communication chain (Figure 8) is developed with SystemC [5]. The objective is to evaluate the performance of the whole algorithm on a lashing of data to get significant results and to generate golden patterns. These reference data are used for the validation of the integrated chain on the SoC.
3) Performance results

Figure 9 are obtained for 19530 data slots (more than 20 millions bits) with CRC of 16 bits long, a channel encoder based on turbo codes, a coding rate of 1/3 and QPSK mapping. In the figure hereunder, the curve labeled “1 Tx” corresponds to the transmission with no combining i.e. without HARQ mechanism. The gain decreases with the number of retransmissions in both CC and partial IR cases.

![Figure 9: FER (turbo coder 1/3, QPSK)](image)

The gain between the first and the second transmission corresponds either to the energy gain (3dB) due to the retransmission for CC techniques or to the energy gain added by the diversity gain (3.6 dB) for partial IR. The gain between the third and the fourth transmissions is around the half of the gain between “1Tx” and “2Tx” for both cases. A trade-off has to be done between the transmission time and the reached gain because both are influenced by the number of retransmissions and this parameter will impact the throughput.

B. Adequation with the NoC architecture

As two FAUST ASICs are embedded in the SoC, all the functionalities integrated in the ASIC and compatible with the need of the application are used. So, FPGA1 and FAUST1 are employed for the transmitter, and FPGA2 and FAUST2 are employed for the receiver (Figure 4).

For the transmitter, in FAUST1, the three IPs: convolutive coder (including a scrambler and a puncturer), bit interleaver and mapper, are used to achieve the PHY layer of the HARQ transmitter chain, as well as the CPU and the memory resources. Two new IPs, the CRC encoder and the ACK manager, must be designed and placed in FPGA1.

As regards the receiver, in FAUST2, the CPU and the memories resources are employed. The three IPs, convolutive decoder (including a de-scrambler and a de-puncturer), bit de-interleaver and de-mapper, are used to achieve the PHY layer of the HARQ receiver chain. Four new IPs must be considered in FPGA2:

- the CRC decoder,
- the HARQ combiner,
- the ACK manager,
- the de-puncturer including in the Faust de-coder should be by-passed and re-implemented in FPGA2 in order to experiment the partial IR algorithm.

Finally, since the six nodes NoC implemented inside offers wide possibilities (Figure 6), since the two IPs needed for the transmitter have their dual functionalities in the receiver and since the FPGA (Xilinx Virtex 4) is large enough, only one FPGA will be designed for the HARQ application. A parameter will be contrived to qualify the use at the transmitter side in FPGA1 or at the receiver side in FPGA2.

C. Design of new IPs

So, the new hardware IPs are designed in VHDL description language and a dedicated FPGA is synthesized.

- The ACK manager is a small improvement in the RFIO block. It can be parameterized to work at receiver side or at transmitter side. At receiver side, when the ACK manager receives a dump from the CRC decoder, indicating the status (ACK or NACK) of the transmitted packet, it sends an interruption to the receiver CPU. Then, the CPU launches the needed re-configurations to the IPs for the next transmission or re-transmission. At transmitter side, when an ACK/NACK status is received by UL, the ACK manager sends an interruption to the transmitter CPU to allow the needed re-configurations.

- The CRC can also be configured as an encoder or a decoder. At receiver side, the data packet is acknowledged if the decoded checksum is equal to a magic number and a dump is launched to the ACK manager. The data flow is transferred to a buffer.

- In a first implementation, the combiner was very simple and realized a bit to bit addition. The double buffering was external. After experimentation, it is established that it is not suitable for a NoC architecture: the implementation of a simple double buffering takes a lot of time, consumes two internal buffers, and needs consequent re-configurations between each packet re-transmission. So, a second implementation is done in order to avoid the external double buffering: a FIFO is placed inside the combiner and a state machine manages the mechanism.

- To allow experimentations of the partial IR algorithm, the combiner must be inserted between the de-puncturer and the de-coder. So, the de-puncturer inside FAUST2 is bypassed and is re-implemented in FPGA2.

D. Data flow progress, co-simulation and co-verification

Figure 10 and Figure 11 depict the specific credit mechanism of the asynchronous NoC protocol applied to the HARQ communication chain for both transmitter and receiver.
At transmitter side (Figure 10), a double buffering mechanism allows to reduce the data access latency: a packet is coded before the transmission, the CRC of the next packet is computed and stored. When an ACK signal is received, the CPU permutes the roles of the two buffers and a new packet is sent.

At receiver side (Figure 11), the external buffers Bu2 and Bu3 are suppressed and replaced by a FIFO inside the combiner. When an ACK signal is received, the data stored in Bu4 are transferred to the host PC.

E. Instrumentation in the SoC

Thanks to the co-simulation framework, all the configuration vectors describing the HARQ application are generated. Each new IP is individually tested before its integration. The CPU code is developed with the ARM tools and the whole application is integrated including the two CPUs (one at transmitter side and the other at receiver side), and allowing many re-transmissions of a same packet. At this step, the co-verification is essential to confirm the good performance of the two CPUs (MAC layer) with the IP units (PHY layer). When the co-verification is completed, the transfer to the hardware platform is done with a lot of functional guarantees.

In the hardware platform, the CPU code and the IP configuration are launched in the memories via the Ethernet link. The two CPUs are started and the data flow begins its progression in the communication chain. When a packet is acknowledged by the CRC decoder, it comes back to the host PC via Ethernet and can be analysed with FER technique. The adjustment of the HARQ application in the SoC platform does not take a lot of time and results are rapidly obtained, proving the pertinence of our methodology.

V. CONCLUSION

This paper presents a complete methodology in order to efficiently implant a communication chain into a SoC. Once the co-simulation and the co-verification frameworks are deployed, the application building is like a “LEGO” assemblage.

This study points out the need of higher level abstraction tools in order to configure more easily the system. The spirit consortium [11] tries to answer to this problematic. The co-verification stage in VHDL/Verilog description takes long but is essential. A global co-verification framework, including a co-simulation and a processor emulator, should be anticipated to handle complex data flow and complex interactions between the MAC layer (CPU) and the PHY layer (IP units).

For the next SoC generation, implemented in nanotechnology, the application deployment methodology and its associated tools should be thought at the chip conception.
REFERENCES


\( \mu \text{SPIDER CAD TOOL: CASE STUDY OF NOC IP GENERATION FOR FPGA} \)

Samuel Evain  
CEA LIST  
CEA Saclay 91191 Gif sur Yvette  
email: samuel.evain@cea.fr

R. Dafali, J-Ph.Diguet, Y. Eustache, E. Juin  
Univ. Européenne de Bretagne / CNRS / LESTER lab  
56100 Lorient, France  
email: jean-philippe.diguet@univ-ubs.fr

ABSTRACT

This paper introduces the \( \mu \text{Spider CAD tool for NoC design under latency and bandwidth constraints and describes the different steps} \) of the associated design flow. We show how the tool can be used to automatically generate a NOC IP compliant with Xilinx EDK tool. We present synthesis results and a real implementation of a video application based on a multi-processor architecture. Finally we conclude about research to be done at application/OS levels above current work to achieve a complete and efficient implementation of a multi-processor embedded system.

1. INTRODUCTION

The concept of Network On Chip (NoC) has been recently introduced as an alternative to bus in order to solve the tedious issue of emerging system on chip (SoC) interconnect design. Different arguments are commonly forwarded to sustain the interest for NoCs [1] based on packet switching. First the NoC can provide new spatial and time parallelism capabilities to cope with expanding bandwidth requirements and the management of an increasing number of inter IPs communications. It also contributes to raise the abstraction level of design tools for flexibility and productivity improvements. This point is crucial, since the design of complex SoC, implementing tens of processors, IPs and memories, means error prone communication schemes that can be intractable for designers under time to market pressure.

Moreover a NoC is based on physical and logical control flows that offer new opportunities for controlling quality of service including real-time, power and reliability. Finally, NoC intrinsically holds the property of scalability, which is a key point for design reuse and SoC configurability.

Today first industrial solutions [2] are available and ongoing researchs investigate further aspects. One of the research issues in which we particularly focus on is design flow to handle complexity and provide designers with CAD tool. Some equivalent work is proposed in Aetereal [3] for ASIC design. This approach is based on an architecture model and a methodology, which is close to ours but differs at level of mutual exclusion considerations and heuristics choices for path and time slot allocation. There is no real work about a specific CAD tools for NoC IP generation targeting FPGA. In [4] FPGA are used as a proof of concept independently from any CAD tool. Other work are based on FPGA for emulation purpose in [5] and to speed-up simulation in [6].

To our point of view, the main issue is the productivity gain for implementing complex communication schemes. Our project was, from the beginning, driven by the objective of providing the SoC designer with an ad-hoc component offering services for communication management at application level. To get over this challenge we have designed a CAD tool and a software layer for a simplified access to the communication medium at application level. On the one hand, it is well known that flexibility has a cost. An area overhead mainly due to interface and router FIFOs and an increase of communication latencies due to the path length and packet routing are generally observed. However, the paradigm can take benefit from a rigorous formalization that enables the implementation of efficient automatic methods. We based our framework development on the guaranty of latency and bandwidth requirements at application level while minimizing the resulting NoC cost.

\( \mu \text{Spider CAD tool for NoC design performs design space exploration and code generation. Design space exploration is implemented in an interactive way based on designer choices for arbiter, routing policies and topology selections. Then automatic procedures are available for time-consuming and error prone tasks such as Time Division Multiplexing (TDM), FIFO sizing and path allocation for guaranteed traffic management. Then based on designer and tool choices, code generation is automatically carried out and can specifically targets Xilinx EDK tool by producing the IP folder including data/IP.mpd and data/IP.pao and /hdl/vhdl/*.vhd files with the correct format. In this paper we present the \( \mu \text{Spider design flow as IP generator for reconfigurable MPSoC implemented on Xilinx FPGA.} \)

2. DESIGN FLOW OVERVIEW

Our NOC model enables the implementation of two kinds of communications: best effort (BE) and guaranteed traffic (GT) based on a TDM technique. Figure 1 gives an overview of
the NoC design flow. An interactive GUI (Fig. 2) helps the designers to easily follow the μSpider design flow.

The first step enables the designer to rapidly specify the application and the NOC parameters. The application is specified with a set of characterized communication tasks, with application throughputs (Application Graph.xml) and if necessary with links between mutual exclusive communications (Mutual Exclusion Graph.xml). The NOC knobs are related to the NOC topology which can be ad hoc or automatically generated as 2D Mesh. NoC parameters are specified by the designer, the main ones are: Path bit width (e.g. 32 bits) Router parameters (ports, routing policy, arbiter policy, etc.), Network Interfaces (NI), Flow control policy (with or without End to End flow control) and wrapper (slave, master, bus standard) and the IP mapping namely the IP/NI association. If real-time constraints are required, then related communications are implemented with virtual channel (GT, BE, BE with priority).

The second step deals with derivation of local latency and bandwidth constraints for each unidirectional communication from application I/O throughputs (e.g. telecom chain or image processing). The objective is to extract local latency and bandwidth constraints for each communication task from global I/O application constraints. The important issue, which is usually omitted in NOC design flows, is in practice necessary for applying following steps 3 and 4, which requires constraints for each individual and unidirectional communication. This work is not trivial since firstly different local decisions are possible to meet global constraints and secondly latency, bandwidth and TDM table size are strongly dependent. Moreover read operations imply two types of heterogeneous communications: the read command (request) and the data response for which two distinct set of constraints must be defined. Thus, the second step first transforms communication tasks into unidirectional ones. This aspect is required for read operations that need a lightweight forward communication for sending a read command and a backward communication for receiving data. Then we produce, for each GT communication, the minimum bandwidth and a set of rules for latency/bandwidth checking. Due to space restriction the step is not detailed in this paper but can be found in [7].

The third step computes the minimum TDM table size required for implementing GT communications and a minimum bandwidth for all BE communications. TDM tables are based on integer bandwidth division, so compared to real constraints higher bandwidths are usually obtained, so we first try to solve the TDM size issue while considering absolute lower bounds. Basically our method is based on a heuristic that starts with a minimum size, which is increased until a solution is found. As a result we obtain minimum latencies taken into account within the next step.

The fourth step [8] automates the more tedious task which is the exploration of time (TDM slots)-space (NOC paths) space in order to allocate time slots to each GT communication. It provides next steps with a complete NOC specification. The exploration / allocation step is based on a two steps heuristic that, for each communication, first evaluates link usage probabilities and then selects a valid path with minimum impact on non allocated paths. The heuristic parameters (cost function, sorting criteria) have been selected for globally minimizing the FIFO costs.

The fifth step is the VHDL code generator, some additional C API codes are also provided for interfacing NOC components with IPs which are compliant with the OPB bus standard.

3. ARCHITECTURE MODEL

The architecture model is a network of bus-based clusters connected through a NoC. In case of Xilinx targets, we use OPB-based clusters connected to a NoC component instantiated as a traditional IP in an EDK project. As depicted in Fig. 3, typically a cluster is composed of a Microblaze (MB) soft processor that can control one or more local RAM memories connected to its OPB bus. So, the NoC IP is in charge of inter-cluster communications, the interface cluster / NoC network interface protocols and the Network Interface, which manages NoC accesses and (un)packetise data. Each cluster can integrate masters able to initiate read or write communications and slaves that can respond to read requests. A cluster can communicate with other clusters through dedicated channels, each input or output channel is implemented as a FIFO. Note that the architecture model is based on first designer choices such as topology, routing policies and link bit width for instance.

In the following we present different aspects of the architecture model.

3.1. Communication model

Three kinds of communications are implemented: write, read and message passing. They are depicted in Fig. 4. Fig. 5
illustrates these different schemes, where the following commands are implemented between masters and wrappers:

- \texttt{Rc}: Read
- \texttt{S}: Status register (In/Out FIFO not full, almost full, not empty, almost empty)
- \texttt{Wc}: Write
- \texttt{D}: Data FIFO
- \texttt{rW}: Write request
- \texttt{rR}: Read request

The first scheme, described in (Fig. 5-a), is a write operation that initiates a master to write data in a remote slave RAM 2 after checking that enough space is available in input FIFO.

The second of one, depicted in (Fig. 5-b), is a read operation, which is quite tricky since a master cannot freeze the OPB bus while waiting for a slave to answer its request. So a real read operation is not appropriate and a write operation is used instead. In practice a master writes a read request in slave wrapper registers, an interrupt is emitted by the local wrapper to the master when requested data are available.

The third scheme, depicted in (Fig. 5-c), is a message passing between two masters, an interrupt is emitted to alert the remote master that a message is available.

Note that an alternative solution, based on polling instead of interrupt, has been developed for wrappers / masters communication. Such a scheme is interesting when a single task is assigned to the master.

### 3.2. Software layer

From an application point of view, these communication mechanisms are available as basic C API, which provide MB with NoC services:

- \texttt{extern int WRAPPER_S_Read_Status(int Wrapper_Address): Returns the wrapper status for each channel.}
- \texttt{void WRAPPER_S_Read(int Wrapper_Address, int channel, int Remote_Address, int *Data, int n): Read n data on channel (remote IP ID), at Remote_Address (local address) and stores in *Data.}
- \texttt{extern void WRAPPER_S_Write_RAM(int Wrapper_Address, int channel, int Remote_Address, int *Data, int n): Write n data on channel (remote IP ID), at Remote_Address (local address) and stores in *Data.}
- \texttt{extern void WRAPPER_S_Recv(int Wrapper_Address, int channel, int Data, int n): Read data in FIFO channel.}
- \texttt{extern void WRAPPER_S_Send(int BaseAddress, int channel, int *Data, int n): Write data in FIFO channel.}

A Hardware Abstraction Layer (HAL) has been developed to separate abstract and physical addresses, thus a single API is used above previous one:
Write(Logical Address, N, RW, *p)
Where Logical Address is then transformed into Wrapper Address, channel and local address if necessary.

3.3. NI / OPB wrappers

Two kinds of wrappers are available to adapt OPB and NI protocols, a Slave wrapper depicted in Fig. 6-a) and a Master wrapper described in Fig. 6-b).

3.4. Network Interface

The Network Interface (NI) model, described in Fig. 7, is generic. A NI is specialized according to NoC specifications. The main configuration parameters are the number of channels, the TDM table. The number of output (resp. input) channels corresponds to the maximum number of receiver (resp. emitter) IPs. For cost reasons no more than two virtual channels are usually used, one for BE traffic and one for GT traffic, each channel can be associated to one virtual channel category. A TDM table is used to allocated contiguous time slots to output channel according to bandwidth and latency constraints as explained in section 2.

3.5. Router

The Router model, given in Fig. 8 is quite largely configurable. Main configuration knobs are number of routers network ports, routing technique (XY or Street sign) and arbiter policy for BE (round robin, first arrived - first served).
Note that a new street-sign path coding model [9] has been designed in order to improve security, reconfigurability and header overhead. Consequently, based on a minimum number of coding bits according to each router arity individually, a path can be used as an identifier [10] and a simple mechanism is available to transform a forward path into a reverse path without knowledge about a possible new configuration (emitter placement, topology, router arity).

4. RESULTS AND EXPERIMENTS

4.1. Introduction

The µSpider design flow has been successfully applied for designing different NoCs in the domain of Telecom for instance [7], we also have recently proposed an architecture for security management in NoC-based reconfigurable SoC [10]. Implementing real MPSoC requires important engineering efforts especially if specific IPs are needed. So, in a first stage generated NOCs have been implemented for generic applications. The test architecture was based on 3 MB, 2 slave RAM and a 2x2 mesh NoC. The architecture is described in Fig. 3.

4.2. Test conditions

These experiments have been completed on a Xilinx Pro FF1152 BOARD based on a Virtex-II VP50-5 device, with Xilinx ISE 6.3 SP3 and EDK 6.3 SP2. In the EDK framework, the NOC is implemented as an usual IP (see Fig. 9).

Our experimental NOC has been parameterized as follows: bit width: 32; Routing: street sign; End to end flow control; Round robin arbiter; 2 routers with 3 ports; 2 routers with 4 ports; 3 NIs with 1 port and 2 Channels; 3 NIs with 1 port and 3 Channels; 3 master wrappers; 3 slave wrappers.

Different synthetic programs have been implemented on MBs in order to test a large set of configurations, each configuration has been validated in terms of data integrity and transfer timing. Tests results have been obtained through an UART connected to a MB. The following five virtual channel configurations have been successfully tested:

- BE: single BE channel with a 2 slots input FIFO;
- GT: single GT channel with a 16 slots TDMA table;
- GT-BE: two virtual channels, BE and GT;
- BE-BE: two virtual BE channels and two priorities;
- BE8: one single BE channel with a 8 slots input FIFO.

4.3. Results

Based on the XML NoC architectural description, VHDL code generation is performed with a delay lower than 2s on a standard PC. NOC specification roughly corresponds to 10.000 VHDL code lines. Moreover, as previously mentioned, a code is produced with the EDK format and can be used directly as any IP. Table 1 synthesizes results for each case study. NoC sizes don’t include wrappers which have the same following features for all cases:
4.4.1. Application Implementation

The aim of this implementation is to proof the ability of our platform to implement real and complex applications. The object tracking application has been initially developed by CEA-LIST in a very generic C code for multi-target prototyping purposes. These experiments have been completed on the same VirtexII board used for previous tests. The target architecture is equivalent to Fig.9 except that the first MB is replaced by a Power PC. The reason of this choice reveals one the current practical locks in the domain MPSoC design, namely the question of interface standards. The NoC IP used is based on GT communications and correspond to case 2 in tab.1.

The application has been partitioned on the target as follows. The hard-mapped PPC processor runs four functions, Image loading & pre-processing (format adaptation), noise filtering based on an average of 2 frames, background subtraction and VGA control. When a new image is ready, a message is sent to the MB 0 that launches three tasks: Image loading from DDR SDRAM connected the PLB PPC bus, adaptive thresholding and load resulting data in its local RAM memory, finally MB0 sends a message to MB1 to inform it that it can load new data. MB1 runs functions Image loading from MB0 RAM, Dilatation, erosion, reconstruction, Gravity center computation, Labeling and Border drawing and storage in DDR SDRAM.

The implementation and the use of the IP NoC was quite plug and play, in that sense the experience was a success since we have implemented processing parts as software running on embedded (soft/hard) processors and communications through the NoC in a very short time. Moreover, the whole application functionality has been checked.

4.4.2. Observations

If the objective was not performances, results we obtained are quite poor since we finally get one image per second. The problem is basically not on the NoC side which is underused and able to provide expected bandwidth and latencies. Actually the causes of performance degradations, which can be solved by the way, are i) interface unavailability, ii) EDK limitations and iii) missing OS services.

**Interface:** The heterogeneity of standards and the availability of interfaces for various peripheral impose constraints to the designer, who, under time and economic constraints, implements possible options instead of his real choices. In our case we have wrappers for OPB bus standard, whereas a PLB bus is required for the PPC communications and the (free) VGA controller. It means that additional master and slaves OPB/PLB bridges have been implemented to cope with this point. This issue can be easily solved with the appropriate IP library, moreover PLB/OPB wrappers are very similar.

**EDK:** EDK tool is an efficient tool if the designer deals with the architecture model where memories are acceded through OPB or PLB buses. However if OPB interfaces are used for interfacing memories, a MB or PPC connected to the bus is required to initialize the address map. As a result all memories are implemented as slaves on OPB or PLB bus and introduce conflict accesses to shared memories, the consequence is that processor runs sequentially whereas a pipeline execution would be theoretically possible. Again this problem can be solved with ad hoc simple wrappers for memory interfacing.

**OS:** The last and much more complex issue is the question of synchronization at application level. Basically a NoC can prevent the system from transaction and transport level deadlocks with end to end credit-based and local handshakes flow controls, however the designer remains in charge of tuning the application control flow in such a way that no deadlocks happen at application level.

A solution to this tedious question is the implementation of a new class of OS services to manage NoC services while taking benefits from OS synchronization and mutex facilities. The last point is quite tricky since it has to be seen in relation with the kind of communications required by application in terms of burst sizes, periodic or sporadic, dynamic or static behaviors, data dependencies and so on. We currently work on this point to bring up NoC services as system level.

<table>
<thead>
<tr>
<th>Case 1</th>
<th>Slices</th>
<th>7204</th>
<th>607</th>
<th>1167</th>
<th>430</th>
<th>645</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freq.</td>
<td>94</td>
<td>120</td>
<td>118</td>
<td>136</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>Case 2</td>
<td>Slices</td>
<td>7206</td>
<td>624</td>
<td>1195</td>
<td>430</td>
<td>643</td>
</tr>
<tr>
<td>Freq.</td>
<td>94</td>
<td>120</td>
<td>118</td>
<td>136</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>Case 3</td>
<td>Slices</td>
<td>9251</td>
<td>630</td>
<td>1181</td>
<td>851</td>
<td>1317</td>
</tr>
<tr>
<td>Freq.</td>
<td>85</td>
<td>93</td>
<td>90</td>
<td>118</td>
<td>118</td>
<td></td>
</tr>
<tr>
<td>Case 4</td>
<td>Slices</td>
<td>9205</td>
<td>621</td>
<td>1170</td>
<td>851</td>
<td>1317</td>
</tr>
<tr>
<td>Freq.</td>
<td>84</td>
<td>93</td>
<td>90</td>
<td>118</td>
<td>118</td>
<td></td>
</tr>
<tr>
<td>Case 5</td>
<td>Slices</td>
<td>7488</td>
<td>622</td>
<td>1203</td>
<td>443</td>
<td>662</td>
</tr>
<tr>
<td>Freq.</td>
<td>100</td>
<td>120</td>
<td>114</td>
<td>132</td>
<td>132</td>
<td></td>
</tr>
</tbody>
</table>
5. CONCLUSION

μSpider is a NoC CAD tool with two main parts, the first one performs architecture design and automatically runs tedious and error prone tasks such as path extraction and coding, TDM sizing, under QoS (latency, bandwidth) constraints while targeting resource minimization (FIFO sizes). Some features such as path minimization and multiplexing based on mutual exclusion properties are exploited to reduce final cost. The second part of the flow, based on the XML NOC architecture description, is a code generator. Moreover, NIs can be configured according to the space-time exploration step to respect communication constraints. VHDL code is automatically produced in such a way, that it can be used as a usual IP within the Xilinx EDK tool. Our approach and tool have been validated with real cases. These experiences have revealed weaknesses of current CAD framework for MPSoC design on FPGA. Interfacing problems will be fixed with CAD tool evolutions, however the definition and formalization of new OS/NoC services remain a real challenge and open promising perspectives to address the design of multiples processes with dynamic behaviors on future (reconfigurable) MPSoC. We currently work on this topic.

6. REFERENCES


Fig. 5. Communication model scenario.
Fig. 6. OPB / NI wrappers.

Fig. 7. Network Interface (NI) Model.

Fig. 8. Router Model.

Fig. 9. IP NOC in EDK.
Algorithm Architecture Adequacy for High Speed 3D Tomography

Nicolas GAC, Stéphane MANCINI, Michel DESVIGNES and Dominique HOUZET

Abstract—Backprojection is a computational costly step in tomography image reconstruction such as Positron Emission Tomography (PET). In this purpose, this paper presents a Pipelined, Prefetch and Parallelized Architecture for PET backprojection (3PA-PET). The main strength of this architecture comes from its original memory access strategy, masking the high memory latency of the external memory. The 3PA-PET architecture is implemented on a System on Programmable Chip (SopC). Time performances are compared with a desktop PC, a workstation and a GPU. We prove that the exploitation of the intrinsic temporal and spatial locality by the 3D Predictive and Adaptative (3D-AP) memory cache succeeds to run efficiently several pipelines of backprojection: each reaches a computational throughput close to 1 operation per cycle.

I. INTRODUCTION

Reconstruction of images in tomography is cpu intensive and usually postponed. However, real-time reconstruction of the acquired data in PET imaging would facilitate positioning of the subject and detection of potential problems during the acquisition. Real time reconstruction is also needed for large scale diffusion of clinical PET examinations, which are used for early detection of cancer, evaluation of disease spread and treatment response. Then, minimization of examination duration can decrease the cost of PET to make its powerful technology more widely available. However Image reconstructions in the field of tomography, including SPECT, CT, multislide CT need to process a costly backprojection step.

There are several implementations of reconstruction systems on PC clusters [1], [2], on DSPs [3], on adapted CPUs for vector processing like Cray or Cell [4], [5], on specific hardware like ASICs or FPGAs [6], [7], [8] or on Graphics Processor Units (GPUs) [9]. All these implementations have to face the memory bottleneck due to the limited bandwidth of the main memory. Without an adapted memory strategy, one rule applies to all these architectures: the greater the computing power is, the more the reconstruction time is penalized by the memory wall. Works previously mentioned are based on Algorithm Architecture Adequacy methods to overcome this issue. For instance, the implementation of 3D cone-beam backprojection done by M.Kachelriess [4] has coupled high computing speed on a cell processor (code parallelization, incremental algorithm, loop unrolling, interpolation pre-processing) with software memory prefetching techniques.

In this paper, we present a Pipelined, Prefetch and Parallelized Architecture for 3D PET backprojection (3PA-PET) implemented on a System on Programmable Chip. This architecture has a computation throughput of about one cycle per operation for each pipeline. It overcomes the memory access bottleneck thanks to a prefetching memory strategy. Results in accuracy of reconstruction and in time performance are evaluated and compared to a desktop PC, a workstation and a GPU.
II. ALGORITHM ARCHITECTURE ADEQUACY

A. 3D PET Backprojection

Data acquired by PET scanners is the Radon Transform of the observed body and is called sinogram [10]. One sinogram element called a bin represents the number of coincidence events counted on two detectors of the scanner. The line what connects two detectors is called a line of response (LOR). The sinogram is a 4D space along \((\Delta, \lambda, \psi, \rho)\). Coordinates \((\Delta, \lambda)\) represent a couple of rings: \(\Delta\) the distance between the two rings (segment number) and \(\lambda\) the mean axial coordinate of the two rings (plane number). Then the coordinates \((\psi, \rho)\) represent one particular LOR: \(\psi\) is the azimuthal angle (angle number) and \(\rho\) is the tangential coordinate (bin number). Backprojection computes the estimated distribution of radio-tracer \(f^*\) for each voxel (VOlume piXEL) \(\vec{x}\) by summing up all the bins corresponding to the voxel projection:

\[
f^*(\vec{x}) = \int \int \text{bin}(\Delta, \lambda, \psi, \rho) \cdot J_\Delta \cdot d\psi \cdot d\Delta \quad (1)
\]

\(J_\Delta\) is a jacobian and bin coordinates \((\rho, \lambda)\) are computed though equation (2).

\[
\begin{align*}
\rho(\psi) &= x \cdot \cos \psi + y \cdot \sin \psi \\
\lambda(\psi, \Delta) &= z - (-x \cdot \sin \psi + y \cdot \cos \psi) \cdot \frac{\Delta}{2R_a} \quad (2)
\end{align*}
\]

B. Memory access strategy

As the sinogram is kept in a SDRAM like external memory, we need an efficient memory management to overcome its latency and allow a high level of parallelism. Standard caches based on a linear accesses to memory can’t be a satisfactory solution because of their complexity and their weakness to load the needed data. Indeed, memory accesses needed to reconstruct one single voxel \(f(\vec{x})\), follow a 3D sinusoid in the 4D sinogram as shown on fig. 1. Such a pattern is of poor address locality. Moreover because of the additional lambda dimension, 3D backprojection accesses are greater and more distributed in the memory space than in the 2D backprojection case [11]. The challenge is to speed-up these 4D memory accesses.

Therefore a new cache mechanism is needed. Predicting which bins the processing unit will use, would help the cache to download the needed bins during the computing process. Thus, we could balance the computational throughput with the access memory throughput.

C. Improvement of spatial and temporal locality

A volume reconstruction by the Voxel-driven Bilinear Interpolation (VBI) standard backprojection is made of three loops: the first on the voxels \(\vec{x}\), the second on the segments Delta and the third on the angles psi. Since voxels can be reconstructed independently, the loop on voxels can be split by two, one on blocs of voxels \((0 .. n_{max})\) and one on the voxels of a bloc \(n \ (x_{n_{min}}(n) .. x_{n_{max}}(n))\).
Algorithm 1 Reordering loops of 3D Backprojection improving spatial and temporal locality

\begin{algorithm}
\begin{algorithmic}
\For {$n = 0 \text{ to } n_{\text{max}}$}
  \For {$\Delta = 0 \text{ to } \Delta_{\text{max}}$}
    \For {$\psi = 0 \text{ to } \psi_{\text{max}}$}
      \For {$\vec{x}_{\min}(n) \text{ to } \vec{x}_{\max}(n)$}
        \State $f(\vec{x}) = f(\vec{x}) + \text{bin}(RHO(\psi, \vec{x}), PSI, LAMBDA(\psi, \vec{x}))$
      \EndFor
    \EndFor
  \EndFor
\EndFor
\end{algorithmic}
\end{algorithm}

The reordering of loops increases the temporal and spatial locality of memory accesses. Indeed, for given $\psi$ and $\Delta$ values, a $\text{bin}(\psi, \vec{x})$ will be used several times since the projection of a 3D Bloc of voxels is a 2D plane in the 4D space of the sinogram.

D. Mean Bin Reuse Rate (MBRR)

The Mean Bin Reuse Rate (MBRR) is defined as the ratio between the number of bins accessed in cache memory and the number of bins loaded in cache memory. The MBRR can be computed analytically. It depends on the shape of the reconstructed bloc of voxels.

![Fig. 2. Mean Bin Reuse Rate (MBRR) estimated versus the size of reconstructed blocs of voxels for each segment of a Siemens HR+ sinogram (span 9 with 96 angles of projection)](image)

Fig. 2 presents the optimal MBRR computed for each segment versus the size of the bloc. For a 16*16*3 bloc, we can expect a MBRR of 13 for segment 0, 10 for segment 1 and 8 for segment 2.

III. 3P Architecture for PET

A. Pipelined Architecture

The pipeline implements the different steps of the VBI standard backprojection: the computation of $\text{rho}(\psi, \vec{x})$ and $\text{lambda}(\psi, \vec{x})$, the bilinear interpolation of the bin, and finally the accumulation of the voxel value. The forward flow control is done by packets passing through each stage of the pipeline.

![Fig. 3. Pipeline of the 3PA-PET Architecture](image)
The accesses to the 4 bins needed for the bilinear interpolation are done through the bridge memory. This bridge controls the cache memory and freezes or not the pipeline whether the data requested is available or not. A backward flow control synchronizes the pipeline and the cache memory.

B. Prefetch Architecture

The 3D-AP cache [12] masks the latency of the external memory. Thanks to it, the pipeline is no more systematically stalled. The bridge memory gets four bins from the cache at each clock cycle.

The 3D-AP cache is a generic cache memory mechanism that prefetches memory access sequences following a continuous path into a 3D memory space. The requests of data from the pipeline are done with spatial coordinates, here (rho, psi, lambda). From the previous coordinate requests, the cache estimates dynamically which data is likely to be requested in the future. This is done by a statistical analysis on each spatial dimension. Moreover, cache data transfers are masked. Indeed, while the new data grabbed by a cache update is transferred from external memory, the data shared by the old and the new cache zone stay available in the cache.

Cache parameters have to be set beforehand by the user to fit to the 3D memory path as close as possible. In this study, we set for each dimension the value of five parameters:

- cut-off frequency and sampling frequency: the mean coordinate is computed by a first order low-pass IIR filter configured by these two frequencies.
- cached zone size: this zone is declared to the bridge memory to be available in cache. In this study, this size is a static parameter.
- guard zone size: each time the mean coordinate is out of the guard zone, the cache zone is updated.
- cache speed: The cache speed has to be set according to the speed of the data accesses performed by the application on each spatial dimension.

C. Parallelized Architecture

To increase the computing power, several pipelines are parallelized. A hierarchical cache reduces as much as possible the memory bus occupation when backprojection units work in parallel. In this hierarchical design, one leaf cache is associated to one 3D backprojection unit while a root cache is feeding each of these leaf caches.

Fig. 4. 3A-AP Cache zones

Fig. 5. Each leaf cache is feeding by the root cache. All backprojection units are synchronized by one single finite state machine (FSM).
The reconstruction of a macro-bloc of neighbor blocs is distributed over the backprojection units. The cache concept presented previously with one unit, applies here in the same manner. The bin needed during a macro-bloc reconstruction draws a 3D sinusoid. Each leaf cache stores a smaller 3D sinusoid needed for its bloc reconstruction as presented on fig. 5.

IV. Results

A. Accuracy of reconstruction

The implemented VBI standard backprojection is a fixed point version of the original algorithm. Moreover the sinogram data is converted from float to short int (16 bits). Accuracy of reconstruction of 3PA-PET is compared to software reconstruction thanks to a bit true software implementation of the FPGA-based architecture.

The reference data set used is a sinogram analytically computed from a 3D Shepp Logan volume of 128*128*63 voxels. This phantom is a standard volume used in tomography to measure the accuracy of reconstruction. The sinogram is obtained from the STIR (Software for Tomographic Image Reconstruction) open source tool kit [13].

![STIR and 3PA-PET comparison](image)

Fig. 6. A slice of the 3D Shepp Logan phantom (128*128*63 voxels) reconstructed by STIR and 3PA-PET backprojection.

On fig. 6 and 7, one can compare the Shepp Logan volumes reconstructed by the STIR voxel-driven backprojection and by the 3PA-PET backprojection.

The accuracy of reconstruction of the 3PA-PET backprojection is measured with two metrics: the mean absolute percentage error (MAPE) and the peak signal-to-noise ratio (PSNR). Both compare a volume \( f_1 \) with a volume of reference \( f_{ref} \) corresponding to the original volume or to a volume reconstructed with a reference 3D-BP implementation. The PSNR corresponds to the ratio between the maximum of \( f_{ref}(\vec{x}) \) (dynamic range) and the mean squared error (MSE) of \( f_1 \) compared to \( f_{ref} \):

\[
PSNR = 20 \cdot \log_{10} \frac{\text{max}(f_{ref}(\vec{x}))}{\sqrt{\text{MSE}}} \quad (3)
\]

On table I, volumes reconstructed by STIR, by VBI with floating point arithmetic (VBI-flt) or VBI with fixed point arithmetic (VBI-fix) are compared to the original phantom and to each other.

All implementations have an error of reconstruction around 3.9% with a PSNR of 10.5 dB compared with the original volume. This error is intrinsic to the method. The difference between floating point and fixed point implementation corresponds to a MAPE of 0.13% and to a PSNR of 23 dB. With different data type, the difference has a MAPE of 1.1% and a PSNR of 19 dB. Thus we can conclude that the 3PA-PET implementation of VBI backprojection leads to an accurate reconstruction system.
\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
compared volumes & data & MAPE & PSNR \\
\hline
STIR / original & float & 3.88 % & 10.5 dB \\
VBI-Flt / original & float & 3.88 % & 10.5 dB \\
VBI-Fix / original & float & 3.88 % & 10.5 dB \\
VBI-Flt / original & int16 & 3.97 % & 10.5 dB \\
VBI-Fix / original & int16 & 3.97 % & 10.5 dB \\
\hline
\end{tabular}
\caption{Accuracy of reconstruction}
\end{table}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
compared reconstructions & & \\
\hline
STIR / VBI-Flt & float & 0.35 % & 21.5 dB \\
VBI-Fix / VBI-Flt & float & 0.13 % & 26.2 dB \\
VBI-Fix / VBI-Flt & int16 & 0.13 % & 23.0 dB \\
VBI-Fix / VBI-Flt & int16/flt & 1.1 % & 19.0 dB \\
\hline
\end{tabular}
\caption{Compared reconstructions}
\end{table}

TABLE I

Accuracy of reconstruction and Compared Reconstructions for the Shepp Logan phantom

\section*{B. 3PA-PET complexity}

Hardware resources used by the 3PA-PET architecture are presented on table II. The main FSM backprojection and the root cache control are shared between the n units of the 3PA-PET architecture, therefore the cost of an additional unit is only 800 slices. Sizes of leaf and root caches are respectively 2 kB and 18 kB. Hence, 9 backprojection units fit in a Xilinx Virtex 2 Pro VP30 chip.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
1 unit & 4 units & 9 units \\
\hline
3D Backprojection & & \\
\hline
CLB slices & 573 & 1817 & 3924 \\
(4.2\%) & (13.3\%) & (28.6\%) \\
Multipliers & 12 & 48 & 108 \\
(9\%) & (35\%) & (79\%) \\
\hline
3D-AP Cache & & \\
\hline
CLB slices & 672 & 2830 & 4804 \\
(4.9\%) & (20.6\%) & (35.1\%) \\
RAMs & 2 kB & 24 kB & 36 kB \\
(0.6\%) & (7.8\%) & (11.7\%) \\
\hline
3D Backprojection + 3D-AP Cache & & \\
\hline
CLB slices & 1245 & 4637 & 8728 \\
(9.1\%) & (32.9\%) & (63.7\%) \\
\hline
\end{tabular}
\caption{Hardware resources used by the 3PA-PET architecture in parenthesis the percentage of occupation of the Xilinx Virtex 2 Pro VP30 resources}
\end{table}

C. Time performance

The 3PA-PET time performances are compared with STIR and our software VBI backprojection on a desktop PC (Pentium 4 Prescott), a workstation (bi-Xeon dual core) and a GPU (Nvidia GTS88800). The efficiency of the architecture is evaluated by the clock cycles per operation where an operation corresponds to a voxel update. The number of voxel updates is equal to the number of voxels multiplied by the number of segments times the number of angles.

The 3PA-PET cycle efficiency is measured on an Avnet development board. A simulated memory bus is implemented to observe how the 3PA-PET architecture reacts with respect to the memory bandwidth and latency. The results presented in fig. 8 are achieved with one backprojecting unit for segments 0 and +2. As we can expect, segment 0 is backprojected more efficiently than segment +2, as the path into the 3D memory space is bounded on lambda dimension. The cache miss rate stays low (about 0.05\% with a memory latency of 5 cycles and a memory delay of 1 cycle). This robustness to high latency and low bandwidth is due to the high spatial and temporal locality of the application as presented on section 2. It explains why the architecture can potentially be highly parallelized.

On table III, time performance in seconds is normalized to the reconstruction of a 128 * 128 * 63 volume. Indeed, STIR reconstructs a cylindrical FOV (Field of View) of 64²π * 63 voxels. Moreover, to fairly compare our FPGA-based architecture with others technologies, the time measured on a Vortex 2 Pro has been scaled to a Vortex 4. Indeed, the Vortex 4 technology is the same generation as the CPU and GPU used in this study.

The software code of the VBI backprojection is carefully optimized. For instance, time performance is improved by a factor two thanks to an incremental computation of coordinates as done by Kachelriess [4] for 3D cone beam backprojection. Thus the implemented backprojection algorithm is competitive with the one used by
STIR. Both have a reconstruction times about 10 s. Then, this code is parallelized to use the four cores of a bi-Xeon dual core workstation. One thread is associated to the reconstruction of one bloc. One strength of the Pentium 4 and Xeon cores is their wide L2 Cache memory of 2 MBytes.

The Nvidia GPU has 12 vector processors, each one having 8 stream processors. A non incremental code is parallelized to run efficiently on these 12 * 8 stream processors. One thread is associated to one voxel reconstruction. Interpolation is hard-wired and each vector processor is associated with a L1 cache memory. Moreover, the GPU offers a high memory bandwidth \(BW_{\text{mem}}=86.4 \text{ GB/s}\) and uses floating point computation.

![Diagram](image)

**Fig. 8.** Performance of 3PA-PET with one unit for backprojection of segment +2 (on the top) and segment 0 (at the bottom) versus the latency and bandwidth of the memory. Bandwidth is represented by the delay between the delivery of two memory words on the memory bus.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>PE (threads)</th>
<th>Time</th>
<th>Cycles/Op</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop PC : Pentium 4 ((freq. = 3.2 \text{ Gzh}, BW_{\text{mem}} = 6.4 \text{ GB/s}))</td>
<td>STIR 1 (1)</td>
<td>5.74 s</td>
<td>34.5</td>
</tr>
<tr>
<td></td>
<td>VBI-float 1 (1)</td>
<td>5.9 s</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>VBI-float 2 (2)</td>
<td>3.1 s</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>VBI-float 4 (4)</td>
<td>2 s</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>VBI-float 4 (8)</td>
<td>1.67 s</td>
<td>40</td>
</tr>
<tr>
<td>Workstation : bi-Xeon dual core ((freq. = 3 \text{ Gzh}, BW_{\text{mem}} = 10.6 \text{ GB/s}))</td>
<td>STIR 1 (1)</td>
<td>5.74 s</td>
<td>34.5</td>
</tr>
<tr>
<td></td>
<td>VBI-float 1 (1)</td>
<td>5.9 s</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>VBI-float 2 (2)</td>
<td>3.1 s</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>VBI-float 4 (4)</td>
<td>2 s</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>VBI-float 4 (8)</td>
<td>1.67 s</td>
<td>40</td>
</tr>
<tr>
<td>GPU : GTS8800 ((freq. = 1.35 \text{ Gzh}, BW_{\text{mem}} = 86.4 \text{ GB/s}))</td>
<td>VBI-float 96 (192)</td>
<td>0.1 s</td>
<td>25.9</td>
</tr>
<tr>
<td>FPGA : Virtex 2 Pro ((freq. = 35 \text{ Mhz}, BW_{\text{mem}} = 40 \text{ MB/s}))</td>
<td>VBI-fx 1</td>
<td>11.86 s</td>
<td>1.05</td>
</tr>
<tr>
<td></td>
<td>VBI-fx 4</td>
<td>6.37 s</td>
<td>1.8</td>
</tr>
<tr>
<td></td>
<td>VBI-fx 8 (*)</td>
<td>3.89 s</td>
<td>2.2</td>
</tr>
<tr>
<td>FPGA : Virtex 4 ((freq. = 200 \text{ Mhz}, BW_{\text{mem}} = 0.8 \text{ GB/s}))</td>
<td>VBI-fx 1(**)</td>
<td>2.6 s</td>
<td>1.05</td>
</tr>
<tr>
<td></td>
<td>VBI-fx 4 (***)</td>
<td>1.11 s</td>
<td>1.8</td>
</tr>
<tr>
<td></td>
<td>VBI-fx 8 (***)</td>
<td>0.68 s</td>
<td>2.2</td>
</tr>
</tbody>
</table>

(*) simulation  
(**) 35 Mhz results scaled to 200 Mhz

**TABLE III**

Comparison Time Performance for the 3D PET backprojection of a 128*128*63 volume from a Siemens HR+ sinogram (5 segments, span 9, 96 angles of projection). FPGA measures are done with a simulated bus (memory latency of 5 cycles and memory bandwidth of 1 cycle/memory word).

**D. Discussion**

On one hand, GPU seems to be the best adapted solution to speed up 3D backprojection with a final reconstruction time of 100 ms and a computational throughput of 0.27 C/Op. This good performance is mainly due to its high degree of parallelization and its high...
memory bandwidth. Parallelization works also well on the bi-Xeon processor. From 36 cycles per operation with one core, the Xeon reaches an efficiency of 10 cycles per operation with 4 cores. As a consequence, the workstation is six times faster than the desktop PC. Our FPGA-based architecture on a Virtex 4 is 7 times slower than the GPU but it remains as in our 2D study [11] a faster solution than classical CPUs: 4.5 times faster than a mono-Xeon dual core and 15 times faster than a Pentium 4.

On the other hand, 3PA-PET is the best cycle efficient architecture per processing element (PE). Compared to the GPU and the Xeon, it is respectively 13 and 20 times more efficient. Indeed, 3PA-PET reaches a computational throughput close to one cycle per PE: 1.05 with one unit, 1.8 with 4 units and 2.2 with 8 units. Besides, as the computational resources are efficiently used, its lower power consumption could become an advantageous solution in another context than medical reconstruction.

V. Conclusion & Future work

The pipelined and parallelized architecture has efficiently sped up 3D PET backprojection without a significant loss of accuracy. The pipelines are seldomly stalled because the 3D Predictive and Adaptative cache overcomes the limitations due to the latency and bandwidth of the external memory. The encouraging results presented in this paper let us hope to better exploit the memory bandwidth and therefore to run efficiently more than 8 units of backprojection in parallel. Afterwards, we could reach a second level of parallelism by increasing the processing resources and the memory bandwidth. This will be done thanks to a board with 7 Virtex 4 (6 processing units and one controller unit), each associated with its own external memory.

VI. Acknowledgement

We would like to thank Anthonin Reilhac from the CERMEP laboratory at Lyon for his help on PET data.

References

SoPC for 3D Point Rendering

Lars Middendorf, Felix Mühlbauer, Christophe Bobda, Georg Umlauf
Department of Computer Science
University of Kaiserslautern
Gottlieb-Daimler-Str. 48
67653 Kaiserslautern, Germany
muehlbauer,bobda,umlauf@informatik.uni-kl.de

Abstract—Real-time 3D visualization of objects or information becomes increasingly important in everyday life e.g. in cellular phones or mobile systems. Care should be taken in the design and implementation of 3D rendering in such embedded devices like handhelds in order to meet the performance requirement, while maintaining power consumption low. In this work, the design and implementation of a vertex shader on a reconfigurable hardware is presented. The main focus is placed on the efficient hardware/software partitioning of the vertex shader computation, in order to maximize the performance while maintaining a high flexibility. The resulting solution must be compatible to existing vertex shaders in order to allow the large amount of existing program to be easily ported to our platform. A prototype consisting of a PowerPC, peripherals and some custom hardware modules is realized on an FPGA-board. The implementation of a point rendering shows considerable speed up compared to a pure software solution.

I. INTRODUCTION

Rendering of three-dimensional objects in real-time requires much arithmetic performance. This is a problem for embedded systems that are running at low clock speed and often lacks dedicated hardware processing modules like a floating point unit (FPU). In desktop computers, the expensive arithmetic computations related to the rendering of 3D objects are done by specialized stream processing hardware in video cards. Those cards are programmable using small programs called shaders. The execution of shaders is the main difference to the CPU. A new instance of the program is invoked for every primitive, vertex or pixel. There are three slightly different types of shaders for these elements. Each instance can be executed independently of the others because there is no communication possible between instances of the same type. This is advantageous when designing the hardware, because it allows the execution of an arbitrary number of instances in parallel, in order to gain the maximum computation speed. Also, pipeline technique can be used to allow some threads to feed parameters to other threads waiting for them in the pipeline. As a result the available hardware can be used more efficiently.

We developed a hardware accelerator for executing vertex shaders that is in particular useful for embedded systems, because it uses very few hardware resources, in this case FPGA slices. It is a kind of coprocessor that is directly connected to the CPU by a fast bus. The main program running on the CPU loads the shader code and all inputs into this coprocessor. While the coprocessor is running the shader, the main program accomplishes further computations in parallel until the results can be read back.

It is important to minimize the resource usage of the hardware, because the number of available slices in a FGPA is very limited. The clock speed is also very low and we have to maximize the utilization of sub-components in all cycles. The scheduling of the threads is therefore pre-calculated and stored as part of the shader code. Hence, the control-logic consists only of the program counter and a few multiplexers, that route the data flow, thus enough space is left on the device to implement floating point calculations. The multiplexer configuration is stored in a table with one row for each cycle. In this work a shader converter that generates the control table from a Direct3D9[1] vertex shader was developed. The shader converter performs the scheduling of all operation on the generated hardware unit, analyzes and optimizes the data flow and maps the calculations to operations of our ALU. Currently we can execute four threads on the ALU in parallel. This allows a speed-up of factor four compare to the software implementation of the shader.

The rest of the work is organized as follows: Section II provides the basics of vertex shader while section III introduces some work related to custom implementation of vertex shaders. Section IV-B explain our implementation. A naive co-design approach is first explained, followed by a more efficient one. Also the design decisions for the hardware software partitioning are explained. The results obtained on a prototype implemented on a Xilinx Virtex 4 evaluation platform are given in VI. Finally section VII concludes the work and provides some indication on the future directions.

II. VERTEX SHADER

In a rendering process, each 3D-point, also called vertex must traversed a set of computing stations, the render pipeline until the final step when it can be drawn on the display. The stations consist of a coordinate transformation (object → world, world → camera) stage, an illumination, a clipping, a projection, a scaling to screen resolution step, and finally the step to approximate the float values to integer values is performed.

To simplify spatial calculations in computer graphics homogeneous coordinates \((x, y, z, w)\) are used. Transformations like translations, rotations, scalings, shearings, projections, etc. can be mapped to \(4 \times 4\) matrices and can be combined to only one matrix by multiplying the corresponding matrices. Thus, a
transformation of a vertex by a certain list of transformations can be realized by one matrix-vector-multiplication.

For illumination calculations the dot product (scalar product) is very important, because the light intensity depends on the angle between surface normals and light sources. Normals can be transformed similar to vertices which is advantageous when filling the surface normals together with the vertices of the scene into the render pipeline to speed up processing.

In conclusion, each stage of the render pipeline executes mainly matrix and vector operations using all values which are involved like coordinates, surface normals, surface attributes, lightning parameters, etc.

There are several vertex shader versions for different hardware. We focus on implementing a subset of the smallest version 1.1 [1]. All versions use a RISC instruction set. Each instruction can read from up to three registers and write to one result register. Almost every register is 128 bits wide and stores four 32 bit floating point numbers. Hence most of the commands operate on vectors with four components. The individual components of a vector can be reordered and duplicated while reading from a register and there is a write mask for every component of the result register. This improves flexibility and allows optimizing calculations. It is for example possible to get a cross product with two instructions. Because our hardware is scalar-based, all write and swizzle-masks are free and should be used to improve performance. The shader converter analyzes the data flow for each individual component and if a result is not used, the calculation is removed on a per-component basis.

There are global and local registers. Each instance of the shader has its own set of local registers consisting of temporary and output registers. It is not allowed to write global registers which makes parallelizing possible, because there is no synchronization required and no operation in one thread depends on results calculated in another thread.

Vertex Shader 1.1 does not support jumps or subroutines. A detailed description of the instruction set can be found in the DirectX SDK [1]. Table I shows some vertex shader commands and a minimal shader that reads the vertex position and performs a vector-matrix multiplication to calculate the projective position of the vertex.

### III. RELATED WORK

Lots of work has been done already in the domain of accelerating graphics applications utilizing FPGAs in general. Some of them are listed in [2], [3]. Often a combination of a desktop computer and a FPGA builds the computing unit. The need of 3D graphics visualization in embedded systems is still growing with the increasing spreading of mobile multimedia systems in everyday life like cellular phones and PDAs. Even the MPEG H.264 standard which is the video coding for next-generation multimedia involves rendering of 2D and 3D deformable mesh geometry [4].

The still continuing miniaturization has led to highly integrated chips and finally to so-called SoCs (system on chip). Here all components and peripherals are placed on a single chip like processors, hardware accelerators, bus and peripheral controllers and allow a PLB (printed circuit board) independent redesign or update of applications which is an important advantage.

Sohn et al. introduced a multimedia co-processor for mobile applications using an ARM-10 processor and fixed-point arithmetic [5]. The company Bitboys developed an vector graphics processor targeting for high-end multimedia cellular phones which is available as IP core for SoCs and is used in several devices for process SVG and OpenVG object data [6].

We are particular interested in a system in which custom hardware can cohabit with software. Also, the system should provide enough flexibility to ease the redesign and also allow a run-time adaptation, while maintaining the performance high and the power consumption low. The next sections explain our solution to this problem.

### IV. IMPLEMENTATION

Our target platform in this project was a Xilinx Virtex 4 evaluation board featuring a Virtex4-FX12 FPGA. This FPGA contains an embedded PowerPC 405 processor, on-chip memory (BlockRAM) and miscellaneous DSP functions[7]. We use the external DDR-RAM as video frame buffer to store 3D object data. A simple system on chip with DDR-RAM controller, VGA out module and system bus needs already half of the available slices of the FPGA. Because, floating point hardware modules are expensive, we tried to avoid or reuse them as much as possible. Thus, an efficient design considering speed and chip area has to be found.

#### A. Basic Design

In a first design a field of 32 registers combined with an adder and a multiplier unit and an instruction memory was drawn up. This co-processor is directly connected to the main processor via the FCM bus, which allows to extend the native PowerPC instruction set with custom instructions that are executed by a user-defined configurable hardware accelerator.

The data words read from the BlockRAM (see Figure 1) specify which registers supply the input values for the

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>addition</td>
</tr>
<tr>
<td>sub</td>
<td>subtraction</td>
</tr>
<tr>
<td>dp3</td>
<td>3D dot product</td>
</tr>
<tr>
<td>dp4</td>
<td>4D dot product</td>
</tr>
<tr>
<td>mad</td>
<td>multiplication &amp; addition</td>
</tr>
<tr>
<td>mul</td>
<td>multiplication</td>
</tr>
<tr>
<td>rcp</td>
<td>reciprocal</td>
</tr>
<tr>
<td>rsq</td>
<td>inverse square root</td>
</tr>
</tbody>
</table>

Table I: Some vertex shader commands and an example
arithmetic units and to which register each result should be written back.

The implementation of this design is very straightforward and also expandable for further operations like division or square root. So the two (or more) operations are executed simultaneously. Unfortunately, the design needed huge multiplexers and address decoders leading to very high resource consumption. The complete chip area was filled by this first version of the design.

\[ x_{n+1} = \frac{1}{2}x_n(3 - x_0x_n^2) \]
the arithmetic sub-units. Multiplications by -1 can be handled directly at the program start, while considering the delays caused by (ALU) operations and long processing chains are moved to output values. Now, vector operations are mapped to scalar points out the dependencies between input, provisional and resulting code, a data flow graph is built up, which is compiled with DirectX SDK[1]. Using the syntax analysis available in multiplication with the inverse, can be processed directly by the ALU. Sometimes algebraic conversions can help to map calculations to the optimized dot product (e.g. \( \frac{a}{b} \)).

### Table II

<table>
<thead>
<tr>
<th>command</th>
<th>result</th>
<th>delay</th>
<th>notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>dot4</td>
<td>( a \cdot b + c \cdot d + e \cdot f + g \cdot h )</td>
<td>19</td>
<td>4D dot product</td>
</tr>
<tr>
<td>dot2</td>
<td>( a \cdot b + c \cdot d )</td>
<td>14</td>
<td>2D dot product</td>
</tr>
<tr>
<td>mult4</td>
<td>( a \cdot b \cdot c \cdot d )</td>
<td>18</td>
<td>multiplication</td>
</tr>
<tr>
<td>mult2</td>
<td>( a \cdot b )</td>
<td>9</td>
<td>multiplication</td>
</tr>
<tr>
<td>div</td>
<td>( a/b )</td>
<td>27</td>
<td>division</td>
</tr>
<tr>
<td>rsq</td>
<td>( 0x5F3759DF -(a \gg 1) )</td>
<td>2</td>
<td>start value for newton iteration of square root</td>
</tr>
<tr>
<td>slt</td>
<td>if ((a \cdot b + c \cdot d &lt; 0)) then ((e \cdot f) ) else ((g \cdot h))</td>
<td>14</td>
<td>input values are needed after 5 clock ticks again</td>
</tr>
<tr>
<td>int2float</td>
<td>( f\text{float}(a) )</td>
<td>6</td>
<td>converts integer to float</td>
</tr>
<tr>
<td>float2int</td>
<td>( int(a) )</td>
<td>6</td>
<td>converts float to integer</td>
</tr>
</tbody>
</table>

### Table III

#### INSTRUCTION FORMAT

<table>
<thead>
<tr>
<th>Word</th>
<th>0/8</th>
<th>9/17</th>
<th>18/26</th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmd0</td>
<td>src0</td>
<td>src1</td>
<td>dst</td>
<td>we</td>
<td>inv0</td>
<td>inv1</td>
<td>inv2</td>
<td>inv3</td>
</tr>
<tr>
<td>cmd1</td>
<td>src2</td>
<td>src3</td>
<td>out</td>
<td>oe</td>
<td>inv4</td>
<td>inv5</td>
<td>inv6</td>
<td>inv7</td>
</tr>
<tr>
<td>cmd2</td>
<td>src4</td>
<td>src5</td>
<td>div</td>
<td>rsq</td>
<td>slt</td>
<td>mult2</td>
<td>dot2</td>
<td></td>
</tr>
<tr>
<td>cmd3</td>
<td>src6</td>
<td>src7</td>
<td>-</td>
<td>(2i)</td>
<td>(2f)</td>
<td>mult4</td>
<td>dot4</td>
<td>-</td>
</tr>
</tbody>
</table>

### C. Vertex Shader Converter

To generate the ALU opcodes given a vertex shader program is compiled with DirectX SDK[1]. Using the syntax analysis for the resulting code a data flow graph is build up, which points out the dependencies between input, provisional and output values. Now, vector operations are mapped to scalar (ALU) operations and long processing chains are moved to the program start, while considering the delays caused by the arithmetic sub-units. Multiplications by -1 can be handled directly by the ALU input stage. Diversions which are not available in Vertex Shader 1.1 and therefore are realized by multiplication with the inverse, can be processed directly by the ALU. Sometimes algebraic conversions can help to map calculations to the optimized dot product (e.g. \((a + b)c \rightarrow ac + bc\)). Also the usage of the \texttt{slt} command is more practical.

### V. GENERATING OPCODES

#### A. Overview

Writing the shader instructions is time-consuming and error-prone, because the format is optimized for simple decoding. Even when using an assembler that translates mnemonics into their corresponding binary format, it would be necessary to pay attention to the latency of the opcodes and the dependence between them. Therefore we developed a shader converter that reads a binary compiled Direct3D9[1] vertex shader and optimizes it for the ALU. Unfortunately it has not been possible to support all instructions, because of the limitations of the FPGA. The Direct3D9 vertex shader format has been chosen, because is well documented[9] and there are several tools like compilers, assembler and linker available[1]. The display driver gets the shader also in this format. Another possibility would have been to compile the shader directly from source code. The shader converter actually decompiles the input shader into a data flow graph and so it would have been only insignificant more complex to generate it from a simple C-style language. But by using a standard format, it is possible to develop and test shaders with Direct3D and to execute already existing shaders.

#### B. Compiling the Shader

The shader is originally written in HLSL (High Level Shading Language), which is very similar to C, except that it has build in support for special types like vectors and texture samplers. The HLSL compiler of the DirectX SDK is then used to compile it into its binary format. The compiler is available both as a standalone command line tool and as a DLL (dynamic link library), that can be used from other applications. Using this approach, the shader converter can directly process HLSL source code.

#### C. Creating the Graph

First the binary Direct3D vertex shader code is read and executed symbolically. The instructions are interpreted, but the calculations are performed on variables instead of real values. Each register stores a node of data flow graph that describes the calculations that were applied to this register. In the beginning there is usually only one node in every register, that is labeled with the register’s type and index. When two registers are added, a new “+”-node is created with both registers as inputs. It represents the result of the calculation and is assigned to the result register for this instruction. After repeating this step for every instruction, each register contains an expression tree for its value at the end of the shader executing. There are of course node that belong to more than one tree, because of common sub-expressions. The union of all these trees is the data flow graph of the shader. It contains less information than the original instruction list, because it describes only the dependencies and not the exact order of operations. So one data flow graph is usually equivalent to a large number of programs which allows the shader converter to select the program most suitable for the ALU.

#### D. Optimizations

Except eliminating common subexpression, the shader converter does mainly architecture-dependent optimizations, because the Direct3D shader compiler already outputs code, that
is very optimized, but written for a more abstract execution model. It is preferable to merge several simple command into one node, because almost every instruction takes one cycle. The ALU can multiply the inputs of every calculation by (-1) for free and so these nodes are pushed against the flow direction of the graph. This means that for an addition instead of only the output both inputs are multiplied by (-1). Although the number of nodes increases, the number of cycles remains does not and there is a higher probability that the resulting nodes can be combined into a larger calculation. There is a greedy algorithm starting at the output nodes that collects additions and multiplications as much as it is possible and creates the smallest equivalent dot2, dot4, mult2 or mult4 node (see Table II). Up to four additions are converted into a dot product. If a summand is the result of a product, the multiplication is also included, otherwise it is simply multiplied by 1, which does not create additional costs, because the multiplication is always done as part of the dot product (see Figure 4). The ALU has got a very expensive full divider, but the vertex shader uses only reciprocals. So multiplication and reciprocals are also combined. Also the distributive law is used to convert the expression \((a + b)c\) into \(ac + bc\) which seems to be more expensive, but takes one instead of two instructions, because it is implemented as the dot2 instruction.

**E. Generating the Instruction Table**

The instruction table is generated recursively. Each instruction is assigned to start at the first free cycle after all of its
inputs are calculated. Because the instruction bits are used directly to control the multiplexers, one logical instruction is distributed over multiply rows in the table and the parameters are delayed according to the internal pipeline of the ALU. The eight input addresses are written in the line the instruction starts, the (−1) inversion bits are written into the next line and the opcode selecting the correct output is inserted after the corresponding number of cycles (see Table II). The minimal count of cycles required to calculate the shader is bound by the length of longest path in the data flow graph. A simple optimization is to generate this path first so that optimally the first and the last instruction belongs to this path and the others can fill the gaps between caused by long latencies. Unfortunately this decreases the total length only by very few cycles. Because of the high latency there are still often large gaps of empty rows in the table where no new operations can be started and the ALU simply waits for intermediate results. To further reduce these gaps, there are always four instances of the shader executed in parallel. This is achieved by simply duplicating the nodes two times after creating the graph. Now even the large latency of the dot product (16 cycles) does not lead to empty rows in the table if there are at least four independent dot products per instance which is very common for matrix-vector multiplication. The hardware does not know about multiple threads but its simplicity allows to improve the performance significantly by generating optimal instructions.

VI. RESULTS

The most important disadvantage of this implementation is the limitation to one result per cycle. This means that a matrix-vector multiplication takes at least four cycles. The high latency of certain operations is not really a problem, but it is different for almost every instruction, so that it can be difficult or impossible to fully load the ALU. Because of the strict requirements, not all commands could be directly implemented in hardware. For example a single multiply instruction that multiplies two vectors component wise can take up to four cycles. But usually the instruction is part of a more complex calculation and the shader converter can merge the previous and following calculations so that the whole block may be mapped to four larger instructions that also take four cycles.

On the other hand the ALU can calculate a 4D dot product every cycle. It has been chosen to be specially optimized, because it is a very important and often used operation. Even the most simple but useful shader does a vector-matrix multiplication that can be calculated using four 4D dot products. A large number of other instructions using only multiplications and additions can be reordered and mapped to dot products. But the most important reason for the dot product is the fact that it has only one scalar result and fits perfectly to the limited register array that can only write one result value. It is also slightly cheaper than a parallel component wise multiplication and addition because it only needs three addition modules.

There is the possibility to output directly the intermediate 2D dot product and to skip the last addition for a lower latency. This can be useful when interpolating between two vectors. The additional multiplier outside of the dot product gives the ability to multiply four floating-point numbers in one cycle. This is important for calculating multi-linear functions that could otherwise only be achieved by a large number of cumbersome repeated high latency dot products.

This design cannot be enhanced any further. Adding another instruction type extends the multiplexer at the output of the ALU and leads to increased complexity. The timing constraints will not be met and the required clock speed of 100MHz cannot be achieved.

The new hardware component has been tested with a mesh viewer. The viewer is running on the PowerPC CPU, but the vertex shader can be calculated either in software or hardware to compare the performance. The triangles are not filled and the mesh is rendered as a point model (bunny model from [10]). We want to measure the speed of the vertex calculations and in a real application the expensive triangle filling would also be done in hardware. For each configuration 100 frames have been rendered several times with different point counts. The time spans are very precisely measured directly on the board with a special 64 bit register that counts the CPU cycles. The vertex shader consists of six instructions that calculate the coordinates and the lighting from a directional light source. Comparing the results both for software and hardware it is obvious that the hardware accelerated version is much faster, see Table IV and Figure 5. The last column of Table IV contains the ratio between software and hardware performance. The ratio is higher when rendering more vertices, because there is a fixed overhead per frame for clearing the color and depth buffers.

<table>
<thead>
<tr>
<th>Vertex Count</th>
<th>Hardware</th>
<th>Software</th>
<th>Software/Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000</td>
<td>3.607s</td>
<td>13.32s</td>
<td>3.694</td>
</tr>
<tr>
<td>10000</td>
<td>4.832s</td>
<td>24.25s</td>
<td>5.019</td>
</tr>
<tr>
<td>15000</td>
<td>6.104s</td>
<td>35.22s</td>
<td>5.770</td>
</tr>
<tr>
<td>20000</td>
<td>7.409s</td>
<td>46.26s</td>
<td>6.244</td>
</tr>
</tbody>
</table>

Table IV

Performance results for 100 frames [sec].

Figure 5. Performance results: Time in seconds for 100 frames and varying vertex counts

VII. CONCLUSION

We have introduced a hardware accelerator for a vertex shader. Our design consumes few resources (slices) on FPGA,
while supporting almost all functions of the common language for such data processing *Vertex Shader 1.1*. Compared to a software only version a significant speed advantage could be achieved. This application is suitable for the domain of embedded systems.

REFERENCES


GAIA Video Processing Algorithms: Design Process and Prototyping Activities

S. Provost(1), M. Le Roy(1), B. Mamdy(1), G. Flandin(1)

(1) EADS Astrium, 31 rue des cosmonautes, 31402 Toulouse Cedex 4, France
Email: severin.provost@astrium.eads.net; bruno.mamdy@astrium.eads.net

Abstract—GAIA is an ambitious mission of the European Space Agency (ESA) whose spacecraft is developed by EADS Astrium. Its objective is to create the largest and most precise three dimensional chart of our Galaxy by providing unprecedented positional and radial velocity measurements for about one billion stars in our Galaxy and throughout the Local Group.

The Video Processing Algorithms (VPA), embedded in the Video Processing Unit (VPU), are part of the payload, dedicated to process the raw data issued from the Focal Plane Assembly, and in charge of controlling it. VPA play a major role in terms of data reduction for GAIA.

Though the phase of systematic and formal validation is only beginning, these functional performances and implementation aspects have been taken into account since the very beginning of the VPA design elaboration. The level of complexity and the numerous constraints to be handled make the VPA designing a pioneering study in the field of real-time image processing for space applications. Since the early study phase, the process followed an iterative scheme, largely based on prototyping activities of different natures.

I. INTRODUCTION

A. GAIA mission

GAIA spacecraft will map about one billion stars in our Galaxy and throughout the Local Group with unprecedented positional (10 μarcsec typical) and radial velocity measurements. Multi-color photometry will also be provided as an observation product. Additionally, near earth objects and extended objects will also be observed.

GAIA spacecraft will be located at 1.5 million km from the Earth around Lagrange point L2. The satellite is rotating around the spin axis (Xs in Fig. 1). It is composed of two instruments that image two different areas of the sky on the same focal plane. Their lines of sight are perpendicular to the spin axis of the satellite and separated with a Basic Angle of ~106°. The spin axis follows a precession around the Sun/Satellite axis with a 45° precession angle. Finally, its rotation with the Sun/Earth axis (365 days) ensures global sky coverage.

During a five years observation period, every star is statistically observed ~80 times.

More information about GAIA mission can be found in [1].

B. Payload module

The instrument is a toroidal full SiC structure. The two beams are combined on a single focal plane from where the data are collected and processed by the Video Processing Unit.
Fig. 3. GAIA Focal Plane Assembly

The FPA (Focal Plane Assembly) is composed of seven rows. Each row is a succession of up to 17 detectors (1966 across-scan x 4500 along-scan pixels each) working on a TDI mode and seeing the complete transit of stars (see Fig. 3). When a star enters the first (resp. the second) instrument field of view, it is first detected in the SM1 (resp. SM2) Sky Mapper (SM) detector. Then it is tracked in the AF1-9 detectors (that observe a superposition of both fields of view) whose purpose is to provide astrometric data. It is also tracked in the BP/RP detectors that provide photometric data of the stars. Finally, the RVS detectors provide very accurate spectra of stars.

All those processing tasks (detection, tracking, and packetisation of acquisition data) are to be performed in real-time by the VPA.

C. VPU functional interfaces

An overview of VPU interfaces is provided on Fig. 4. The VPU has the main functional interfaces with the following equipments:

- The FPA (Focal Plane Assembly). Samples are commanded to the Interconnection Module of the FPA at every TDI. Other VPU commands have an impact on the conditions of acquisition (e.g. command of gates to reduce the signal of bright stars).
- The CDMU (Central Data Management Unit). Satellite velocities are provided from the CDMU to the VPU, so as to allow the VPU to propagate the windows in the focal plane. In return, the VPU evaluates the speed of stars (by comparing positions in SM and AF1) and sends the speed measurements towards the CDMU.
- The PDHU (Payload Data Handling Unit). The PDHU is in charge of star packets storage. Star packets are produced by the VPU from the focal plane raw data and stored in files after compression. Several files are available within the PDHU to store the start packets, the repartition depending on the star packet ‘priority tag’. This allows dispatching the data in real time, as well as prioritising the star packets downlink. Basically the priority is based on the star magnitude.

![Fig. 4. Overview of VPU interfaces](image-url)
D. VPA implementation principles

VPA include the main processing tasks to be implemented in the VPU. First evaluations of the computing requirements lead to a need for more than 1.5 Gips without margins. Such a computing power being by far not available for space applications, it has been decided to early design VPA with the objective to optimize the HW/SW partitioning in order to ensure the feasibility using flight compatible technologies while keeping the design flexible as far as possible.

The HW part of the processing consists in algorithms implemented in Actel FPGA. In order to confirm both timing and logic consumption, an extensive prototyping up to the VHDL level and then the FPGA synthesis has been performed. This has demonstrated that the expected performances where met and that the algorithms were performing as expected.

The SW part has been implemented for a PowerPC processor. Evaluations have shown that a board in the class of the Maxwell SCS 750 was able to cope with the requirements.

II. OVERVIEW OF VIDEO PROCESSING EMBEDDED ALGORITHMS

A. Functional architecture of the VPA

The functional architecture of the VPA is provided in Fig. 5. More details on the algorithmic processing can be found in [2].

SM1 and SM2 input data are separately processed. First a pre-processing algorithm allows correcting bad pixels and applying a linear correction for the gain and bias over the samples. Gain and bias values are look-up tables, which may be uploaded from the ground. Then the detection, applied on SM1 and SM2 samples, provides detected sky objects and discards non-sky objects (e.g. galactic cosmic rays or solar protons impacting the detectors) or too faint objects. Priorities are then assigned to objects before merging and sorting both SM object flows. Priorities are typically based on object magnitude (magnitude being estimated on-board by the VPU).

Fig. 5. Functional architecture of VPA
Then the **resource allocation** determines, from the objects list, which one will be read in AF1. Resource allocation process is needed because only a restricted (and constant) number of samples shall be read from each CCD. Resource allocation process optimizes the priorities of objects by buffering the objects on a given TDI horizon, and allocating resources by decreasing order of priorities.

The **confirmation** is applied only in AF1. Confirmation step allows to discard the objects that may have been wrongly detected in SM (e.g. cosmic rays). This allows to reduce to minimum the number of useless star packets. AF1 data processing also allows the detection of SMO (Suspected Moving Object, i.e. object with proper motion), and the computation of star speed measurement. Star speed is evaluated by comparing the predicted and actual position of stars in AF1 CCD.

After confirmation, **commands** are elaborated in order to track the confirmed stars in further CCD (AF2-9), then BP-RP, then RVS1-3. This tracking accounts for the real-time integration of AOCS speed updates sent by the CDMU.

From RP observations, a new priority, which accounts for magnitude of objects in the wavelength range of RVS, is assigned to objects. 3 instances of resource allocation process are then called in the whole processing chain: Astro resource allocation, BP-RP resource allocation (based on Gaia G magnitude estimate) and RVS resource allocation (based on Gaia GRVS magnitude estimate).

The **Input / Output data management** consists in preparing commands to the focal plane, assembling windows from read samples and constituting star packets (gathering all the data associated to a star) before compression and storage in the Payload Data Handling Unit (PDHU).

**Fig. 6. Detection / allocation / confirmation principles**

B. **HW and SW sharing of VPA tasks**

HW and SW sharing of VPA is represented in Fig. 5 (processing tasks marked in grey are implemented in HW).

There are three main reasons for which a function cannot be implemented in software:

- It requires more processing power than available in a space compatible processor,
- It requires the respect of a shorter latency than allowed in a space compatible processor,
- It interfaces with another function implemented in hardware, and the presence of a hardware/software interface is not compatible with the performance.

The following algorithms have been selected and optimised to be implemented in hardware instead of software:

- Most of the systematic processing of SM data: (pre-processing of SM1 and SM2 samples, detection of faint objects, and extremities of saturated stars, pre-selection of SM objects),
- Sending of FPA commands / reception of the corresponding samples.

All the other algorithms have been defined in order to remain implementable in software. The main advantage of software processing is of course flexibility, since software patches (during VPU development or during in-orbit life of the satellite) may allow the adaptation to evolving processing needs.

The SCS750 board requires periodic resynchronisation of its 3 microprocessor (functioning in majority vote), each resynchronization suspending the software execution during almost 1 ms, i.e. approximately one TDI. As a consequence of this, the SW processing will not be able to respect a latency in the order of one TDI. The hardware algorithms are working almost synchronously with respect to the TDI cycle: in each cycle, they are reading and/or writing data at offsets in the data buffers that depend only on the current TDI cycle number. The software algorithms are trying to do the same, but are allowed to run late in their processing by several TDI cycles. This then allows to cope for the periodic resynchronization of the processors and also allows the smoothing of the processing load.

The maximum delay of SW algorithms is constrained by the maximum functional latency: data received at a given TDI shall be processed within a limited numbers of TDI since decisions are to be taken depending on the processed data (e.g. application of gates, application of charge injection at the inlet of the CCD, positioning of the FPA sample commands, refer to Fig. 7). This yet allows typically the SW processing to run late by more than 100 TDI (and thus to be “smoothed” on more than 100 TDI).
Despite new data become available at each cycle, the algorithms have hundreds or thousands cycle available to execute.

| (1) Descriptors of objects detected in ASM1 received | Time available to compute for charge injection, gate and readout commands |
| (2) Descriptors of objects detected in ASM2 received |
| (3) Deadline for sending AF1 charge injection command |
| (4) Deadline for sending the first AF1 gate command |
| (5) Deadline for sending the AF1 readout command |
| (6) Deadline for sending AF2 charge injection command |

**Fig. 7. Example of functional latency constraints**

### III. DESIGN PROCESS

Space environment is a key point in the design of the VPA. Failure is unacceptable in space embedded systems, thus only space compatible electronics, and particularly processors, are available. Moreover, a special care is required for the HW part of the system, which requires serious validation since it cannot be updated after being put under real operational conditions.

In this context, design of the GAIA video processing unit is very challenging. Scientific needs are very constraining and induce a high level of complexity in algorithms design, while a non-extensible computing load forces us to stay in a “reasonable” range of possibilities. Most of the challenge lies in defining what “reasonable” means.

Complexity and a high level of coupling between the different algorithmic tasks, between the algorithms and architecture design, make the problem very specific. Feasibility cannot actually be demonstrated without evaluating future code optimisations. The potential gain of these optimisations in terms of CPU load is itself dependant on general algorithms organisation and data formats. Design must therefore be considered as a whole, every aspect of the study having to be taken into account since the beginning, in an interdependence scheme.

The VPA have to deal with the following constraints:
- Feasibility, which is strictly non-negotiable,
- Scientific need, which is negotiable only in a limited range,
- Flexibility, a major constraint, which must be understood in the following way: at any time from early study to operational phase, we must dispose for strong actuators, able to take new elements into account, and adapt VPA in order to preserve performances, with a minimal effort. New elements can include results from different stages of prototyping, revealing critical points, necessary updates of any part of the design, evolutions of the expressed scientific need, etc,
- Robustness, since complexity induces a lot of hypothesis on the operating conditions, which may be partially subject to evolve,
- Testability.

Each of these constraints must be kept in mind along the design phase, the process becoming a kind of arbitration between these, punctuated by specific analysis and prototyping. Optimality is a constant concern in this process, unnecessary complications being systematically avoided.

Rather than a straight process, going from a general approach to detailed definition, we could describe an iterative process, with the following specificities:
- Each step requires nearly the same level of definition: general architecture, data formats, algorithmic details. However a certain level of redundancy allows some time-saving projections in the early phase,
- Prototyping is carried out further on at each step, representativity level being increased,
- Information is collected from new definitions, interactions, formats, prototyping, performances, and re-injected as an entry for next step.
- The process converges to the final system definition, with associated prototypes and validation elements.

As it could be expected, this process is largely based on prototyping activities.

### IV. PROTOTYPING ACTIVITIES

Previous studies have demonstrated the need for a simulation of the complete chain in order to evaluate the performances of the specified algorithms. This has been particularly true concerning the GAIA VPA definition, and prototyping has been put in the heart of the design process.

#### 4. Functional prototyping

In order to manage and monitor the functional performances of the algorithmic chain, a functional prototype has been developed and maintained from the very early phase to the final definition. This has been coded in Matlab, since such a high-level language was necessary to provide flexibility and reactivity. Particularly on the early phase, a wide range of algorithmic solutions have been tried in a short time
interval. Results from the scientific performances side obtained directly by this way were of crucial impact on many early trade-off.

The prototype and associated evaluation tools have been maintained all along the study. Any possible update could then be quickly characterized in scientific performances terms.

The Matlab implementation of a number of HW tasks was completed by a SystemC implementation, in order to define and check for detailed HW data types, thus providing functional performances results and entries to HW implementation prototype.

B. VPA-RTP

A simulator which is representative of the HW/SW implementation of VPA has also been developed (VPA-Real Time Prototype, referred to as VPA-RTP), simultaneously to the functional prototyping activities.

The algorithms to be implemented in hardware have been prototyped in Handel-C [5], and then synthesised to assess the hardware resources required and preliminary timing information.

The algorithms to be implemented in software have been developed in C, optimized for real-time execution and ported to a (prototype) SC5750 board from Maxwell technologies [4] based on three PowerPC 750FX operated in triple redundancy, using majority voting to provide robustness e.g. to single events upsets encountered in space. Their execution time has been monitored and analysed at various steps during the study. In addition to global feasibility elements, this provided precious information about time consumption of elementary algorithms, allowing potential feedback on general design.

![Diagram of VPA validation test bench](image)

**Fig. 8. Overview of VPA validation test bench**

As the VPA-RTP will finally be representative of the whole algorithmic chain, both from a functional and implementation point of view, it will be used for final high-level validation campaign.

Moreover, VPA-RTP should be used as a starting point for the development of the future VPU flight software.

V. CONCLUSIONS

Obviously, the performances of the on-board detection chain have a crucial impact on the GAIA mission. Delivering the best scientific results, while staying in the feasibility domain, would be an exact manner for describing the algorithmic design process. Iterative scheme and large prototyping usage imposed themselves as necessary tools, to arbitrate what could be seen as a rough negotiation between functional performances and implementability considerations.

The GAIA VPA design is definitely a major study in the field of real-time image processing designed for a mixed HW and SW implementation, under space application constraints.

ACKNOWLEDGMENT

This study took benefit of the Pyxis developments from the Observatoire de Paris Meudon (ObsPM) [3].

REFERENCES

[1] GAIA ESA Webpage: [http://www.esa.int/science/gaia](http://www.esa.int/science/gaia)
Design and Simulation of High Speed Multi-Processing CMOS Image Sensor

Jérôme Dubois, Dominique Ginjac, Michel Paindavoine
Laboratoire LE2I - UMR CNRS 5158
Université de Bourgogne
21078 Dijon Cedex - FRANCE
E-mail : (jerome.dubois,dginhac,paindav)@u-bourgogne.fr

Abstract—A high speed Analog VLSI Image acquisition and pre-processing system is described in this paper. A 64 × 64 pixel retina is used to extract the magnitude and direction of spatial gradients from images. So, the sensor implements some low-level image processing in a massively parallel strategy in each pixel of the sensor. Spatial gradients, various convolutions as Sobel filter or Laplacian are described and implemented on the circuit. The retina implements in a massively parallel way, at pixel level, some various treatments based on a four-quadrants multipliers architecture. Each pixel includes a photodiode, an amplifier, two storage capacitors and an analog arithmetic unit. A maximal output frame rate of about 10 000 frames per second with only image acquisition and 2000 to 5000 frames per second with image processing is achieved in a 0.35 μm standard CMOS process. The retina provides address-event coded output on three asynchronous buses, one output is dedicated to the gradient and both other to the pixel values. A prototype based on this principle, has been designed. Simulation results from Mentor Graphics™ software and AustriaMicrosystem Design kit are presented.

keywords : CMOS Image Sensor, Parallel architecture, High-speed image processing, Analog arithmetic unit.

I. INTRODUCTION

Today, improvements continue to be made in the growing digital imaging world with two main image sensor technologies : the charge coupled devices (CCD) and CMOS sensors. The continuous advances in CMOS technology for processors and DRAMs have made CMOS sensor arrays a viable alternative to the popular CCD sensors. New technologies provide the potential for integrating a significant amount of VLSI electronics onto a single chip, greatly reducing the cost, power consumption and size of the camera [1]–[4]. This advantage is especially important for implementing full image systems requiring significant processing such as digital cameras and computational sensors [5]. Most of the work on complex CMOS systems talks about the integration of sensors providing a processing unit at chip level or column level [6][7]. Indeed, pixel level processing is generally dismissed because pixel sizes are often too large to be of practical use. However, integrating a processing element at each pixel or group of neighbouring pixels seems to be promising. More significantly, employing a processing element per pixel offers the opportunity to achieve massively parallel computations. This benefits traditional high speed image capture [8][9][10] and enables the implementation of several complex applications at standard video rates [11][12].

Vision chips are designed based on the concept that analog VLSI systems with low precision are sufficient for implementing many low level vision algorithms. The precision in analog VLSI systems is affected by many factors, which are not usually controllable. As a result, if the algorithm does not account for these inaccuracies, the processing reliability may be severely affected. Vision chips implement a specific algorithm in a limited silicon area. They are always full custom designed which is a challenging task, known to be time consuming and error prone. One should consider issues from visual processing algorithms to low level circuit design problems, from photo transduction principles to high-level VLSI architectural issues[13].

This paper describes the design and the implementation of a 64 × 64 active pixel sensor with per-pixel programmable processing element. Both the circuit design and layout are targeted for manufacturing in a standard 0.35 μm double-poly quadruple-metal CMOS technology. The main objectives of our design are : (1) to evaluate the speed of the sensor, and, in particular, to reach a 10 000 frames/s rate, (2) to demonstrate a versatile and programmable processing unit at pixel level, (3) to provide an original platform dedicated to embedded image processing.

The rest of this paper is organized as follows : the section II describes the main characteristics of the overall architecture. The section III is dedicated to the description of the operational principle at pixel level in the sensor. In the following, the sections IV and V respectively describe the details of the pixel design and the Analog Arithmetic Unit embedded in each pixel. Finally, some simulation results of high speed image acquisition with processing at pixel level are presented in the last section of this paper.

II. OVERALL ARCHITECTURE

The core circuit is, obviously, the bidimensional array of pixels. This array is organized into a cartesian arrangement of 64×64 pixels and contains 160000 transistors on a 3.5 mm × 3.5 mm die. The full layout of the retina is shown in Figure 1 and the main chip characteristics are listed in Table I. Each pixel contains a photodiode for the light-to-voltage transduction and 38 transistors integrating all the analog circuitry necessary to implement the algorithm described in section III.
This amount of electronics includes a preloading circuit, two “Analog Memory, Amplifier and Multiplexer” ([AM]²) and an “Analog Arithmetic Unit” (A²U) based on a four-quadrant multiplier architecture. The full pixel size is 35 \( \mu \text{m} \times 35 \mu \text{m} \) with a 25 % fill factor.

The left part of the sensor is dedicated to a row decoder for addressing the successive rows of pixels. Below the chip core are the readout circuits with the three asynchronous buses described in section IV. The chip also contains test structures used for detailed characterization of the photodiodes and processing units. The test structures can be seen on the bottom left of the chip.

![Fig. 1. Layout of the Retina in a standard 0.35 \( \mu \text{m} \) CMOS technology](image)

TABLE I

<table>
<thead>
<tr>
<th>Chip Characteristics</th>
<th>0.35( \mu \text{m} ) 4-metal CMOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td>0.35( \mu \text{m} ) 4-metal CMOS</td>
</tr>
<tr>
<td>Array size</td>
<td>64 \times 64</td>
</tr>
<tr>
<td>Chip size</td>
<td>11 mm(^2)</td>
</tr>
<tr>
<td>Number of transistors</td>
<td>160 000</td>
</tr>
<tr>
<td>Number of transistors / pixel</td>
<td>38</td>
</tr>
<tr>
<td>Pixel size</td>
<td>35 ( \mu \text{m} \times 35 \mu \text{m} )</td>
</tr>
<tr>
<td>Sensor Fill Factor</td>
<td>25 %</td>
</tr>
<tr>
<td>Dynamics power consumption</td>
<td>250 mW</td>
</tr>
<tr>
<td>Supply voltage</td>
<td>3.3 V</td>
</tr>
<tr>
<td>Frame rate</td>
<td>10 000 fps</td>
</tr>
</tbody>
</table>

**III. OPERATIONAL PRINCIPLE**

**A. Photodiode Structure**

In a standard design, a pixel includes a photodiode (called PD in our chip) and a processing unit (implemented in the zone called free surface) as shown in Figure 2(a). With the proposed approach, we focus on the optimization of the processing unit mapping. Each pixel integrates image processing based on neighborhood. So, each processing unit must easily have access to the nearest neighbors that’s why an original structure was chosen as shown in Figure 2(b). The major advantage of this structure is the possibility to limit the length of metal interconnection between adjacent pixels and the processing units, contributing to a better fill factor.

![Fig. 2. (a) Photosite classical structure, (b) Considered approach in our system](image)

N-type photodiodes consist of a \( \text{n}^+ \)-type diffusion in a p-type silicon substrate. During the preloaded period, the depletion region is formed in the neighborhood of photodiode cathode. However, considering the layout constraints, choosing a cross shape for the photodiode appears to be not realistic. Finally, a quasi-octagonal structure, shown in Figure 3, was selected because of three main properties:

1) The reserved surface dedicated to the interconnections is about 12 % lower compared to a square shape,
2) The depletion region is more efficient at the edges of the photodiode
3) This shape, based on 45° structures, is technologically realizable by the founder.

![Fig. 3. (a) Photodiode shape, (b) Photodiode layout](image)

**B. Spatial Gradients**

The structure of our processing unit is perfectly adapted to the computation of spatial gradients. The main idea for evaluating these gradients [14] in-situ is based on the definition of the first-order derivative of a 2-D function performed in the vector direction \( \xi \), which can be
expressed as:

\[
\frac{\partial V(x,y)}{\partial \xi} = \frac{\partial^2 V(x,y)}{\partial x^2} \cos(\beta) + \frac{\partial^2 V(x,y)}{\partial y^2} \sin(\beta) \]

(1)

where \( \beta \) is the vector’s angle. A discretization of the equation 1 at the pixel level, according to the Figure 5, would be given by:

\[
\frac{\partial V_i}{\partial \xi} = (V_1 - V_2) \cos(\beta) + (V_2 - V_4) \sin(\beta) \]

(2)

where \( V_i, \ i \in \{1; 4\} \) is the luminance at the pixel \( i \), i.e., the photodiode output. In this way, the local derivative in the direction of vector \( \xi \) is continuously computed as a linear combination of the derivatives along the vertical axis (I_2) and the horizontal axis (I_1). According to the Figure 5, the processing element implemented at the pixel level carries out a linear combination of four adjacent pixels by the four associated weights (coefficients, \( i \in \{1; 4\} \)). To evaluate the equation 3, the following values have to be given to the coefficients:

\[
\begin{pmatrix}
\text{coef1} & \text{coef2} \\
\text{coef3} & \text{coef4}
\end{pmatrix} =
\begin{pmatrix}
\sin(\beta) & \cos(\beta) \\
-\sin(\beta) & -\cos(\beta)
\end{pmatrix}
\]

(4)

**C. Sobel operator**

The structure of our architecture is also adapted to various algorithms based on convolutions using binary masks on a neighborhood of pixels. As example, the Sobel operator is described here. With our chip, the evaluation of the Sobel algorithm leads to the result directly centered on the photosensor and directed along the natural axes of the image according to the Figure 5. In order to compute the mathematical operation, a 3x3 neighborhood is applied on the whole image. To carry out the derived operation discretized in two dimensions, along the horizontal and vertical axes, it is necessary to build two 3x3 matrices called \( h_1 \) and \( h_2 \):

\[
\begin{pmatrix}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1
\end{pmatrix} \quad \begin{pmatrix}
-1 & -2 & -1 \\
0 & 0 & 0 \\
1 & 2 & 1
\end{pmatrix}
\]

(5)

Within the four processing elements numbered from 1 to 4 (see Figure 5), two 2x2 masks are locally applied. According to the equation 5, this allows the evaluation of the following series of operations:

\[
\begin{pmatrix}
I_{11} = I_2 + 2I_4 + I_7 \\
I_{13} = I_1 + 2I_4 + I_7 \\
I_{22} = I_1 + 2I_2 + I_3 \\
I_{24} = I_7 + 2I_8 + I_9
\end{pmatrix}
\]

(6)

with the \( I_{1k} \) and \( I_{2k} \) provided by the processing element \( k \). Then, from these trivial operations, the discrete amplitudes of the derivatives along the vertical axis \( (I_{11}=I_{11},I_{13}) \) and the horizontal axis \( (I_{22}=I_{22}+I_{24}) \) can be computed. All these operations can be carried out in two cycle’s retina. We call cycle’s retina, all operations of treatments carried out at the end of the screen of acquisition.

**D. Second-order detector : Laplacian**

Edge detection based on some second-order derivatives as the Laplacian can also be implemented on our architecture. Unlike spatial gradients previously described, the Laplacian is a scalar (see equation 7) and does not provide any indication about the edge direction.

\[
\delta =
\begin{pmatrix}
0 & 1 & 0 \\
1 & -4 & 1 \\
0 & 1 & 0
\end{pmatrix}
\]

(7)

From this 3x3 mask, the following operation can be extracted according to the principles used previously for the evaluation of the Sobel operator; \( \delta = I_1 + I_2 + I_3 + I_4 - 4I_5 \). This operation can be carried out in only one cycle’s retina.

**IV. ARCHITECTURE AND OPERATION**

The proposed chip is based on a classical architecture widely used in the literature as shown on the Figure 6. The CMOS image sensor consists of an array of pixels that are...
typically selected a row at a time by a row decoder. The pixels are read out to vertical column buses that connect the selected row of pixels to an output multiplexer. The chip includes three asynchronous output buses, the first one is dedicated to the image processing results whereas the two others provides parallel outputs for full high rate acquisition of raw images.

As a classical APS, all reset transistor gates are connected in parallel, so that the whole array is reset when is active the reset line. In order to supervise the integration period (i.e., the time when incident light on each pixel generates a photocurrent that is integrated and stored as a voltage in the photodiode), the global output called Out\_int provides the average incidental illumination of the whole matrix of pixels. So, if the average level of the image is too low, the exposure time may be increased. On the contrary, if the scene is too luminous, the integration period may be reduced.

With these circuits called [AM]\(^2\) (Analog Memory, Amplifier and Multiplexer), the capture sequence can be made in the first memory in parallel with a readout sequence and/or processing sequence of the previous image stored in the second memory (see Figure 8). With this strategy, the frame rate can be increased without reducing the exposure time. Indeed, it is possible to double the speed of acquisition. Simulations of the chip show that frame rates up to 10,000 frames per second can be achieved with a brightness superior to 15,000 luxes.

The chip operates at a single 3.3 V power supply. In each pixel, as seen on Figure 9, the photosensor is a NMOS photodiode associated with a PMOS transistor reset, which represents the first stage of the capture circuit. The pixel array is held in a reset state until the “reset” signal goes high. Then, the photodiode discharges according to incidental luminous flow. This signal is polarized around \(V_{DD}/2\), so the half power supply voltage. While the “read” signal remains high, the analog switch is open and the capacitor \(C_{AM}\) stores the pixel value. The \(C_{AM}\) capacitors are able to store, during the frame capture, the pixel values, either from the switch 1 or the switch 2. The following inverter, polarized on \(V_{DD}/2\), serves as an amplifier of the stored value and provides a level of tension proportional to the incidental illumination to the photosite.
V. ANALOG ARITHMETIC UNIT: A²U

The analog arithmetic unit (A²U) represents the central part of the pixel and includes four multipliers (M1, M2, M3, and M4), as illustrated on the Figures 10 and 11. The four multipliers are all interconnected with a diode-connected load (i.e., a NMOS transistor with gate connected to drain). The operation result at the “node” point is a linear combination of the four adjacent pixels.

Figures 12 and 13 show the simulation results of this multiplier structure with cosine signals as inputs, i.e.,

\[
\text{coef}_1 = A \cdot \cos(2\pi f_1) \quad \text{with} \quad f_1 = 2.5\, \text{kHz} \quad (8)
\]
\[
V_i = B \cdot \cos(2\pi f_2) \quad \text{with} \quad f_2 = 20\, \text{kHz} \quad (9)
\]

In this case, the output Node value can be written as following:

\[
\text{Node} = \frac{A \cdot B}{2} \left[ \cos(2\pi (f_2 - f_1)) + \cos(2\pi (f_2 + f_1)) \right] \quad (10)
\]

The signal’s spectrum, represented on Figure 13 contains two frequencies (17.5 kHz and 22.5 kHz) around the carrier frequency. The residues which appear in the spectrum are known as “inter modulation products”. They are mainly due to the nonlinearity of the structure (around 10 kHz and 30 kHz) and the defects in input pads insulation (at 40 kHz). However, the amplitude of these intermodulations products are significantly lower than the two main frequencies.

Furthermore, in order to obtain the best linearity of the multiplier, the amplitude of the signal \(V_i\) is limited to a range of 0.6-2.6 V. In the full chip, the signal \(V_i\) corresponds to the voltage coming from the pixel and can be easily included in the range described before.

VI. LAYOUT AND PRELIMINARY RESULTS

The layout of a 2x2 pixel block is shown in Figure 14. This layout is symmetrically built in order to reduce fixed pattern noise among the four pixels and to ensure uniform spatial sampling. An experimental 64x64 pixel image sensor has been developed in a 0.35\(\mu\)m, 3.3 V, standard CMOS technology with poly-poly capacitors. This prototype has been sent to foundry at the beginning of 2006 and will be available at the end of the second quarter of the year. The Figure 15 represents a simulation of the capture operation. Various levels of illumination are simulated by activating different readout signals (read 1 and read 2). The two outputs (output 1 and output 2) give the levels between GND and \(V_{DD}\), corresponding to incidental illumination on the pixels. The calibration of the structure is ensured by the biasing (Vbias=1.35V). Moreover, in this simulation, the “node” output is the result of the difference operation (out1-out2). The factors were fixed at the following values: coef1=coef2=V_{DD}
and \( \text{coef3} = \text{coef4} = \frac{V_{DD}}{2} \). MOS transistors operate in sub-threshold region. There is no energy spent for transferring information from one level of processing to another level. According to the simulation’s results, the voltage gain of the amplifier stage of the two \([\text{AM}]^2\) is \( A_v = 10 \) and the disparities on the output levels are about 4.3%.

![Fig. 14. Layout of a pixel](image)

Fig. 14. Layout of a pixel

Chip simulations reveal that raw images acquisition at 10,000 frames per second can be easily achieved using the parallel \( \text{A}^2\text{U} \) implemented at pixel level. With basic image processing, the maximal frame rate slows to reach about 5000 fps.

The next step in our research will be the characterization of the real circuit as soon as the chip comes back from the foundry. Furthermore, we focus on the development of a fast analog to digital converter (ADC). The integration of this ADC on future chips will allow us to provide new and sophisticated vision systems on chip dedicated to digital embedded image processing at thousands of frames per second.

**REFERENCES**

High Performance Embedded Co-Processor Architecture for CMOS Imaging Systems

Romuald Mosqueron and Marco Mattavelli
Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH 1015 Lausanne Switzerland
Email: romuald.mosqueron@epfl.ch
marco.mattavelli@epfl.ch

Julien Dubois
Université de Bourgogne,
Laboratoire LE2I
21000 Dijon, France
Email: julien.dubois@u-bourgogne.fr

Abstract—This paper describes an image acquisition and processing system based on a new co-processor architecture designed for CMOS sensor imaging. The system exploits the full potential CMOS selective access imaging technology, to integrate the co-processor unit into the image acquisition loop. The acquisition and co-processing architecture is compatible with the majority of CMOS sensors. It allows to implement a wide variety of acquisition modes (random region acquisition, variable image size, variable acquisition modes line/region based, multi-exposition image) as well as high-performance image pre-processing (calibration, filtering, de-noising, binarisation, pattern recognition). Furthermore, the processing and data transfer, from CMOS sensor to processor, can be operated simultaneously to increase achievable performances. The co-processor architecture has been designed in order to obtain a unit that can be configured on the fly, in terms of type and number of chained processing, during the image acquisition process that is defined by the application. Acquisition and processing performances for a co-processor FPGA implementation, are reported and compared to classical image acquisition systems based on standard modular PC platforms.

I. INTRODUCTION

Nowadays, smart cameras are more and more used for their performances and their processing capabilities in different applications. We can distinguish three typical classes of smart cameras.

- Artificial retinas: in which dedicated processing is directly integrated aside the pixel acquisition circuitry transistors. The processing capabilities are usually fixed or limited to a few simple and local functions [1], [2].
- Standard cameras directly connected to a computer via a standard interfaces: all the processing is performed into the computer CPU,
- Cameras including embedded processing units: the processing is performed into the camera and only the processing results are transferred outside the camera (Fig.1 [3]).

For the class of the retinas, the image processing capabilities, or better the pixel imaging capabilities, are usually fixed, locally to a small pixel neighborhood and remain very limited in scope. No application specific processing can be added at the image acquisition stage. However, such kind of sensors can achieve very high frequency acquisition rates that are necessary for some class of applications. In the case of a standard camera interfaced with a computer, the data transfer between the camera and the computer is limited by the connection interface. When dealing with applications requiring high frame-rate or very high resolution cameras, usually the problem is the amount of data that need to be transferred to the CPU that may largely exceed the available standard interface bandwidth. For such class of applications, different camera architecture that include embedded processing units have been developed. They can be classified as follows: Cameras including an embedded ASIC, where fixed processing is executed. The flexibility is very low since only a few processing parameters can be configured. Such systems can be considered as very similar to a retina sensor. There also exist cameras coupled with an embedded processing unit (DSP, FPGA). The problem of such architectural solutions is that the processing capability remains very limited. Cameras with embedded coprocessors enable the implementation of more powerful processing due to the high degree of flexibility and to the clear task separation between the different units. The camera developed in this work belongs to this class of systems (Processor+FPGAs based co-processor).

For very high-performance image-processing applications an adaptive image acquisition stage is very often the key feature to achieve real-time performance and thus satisfy the application requirements. The large volume of data to be transferred to the central processing unit from the image sensor is often the system bottleneck in terms of performance. Moreover, the response time of the system is too slow to adapt the acquisition stage to the image sequence content because the transfer time is too large. This problem can also be seen in terms of system costs, the (very large) bandwidth required is
very often too expensive in terms of equipment and interfaces. Although we have assisted in the past to a continuous increase of processing performances of single core processors, this trend is approaching its end for the difficulty to brake and approach the 4GHz barrier. In the meantime we observe now the wide availability of low-cost high-speed high-resolution sensors. This fact not only pushes the required processing performance to higher and higher levels so as to cover new demanding applications, but require new architectural approaches to reduce the costs of the interfacing and processing stages that are now the real bottleneck of such systems.

The co-processing approach has been investigated in the last few years by several authors. Some works presented in literature are based on hardware co-processing designs specifically dedicated to a single application [4]–[6]. The performance improvements reported in literature are quite relevant, when comparing architectures with or without co-processor, those have the speed-up factor of several hundreds. Other authors have proposed generic systems whose property is the possibility to implement different algorithms on a coprocessing based architecture [7]. The performance of such implementations, in terms of speed-ups factors, can be higher than for some specific processing. In the class of "generic" co-processor units, only a few authors have mentioned the possibility to control the image acquisition stage simultaneously with the processing stage. Gorgon proposed a co-processor unit to control the acquisition stage of Charge Coupled Devices (CCD) sensor [8]. Jung et al. presented a preprocessing unit to control CMOS sensor [9], but the achieved functionality operates only on the specific image corrections used to compensate physical limitation of the CMOS sensor. Although CMOS sensors present very attractive properties, no works presented in literature have shown that acquisition can be adapted to the processing providing a processing stage similar to the one we can find in "retina" sensor approaches [10].

This paper describes a co-processor unit design (COP) providing an interface for the full control of the sensor acquisition process driven from the main application CPU. The main processor and the co-processor are respectively in charge of the high-level tasks, the acquisition and processing decision imposed by the application, and the lower-level tasks, characterized by high level of processing regularity and parallelism. The co-processor implementation is based on a standard Field-Programmable Gate Array (FPGA) technology.

The first interesting result achieved by implementing this architecture is that relevant speed-up factors are obtainable for reconfigurable processing modules, thus providing enough flexibility in term of choice of processing and in terms of acquisition mode defined on the fly by the application itself (selection and preprocessing of any kind of area of interest). The second is that such on the fly adaptation of the acquisition mode yields a further bandwidth reduction for the transfer of the image data to the central CPU. This feature represents for some application a further speed-up in the overall system performance in terms of reduction of processing or increase of the achievable acquisition/processing frame rate.

The co-processor commands and the data are transferred between the main CPU and the by a common bus. The command word bandwidth is negligible compared to the image data volume. The co-processor operations are determined by command received by the main processor together with the acquisition commands.

The paper is organized as follows: section IV presents how the inclusion of processing into the acquisition loop enables to exploit the features and innovations of CMOS based imaging. In section V the co-processor architecture is presented and his features are discussed in detail. Finally, the performance of the co-processor architecture obtained by simulations, are reported in Section VI and compared to a classical image acquisition and processing scheme.

II. ARCHITECTURE OF THE CO-PROCESSOR CAMERA

Figure (2) illustrates the main architectural components of the camera with embedded co-processing stage.

The system is composed by an embedded frame-grabber equipped, at different levels, of processing capabilities for the image acquired by the sensor and it is illustrated Figure (3). This system in the experimental configuration is made of a compact stack of 4 boards, to allow to interface easily various types of sensor/cameras and thus to answer various resolution and acquisition speed requirements in the most modular and economic way. The four boards include:

- the motherboard containing the main processor
• the communication board
• the board including the co-processor
• the camera interface board

The main additional advantages of this system, beside the capability of controlling the acquisition loop and the achievable processing performances compared to a traditional modular PC system, consist of

• a low dissipated power,
• compact dimensions,
• a greater robustness (mean time between failures) because it does not integrate mobile components (ventilators, hard disks)
• a greater commercial lifespan because components in the computers world are very volatile and cannot be replaced with components having the same characteristics after only a few years sometimes requiring for critical applications partial redesign of the system.

The mother board contains a Nexperia processor and include functions for the sound and image processing. Around this DSP, we can find many communications like Ethernet, ISDN, etc., as well as acquisition and restitution of video images and analogical sound. The second board is based on a FPGA Spartan XL to manage the PCI arbiter, communications like USB2.0 and Firewire can be driven to connect camera with digital standard interfaces. On the third board, two FPGAs are used to acquire the pixels and to process the image coming from the camera sensor. The fourt board is a simple interface board between the FPGA board and the camera. A compact solution with only two boards is possible for low cost and compactness.

The main boards communicate through bus PCI v2.2 allowing to transfer a large number of data (upto 133Mbytes/s) to the host processor. However, the main idea of the system architecture is indeed to reduce as much as possible the data rate after the co-processor unit by transmitting only the processed image sections or by controlling the acquisition and to let the room on the bus for other interfaces such as the USB, the IEEE1394 that are supported and communicate with the host processor by the PCI bus.

This architectural solution provide exceptional processing potential and offers wide communication possibilities (RS-232, RS-485, USB, and Ethernet interface). The connectivity is achieved with a standard PCI bus. The co-processor unit is in charge of image acquisition and pre-processing. It implements a wide variety of acquisition modes (random region acquisition, variable image size, variable acquisition modes line/region based, multi-exposition image) and high-performance image pre-processing (calibration, filtering, de-noising, binarisation, pattern recognition). The pre-processing part is independent from the acquisition part. The processing part is built with pipelined or parallel HW processing modules to obtained high-performance. Furthermore, a processing and data transfer, from CMOS sensor to processor, can be operated in parallel to increase performance. Eventually, the co-processor architecture has been designed so as to obtain a unit that can be configured on the fly, in terms of type and number of chained processing, during the image acquisition process that is defined by the application.

Following this section is a description of the three main Component board.

A. Motherboard

The processor Nexperia PNX1500 [11] has been selected for the powerful VLIW core and for the variety of supported integrated interfaces such as Ethernet controller, DDRam controller and PCI. In addition, Nexperia integrates a graphic 2D engine able to display up to a resolution of 1024 × 768 to 60 Hz. The motherboard contains a 64 Mbytes DDR Ram (333 MHz). In addition, there are 32 Mbytes of flash memory used to store the programs or different information.

B. Communication Board

This board provides four functionalities. It extends the communication of the motherboard with standards USB 2.0, IEEE1394 and Ethernet. The ethernet connection is important, because systems and a computer can communicate by this interface with a simple IP number. It also provides a centralized alimentation for the system. On this board, a converter N/A is used to display on a standard VGA monitor and this board is also used as PCI arbiter by the intermediary of a FPGA (Spartan XL) [12].

C. Acquisition and processing board

This board is in charge of the acquisition and the preprocessing stages of the video signal coming from the CMOS sensors or cameras. Figure 4 illustrates the architecture of the board. The main functions are partitioned onto two FPGAs. The first FPGA is a Virtex2Pro VP4-fg456-5 [13], their functions are to acquire images and communicate the configuration and orders to the camera. The second FPGA is a Virtex2Pro vp20-fg676-5 [13], the image pre-processing is its principal function. Both FPGAs communicate through two high speed serial channels, specific to Xilinx, called RocketIO [14].

This board contains, in addition of FPGAs:

• 2 SDRAM until 128 Mbytes associated with the first FPGA.
• 2 ZBT until 8 Mbytes associated with the co-processing FPGA.

The system can be used with several CMOS image sensors, for the results described in this paper, a sensor IBIS4 [15], with a resolution of 1280 times 1024 pixels and a 40 MHz pixel frequency has been used.

III. SYSTEM AND APPLICATIONS

The overall system can be described as an autonomous intelligent camera with powerful embedded processing when compared with modular systems associated with a computer.

The system has been thought for monitoring applications such as: road monitoring [16] or intrusion detection or any other similar application. Quality control and control of industrial processes where very high frame-rate on specific image
sections are required is another application field of this system. For this type of processes, only the "relevant" portion of the images are necessary to be transmitted to the host CPU for further processing. In some cases only the result of the preprocessing, or of the processing (i.e. the detected feature) is needed to be transmitted outside the system to a host PC.

Several schemes of system connections are possible using the Ethernet network:

- a vertical architecture: several camera works on the same sector either with different processes, or with the same processes, but placed at different position. These cameras can communicate with each other and with a central unit which can be a standard PC;
- a horizontal architecture: Several cameras are used on different work sector. In this case they communicate with each other and can be connected to a standard PC;
- a vertical and horizontal architecture illustrated in figure 5: This architecture is the composition of the previous topology connection. It finds interesting applications in the inspection of large areas for the control of industrial process chains.

The association of the processing and acquisition stages aims to reduce the pixel rate for applications where "irrelevant" image portions are detected by the co-processor. The processing is then complemented by the Nexperia processor for higher level tasks at a possibly lower pixel-rate. The partition of the tasks is made by exploiting the specificity of each elements, to use it as well as possible, thus reducing the pixel-rate where possible and the processing time so as to increase the overall throughput.

This architectural approach to the processes of sequence is particularly useful and performing, but not limited to the following application examples:

- Tracking applications: to follow events of objects on a camera with transmission of the results to another camera which will resume the tracking,
- Pattern recognition applications: to recognize an object in a scene for which only a portion of the image need to be further processed,
- Compression applications: to visualize or store sequences on a computer as for the video monitoring,
- Profilometry applications: where detection of objects profile depth and images need to be acquired on the same cameras.

Compression of video signals is generally used in camera systems to reduce the bandwidth of the data transfer and thus being able to use a standard communication channel without addition of acquisition boards such as the camera-link for instance. However, with high performance sensors, there is immediately the problem of the connection that become now the system "bottleneck", and prevents from transferring the images rate provided by the sensor. The system described in this paper supports the implementation of a compression stage that makes it possible to approach to the sensor limit capabilities [3].

IV. CO-PROCESSOR INTO PROCESSING ALGORITHM/ACQUISTION LOOP

The integration of a co-processing element into the image acquisition loop of a CMOS sensor has very interesting features. For several applications such flexibility can be successfully exploited so as to reduce the data transfer to the central CPU thus considerably reducing the necessary data bandwidth. As consequence the overall processing requirement of the application has just to process a limited portion of the original image. The key to achieve such results is to be able to provide to the main application the necessary information to adapt the acquisition stage without the need to transfer the full image to the central CPU. In other words, CMOS imaging can achieve:

- a selective image acquisition stage depending on the image content itself and on the requirements of the application,
- a relevant reduction of the data volume to be transmitted to the central CPU once the selective acquisition stage has been activated.

The condition for which such features can be achieved is that a "co-processing" element is inserted in the image acquisition loop driven by the "high level" application. In such architecture the "co-processing" unit beside the control of the acquisition stage becomes naturally in charge of the
standard low-level repetitive tasks such as filtering, de-noising, binarisation, etc. In fact the full control of the acquisition stage enables the right control of the pre-processing tasks usually performed at the level of the central CPU or high-level application.

The challenging aspects of the co-processor design are mainly related to the variable acquisition mode (i.e. input image format and layout). Obviously, The bandwidth associated to a window processing can be optimized, moreover the nature, the complexity and the number of possible processing stages can be adapted at each acquisition mode. In the examples of co-processing performance provided in this paper the acquisition command word set generated by the processor are constituted essentially by two parts: the processing order with the parameters and the acquisition part. Each acquisition field is coded on 16 bits. Many different acquisition modes are then available. In all modes, a window can be selected in the full-range image, the size and the integration time are defined and moreover a sub-sampling (on Y and X) can also be specified. In simple multi-exposition mode, the same window is acquired several times or periodically and the delay between two acquisitions can be defined. Also in the tracking multi-exposition mode, the window can be translated. Such modes allow to create a "sub-image" image by row or column accumulation when the sensor is used as line sensor even with lines varying their position during the acquisition itself.

V. CO-PROCESSOR DESIGN

The essential problem of the co-processor architecture is the trade-off between processing efficiency and flexibility required to exploit the CMOS potential features. Two different parts essentially constitute the COP architecture (Fig. 2): the processing and the acquisition parts. The following functional blocks constitute the processing part: a processor interface (PCI interface), a command controller, a processing controller, a processing structure, a CMOS sensor interface.

The command controller receives the acquisition commands, the processing commands from the main application. The task scheduling is controlled by the processing controller and is executed by the processing structure unit configured according to the received commands. The data and image portions, provided by the main CPU and used by the co-processor for the actual processing tasks, are transferred to the processing structure via the bus bridge and via the processing controller. This feature enables to implement a true co-processing stage and not a simple pre-processing.

The link between sensor and acquisition part is specific for each image sensor, consequently it should be modified after any sensor change. The connection between acquisition and processing is standard therefore independent of the sensor. Acquisition commands are constituted of parameters defined to cover a large number of acquisition modes to enable to
The co-processor is implemented in the other FPGA. All the processing of the co-processor are used for about 90%. No processing is implemented on the system architecture. In section VI, some simulation results of less an extension is made by adding new components to the system. This system is adopting this principle, nevertheless an extension is made by adding new components to the system architecture. A description of the global co-processor functionality is made in [17]. This system is adopting this principle, nevertheless an extension is made by adding new components to the system architecture. In section VI, some simulation results of the system performances are provided.

VI. EXAMPLE OF ACHIEVABLE PERFORMANCE

The image acquisition FPGA contains the camera driver, a PCI core and a camera IO core. This FPGA resources are used for about 90%. No processing is implemented on this component. All the processing of the co-processor are implemented in the other FPGA.

Three different processing types have been implemented in the co-processor:

- a median filter on different basic kernels (1 × 3, 1 × 5, 3 × 3),
- a local adaptive binarisation (Niblack algorithm) with a neighbourhood of 8 × 8 or 16 × 16 pixels [18],
- a binary pattern recognition based on block matching with 32 × 32 and 64 × 64 pattern size.

The performances and the required hardware-resources obtained by the co-processor architecture are reported in Table I for the median filter, Table II for the local adaptive binarisation and Tables III, IV for the pattern recognition.

The median filter is a simple sliding-window spatial filter that replaces the center value in the window with the median of all the pixel values in the window. The median filter is normally used to reduce noise in an image. In the median filter implementation, two kinds of filter are implemented: one dimensional (1 × 3, 1 × 5) and a two dimensional (3 × 3). The required resources for each kind are 313 and 265 slices for processing, the reunification use 893 slices which included all the processing tasks and the handling of all transfers and local buffers.

Used in the pre-processing stage, the local adaptive binarisation not only provides a very performing threshold algorithm in presence of illumination or object variations, but also allows to reduce the bandwidth to the central CPU like a retina sensor can perform. For example, a 1024 × 1024 full-range image requires 1 Mbytes to be stored but the binarized image only 1 Mb. If an area can be selected in the full-range image, for example a 256 × 256, the result image size would reduce to 64 Kbits. This process allows to gain a factor 64 on the original bandwidth.

<table>
<thead>
<tr>
<th>Number of points</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of slices</td>
<td>477</td>
<td>965</td>
<td>1605</td>
</tr>
<tr>
<td>Number of mul16 × 16</td>
<td>5</td>
<td>9</td>
<td>17</td>
</tr>
<tr>
<td>Frequency (MHz)</td>
<td>25</td>
<td>25</td>
<td>25</td>
</tr>
</tbody>
</table>

**Table I**

TIME PROCESSING OF MEDIAN FILTER 3 × 3.

<table>
<thead>
<tr>
<th>Image</th>
<th>Block</th>
<th>Time processing (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>512 × 512</td>
<td>M16 × 16</td>
<td>158.09</td>
</tr>
<tr>
<td></td>
<td>M8 × 8</td>
<td>40.80</td>
</tr>
<tr>
<td>256 × 512</td>
<td>M16 × 16</td>
<td>76.66</td>
</tr>
<tr>
<td></td>
<td>M8 × 8</td>
<td>20.12</td>
</tr>
<tr>
<td>256 × 256</td>
<td>M16 × 16</td>
<td>37.75</td>
</tr>
<tr>
<td></td>
<td>M8 × 8</td>
<td>9.92</td>
</tr>
<tr>
<td>128 × 128</td>
<td>M16 × 16</td>
<td>8.17</td>
</tr>
<tr>
<td></td>
<td>M8 × 8</td>
<td>2.34</td>
</tr>
<tr>
<td>64 × 64</td>
<td>M16 × 16</td>
<td>1.54</td>
</tr>
<tr>
<td></td>
<td>M8 × 8</td>
<td>0.52</td>
</tr>
</tbody>
</table>

**Table II**

LOCAL ADAPTATIVE BINARISATION.

For the third processing, the binary shapes search, different...
tests have been made to compare with different image size and research size. The goal of this processing is to recognize one or more shapes in a binary image. The entire processing is implemented and require 3021 slices and 9 Blocks RAM of 18 kb each. In the table, only the required slices for a single processing are reported.

<table>
<thead>
<tr>
<th>number of shapes</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>256 × 512</td>
<td>68.52</td>
<td>59.88</td>
<td>51.24</td>
<td>42.6</td>
<td>33.96</td>
</tr>
<tr>
<td>256 × 256</td>
<td>32.67</td>
<td>28.64</td>
<td>24.61</td>
<td>20.58</td>
<td>16.55</td>
</tr>
<tr>
<td>128 × 128</td>
<td>6.73</td>
<td>5.99</td>
<td>5.24</td>
<td>4.49</td>
<td>3.74</td>
</tr>
</tbody>
</table>

TABLE IV
TIME TO SEARCH A SHAPE IN AN IMAGE (MS).

A comparison has been done between the performance obtained by the co-processor architecture (COP) and a PC, Bi-Xeon 1.7 GHz, 256 Mo Ram, Rambus 800 MHz (2 × 400MHz). The performance results reported in Table V do not consider camera frame-grabber transfer time. The comparison shows that, besides the achieved speed-up factor up to a factor of 5 that would certainly result higher considering the frame-grabber transfer time, the central CPU in the co-processor approach is fully available for further processing. Moreover, when a bandwidth reduction is possible by means of adaptive acquisition the co-processor approach provides much higher speed-up gains.

<table>
<thead>
<tr>
<th>Processing</th>
<th>PC (Mpixel/s)</th>
<th>COP (Mpixel/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median 1 × 3</td>
<td>41</td>
<td>100</td>
</tr>
<tr>
<td>Median 1 × 5</td>
<td>28</td>
<td>100</td>
</tr>
<tr>
<td>Median 3 × 3</td>
<td>27</td>
<td>50</td>
</tr>
<tr>
<td>Niblack 8 × 8</td>
<td>5</td>
<td>25</td>
</tr>
<tr>
<td>Niblack 16 × 16</td>
<td>4</td>
<td>14</td>
</tr>
</tbody>
</table>

TABLE V
PROCESSING COMPARISONS.

VII. CONCLUSION

Despite the increasing speed PC processors and bus frequency, the implementation of co-processor systems expressly conceived for image sensors and inserted in the acquisition loop keeps several advantages. Very high processing speed and reduced image data bandwidth are achievable and maintain a high degree of flexibility in the pre-processing stage for the different acquisition modes specific of CMOS imaging. The described acquisition and co-processing architecture is completely operational and the potentiality of such a new architecture are far from being fully exploited and are currently investigated in several challenging applications.

REFERENCES

Abstract—Today's FGDRAs1 could now be regarded, not only as prototyping platforms, but also as reliable alternative to ASICs for consumers products production platforms. To deal with such a system, we propose a middleware layer (RTOS) named SMILE, which could manage not only software process but also hardware tasks running on an FGDRAs. Preemption issues for hardware tasks on such a system will be treated, introducing the concept of PDR-SoC2. In this paper, our work on hardware FGDRAs based task contexts, its management and its evaluation, is exposed.

I. Introduction

The idea of reconfigurable computing has been introduced by Gerald Estrin in [1] in 1960. More recently, popularization of FPGA technologies as brought the subject on the foreground of today's research themes. The application field of this concept is very large, it goes from HPC3 [2], [3], [4] to embedded SoC [5], [6] including telecommunication systems [7]. Several research have been led on this subject, some using fine grained reconfigurable platforms like FPGA [8], [9], [10], others using coarse grained reconfigurable platform [11], [12].

With the latest platform FPGA generation that integrate hardwired CPU core with the ability to manage the FPGA logic array's configuration (ex : Virtex II pro), the concept of PDR-SoC could now be a reality.

But today, most of time, DRAs4 are considered as hardware accelerator for very specific tasks, like video encoding as an example. One of the keystone of our project is that we consider a DRA as a full-fledged computational resource along with CPUs or DSPs. Several experimentation have shown that a fine grained DRA, like an FPGA, can perform any algorithm. Moreover, for any algorithm we can build an optimized architecture running on a FGDRAs. In this context, a FGDRAs may be seen as a general purpose computing device as well as a microprocessor or a DSP.

From this point of view, it is obvious that dynamical reconfiguration gives a better versatility to a FGDRAs and that an operating system is required to manage the system optimally.

The SMILE project, in which this work take place, aim at developing a distributed RTOS kernel which is able to manage a whole DR-SoC including FGDRAs. The FGDRAs' kernel must handle any algorithm and thus require preemption ability.

The main challenge of preemption being context management, the present article relates our work on this particular subject. After brief theoretical considerations about hardware task context on fine grained DRA and a short state of the art on the subject, a solution called CSB and several original improvements we made to this method will be presented.

II. Concept of hardware task's context

From a previous study led in our laboratory for the SMILE project, we were able to define what hardware task context exactly is. A task context, in the general case, relates to all the informations that must be saved to continue this task after an interruption, without modifying its work. In the case of a hardware task, this concept needed to be defined.

According to the Huffman model a synchronous architecture consists of several flip-flops (registers) and combinatorial logic blocks. For combinatorial parts, for each input vector we will always get a unique output result, so that for a totally combinatorial task, we do not need to save anything.

In case of a sequential task, the output result depends not only on the input signals, but also on the current state of the system. The current state of a system relates not only on the current control state but also on the current value of variables. This current state thus need to be saved and then restored to be able to stop a hardware task at any time, without any data corruption. Although this current state is defined by both registers and memories, we will focus only on the registers, considering that memory's issues could be managed by some kind of MMU as it's often performed in micro-processor based system.

---

1FGDRA : Fine Grained Dynamically Reconfigurable Architecture, the most known being FPGAs
2PDR-SoC : Preemptive Dynamically Reconfigurable System on Chip
3HPC : High Performance Computing
4DRA : Dynamically Reconfigurable Architecture
The hardware task’s context can thus be defined as the entire set of information memorized by all the flip-flops that compose this task.

III. State of the art

This section mainly summarizes the survey of the IRISA lab about hardware preemption on FPGA.[13]

A. Bitstream based solution[14], [15], [16]

This method consists on reading back the configuration bitstream of the FPGA as it contain all data we want to save. The bitstream transfer can use either the FPGA JTAG chain or a parallel interface such as the SelectMAP/ICAP interface on Virtex Family. But this method presents a lots of drawbacks in both cases, especially for the saving process.

The first drawback is due to the fact that the useful information is scattered among a large quantity of data. So that a lot of useless information must be transferred and the task context must still be extracted from it.

The second drawback is that the structure of this bitstream varies from one FPGA’s family to another. Moreover, the read back bitstream is not the same than the configuration bitstream, and not all FPGA family support a read back process.

B. Ad-Hoc solution[17], [18]

This method consists in providing a predefined interface to the application designer (API), the designer must then take context management of his application into account in respect of the given API.

These methods can not be used here as our goal is to simplify the design process of an application, and not to make it more complex. We cannot then relieve us of a hard job on the application designer.

C. Scanpath based solutions

This method is inspired by circuit testing. It consists on providing additional logic to all flip-flops of the design, allowing both a 'normal' running mode and a 'scan' mode. During the scan mode, all flip-flops are chained to build a shift register. We then just need to shift as many bits as there are flip-flops in the design, to read or to write the required information.

This method has many advantages:

- First, it can be easily implemented directly into any existing FPGA just using an HDL definition of special flip-flop. It is thus an universal method.
- Secondly, modifications that must be attempted to a standard design to make it into a preemptable task in a system based on this method can be easily automated.

But this method has also some drawbacks. It introduces overheads in both size and maximum speed of the task and most of all it suffers of poor transfer speed. In this article this last point is addressed.

5API : Application Programmer Interface

IV. Retained solutions and Implementation

A. Scanpath

Taking into account all that we have just seen in the preceding section, we have chosen to concentrate on the use of a scanpath. As we wanted to be able to test different kinds of chainable flipflop, we chosen to work at the lowest possible level.

A previous study has been performed in our lab about hardware task’s context saving, that use the JTAG Boundary Scan bus for context transfer. From this previous study, it comes out that the use of the JTAG’s Boundary Scan standard increased in an useless manner both data transfer and the surface occupied in the FGDTA. Indeed, each flip-flop having to be doubled, the size of the application was also doubled. Although this experiment validated the use of a scanpath to save and restore an hardware context from a functional point of view, it appeared that there was some weaknesses in this system.

A solution consisted in putting aside the JTAG protocol and preserving only the scanpath to reduce at the same time the configuration sequencer, which does not have to manage JTAG protocol any more and the transfer since all the protocol part is removed.

B. CSB

The main idea of our preemption scanpath is to modify chainable flip-flops to minimize both occupied surface and the number of control signals. We have then introduced the CSB.[6] Whereas previously set up JTAG chainable flip-flops were each made up of three multiplexers, two flip-flops and a combinatorial block, the new CSB flip-flops only require one flipflop and one multiplexer as it can be seen on (fig. 1). Moreover, the old design required five additional control signals versus one for the new one, thus simplifying drastically the place&route process. We just had to this a global clock multiplexer in order to be able of using a different clock frequency in run mode.

The question of the relative slowness of the transmission still remains. To circumvent this issue, two approaches have been considered:

- first saving the contexts of the various hardware tasks, directly into internals ram blocks(eg: BRAM in FPGA). In this way, the operating frequency of our scanpath can be maximized. The transfer time of a context by a scanpath being directly related to its working frequency, this enables to save time.
- secondly using several scanpath working simultaneously. Memories often requiring an 8bit data access,
we have chosen to use an 8 scanpath architecture called PCS8.  

C. CMU with embedded Block RAM  

Finally, a configuration controller allowing to save or to restore a given context on request have been defined and implemented. This sequencer is directly linked with a BRAM memory thus forming a complete context management block called CMU. Using an external internal BRAM permits to maximize the working frequency of the CMU as it only require internal signals which are quicker than external ones. Moreover, this topology authorizes to place the memory close to the sequencer resulting in a shorter critical path.

This CMU can be directly controlled using a single 16 bits register that can be accessed by a standard micro-processor bus. This register contains four fields, “nb” to specify the number of shift required, “CID” which identifies the context that we want to save/restore, “S/R” to choose whether we want to save or restore the context, and a “run” bit that must be set to 1 to begin the transfer. This bit gets back to 0 automatically when the transfer is completed. This configuration allow to deal with 16 tasks of 1024 flip-flops each for a simple CSB configuration or 8192 flip-flops each for a PCS8 configuration. Note that those numbers have been chosen arbitrarily and can easily be increased if necessary, the only limitation being the restricted amount of RAM in the targeted FG DRA. The CMU offers a simple connection with another system like a micro-processor and provides a simple and classical programmer view of context management. In the long term, CMU should be integrated inside the FG DRA, like MMU are in most microprocessor.

V. Design flow, methodology and tools  

As a designer realizes an IP, he spends his time on the functionality of the IP, he does not want nor have the time to add specific material to make his IP preemptable. It is thus necessary to describe a design flow that takes into account this point of view. In such a flow, the flip-flops replacement and chaining must then be automatic. Although some solutions exist for DFT purposes, those solutions were not applicable here. We have thus decided to set up our own solution.

For that purpose, we based our work on a standard workflow as can be seen on figure 3a. The synthesis is done in two stages, RTL synthesis and Technological synthesis (Place & Route). Each stage has two outputs, one for implementation and one for simulation. As an example, the RTL synthesis provides an RTL VHDL model of the design, in our case, as we targeted a Xilinx's FPGA, this model is based on the UNISIM library which is provided by Xilinx.

A TCL script have been developed to add chaining signals to the RTL simulation model, and a technological library called CSBlib that contains a synthesisable VHDL model of every component that can be instantiated at this level have been implemented.

Based on CSBlib and on our conversion script, a new designflow can be proposed (figure 3b) in which only the RTL synthesis is first performed. The conversion script is then executed on the RTL simulation model provided. The whole synthesis (RTL and Technological) must then be performed.

This method has been implemented for CSB and the whole process is quasi-automatic. The final version should be able to deal with a PCS8 context management configuration and should be fully automatic. The current version have been successfully tested on an UART downloaded on OpenCores.org. This show that IP designer do not need to bother about context management while designing an application as any IP can be converted as long as a VHDL description of this IP is provided.

VI. Tests and comparisons  

In order to evaluate the proposed architecture, a 7th order fully pipelined FIR filter have been implemented in three different configurations. The first configuration use standard design method (not preemptable), the two others use respectively a CSB and a PCS8 configuration. Note that in order to simplify the design we chose to fix all coefficients to one.

For each test, pre and post-synthesis simulation were performed. At the light of those tests, we can say that both CSB and PCS8 effectively allow to save and to restore an hardware task’s context. Quantitative results about used place and speed are extracted from synthesis reports. Note that those tests were made targeting a Xilinx Virtex II-Pro XC2VP30FF896 FPGA, but we can use any other FG DRA as long as it provide sufficient internal memory.

1 PCS : Parallel Configuration Scan 8bit  
2 CMU : Context Management Unit

- 133 -
A. Area considerations

As expected, the use of both methods implies an overhead in term of FGDRA occupation. The main problem with area occupation on FGDRA is that all results highly depend on optimizations performed during the synthesis process. In table II we can see that the PCS8 method uses less LUT that CSB. But after further investigations it appears that due to the FIR architecture, seven 8 bits registers from the first pipeline stage are chained the same way in run mode or in scan mode. So that during synthesis process, data input multiplexers for those seven register (representing 56 flipflops) are actually not implemented.

Another optimization artifact is replica flipflops. As an example, the CSB based CMU alone require 37 flipflops and the FIR use 128, so the complete task including both should require 165 flipflops, but it actually requires 180. This is due to the fact that, for timing optimization purpose, the synthesizer creates replicas of some flipflops of the design. To study the real impact of our context management method, we then should not consider replicas as they are due to synthesis optimization, not to the context management method.

To conclude, we can give an estimation of area required for a given task when using CSB or PCS8 context transfer method and a CMU. Given $N_{ff_{STD}}$ and $N_{ff_{STD}}$ respectively the number of flipflops and LUT used for the standard version of the task, equations 1 and 2 gives the number of flipflops and LUT required for the CSB based preemptable version of this task.

$$N_{ff_{CSB}} = N_{ff_{STD}} + 37$$
$$N_{lut_{CSB}} = N_{lut_{STD}} + N_{ff_{STD}} + 1 + 40$$

Equation 3 and 4 give similar result for a PCS8 based preemptable task.

$$N_{ff_{PC S8}} = N_{ff_{STD}} + 33$$
$$N_{lut_{PC S8}} = N_{lut_{STD}} + N_{ff_{STD}} + 1 + 35$$

Note that those equation do not take synthesis optimization in account so that actual result can be slightly different. Memory used for the CMU is not mentioned here, when using a Virtex II pro, both PCS8 and CSB CMU use one BRAM in the actual configuration.

B. Time consideration

Concerning time there are two important parameters, the first is context transfer time, directly proportional to the scan mode max frequency, and the second is the run mode max frequency.

When adding CSB or PCS8 preemption ability to a given task, the maximum application’s working frequency is slowed down. This could be a serious problem and probably constitutes the main drawback of our system. This speed penalty is less important for CSB than for PCS8 but if the working frequency is not of a big issue here the penalty in term of overhead is largely compensated by the 8x gain due to parallelization.
Those results demonstrate that PCS8 has a clear advantage in terms of overhead. But if the applications had to work at a frequency close to the FGDRAs limits, a CSB configuration should be considered.

VII. Conclusion

A complete hardware task-context management method have been set up, based on a management unit, the CMU, and using two different context transfer architectures, one based on a simple scanpath, the CSB, and the second based on eight scanpath running in parallel, the PCS8. A quasi automatic designflow have also been proposed to build a preemptable hardware task, using our context management method, from a common VHDL description, using standard synthesis tool, a custom build library and a custom conversion script.

Our method has the particularity to be universal as it can use any synthesizable VHDL IP as input and can target any existing FPGA as long as it has sufficient resources for the given applications. The most noticeable drawback of this method being that it slows down the application's maximum frequency.

Using this method, a hardware task can be preempted at any time. In some case a given task should not be preempted while performing a particular action, an access to an external pipelined memory is an example of such an action. For this purpose, the next version of the CMU will provide a special flag to indicate that the task is in a critical section.

At this time, we work on a midlevel side of RTOS for PDR-SoC and the real time scheduling of hardware task on FGDRAs. The second part of our work will concentrate on modifications of hardwired architecture of an FGDRRA to make it more efficient for dynamical reconfiguration. We work on a new FGDRRA architecture that directly implement hardwired PCS8 and CMU.

References

A modelling of the flexibility of an RTR-FPGA implementation compared to a software implementation

Yves BERVILLER, Oscar PEREZ and Serge WEBER
Nancy Universités
LIEN
BP 239 - 54506 VANDOEUVRE lès NANCY Cedex
Email : {first name}.{last name}@lien.uhp-nancy.fr

Résumé—In this paper we present both a definition and a modelling of the suppleness and the flexibility of a data processing architecture. Based on this modelling, we can characterise various run-time reconfiguration scenarios for the implementation of an algorithm on an FPGA that allows partial reconfiguration. In this way we show that run-time reconfiguration, in some cases, can be used to extend the suppleness in the low throughput area of the design space, but with orders of magnitude improvement compared to a general purpose processor. The cost of this increased suppleness is a larger configuration memory and an additional temporary data memory. The last one is a good design parameter to trade-off throughput versus cost. Our modelling is the basis of a design methodology that can give some advices to a designer about the opportunity to whether use the run-time reconfiguration feature or not.

I. INTRODUCTION

One of the claimed goals of the run-time reconfiguration (RTR) feature allowed by some FPGA architectures is that this technique provides more flexibility than ASICs and more processing power than general purpose processors (GPP) [7]. The reasons given to justify this, are that more specialisation leads to more efficiency and a more generic architecture leads to less efficiency [5], ASICs and GPP being the extreme architectural cases. If this sounds, at least qualitatively, quite intuitive, from a quantitative point of view little work, if even, has been made in order to prove it. Even worse, we did not find any formal definition for the expression “flexibility of a processing architecture”. However, we found some more specific definitions like, for example, the flexibility of the interconnection in [6]. There, the flexibility of the memory/logic interconnect, is defined as the number of vertical tracks to which each memory pin can connect, and the flexibility of a switch block is defined as the total number of possible connections offered to each incoming wire. We found also some definitions in fields that do not belong to the computer architecture domain. The factory automation working field provides a definition for the flexibility of a manufacturing chain : flexibility measures the degrees of freedom that are presents during the reactive phase of the scheduling [9]. If such a definition seems of little help to us, it introduces the time dimension of the scheduling in the definition of the flexibility. Thus we propose to define the flexibility of a data processing architecture as the number of different algorithms that can be implemented at a given processing throughput. With such a definition it is no more obvious that run-time reconfiguration increases the flexibility of an architecture. Indeed, if it is obvious that the number of implementable algorithms is increased by the concept of virtual hardware allowed by RTR [7], the data processing throughput can also be decreased by the use of RTR. A high level modelling of the flexibility of a data processing architecture would help a designer to decide when it is a good choice to use RTR and when it is not. Some metrics to assess the efficiency of a given architecture have been proposed [4] but they are mainly located in the time-area domain [2] and based on benchmarks. This can be very effective for the designer of an FPGA circuit in order to chose, for example, the optimal LUT size [3], but for the designer of a system that has to choose such an FPGA the problem is quite different. Here we want to propose a metric for helping a designer, of a system that includes an FPGA, to decide if it is wiser to implement some part of the application by using RTR or a GPP. Indeed, in both cases the implementation of the algorithm resides in a memory (fig. 1) and the problem could be posed as : for a given amount of memory, which architecture provides the largest flexibility? We try to answer to this question in this article.

The remainder of the paper is organised as follows : in section II we present the modelling used to assess the suppleness of a GPP and an FPGA. Then, in section III we present the modelling of the flexibility of both architecture. Section IV we study the case of some run-time reconfiguration scenarios and we conclude in section V.

II. SUPPLENCE MODELLING

We define the suppleness of a data processing architecture as the number of different algorithms that can be implemented on this architecture. We use the word algorithm in its broadest definition : a succession of elementary operations. We do neither worry about the usefulness of each algorithm, nor about the semantic behind the counted algorithms. Thus, our model gives only an approximated mean to compare
the suppleness of data processing architectures independently from any benchmark. However, the model seems coherent with many trends in current and future processing architectures.

A. Suppleness of a general purpose processor

1) Assumptions: The configuration of a general purpose processor corresponds to the program that it is executing. Thus the main parameters that define its suppleness are: the number \( N_P \) of instructions that are executed and the number \( N_I \) of instructions in the instruction set. For the sake of simplicity, we assume that two different addressing modes of the same instruction can be counted as two different instructions. In order to make comparison between two architectures at a given configuration memory footprint, we express the number \( N_P \) of instructions that are executed, according to the size \( B_M \) of the configuration memory in bits and the number \( B_I \) of bits constituting each instruction word. Thus for an Harvard architecture processor, the last one is quite simple to determine, but for a Von Neumann architecture processor this can be more tricky. In the later case we could, for example, use the mean value of the size of the pure control field part of each instruction word. But there are many instructions that do not include an operand address, thus again we simplified the problem by assuming that all instructions have the same size, that of the size of the greatest memory word accessible in one clock cycle.

2) Modelling: With the assumptions exposed previously, we can express the suppleness \( S_{GPP} \) of a given microprocessor using equation 1. Where \( N_I \) is the number of instructions in the instruction set, \( N_P \) is the number of instructions in the executed program, \( B_M \) is the number of bits in the program memory and \( B_I \) is the number of bits of the instruction bus of the microprocessor. As we will see later, this leads to very high values for the suppleness \( S_{GPP} \), thus we will more conveniently use the logarithm in radix \( N_I \), as given by equation 2.

\[
S_{GPP} = (N_I)^{N_P} = (N_I)^{B_M / B_I} \quad (1)
\]

\[
\log_{N_I}(S_{GPP}) = N_I = \frac{B_M}{B_I} \quad (2)
\]

B. Suppleness of an FPGA

1) Assumptions: The configuration of an FPGA contains various information: the functionality of each logic cell, the routing of signals between cells, local functionality of embedded blocks and the content of embedded memories. We defined the suppleness of an FPGA as the number of different sequences of operations that it can apply to data. Since the common denominator between all FPGA families are the logic cells, we only consider this type of resources in our modelling. All logic cells contain one or more LUTs of \( K \) inputs and an optional associated flip-flop. The bits needed for the routing of signals is taken into account for the configuration memory requirements, but we assume that they do not provide a functionality, they make only the functionality of the logic cells usable.

2) Modelling: If the number of inputs of the look-up tables is \( K \), if each LUT has an associated by-passable flip-flop and if the number of such LUTs available in the FPGA is \( N_C \) then equation 3 gives the suppleness of the FPGA. Again, since the values are generally very high, it is easier to take the logarithm and, in order to make comparisons, we can use the same radix for the logarithm as in the case of the GPP. If we assume that the data-path of the algorithm implemented on the FPGA has a width of \( B_O \) bits, the resulting suppleness is given by equation 4. For the sake of simplicity, we assume that each LUT and its associated flip-flop can only process a single bit. This can be easily corrected, indeed many currently available FPGAs can process two bits of data within a single LUT, in such a case one needs only to substitute \( B_O \) with \( \frac{B_O}{2} \) in equation 4.

\[
S_{FPGA} = (N_I)^{N_P} = (N_I)^{B_M / B_I} \quad (3)
\]

\[
\log_{N_I}(S_{FPGA}) = N_I = \frac{B_M}{B_I} \quad (4)
\]
\[ S_{FPGA} = \left(2^{(2^K)+1}\right)^{N_C} = \left(N_L N_C\right)^{N_C} \text{ with } N_L = 2^{(2^K)+1} \]  
\[ S_{FPGA} = \left(N_L\right)^{N_C} \]  
\[ (3) \]
\[ (4) \]

C. Suppleness of an RTR FPGA

In the case of an RTR implementation on an FPGA, the suppleness has the same expression than the one given by equation 4, except that the value of \( N_C \) is multiplied by the number of reconfigurations. Thus, if the number of temporal partitions is \( N \), the value of \( N_C \) must be substituted with that of \( N \cdot N_C \).

D. Examples

In order to illustrate our modelling, we give two examples, one for a microprocessor and one for an FPGA. The memory footprint used for both example is the size of the bit-stream required to configure the CLB part of the FPGA. This means, that we do not take into account the configuration bits that are used by the embedded BlockRAM and multiply and accumulate blocks.

1) FPGA example: Here we give an example based on the Xilinx Virtex II family of FPGAs. We use an XC2V6000, but the main difference between members of a given family is the number of elementary logic cells, so the suppleness of a given cell will be the same, but the size of the configuration memory of the whole circuit will be different. As stated above, we only take into account the routing and configuration of the logic cells part (slices in the Xilinx terminology). For the XC2V6000 we have the following characteristics: 33 792 slices containing two 4 - LUT and two flip-flops each, the size of the bit-stream is 21 849 504 of which 6 609 312 are required for the various embedded and I/O blocks. This means that the configuration memory of the logic cell part has a size of 15 240 192 bits.

By substituting these values in equation 3, we obtain a suppleness of about \( 10^{335 \text{,} 689} \).

2) GPP example: Here we give an example based on the Xilinx Microblaze 32-bit RISC embedded microprocessor. This processor has a 32-bit instruction-word bus and its instruction set comprises 106 instructions if we take into account the different addressing modes. The program memory size to consider is that of the configuration of the logic cell portion of the FPGA.

By substituting these data in equation 1 we obtain a suppleness value of about \( 10^{364 \text{,} 564} \). Thus we can see that a GPP seems to be much more supple than an FPGA, and this can be easily explained by the fact that a GPP has mainly a fixed interconnect in contrast to the FPGA. One could object that a single instruction executed on a GPP is not sufficient for processing, because there is, roughly, the need for a load or a store instruction for each processing instruction. But even if we make such an assumption, the GPP in our example would have a suppleness of about \( 10^{482 \text{,} 282} \), still outperforming the FPGA.

III. FLEXIBILITY MODELLING

Our definition of the flexibility of a data processing architecture rely on the suppleness and the data throughput of the architecture. This implies that we have to model the throughput of an architecture with respect to its suppleness. This is what we are going to do for a GPP and for an FPGA in this section.

A. Throughput modelling of a GPP

Here we make the very simple assumption that a GPP can load, apply a single processing instruction and output (or store) the resulting data whose size is the same than that of the data-path of the GPP at each clock cycle. This implies that if the processing algorithm needs \( N_P \) instructions in order to complete, the throughput will be \( N_P \) times lower than the one obtained with a single instruction. The duration of a clock cycle is \( T_{GPP} = \frac{1}{Clk_{GPP}} \). Here again, one can object that the pipeline architecture of most GPP would give some improvements. This is true, and it is always possible to assume that all the pipeline stages are always fully supplied with data, thus providing an improvement factor equal to the number of pipeline stages available. Even with such a conservative assumption, it would be difficult to obtain a sustained throughput of one data word per clock cycle for an algorithm of more than 10 instruction. But we will see that there are generally many orders of magnitude between the throughput of a GPP and the one of an FPGA.

Hence, the data processing throughput \( T_{GPP} \) of a GPP can be modelled by equation 5. We can see that this could explain the trend towards very long instruction words and data-path, because the logarithm of the suppleness is only divided by this size \( (B_I) \), but the throughput is multiplied by the square of this size. Thus, as the flexibility can be interpreted as being the product of the throughput by the suppleness, increasing the size of the data-path increases the flexibility.

\[ T_{GPP} = Clk_{GPP} \cdot \frac{B_I}{N_P} = Clk_{GPP} \cdot \frac{B_I^2}{B_M} \]  
\[ (5) \]

B. Throughput modelling of an FPGA

In an FPGA we can theoretically choose any size for the data-path width. In a real case however, there are some limiting factors. First, there is a limited number of I/O pins available to the designer, secondly, and may be most importantly, the data buses pushing and pulling data to and from the FPGA are generally not very wide. We assume that the data-path is \( B_O \) bits wide, that it is a total pipeline and that it operates at a clock frequency \( Clk_{FPGA} \). Hence the throughput \( T_{FPGA} \) can be modelled like in equation 6. Again, if we think in terms of logarithm for the suppleness of the FPGA we can see that its multiplication by the throughput (i.e. the flexibility) does not depend on the value of \( B_O \). This is in contrast with the GPP, where it was a clear advantage to increase \( B_I \). But, one can also see that it is interesting to have \( LUTs \) with more inputs,
since this increases the suppleness for a given throughput or the throughput for a given suppleness.

\[ Th_{\text{FPGA}} = Fclk_{\text{FPGA}} \cdot B_O \]  

\[ \text{C. Throughput modelling of an RTR FPGA} \]

In the case of run-time reconfiguration of an FPGA, the throughput modelling depends on four more parameters: the number \( N \) of temporal partitions, the size \( D \) that is the number of words in each data block processed between reconfigurations, the number \( n \) of pipeline stages in the data-path and the reconfiguration time \( T_{\text{reconf}} \) that is needed for reconfiguring the logic cells used in a temporal partition of the algorithm. Hence, we can model the average throughput, which is the number of processed bits per time unit, of an RTR FPGA implementation by means of equation 7. Thus, we can see a double overhead in the use of RTR. The first one is reduction by a factor of \( N \) of the throughput if we neglect the reconfiguration time. The second one is the value of \( T_{\text{reconf}} \), that in practice, is very great compared with \( T_{\text{clkFPGA}} = \frac{1}{Fclk_{\text{FPGA}}} \), thus requiring a very big value for \( D \) in order to achieve an interesting throughput. The last case implies that \( n \) can generally be neglected with respect to \( D \).

\[ Th_{\text{RTR FPGA}} = \frac{D \cdot B_O}{N \cdot ((D + n) \cdot T_{\text{clkFPGA}} + T_{\text{reconf}})} \]  

\[ \text{D. Comparisons of the flexibility of a GPP and the one of an FPGA} \]

Figure 2 gives an illustration of the variations of the suppleness with respect to throughput for our three examples. We made the following assumptions:

- The clock frequency of the GPP was ten times greater than that of the FPGA (100 MHz)
- The maximum data-path width in the FPGA was ten times greater than that of the GPP (32 bits)
- The number \( D \) of words in a data block for the RTR FPGA implementation was chosen such that the processing time is equal to the reconfiguration time (5 ms)
- The FPGA can not process data of less than one bit, thus if we want a throughput of less than 100 Mbps we have to reduce the clock frequency, but then the suppleness remains constant.

In the case of the RTR FPGA implementation, the number \( N \) of temporal partitions is the parameter that allows both a reduction of the throughput and an increase of the suppleness.

There are many observations that we can make based on figure 2. First, for a given throughput value, the FPGA is generally more supple than a GPP except for very low throughput values. This is not a surprise, because the suppleness of the GPP is directly related to the number of executed instructions, but the throughput depends on the reciprocal of this number. Secondly, the RTR seems not interesting for very high throughput values in comparison with a static FPGA.
FIG. 3: the three scenarios for the RTR on two identical areas with three partitions

implementation, but it provides more suppleness for a given throughput than the GPP. What is not visible on this figure is that the curve for the RTR case can be adapted by changing the value of $D$. This means that the designer of an RTR system can trade-off the size of the temporary memory in order to achieve the desired throughput with a given value of suppleness.

IV. RTR SCENARIOS

A. RTR with a globally reconfigurable FPGA

This case corresponds to the model presented in section III-C. We do not analyse it further here, but we enumerate it because this case was the first step towards the partially reconfigurable RTR. Indeed, the partially reconfigurable RTR can be seen as a multi-FPGA system in which an FPGA can process data while the other one in being reconfigured. This leads to some techniques for, at least partially, hiding the reconfiguration overhead. Figure 4 illustrates some possibilities offered to a designer by the use of different types of reconfiguration in comparison to the GPP and the static FPGA. Again, a parameter not visible on this figure is the number of data required in the block to process. It is nevertheless this parameter that makes sense to the masking of reconfiguration steps. Indeed, by observing figure 4 it seems that the case annotated suppleness_FPGA_RTR_hidding_2_3 is absolutely not interesting, since it provides always less suppleness than the globally reconfigurable case annotated suppleness_FPGA_RTR and its maximum throughput is smaller. But in fact, it can provide a given throughput with a smaller data block size than the globally reconfigurable implementation. This is shown on figure 6 where we did not plot the value for a throughput near the maximum achievable,
because a throughput of 4.9Gb/s requires more than 2Gbits of temporary memory for the globally reconfigurable case.

B. RTR with a partially reconfigurable FPGA

1) Two reconfigurable areas - three partitions: In this case, the only interesting area ratio is $\alpha = \frac{1}{2}$, because for all other values, one of the two reconfigurable areas can never be reconfigured. As shown on figure 3, there are still three sub-cases depending on the value of $D$. We assume that there is a task dependency between partitions, that is : a partition can only begin to process its data when the former partition has finished.

The suppleness along with the throughput obtained for each case is given in table I. In all cases there is a slight increase in the suppleness compared to a static FPGA implementation, since the virtual area is increased by 50%. Case 1 corresponds to a situation where the reconfiguration times are over-hidden by the processing times. This can be seen on the figure by the wait states. Thus the throughput is the same than the one obtained with a global reconfiguration with 3 partitions and a null reconfiguration time. This sounds interesting, but this over-performance has a cost, the size of the memory needed to store the $D$ data words processed by each partition. The asymptotic value of the throughput is one third of that of a static implementation. Case 2 corresponds to a small size of the data block, thus the total processing time depends only on the reconfiguration time. This means that the overall performance will be poor, since the reconfiguration time dominates. However, there is still an improvement in comparison to a similar case with global reconfiguration (III-C with $N = \frac{3}{2}$). Case 3 is a kind of optimal one, since the size of the data block is such that the partition processing time exactly hidden the reconfiguration time. This means that we obtain the same throughput as the one we would expect from a global reconfiguration in three partitions with null reconfiguration time and this is obtained with the minimal size of the data block.

2) Two reconfigurable areas - four partitions: In this case, we could set $\alpha$ to an arbitrary value belonging to $[0; \frac{1}{2}]$, since the domain $[\frac{1}{2}; 1]$ corresponds to the symmetrical case (permutation of the reconfigurable areas $A_1$ and $A_2$). But for the sake of simplicity we only present the case with $\alpha$ belonging to $[0; \frac{1}{2}]$ on figure 5. However we made a different assumption than in IV-B1 on the reconfiguration feature of the FPGA. Indeed, in IV-B1 we assumed that there was only one reconfiguration interface in the FPGA, thus we could not reconfigure simultaneously two areas. But in the case of two reconfigurable areas of different sizes with four partitions, if there is only one reconfigurable interface, there would be too many wait states on the smallest area for usable values of $D$. Again, case 1 corresponds to an over-hidden reconfiguration time. This explains the various wait states on figure 5 for both areas. Case 2 corresponds to the case where only a portion of the reconfiguration time hiding is achieved and case 3 is a kind of optimal case, at least with respect to the reconfigurable area $A_1$. The corresponding results are summarised in table II.

V. Conclusion

In this paper we have presented both a definition and a modelling of the suppleness and the flexibility of a data processing architecture. Based on this modelling, we can characterise various run-time reconfiguration scenarios for the implementation on an FPGA allowing partial reconfiguration. In this way we show that run-time reconfiguration can be used to extend the suppleness in the low throughput area of the design space, but with orders of magnitude improvement compared to a general purpose processor. The cost of this increased suppleness is a larger configuration memory and an additional temporary data memory. The last one is a good design parameter to trade-off throughput versus cost. We give also some reconfiguration scenarios, that a designer can use to try some partitioning and reconfigurable management stra-
Figure 4: Supplesness versus throughput comparison for cases IV-B1

Figure 6: Comparison of the temporary memory size for the global and partial RTR cases

Strategies. Currently, we try to define a strategy and its modelling in the case of a simple data dependency between partitions. We call “simple data dependency” the fact that a partition can begin its processing as soon as the preceding one has output some data, but not necessarily all the D data as we assumed in this paper. Our future works concentrate on the integration of this modelling in a temporal partitioning tool, in order to help the designer to assess the advantages and drawbacks of run-time reconfiguration very early in the design schedule.

Références


Deploying a Telecommunication Application on Multiprocessor Systems-on-Chip

Daniela Genius, Etienne Faure, Nicolas Pouillon
Laboratoire LIP6/Département SOC
Université Pierre et Marie Curie
E-mail: {daniela.genius,etienne.faure,nicolas.pouillon}@lip6.fr

Abstract

The particular form of the task graph of many telecommunication applications permits a high level of coarse grained parallelism. We consider a classification application on a telecommunication oriented multi-processor system-on-chip (MP-SoC) platform. The hardware architecture on which this type of application is executed can contain a variable number of programmable processors and of dedicated hardware co-processors, sharing the same address space. Inter-task communications are implemented via Multi Writer Multi Reader FIFOs placed in shared memory. This paper analyzes additions and modifications required to enable the Design Space Explorer DSX to meet the requirements for the deployment of such an application.

1 Introduction

Telecommunication applications can be considered a special case of streaming applications; They are usually processing packet streams, where the same processing is performed on each packet, but the actual computing depends on the packet contents. Throughput requirements are variable: backbone equipments, such as routers, require high throughput and low processing per packet, while traffic analysis requires less throughput but intensive computation per packet. For [17], this variable processing time, depending on the packet type, is the main characteristic of network applications.

We focus on telecommunication applications written in the form of a set of coarse grain parallel threads communicating with each other. Inter-task communications can be done through message passing like in STepNP [12], modeled in the form of data flow graphs like StreamIt [6] and Ptolemy [3] or can use the shared memory capabilities of the multi-processor hardware architecture.

Kahn Process Networks (KPN [9]) propose a semantics of inter-task communication through infinite FIFO channels with nonblocking writes and blocking reads. Such infinite channels are impossible to implement, thus KPN formalism has been adapted for example by YAPI [5]. To deal with the select problem YAPI introduces the select function, which makes the model nondeterministic. Implementations of YAPI are COSY [2] and SPADE [18]. Disydent (Digital System Design Environment [1]) is also based upon KPN and uses point to point FIFOs. While the KPN formalism is well suited for video and multimedia applications, that can be modeled by a task graph where each communication channel has only one producer and one consumer, it is not convenient for telecommunications applications where several tasks access the same communication buffer, in order to consume (or produce) packet descriptors.

MWMR (Multi-Writer / Multi-Reader) channels [8] are software FIFOs that can be accessed by several reader and writer tasks. The communication protocol is described in more detail in [7]. The generic MWMR communication channel supports both hardware or software producer or consumer, making possible to decide quite late whether a task should be implemented in software or hardware.

DSX (Design Space Explorer, [10]) is based upon the MWMR concept. It extends Disydent by a comfortable user API for design space exploration. The aim of DSX is to have one application description, one hardware description, and mapping rules which can be easily modified, all written down in one common language. It offers extensive debug capabilities: the arduous task of deployment comes before the fine-tuning of parameters.

2 Application Specification

In order to extract the coarse grain parallelism from a sequential application, two basic approaches exist. The first one relies on the coarse-grained segmentation of the sequential application. The algorithm is split into functional tasks that execute sequentially. This is called pipeline parallelism. The other consists in duplicating the whole sequential application into many clones. All the tasks are doing the same job, but each one on different data. This is known as task farm parallelism. This task farm model is convenient
for telecommunication applications processing successive and independent packets in a Gigabit Ethernet stream.

**Task graph** Task farm and pipeline parallelism can be combined to yield any hybrid of graph between these two forms such as figure 1. All communication between tasks use point to point channels, that can be implemented as software FIFOs, in order to handle the asynchronous behavior of the tasks. Figure 1 represents communication channels by arrows between tasks. The FIFOs implementing the communication channels are implicit.

**MWMR channels** In many cases, the data produced by a task is not destined to one particular task, but rather to a class of tasks. Assume that tasks T00, T01 and T02 in figure 1 are three instances of the same computation, and that T10, T11 and T12 are three instances of another computation. In this case, the first three tasks can send their output to any of the three others. It is evident that we should try to replace the nine separate communication channels by one single, multi-access communication channel. Figure 2 shows one single FIFO, shared by three producers and three consumers.

![Figure 1. Example of hybrid parallelism](image1)

![Figure 2. explicit MWMR channels](image2)

This new task and communication graph (TCG) is now a bipartite graph that describes the intrinsic coarse grain parallelism of the application, without precisign the type of implementation: As both programmable processors and hardware coprocessors can read from or write to a given software MWMR, each task can be implemented either as a software task (running on a programmable processor), or as a dedicated hardware coprocessor. KPN channels can be implemented as a special case of the proposed MWMR communication formalism: In order to implement the KPN semantic, the task graph must have only one producer and one consumer per channel, and all the accesses to the FIFOs must be enclosed into a loop [11].

### 3 The Target Hardware Architecture

The target hardware architecture is a multi-processor system on chip based on SoCLib [16] components and running the MUTEK [14] micro kernel. It contains a variable number of 32 bits processors (currently MIPS R3000), a variable number of embedded RAM banks and other components such as lock engine, interrupt controller, and several I/O coprocessors. All these components are communicating through a VCI/OCB compliant micro-network [19]. There are two types of components: initiators and targets. Initiators send request packets, routed to the appropriate target by the interconnect, targets send response packets.

All initiators and targets share the same address space. In such hardware platform, using a large amount of processors, coprocessors and ram banks, the central interconnect has to provide a large throughput between initiators and targets. A conventional bus cannot offer such a throughput as it can only serve one communication at time. Thus we replace it with a NoC, which provides the required throughput. The NoC prevents us from using a snoop mechanism to ensure data coherency.

The communication protocol is based upon a shared memory multi-processor architecture. All MWMR channels are mapped in shared memory and access is protected by a single lock (one lock per channel). Each channel may have several readers and writers, but ignores their number. As illustrated by the following write request, the MWMR protocol requires five steps:

1. get the lock protecting the MWMR (READ access).
2. test the status of the MWMR (READ access).
3. transfer a burst of data between a local buffer and the MWMR (READ/WRITE access).
4. update the status of the MWMR (WRITE access).
5. release the lock (WRITE access).

For performance reasons, all transfers to or from a MWMR channel must be a multiple of 32 bit words. Figure 3 shows the hardware architecture with two processors and one memory bank. On the TTY the progress of the application can be observed, the Locks Engine manages the locks when more than one initiator is present (here four initiators: two processors and two coprocessors). These two components are VCI targets. The MWMR channel is located in on-chip RAM; it implements a communication channel between a software task running on CPU0 and a hardware task executed by coprocessor 1.
For performance reasons, MWMR channels are located in cacheable memory. Their coherency is guaranteed by software mechanism. Each cache line containing MWMR channel data is invalidated twice: before the five step access to ensure that data is read from memory and after the access to ensure that memory is updated. The latter is only necessary in the case of a write back cache mechanism.

4 The Telecommunication Platform

Until now, we have presented the generic hardware architecture. The telecommunication platform can be obtained by replacing the two coprocessors in Figure 3 by two application specific coprocessors called InputEngine and OutputEngine.

As usual in the domain of network processors, packets are cut into chunks of equal size which can be handled more efficiently and economize on-chip memory [4]. The basic idea of our coprocessors is that apart from this economy of space, time can be economized, too. A packet is represented by an eight byte descriptor, containing only a pointer to the beginning of the packet and the mandatory information to retrieve it. Necessary data are the address of the next slot, the total size of the packet, and an offset for eventual additional headers like for Multi Protocol Label Switching [15]. This leaves 120 bytes for the payload and an eventual offset. The use of descriptors allows us to avoid the copying of packets in memory most of the time. Consequently, the MWMR channels contain only descriptors.

Thus, I/O coprocessors must have a MWMR interface to send and retrieve descriptors as well as a VCI interface, the latter directly connected to the on-chip interconnect. SoCLib components are required to have a VCI interface, while coprocessors use FIFO interfaces. Thus, to implement MWMR channels a hardware wrapper with two VCI interfaces is required: a target interface for configuration and an initiator interface to fetch descriptors from the software channels located in on-chip RAM by issuing VCI requests. This wrapper is completely transparent for the user of DSX. Figure 4 shows on the lower right an InputEngine with one outgoing VCI and three FIFO channels. InputEngine and OutputEngine are symmetric concerning the interfaces.

A fundamental assumption of our architecture is that for a large majority of networking applications it is sufficient to consider the beginning of a packet. We privilege this first slot in so far as we store it in on-chip RAM, taking into account that this penalizes applications such as encryption/decryption or content control. Nevertheless the policy can be modified easily as only one bit has to be manipulated in order to determine whether a slot is to be stored on-chip or not.

5 The Classification Application

Classification is an important and resource-consuming part of many telecommunication applications [4] which
takes place just after a packet arrives at the framer chip (input coprocessor in networking terminology). Packet headers are analyzed and sometimes more in-depth analysis, useful for traffic management, is performed on the packet content. The packet is then sent on to one of several priority queues. We restrict to header analysis; more precisely, the destination address determines to which of twelve priority queues the packet will go.

Besides input and output tasks, there are two levels of tasks, which makes the parallelism expressed in our task graph a hybrid of pipelined and task farm parallelism (see section 2). There is also a bootstrap task that organizes startup. Figure 7 shows the task graph of the classification application. In the following we will describe the five different types of tasks in some more detail.

**Input Task**  The input task reads a packet from a file, determines its size, performs some basic checks, then computes the number of slots required taking into account the offset, and finally copies the slots to internal and external memory, respectively. In a last step, the slot is generated, and only its eight byte descriptor is sent to the outgoing MWMR channel, thus limiting the size of the memory allocated to the FIFOs. This write is non-blocking: if the channel is full, the packet is dropped and its address recycled. Addresses for the slots are obtained from either of three sources. Bootstrap: in the beginning, the input task needs to have a pool of internal and external addresses to allocate the slots to. Output task: when a packet leaves the system without having been dropped or discarded, the addresses of all its slots can be liberated. Classification task: errors are detected and packets can be discarded.

**Classification Task**  The classification task reads one or more descriptors and suspends itself if this is not possible. When it succeeds, it reads the first slot from on-chip memory. The validity of the checksum bits contained in the header is verified. The packet has to be dropped if one of those checks fails. Each slot begins with a descriptor containing the address of the next slot and one INTERNAL bit. Deallocation proceeds along these addresses from one slot to the next until the last slot is reached (next slot address is NULL). Next, depending on its destination address, the outgoing IP route is determined by consulting a routing table. This table is located in a shared memory segment so that all classification tasks can have a local copy in the cache of the processor on which they run. Update is assured by having the classification task sending a signal to all other classification tasks to invalidate their data cache. Our routing tables are relatively small: for lookup a simple iteration loop suffices. For fast lookup of larger routing tables, specific hardware is required. Finally the classification task writes the descriptor to one of the twelve priority queues.

**Scheduling Task**  The scheduling task uses algorithm which ponders by the priority of the current queue and the number of packets that have already left for the output task in order to reach the percentage assigned (example: the highest priority queue is read 40% of the time in the long run, the lowest 5% etc.). The FIFOs are tested for eligibility in a round robin manner, necessitating a non blocking read primitive in order not to suspend execution on an empty queue. The descriptor of the eligible packet is then written to the unique output queue.

**Output Task**  The output task is a coprocessor and as such a VCI initiator. It constantly reads the output queue of descriptors and blocks if this queue is empty. The address contained in the first part of the descriptor tells the memory location of the first slot. The address of the descriptor contained in the first eight byte of this slot tells the address of the second slot, and so forth. A buffer contains the packet during reconstitution. Finally, the packet is written to an

---

1For reasons of readability, Figure 7 shows the writing of liberated addresses to their respective FIFOs only for the last classification task.
output file, with optional information on latency and other data useful for performance evaluation. Each time a slot is read and sent to the buffer its released address is sent to either of the two internal or external address FIFOs.

**Bootstrap Task** This task is responsible for everything having to do with the startup of the application, originally located in the application main(). The input task will need addresses to which it can allocate the first and following slots, otherwise the packet treatment cannot start. The bootstrap task is responsible for this; Figure 7 shows that it writes into two FIFOs: addresses of internal slots into the FIFO slint and addresses of external slots into the FIFO slext. The generated addresses have 128 bytes distance between them, to accommodate one slot.

6 **DSX Design Space Explorer**

DSX describes the task graph in a completely static manner. The number of tasks is fixed, also the association of channels to ports and their sizes. The management of software tasks which are implemented as POSIX threads created in the main() function of embedded code is automatic, the configuration of the MWMR wrappers is also done automatically. Debug messages are filtered through different levels of verbosity. Furthermore, the instantiation of the SoCLib components and the rather error-prone task of connecting all signals to their respective ports in the SystemC netlist is automatically generated and has completely disappeared from the designer’s immediate view. The designer provides a descriptions of the hardware, the task graph and the mapping, in the Python language [13].

The following few lines describe most of the mono processor architecture. It includes the description of cached and uncached memory segments. The last five lines describe the interconnection of interfaces between processor and cache and around the VCI interconnect where the cache is initiator and all others are targets.

```python
def architecture(self):
    vgmn = Vgmn("vgmn",
            latency = self.getParam('min_latency'))
    cache = Xcache('cache1',
            dcache_lines=64, dcache_words=8,
            icache_lines=1024, icache_words=16)
    mips0 = Mips('mips0')
    cram0 = Segment('cram0',Cached)
    uram0 = Segment('uram0',Uncached)
    reset = Segment('reset',Cached,addr=0xbfc00000)
    excep = Segment('excep',Cached,addr=0x80000080)
    ram0 = MultiRam('ram0',cram0,uram0,reset,excep)
    locks = Locks('locks')
    tty = MultiTty('tty', 'tty0')
    mips0.cache // cache.cache
    cache.vci // vgmn.getTarget()
    ram0.vci // vgmn.getInit()
    locks.vci // vgmn.getInit()
    tty.vci // vgmn.getInit()```

Each task requires a short description of its model: name, ports, and some precisions on implementation. or instance, the InputEngine is defined by:

```python
input_task = TaskModel('input_engine',
            infifos = ['slint','slext'],
            outfifos = ['desc'],
            impl = SwTask('inputengine',
                stack_size=0',
                sources = ['src/input_engine.c'])
```
The Input Task reads from the FIFOs connected to the ports slint and slext and writes to the FIFO connected to the port desc. It exists in a software (keyword SwTask) and a hardware (keyword HwTask) implementation. The code of the software tasks has to be supplied by the programmer and indicated where it can be found (here, the src subdirectory). The define contains file names and constants. MWMR channels are connected to ports, forming the TCG.

tasks = Task('ie0', models.input_engine,
    portmap = {'slin':cin,'slext':cemt,
        'desc':cdesc},
    defines = {'FILE':"packets"}),

All Makefiles are generated automatically. DSX dimensions the memory space to what really is required by making a second compilation pass after mapping. Finally, if not demanded otherwise, it generates an executable for both a version consisting entirely of POSIX threads and a version with SoClib models of hardware coprocessors.

DSX provides both blocking and non blocking primitives to access MWMR channels. Non-blocking primitives are required when the scheduling task reads from a priority file if there is a descriptor available. The blocking read and write primitives return when the requested transaction is complete, i.e. the requested number of words was successfully read or written, whereas the corresponding non blocking functions will always return an integer that indicates the number of words that have been accessed, even if the request is not satisfied. If the returned number is less than required, it is up to the software task to decide.

In order to distinguish between internal and external address spaces, DSX allows to define memspaces. Memspaces are located on one memory segment, either cacheable or uncachable. Barriers ensure task synchronization. Like channels, memspaces and barriers are connected to ports, again their declarations are completely static. Strobe signals for starting up the hardware InputEngine and OutputEngine are adequately placed in the bootstrap task. These signals go to the MWMR wrapper, their names are deduced from coprocessor names. The bootstrap task runs only once, then suspends, whereas DSX tasks run continuously.

On the one hand the MIPS R3000 does not support multiple contexts, context switching is thus expensive. On the other hand the MIPS R3000 architecture is relatively simple, we can thus afford to add a large number of processors and allocate only one task per processor. For any task, its software objects are mapped to memory banks. For example, all classification tasks can be mapped to their respective processors and memory banks in a loop.

for i in range(nclassif):
    mapper.map("classify%d" % i,
    run = hardware.mips[i],
    stack = hardware.cram[i],
    desc = hardware.cram[i],
    status = hardware.uram[i],
    code = hardware.uram[i])

Lock, status and description of a MWMR channel or a barrier are equally mapped to memory banks. Note that the memory banks of the lock and the channel appear explicitly in the mapping, which is a significant improvement over SPADE where only tasks can be mapped. Memspaces are mapped to one memory bank each.

7 Validation and Performance Results

All hardware components are described by simulation models from the SoClib library [16]. A direct mapped write through cache policy is used for both the 16 Kbytes instruction cache and 512 bytes data cache. This article does not aim at design space exploration; the choices were guided by the results obtained in [7]. Performance results (Figure 9) were obtained for a mono processor platform with three memory banks (one for the application code, one for internal slots, one for external slots) running one task of each type, and two multiprocessor platforms running 20 classification, 3 scheduling and the bootstrap task, one task per processor. The first multiprocessor platform has only one level of interconnection network, around which the processors, eight memory banks, the TTY, lock engine and co-processors are grouped. For the clustered platform with its six processing clusters, the optimal mapping places the three scheduling tasks on the last cluster, together with all priority queues and the output channel (Figure 8). The input file contains 8,000 packets of 54 bytes, which is the minimal size for Ethernet encapsulated IPv4 packets: 40 bytes plus 14 bytes header. This represents the worst case for our type of platform: the number of headers to verify and of packets to classify is the highest possible for the volume of transmitted data.

The graphs and histograms in Figure 9 show the evolution of latencies of packet traversal. For better readability, the left hand diagrams show the evolution of mean latency taken over 50,000 simulation cycles. Packets are timestamped before leaving the InputEngine and these timestamps compared to the time they leave for the output file. Measurement begins once the bootstrap task has completed its work; to ensure this, we use barrier synchronization. Performances on the mono processor platform are weak as expected (Figure 9.a). The first packet arrives after 5 million cycles, then latencies are increasing without ever reaching a steady state. The channel between the input task and the classification task quickly overflows, leading to excessive

---

2DSX considers hardware coprocessor and software implementation as two implementations of the same task with the same functionality and identical interfaces.
packet loss (of 8,000, only 1,552 arrive). When scaling to 24 processors grouped around a one level interconnect, performances improve as expected, but are still not satisfactory. After reaching a steady state rapidly, they vary considerably around 450,000 cycles (Figure 9.b). Results for the clustered platform (Figure 9.c) are weaker than those obtained in [7]. A possible explanation is that the application code is around 15% larger (statically allocated space for the memory banks not counted) accounting for more misses in the instruction cache. A smaller peak around 230,000 cycles indicates packets for which the effect of NUMA was not compensated, neither do we observe a decrease of latencies after an initial phase.

8 Conclusion and Perspectives

With the help of DSX we achieved within less than a month the deployment of a multi threaded telecommunication application on several variants of our MPSoC platform. Our goal was to prove that DSX can handle task graphs that do not use the KPN semantic, as this kind of graph is often used in telecommunication applications.

Two telecommunication specific hardware coprocessors have been integrated into DSX in a first place. Some specific needs of our application were already met by DSX.

Reading from and writing to a such a large number of tasks efficiently is guaranteed by using MWMRs. Barrier primitives for synchronization and memspaces for placement of slots are part of the original version of DSX. Our application also requires mechanisms which enable tasks to run during a limited time only, whereas DSX tasks run continuously. The integration of a generic bootstrap task into DSX itself, which manages the entire initialization process (starting up coprocessors, initializing variables and filling up channels if required), is currently discussed. Primitives to ensure cache coherency by selective invalidation have been added. The impact of the larger DSX executable on the instruction cache will have to be investigated in more detail. For performance reasons, the code of the tasks should be replicated on each cluster, which is not yet possible.

With these adaptations, DSX will significantly ease the addition of application specific hardware coprocessors such as table lookup and cryptography.

References

Figure 9. Evolution (left) and histograms (right) of packet latencies: (a) mono processor (b) 24 processor non clustered (c) 6*4 processor clustered platform


Application of Advanced Logic Synthesis in FPGA-based Implementations of Digital Filters

Mariusz Rawski  
Warsaw University of Technology  
Warsaw, Poland  
Email: rawski@tele.pw.edu.pl

Paweł Tomaszewicz  
Warsaw University of Technology  
Warsaw, Poland  
Email: p.tomaszewicz@tele.pw.edu.pl

Bogdan J. Falkowski  
Nanyang Technological University  
Singapore  
Email: efalkowski@ntu.edu.sg

Tadeusz Łuba  
Warsaw University of Technology  
Warsaw, Poland  
Email: luba@tele.pw.edu.pl

Abstract—General functional decomposition has been gaining more and more importance in recent years. Though it is mainly perceived as a method of logic synthesis for implementation of Boolean functions into FPGA-based architectures, it has found application in many other fields of modern engineering and science. In this paper, application of logic decomposition to efficient implementation of digital filters in modern FPGA architectures is described. We have adapted an algorithm called Demain that can be used as a preprocessing tool in the FPGA synthesis flow. Wavelet filters are used to test Demain with Altera's Stratix device family and the experimental results are compared with those obtained by Altera's Quartus tool. When Demain is used as a premapping processing, it shows 50% improvement in terms of the area covered by embedded memory blocks.

I. INTRODUCTION

Traditionally, digital signal filtering algorithms are being implemented using general-purpose programmable DSP chips. Alternatively, for high-performance applications, special-purpose fixed function DSP chipsets and application-specific integrated circuits (ASICs) are used. Typical DSP devices are based on the concept of RISC processors with an architecture that consists of fast array multipliers. In spite of using pipeline architecture, the speed of such implementation is limited by the speed of array multiplier. Digital filters are implemented in such devices as multiply-accumulate (MAC) algorithms [9]. However, the technological advancements in Field Programmable Gate Arrays (FPGAs) in the past decade have opened new paths for DSP design engineers.

The FPGA maintains the advantages of custom functionality like an ASIC while avoiding the high development costs and the inability to make design modifications after production. The FPGA also adds design flexibility and adaptability with optimal device utilization while conserving both board space and system power, which is often not the case with DSP chips. When a design needs to use digital signal processing and time-to-market is critical, or design adaptability is crucial, then FPGA may offer a better solution.

Particularly FPGAs with embedded memories are well suited for arithmetic operation, including Multiply&Accumulate (MAC) intensive digital signal processing functions. A wide range of arithmetic functions (such as fast Fourier Transform’s (FFT’s), convolutions, and other filtering algorithms) can be integrated with surrounding peripheral circuitry. The FPGA can also be reconfigured on the fly to perform one of many system level functions. When building a Digital Filter in an FPGA, the design can take advantage of parallel structures and Distributed Arithmetic (DA) algorithms to exceed the performance of multiple general-purpose DSP devices. While the FPGA has no dedicated multiplier, the use of Distributed Arithmetic for array multiplication in an FPGA is one technique used to implement and increase the function’s data bandwidth and throughput by several order of magnitudes over off-the-shelf DSP solutions [9].

DA approach significantly increases the performance of implemented filter, by removing general purpose multipliers and introducing combinational blocks that implement the scaling. These blocks have to be efficiently mapped onto FPGA's logic cells. This can be done with the use of such advanced synthesis methods as functional decomposition [14], [16].

In the case of applications targeting FPGA structures based on look-up tables (LUT), the influence of advanced logic synthesis procedures on the quality of hardware implementation of signal and information processing systems is especially important. Direct cause of such a situation is the imperfection of technology mapping methods that are widely used at present, such as minimization and factorization of Boolean function, which are traditionally adapted to be used for structures based on standard cells. These methods transform Boolean formulas from sum-of-products form into multilevel, highly factorized form that is then mapped into LUT cells. This process is at variance with the nature of LUT cell, which from the logic synthesis’ point of view is able to implement any logic function of limited input variables. For this reason, in the case of implementation targeting FPGA structure, decomposition is a much more efficient method. Decomposition allows synthesizing the Boolean function into multilevel structure that is
built of components, each of which is in the form of LUT logic block specified by truth table. Efficiency of functional decomposition has been proved in many theoretical papers [1], [2], [12], [17]. However, there are relatively few papers where functional decomposition procedures were compared with analogous synthesis methods used in commercial design tools. The reason behind such a situation is the lack of appropriate interface software that would allow transforming description of project structure obtained outside commercial design system into description compatible with its rules. Moreover, the computation complexity of functional decomposition procedures makes it difficult to construct efficient automatic synthesis tool. These difficulties – at least partially – have been eliminated in so called balanced decomposition [8], [10], [13].

In the paper, first the basic information concerning balanced decomposition is presented. Following that digital filters are discussed with focus on application of distributed arithmetic concept to their implementation. Next, the results of balanced decomposition application in implementation of digital filters in logic cell resources of FPGA devices are presented. The results of application of decomposition in digital filters implementation in embedded memory blocks are also shown.

II. BALANCED FUNCTIONAL DECOMPOSITION

Balanced decomposition relies on partitioning of a switching function with either serial decomposition or parallel decomposition applied at each phase of the synthesis process. In the serial decomposition (Fig. 1), the set of input variables $X$ is partitioned into subsets, $A$ and $B$, and functions $G$ and $H$ are derived so that the set of input variables of $G$ is $B \cup C$, where $C$ is a subset of $A$, the set of input variables of $H$ is $A \cup Z$, where $Z$ is the set of output variables of $G$, and $H$ has fewer input variables than the original function $F$, i.e. $F = H(A, G(B, C))$.

In the parallel decomposition (Fig. 2), the set of output variables $Y$ of a multi-output function $F$ is partitioned into subsets, $Y_g$ and $Y_h$, and the corresponding functions, $G$ and $H$, are derived so that, for either of these two functions, the input support contains fewer variables than the set of input variables $X$ of the original function $F$. An objective of the parallel decomposition is to minimize the input support of $G$ and $H$.

Intertwining of serial and parallel decomposition strategies opens up several interesting possibilities in multilevel decomposition. Experimental results show that the right balance between these two strategies and the selection of a set of bound variables in serial decomposition severely influence the area and depth of the resultant network. The overall performance can be significantly improved by applying non-disjoint serial decomposition.

A. Example: Application of balanced decomposition

The influence of the right balance on the final result of the FPGA-based mapping process will be explained with the function $F$ given in Table I, for which cells with 4 inputs and 1 output are assumed (this is the size of Altera’s FLEX, Cyclone and Stratix FPGAs). As $F$ is a ten-input, two-output function, in the first step of the decomposition either parallel or serial decomposition can be applied. If we apply serial decomposition at first, then the resultant network (Fig. 3) can be built of 7 logic cells (LCs).

The same function decomposed with parallel decomposition as the first step shown in Fig. 4 – leads to a completely different structure. Parallel decomposition applied directly to
function $F$, generates two components both with 6 inputs and one output. Each of them is subject to two-stage serial decomposition, so the obtained network can be built of 4 L
cells.

Such a considerable impact on the structure results from the fact that the parallel decomposition simultaneously reduces the number of inputs to both the resulting components, leading to an additional improvement in the final representation.

It is worth noticing that the same function synthesized directly by commercial tool e.g. Quartus can be mapped onto 32 logic cells.

The above presented method was implemented as software package called DEMAIN [10]. Recently the package was improved to help designers to deal with large truth tables. All described methods of truth tables transformations can be performed easily, and results are shown immediately on the screen for further work. It is designed for performing manual operations on functions, and therefore is meant to be maximum user friendly, as well as cross-platform compatible.

After choosing the operation, a dialog pops up which can be used to input the parameters of the operation (for example in the case of serial decomposition it allows to choose the number of $G$ block inputs and outputs). After the actual operation is performed, its results are displayed in the project window. The results can be interpreted in the form of a tree of operations. For example, Fig. 5 shows the results of decomposition process for example function decomposed with a strategy where parallel decomposition is performed at first (see Fig. 4).

The structure can be understood as follows. The top entry $F(10, 2)$ indicates the number of inputs and outputs of the primary function. In the first iterative step, parallel decomposition is applied to split the $F(10, 2)$ into two $H(6, 1)$ blocks. This is illustrated by two arrow marks with the common starting point going to different directions. In the second step each of the obtained components is decomposed serially. Two brackets, which are given on the bottom of the arrows show the number of inputs and outputs for functions $G$ and $H$.

III. DIGITAL FILTERS

In recent years digital filters have been recognised as primary digital signal processing operation [5]–[7]. They are widely used in many tasks of image and signal processing systems, e.g. wavelet analysis is being successfully used in the area of speech and image coding, as well as signal detection and identification, and will be adopted in many emerging standards, starting with the compression standard JPEG2000 [11], [15].

Digital filters are typically used to modify the attributes of a signal in the time or frequency domain through a process called linear convolution [9]. This process is formally described by the following formula:

$$y[n] = x[n] * f[n] = \sum_k x[k] \cdot f[n - k] = \sum_k x[k] \cdot c[k],$$

(1)

where the values $c[k] \neq 0$ are called the filter’s coefficients.
There are only a few applications (e.g. adaptive filters) where general programmable filter architecture is required. In many cases the coefficients do not change over time - linear time-invariant filters (LTI). Digital filters are generally classified as being finite impulse response (FIR) or infinite impulse response (IIR). As the names imply, an FIR filter consists of a finite number of sample values, reducing the above presented convolution to a finite sum per output sample. An IIR filter requires that an infinite sum has to be performed. In this paper implementation of the LTI FIR filters will be discussed.

The output of an FIR filter of order (length) \( L \), to an input time-samples \( x[n] \), is given by a finite version of convolution sum:

\[
y[n] = \sum_{k=0}^{L-1} x[k] \cdot c[k].
\]  

(2)

The \( L^{th} \) order LTI FIR filter consists of a collection of delay line, adders and multipliers.

Many digital filter software are available that enable very easy computation of coefficients for a given filter. However, the challenge is mapping of the FIR structure into suitable architecture. Digital filters are typically implemented as multiply-accumulate (MAC) algorithms with the use of special DSP devices.

Efficient hardware implementation of a filter’s structure in programmable devices is possible by optimizing the implementation of multipliers and adders. In modern programmable structures specialized embedded blocks can be used to implement multipliers, increasing the performance of the designed system. Moreover, in the case of Altera’s devices, a whole MAC unit can be implemented in embedded DSP block, making the design methodology very similar to the one used in the case of DSP processors.

In the case of programmable devices, however, direct or transposed forms are preferred for maximum speed and lowest resource utilization. This is because of the fact that the approach enables exploitation of prevalent parallelism in the algorithm.

A completely different FIR architecture is based on the distributed arithmetic concept. The distributed arithmetic method is a method of computing the sum of products. In many digital signal processing applications, a general purpose multiplication is not required. In the case of filter implementation, if filter coefficients are constant in time, then the partial product term \( x[n] \cdot c[n] \) becomes a multiplication with a constant. Then, taking into account the fact that the input variable is a binary number:

\[
x[n] = \sum_{b=0}^{B-1} x_b[n] \cdot 2^b,
\]  

(3)

where \( x_b[n] \in [0, 1] \), the whole convolution sum can be described as shown below.

\[
y[n] = \sum_{b=0}^{B-1} 2^b \cdot \sum_{k=0}^{L-1} x_b[k] \cdot c[k] = \sum_{b=0}^{B-1} 2^b \cdot \sum_{k=0}^{L-1} f(x_b[k], c[k]).
\]  

(4)

The efficiency of filter implementation based on this concept strongly depends on the implementation of the function \( f(x_b[k], c[k]) \). The preferred implementation method is to the mapping \( f(x_b[k], c[k]) \) as the combinational module with \( L \) inputs. The schematic representation of DA filter structure is shown in Fig. 6, where the map \( f \) is presented as a lookup table that includes all the possible linear combinations of the filter coefficients and the bits of the incoming data samples [9]. The utility programs that generate the lookup tables for filters with given coefficients can be found in the literature.

In the experiments presented in this paper a variation of DA architecture has been used. It increases the speed of a filter at the expense of additional LUTs, registers and adders. The basic DA architecture, for computing the sum-of-products, accepts one bit from every \( L \) input words. The computation speed can be significantly increased by accepting for the computation more bits of each word. Maximum speed can be achieved with a fully pipelined parallel architecture as shown in Fig. 7. Such an implementation can outperform all commercially available programmable signal processors.

The HDL specification of the lookup table can be easily obtained for the filter described by its \( c[i] \) coefficients. Since the size of lookup tables grows exponentially with the number of inputs, efficient implementation of these blocks becomes crucial to the final resource utilization of filter implementation. Here advanced synthesis methods based on balanced decomposition can be successfully applied for technology mapping of DA circuits onto FPGA logic cells.

IV. EXPERIMENTAL RESULTS FOR CELL-BASED FPGAS

Experimental results for FIR filter implementation with different design methodologies are presented in this section. Filter with length (order) 15 has been chosen for the experiment. It has 8-bit signed input samples and its coefficients can be found in [4]. For comparison, the filter has been implemented in Stratix EP1S10F484C5, Cyclone EP1C3T100C6 and CycloneII EP2C5T144C6 structures using Altera Quartus2.
Table II presents the comparison of implementation results for different design methodologies. Column falling under the "MAC" label presents the results obtained by implementing the multiply-and-accumulate strategy with the use of logic cell resources, without utilizing the embedded DSP blocks. Multipliers, as well as accumulators, were implemented with use of logic cells. This implementation, due to its serial character, requires 15 clock cycles to compute the result. It requires relatively large amount of resources, while delivering the worst performance in comparison to other implementations.

Next column – "MULT block" – holds the implementation results of a method similar to "MAC" with such a difference that multipliers were implemented in dedicated DSP embedded blocks. It can be noticed that the performance of the filter increased at the cost of additional resources in form of DSP embedded blocks. Results in the column falling under "DSP block" were obtained by implementing the whole MAC unit in the embedded DSP block. Further increase in performance could be noticed, but still 15 clock cycles have to be used to compute the result.

Results given in "Parallel" column were obtained by implementing the filter in a parallel manner. In this case, the filtering results are obtained in a single clock cycle. Even though, the maximal frequency of this implementation is less than in the previous ones, it outperforms these implementations due to its parallel character.

Application of the DA technique results in the increased performance since the maximal frequency has increased. However, in this approach more logic cell resources have been used, since multipliers have been replaced by large combinational blocks and no DSP embedded modules have been utilized.

Finally, results presented in the column under "DA decomposed" label demonstrate that the application of the DA technique combined with advanced synthesis method based on balanced decomposition results in a circuit that not only outperforms any other implemented circuit but also reduces the necessary logic resources. The balanced decomposition technique was applied to decompose the combinational blocks of the DA implementation.

In Table III, the experimental results of 9/7-tap bio-orthogonal filter banks are presented. Filters 9/7 are in two versions: (a) analysis filter and (s) synthesis filter. All filters have 16 bit signed samples and have been implemented with the use of distributed arithmetic concept. Balanced decomposition software was also added to increase efficiency of the DA tables’ implementations.

Table III presents the result for filter implementations using Stratix EP1S10F484C5 device, with a total count 10,570 of logic cells. In the implementation without decomposing the filters, the new method was modeled in AHDL and Quartus2 was used to map the model into the target structure. In the implementation using decomposition, the automatic software was used to initially decompose DA tables and then Quartus system was applied to map the filters into FPGA.

V. EXPERIMENTAL RESULTS FOR EMBEDDED ROMS

Modern FPLD devices have very complex structure. They combine PLA-like structures as well as FPGA’s and even memory-based structures. In many cases designers cannot utilize all of these possibilities such complex architectures provide due to the lack of appropriate synthesis methods. Embedded memory arrays make possible an implementation of memory like blocks such as large registers, FIFO’s, RAM or ROM modules [9].

These memory resources make up considerably large part of the devices, i.e., EP20K1500E devices provides 51,840 logic
cells and 442 Kbit of SRAM. Taking under consideration the conversion factors of logic elements and memory bits to logic gates (12 gates/logic element and 4 gates/memory bit) it turns out that embedded memory arrays make up over 70% of all logic resources. Since not every design consists of such modules as RAM or ROM, in many cases, these resources are not utilized, though. However, such embedded memory blocks can be used for implementation of DA blocks in a way that requires less resources than the traditional cell-based implementation. This may be used to implement “non-vital” combinational parts of the design, saving logic cell resources for more important sections. Since the size of embedded memory blocks is limited, such an implementation may require more memory than is available in a device. To reduce a memory usage in ROM-based DA implementations, a structure with combinational logic partially implemented in the ROM and partially implemented in logic cells was proposed.

In Table IV, the experimental results of Daubechies’ 9/7-tap bio-orthogonal filter banks are presented. All filters have 16 bit signed samples and have been implemented with the use of distributed arithmetic concept. Balanced decomposition software was also added to increase efficiency of the DA tables’ implementations [3].

Table IV presents the results for filter implementations using Stratix EP1S10F484C5 device. In the implementation without decomposing the filters, the method was modeled in HDL and Quartus2 was used to map the model into the target structure. In the implementation using decomposition (denoted dec), DEMAIN software was used to initially decompose DA tables and then Quartus system was applied to map the filters into FPGA.

Filters 9/7 are in two versions: (a) analysis filter and (s) synthesis filter. The implementation of filters is characterized by the number of logic cells (LCs) and Flip-Flops (FFs), memory bits, the number of memory modules (ROMs) and operating frequency. In all cases, decomposition reduces the sizes of memory and the number of memory modules. For example, implementation of ahp filter requires 204 LCs and 4 M512 embedded memories if performed by Quartus software. Application of DEMAIN tool allows implementing DA logic of this filter (Fig. 8) with 2 M512 memories and 11 LCs. This allows implementing the whole filter with 210 LCs and 2 M512 memories.

The structure of the balanced decomposition process for DA function of ahp filter is shown on Fig. 9.

<table>
<thead>
<tr>
<th>Filter</th>
<th>Order</th>
<th>LCs</th>
<th>FFs</th>
<th>bits</th>
<th>ROMs</th>
<th>$f_{\text{max}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>9/7, a low-pass</td>
<td>9</td>
<td>236</td>
<td>181</td>
<td>8192</td>
<td>7×M512, 1×M4K</td>
<td>133.51</td>
</tr>
<tr>
<td>9/7, a low-pass dec</td>
<td>9</td>
<td>248</td>
<td>181</td>
<td>4096</td>
<td>1×M4K</td>
<td>140.51</td>
</tr>
<tr>
<td>9/7, a high-pass</td>
<td>7</td>
<td>204</td>
<td>149</td>
<td>2048</td>
<td>4×M512</td>
<td>155.04</td>
</tr>
<tr>
<td>9/7, a high-pass dec</td>
<td>7</td>
<td>210</td>
<td>153</td>
<td>1024</td>
<td>2×M512</td>
<td>157.53</td>
</tr>
<tr>
<td>9/7, s low-pass</td>
<td>7</td>
<td>204</td>
<td>149</td>
<td>2048</td>
<td>4×M512</td>
<td>155.04</td>
</tr>
<tr>
<td>9/7, s low-pass dec</td>
<td>7</td>
<td>211</td>
<td>153</td>
<td>1024</td>
<td>2×M512</td>
<td>161.21</td>
</tr>
<tr>
<td>9/7, s high-pass</td>
<td>9</td>
<td>236</td>
<td>181</td>
<td>8192</td>
<td>7×M512, 1×M4K</td>
<td>133.51</td>
</tr>
<tr>
<td>9/7, s high-pass dec</td>
<td>9</td>
<td>246</td>
<td>181</td>
<td>4096</td>
<td>1×M4K</td>
<td>134.25</td>
</tr>
</tbody>
</table>
VI. CONCLUSION

The modern programmable structures deliver the possibilities to implement digital signal processing algorithms in dedicated embedded blocks. This makes designing of such algorithms an easy task. However flexibility of programmable structures enables more advanced implementation methods to be used. In particular, exploitation of parallelisms in the algorithm to be implemented may yield very good results. Additionally, the application of advanced logic synthesis methods based on balanced decomposition, which is suitable for FPGA structure leads to results that can not be achieved with any other method.

The presented results lead to the conclusion that if the designer decides to use the methodology known from DSP processor application, the implementation quality will profit from the utilization of specialized DSP modules embedded in the programmable chip.

However, best results can be obtained by utilizing the parallelisms in implemented algorithms and by applying advanced synthesis methods based on decomposition. Influence of the design methodology and the balanced decomposition synthesis method on the efficiency of practical digital filter implementation is particularly significant, when the designed circuit contains complex combinational blocks. This is a typical situation when implementing digital filters using the DA concept.

The most efficient approach to logic synthesis of FIR filter algorithms discussed in this paper relies on the effectiveness of the functional decomposition synthesis method. These methods were already used in decomposition algorithms; however they were never applied together in a technology specific mapper targeted at a look-up table FPGA structure. This paper shows that it is possible to apply the balanced decomposition method for the synthesis of FPGA-based circuits directed towards area or delay optimization.

ACKNOWLEDGMENT

This paper was supported by Ministry of Science and Higher Education financial grant for years 2006-2009 (Grant No. SINGAPUR/31/2006) as well as Agency for Science, Technology and Research in Singapore (Grant No. 0621200011).

REFERENCES


Hardware implementation of a multi-mode hash architecture for MD5, SHA-1 and SHA-2

Sylvain Ducloyer, Romain Vaslin, Guy Gogniat, Eduardo Wanderley
LESTER CNRS FRE 2734
Lorient, FRANCE
name@univ-ubs.fr

Abstract— In this paper, we propose a unified architecture adapted to the field of embedded systems which combines commonly used hash algorithms in a single design in order to reduce area requirements and optimize the maximum frequency. We present an implementation of three hash functions: MD5 [8], SHA-1 [9] and SHA-2 [10]. Many similarities exist between these algorithms which help us to move towards a unique architecture. The design was implemented on an Altera Stratix II device. It only requires 3311 ALUTs and operates at a frequency of 93 MHz which provides a throughput of 567 Mbps for MD5/SHA-2 and a 476 Mbps throughput for SHA-1 (including padding operations).

I. INTRODUCTION

Cryptography field has been strongly active these last years, essentially in consequence of an increasing need for security issues. For example, most of the digital communications in mobile or network applications require some protections against potential threats (virus, worms and hackers).

Hash functions are well known cryptographic solutions used to guarantee message integrity (see Figure 1). Several hash algorithms exist, however MD5 and SHA-1 are currently the most used. Many processor-based solutions have already been proposed but they are not adapted to the specific constraints of embedded systems (efficiency, area, power consumption).

There are different ways to implement those kinds of protection in a system. The first one is using a coprocessor for each algorithm. In this case the area overhead will be significant. The second solution would be to use a reconfigurable hardware device like FPGA and to dynamically reconfigure the FPGA for switching from one algorithm to another. It means that the system will need extra memory to store the different configurations for the FPGA.

The last alternative is a configurable coprocessor which provides a few different configurations for those algorithms. This solution provides a trade-off with the two previous solutions presented. Due to the common features of MD5, SHA-1 and SHA-2, it is possible to minimize the area required to implement these algorithms, to limit the storage (one FPGA context instead of three) and to provide flexibility. For SHA-2 we focus our work on SHA-224/256 version because they are based on a 32 bits datapath.

In this paper, we present the first hardware architecture for integrity checking based on three hash functions MD5, SHA-1 and SHA-2. The paper is organised as follows: in section II, hash functions are presented with hash algorithm descriptions. The proposed architecture is detailed in section III. Section IV focuses on throughput, area results and maximum frequency. Finally, related work is presented in order to compare our architecture with previous works in section V.

II. HASH FUNCTIONS

Hash functions are designed to obtain a signature depending on an initial message. In addition, the functions are one-way-function in order to generate a unique fixed-length bit vector as an output. This signature is then sent with the message. The addressee will check the message integrity by comparing the received signature with the one he will compute based on the received message (Figure 1).

In our case, we combine three hash functions MD5, SHA-1 and SHA-224/256 in a configurable global architecture. In order to obtain a signature corresponding to a message, we perform several steps: a pre-processing of the message and
then the hash computation. The 3 selected algorithms rely on
the same way to obtain the signature. The pre-processing
phase is the same for all of them. The hash computation differs
for each algorithm.

A. Pre-processing of the message

The first step is padding. The goal of padding is to obtain a
message that will have a size modulo 512 bits. It needs be
modulo 512 bits because it is the size of the input of these al-
gorithms. The last 64 bits of the padded message are reserved
to store the size of the initial message. Figure 2 shows an ex-
ample of a padded message. To fill the gap between the end of
the original message and the last 64 bits, a bit set to 1 is in-
serted with a succession of bits set to 0.

![Fig.2 Padding](image)

The second step is to fill the last 64 bits with the size of the
message. In some cases, there are not enough bits left to store
the length (an example is a message of 1008 bits). Then a new
512 bits word will be created. This word will be filled with
some 0 until the space reserved to store the size is reached.

As soon as the pre-processing ends, the message is ready to
be hashed. As it is said previously, the pre-processing phase is
the same for MD5, SHA-1 and SHA-2.

![Fig.3 Initialization of buffers values](image)

B. Hashing of the message

It is the heart of a hash algorithm. During this step, the
computation part of the algorithm will be done. This module
consists of four processing rounds of 16 steps each for MD5
and SHA-2; 20 steps each for SHA-1. In this section, we detail
the different ways the signatures are computed. It will help to
extract the similarities between the 3 algorithms in order to
move towards a unified hardware architecture.

Each 512 bits word composing the message will pass
through the algorithm. At the beginning some buffers are ini-
tialized (see Figure 3 for values). When the first 512 bits word
is hashed, the second word is hashed but the buffers are not
reset to the initial values. The result from the previous 512 bits
word is used. The process is iterated until all the 512 bits
words have been hashed. In all the following text the letter W
will mean a 32 bits word from a 512 bits word. If an index
appears, it means for example the second word of the 512 bits
word (W₂). K is a different constant for every step t. The val-
ues of K can be found in the official algorithm RFC [8] [9] or
FIPS [10].

Concerning the SHA-2, as said previously we have focused
our work only on the SHA-224/256 version in order to sim-
plify the architecture. Indeed, the SHA-512 will require a 64
bits architecture.

1) MD5

MD5 (Message Digest) was developed by Rivest in 1991 [8]. It is a hash function which produces a digital signature of
128 bits for an arbitrary-length message. Each round the op-
erations of Equation 1 are performed. Table 1 summarizes all
the values of the parameters.

\[
T = B + (A + F_{md5}(B, C, D) + W_t + K_t) \ll S
\]

Variation of \( \ll \) depends on the round.

<table>
<thead>
<tr>
<th>Round</th>
<th>( F_{md5}(B, C, D) )</th>
<th>( S )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 &lt; t &lt; 15 → Round 1</td>
<td>( (B \text{ and } C) \text{ or } ((\text{not } B) \text{ and } D) )</td>
<td>7, 12, 17, 22</td>
</tr>
<tr>
<td>16 &lt; t &lt; 31 → Round 2</td>
<td>( (B \text{ and } D) \text{ or } (C \text{ and } (\text{not } D)) )</td>
<td>5, 9, 14, 20</td>
</tr>
<tr>
<td>32 &lt; t &lt; 47 → Round 3</td>
<td>( B \text{ xor } C \text{ xor } D )</td>
<td>4, 11, 16, 23</td>
</tr>
<tr>
<td>48 &lt; t &lt; 63 → Round 4</td>
<td>( C \text{ xor } (B \text{ or } (\text{not } D)) )</td>
<td>6, 10, 15, 21</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MD5</th>
<th>SHA-1</th>
<th>SHA-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 67452301</td>
<td>A = 67452301</td>
<td>A = 6a09e667</td>
</tr>
<tr>
<td>B = efcdba89</td>
<td>B = efcdba89</td>
<td>B = bb67ae85</td>
</tr>
<tr>
<td>C = 98badcfe</td>
<td>C = 98badcfe</td>
<td>C = 3c6ef372</td>
</tr>
<tr>
<td>D = 10325476</td>
<td>D = 10325476</td>
<td>D = a54ff53a</td>
</tr>
<tr>
<td>E = 00000000</td>
<td>E = c3d2e1f0</td>
<td>E = 510e527f</td>
</tr>
<tr>
<td>F = 00000000</td>
<td>F = 00000000</td>
<td>F = 9b05388c</td>
</tr>
<tr>
<td>G = 00000000</td>
<td>G = 00000000</td>
<td>G = 1f83d9ab</td>
</tr>
<tr>
<td>H = 00000000</td>
<td>H = 00000000</td>
<td>H = 5be0cd19</td>
</tr>
</tbody>
</table>

Tab.1: Parameters for MD5

2) SHA-1

SHA-1 (Secure Hash Algorithm) was developed by NSA
(National Security Agency) in 1993 [9]. It is a hash function
which produces a digital signature of 160 bits for an arbitrary-length message.

\[ T = (A << 5) + \text{Fsha1}(B,C,D) + E + W_t + K_t \]
\[ A = T \]
\[ B = A \]
\[ C = \text{left rotation of 30 bits on B} \]
\[ D = C \]
\[ E = D \]

Eq. 2: Hash operations of SHA-1

0 < t < 19 → Round 1  
\[ \text{Fsha1}(B,C,D) = (B \text{ and } C) \oplus (\neg B \text{ and } D) \]

20 < t < 39 → Round 2  
\[ \text{Fsha1}(B,C,D) = B \oplus C \oplus D \]

40 < t < 59 → Round 3  
\[ \text{Fsha1}(B,C,D) = (B \text{ and } C) \oplus (B \text{ and } D) \oplus (C \text{ and } D) \]

60 < t < 79 → Round 4  
\[ \text{Fsha1}(B,C,D) = B \oplus C \oplus D \]

Tab. 2: Parameters for SHA-1

\[ W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) << 1 \]

Eq. 3: Computation of \( W_t \) for \( t > 16 \)

3) SHA-2

SHA-2 (Secure Hash Algorithm) was developed by NSA in 2000 [10] because collisions have been found in SHA-1 which means that the algorithm is not secured enough. The SHA-224/256 version provides a digital signature of 224/256 bits for an arbitrary-length message. SHA-2 uses different shifts and constants, but its structure is almost identical to SHA-1.

\[ T_1 = H + \sum_0(E) + \text{Ch}(E, F, G) + W_t + K_t \]
\[ T_2 = \sum_0(A) + \text{Maj}(A, B, C) \]
\[ A = T_1 + T_2 \]
\[ B = A \]
\[ C = B \]
\[ D = C \]
\[ E = D + T_1 \]
\[ F = E \]
\[ G = F \]
\[ H = G \]

Eq. 4: Hash operations of SHA-2

\[ \text{Ch}(E, F, G) = (E \text{ and } F) \oplus (\neg E \text{ and } G) \]
\[ \text{Maj}(A, B, C) = (A \text{ and } B) \oplus (A \text{ and } C) \oplus (B \text{ and } C) \]
\[ \sum_0(A) = (A >> 2) \oplus (A >> 13) \oplus (A >> 22) \]
\[ \sum_t(E) = (E >> 6) \oplus (E >> 11) \oplus (E >> 25) \]

Eq. 5: Logical operations of SHA-2

Like in SHA-1, there are some operations to obtain \( W_t \). If \( t < 16 \), \( W_t \) is the \( t \) th 32 bits word of the message block. Else if \( t \geq 16 \), \( W_t \) is computed according to Equation 6.

\[ W_t = \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16} \]
\[ \sigma_0(W) = (W >> 6) \oplus (W >> 11) \oplus (W >> 25) \]

Eq. 6: Computation of \( W_t \) for \( t > 16 \)

SHR stands for shift right

III. HARDWARE ARCHITECTURE

There are many algorithmic similarities between MD5, SHA-1 and SHA-2. We have attempted to study the algorithms in order to implement them in a single design while minimizing the required resources and optimizing the throughput.

Figure 4 shows the unified architecture we propose. The architecture is divided in 6 blocks. The PADDING and FIFO blocks are not configurable because for all the algorithms the execution will be the same (see II-1). Concerning the BUFFER SCHEDULER, MESSAGE SCHEDULER and MACROFUNCTION, those blocks can be configured depending on the selected algorithm. The ROM stores all the constants used by the algorithms. With our solution we should limit the logic amount necessary to implement the 3 algorithms due to the architectural recovery between these algorithms.

A. Padding and storage

Padding enables to prepare the message for the hash as presented in section II – 1. After the pre-processing step, the
words are stored in a 512 bits FIFO. The architecture of padding and storage is designed according to Figure 5.

B. Message scheduler

This structure enables to select a 32 bits word in the word of 512-bits according to hash algorithms and the current executed round.

As presented in section II – 2, for MD5 no computation is required to get the 32 bits message to send for hashing. For SHA-1 and SHA-2, Equations 3 and 6 are implemented in the MESSAGE COMPUTATION block to obtain the computed value of the message. Figure 6 gives an overview of the MESSAGE SCHEDULER architecture. There is a large need of logic to control the system inputs. It helps to manage the configuration depending on the current algorithm running and the current round being executed.

C. Buffer scheduler

This function only enables a simple shift according to the algorithm selected. These buffers are used in all the computation process of the algorithm. At the beginning they will be initialized by the values presented in Figure 3.

D. Macro function

It is the heart of the architecture; indeed all calculations according to the algorithms (MD5, SHA-1 and SHA-2) are done by this architectural block. In this part we have extended the proposition of [6]. As the authors had limited their work to MD5 and SHA-1. Our proposition gives us the possibility to choose between three algorithms. The first step is to extract the computation similarities. In section II – 2, the logic equations to be performed with the different buffers (Table 1 and 2, Equation 5) are detailed.

<table>
<thead>
<tr>
<th>MD5</th>
<th>SHA-1</th>
<th>SHA-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>X xor Y xor Z</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>(X and Y) xor ((not X) and Z)</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>(X and Z) xor (Y and (not Z))</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Y xor (X or (not Z))</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>(X and Y) xor (X and Z) xor (Y and Z)</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Tab.3: Logical equations recovery between algorithms

Based on Table 3, we can extract the combinational function \( F_{comb} \) (Equation 7) which contains all the logical equations required to be able to perform the computation for the 3 algorithms.

\[
F_{comb} (X, Y, Z) = \begin{cases} 
(X and Z) or (Y and (not Z)) \\
X xor Y xor Z \\
Y xor (X or (not Z)) \\
(X and Y) xor ((not X) and Z) \\
(X and Y) xor (X and Z) xor (Y and Z)
\end{cases}
\]

Eq.7: combinational function
The second step toward a unified architecture is to propose a macro function (Tcomb) based on the Fcomb. Equation 8 shows the macro function. Table 4 summarizes the values of the function parameters depending on the algorithm running.

\[ T_{comb} = I + \left( J \ll S1 + Fcomb(X, Y, Z) + \sum (N) + W + K \right) \ll S2 \]

Eq.8: Macro function

For SHA-2, we need to have a second dedicated combina-
tional function (Equation 9) in order to have a result for T2 in Equation 4.

\[ T_{comb2} = \sum(a(A) + Maj(A, B, C)) \]

Eq.9: Dedicated SHA-2 function

<table>
<thead>
<tr>
<th></th>
<th>MD5</th>
<th>SHA-1</th>
<th>SHA-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>B</td>
<td>E</td>
<td>H</td>
</tr>
<tr>
<td>J</td>
<td>A</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>S1</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Fcomb (X,Y,Z)</td>
<td>(B,C,D)</td>
<td>(B,C,D)</td>
<td>(E,F,G)</td>
</tr>
<tr>
<td>( \sum (N) )</td>
<td>0</td>
<td>0</td>
<td>E</td>
</tr>
<tr>
<td>W</td>
<td>( W_t )</td>
<td>( W_t )</td>
<td>( W_t )</td>
</tr>
<tr>
<td>K</td>
<td>( K_t )</td>
<td>( K_t )</td>
<td>( K_t )</td>
</tr>
<tr>
<td>S2</td>
<td>S</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Tab.4: Parameters values depending on the algorithm

Figure 8 gives an overview of the macro function archite-
cture. The main output is OUT_T which is the Tcomb value. OUT_C is the result for BUFFER C with SHA-1. OUT_E provides the result of Tcomb2.

E. Unified architecture features

All architectural blocks described previously need to be as-
sociated with state machines to manage all the internal signals. FSM2 (see Figure 4) manages the signal for the hash part. FSM1 manages the activation of the PADDING and FIFO_512 blocks. Thus, in this top-level architecture, we can realize three hash functions MD5, SHA-1 and SHA-2 thanks to the Sel_Algo signal. Moreover, the proposed architecture allows a pipelined implementation. Indeed, during hash compu-
tation, the system can achieve padding for the next 512 bits word. The latency of the pre-processing is saved for each 512 bits word. Note that, this padding penalty will always be pre-
SENT at the beginning of the message hashing.

The total latency to hash a 512 bits word depends on the chosen algorithm. 66 cycles are required for MD5 and SHA-2. With these 2 algorithms, the architecture performs one round per clock cycle. It is the same with SHA-1 but it requires 82 cycles as there are about 80 rounds to be performed. 18 more cycles are necessary to do the pre-processing on a 512 bits word before starting to compute the hash. As said before, this latency can be overlapped by the hash computation in the case of a message with a size bigger than 512 bits.

The last point to notice concerns the potential weakness of our proposition. Due to the specificity of our proposition which mainly relies on the recovery between the 3 algorithms; we might expect to have a longer critical path. Indeed, the solution implements 3 algorithms. It means that if we compare the critical path with a dedicated application, the critical path should be longer and the design may operate at a lower fre-
frequency. It may limit the throughput of the architecture. The next section presents further details concerning this last point.

IV. RESULTS

The proposed architecture has been synthesized and imple-
mented on an Altera Stratix II - EP2S60F672C3 device. We obtained a maximum achieved operating frequency of 93 MHz for our architecture. The throughputs are 567 Mbps for MD5 and SHA-2, and 476 Mbps for SHA-1. The throughput is cal-
culated with Equation 10. All the throughput values presented in Table 5, have been obtained for 512 bits messages. It means that the time to perform padding and so on is included in the Number of clock cycles per block value. So the throughput value would be higher if the values were calculated for a 1024 bits word. In this case due to the way we have designed our architecture the pre-processing of the second 512 bits word
would have been overlapped by the hashing operations of the first 512 bits word.

Moreover, the loss concerning the throughput is very negligible. In the worst case (SHA-1), the throughput goes from 695 Mbps to 565 Mbps (-20%). This loss was expected because

$$\text{Throughput} = \frac{\text{Block size} \times \text{Max frequency}}{\text{Number of clock cycles per block}}$$

Eq.10: Equation to obtain the throughput of architecture

The first part of Table 5 gives a summary of relevant figures about our work. The optimized multi-hash core is the architecture which was presented in the previous sections. It clearly appears that the work done on the recovery of the algorithm gives very good results for the area of the design. The area is about 2 times less important than the non optimized architecture with 3 hash cores. Furthermore, we also provide figures of dedicated cores we developed in order to compare the gain of our proposition. The MD5 core is a quite small core compared to the SHA-1 and SHA-2 ones. The overhead added by the possibility to manage 3 algorithms with our architecture seems to be negligible as the size of a SHA-2 core is around 1380 ALUTs.

As discussed in section III – 5, due to the longer critical path, the frequency results are not always in favour of our proposition. There are some losses compare to SHA-1 and SHA-2 dedicated cores. But this loss is balanced by the gain in area. Some differences may also come from the technological difference between FPGA (ALTERA, Xilinx).

In [6], the authors propose an architecture for MD5 and SHA-1 based on the idea of the study recovery between the two algorithms. The extensions we propose to their work improve the results they have obtained. Our solution provides an efficiency of 0.43 and 0.34 compared to the 0.31 and 0.25 of [6]. In addition our solution is able to manage the SHA-224/256.

Another paper references a multi-hash core. In [11], authors implemented all the SHA-2 versions (224, 256, 384 and 512). Again our proposition provides a better efficiency even if we consume a little more area space. This difference mainly comes from the logic required for MD5 and SHA-1. Indeed, MD5 and SHA-1 architectures are very close and the amount of logic necessary to add SHA-2 to those 2 algorithms is important (especially T2 in equation 9 which does not have any recovery with other algorithms).

V. RELATED WOTK

In [6], the authors propose an architecture for MD5 and SHA-1 based on the idea of the study recovery between the two algorithms. The extensions we propose to their work improve the results they have obtained. Our solution provides an efficiency of 0.43 and 0.34 compared to the 0.31 and 0.25 of [6]. In addition our solution is able to manage the SHA-224/256.

Another paper references a multi-hash core. In [11], authors implemented all the SHA-2 versions (224, 256, 384 and 512). Again our proposition provides a better efficiency even if we consume a little more area space. This difference mainly comes from the logic required for MD5 and SHA-1. Indeed, MD5 and SHA-1 architectures are very close and the amount of logic necessary to add SHA-2 to those 2 algorithms is important (especially T2 in equation 9 which does not have any recovery with other algorithms).
If we have a look at the proposition done in [3], [4] and [5], it appears that the efficiency of our solution is similar or better than those solutions. The fact that we are able to manage 3 algorithms makes up for the little area overhead or the throughput loss.

In [7], an ASIC solution is proposed in 0.18µm and 0.13µm. Thus, these architectures propose the best performances. However the price of such architecture is really high and does not provide any flexibility.

The last comparison to do is with an architecture which uses dynamic reconfiguration. An advantage of such solution is the flexibility and the reduce needs for logic. Throughout this paper, we show how we provide flexibility for the algorithm. Furthermore, in Table 5, it is shown that the area required to gather the 3 algorithms is not important. It means that our solution can be considered to be a very good alternative to the solution with dynamic reconfiguration. Last important point, our solution does not need more memory. Indeed, with the dynamic reconfiguration, extra memory is necessary to store all the bitstreams for the 3 algorithms.

VI. CONCLUSIONS

In this paper, we have presented an architecture which gathers three hash functions MD5, SHA-1 and SHA-2. The large similarities between them lead us to study the algorithms in order to implement them in a single design.

The architecture which was implemented on an Altera Stratix II device only requires 1662 ALUTs and operates at 93 MHz. Moreover, we obtained throughput of 721 Mbps for MD5 and SHA-2 and 580 Mbps for SHA-1.

The comparison with previous works shows that our implementation is a good compromise for flexibility, area and throughput. It also confirms that the recovery between algorithms might lead to an optimized architecture. This point is essential due to the specific field of embedded systems with limited resources.

REFERENCES

Evaluation of dynamic partial reconfiguration in professional electronics applications

Philippe Manet¹
Philippe.manet@uclouvain.be

Marco Di Ciano⁴
m.diciano@tno.it

Jean-Didier Legat¹
Jean-didier.legat@uclouvain.be

Raffaele Liberati⁹
raffaele.liberati@elt.it

Daniel Maurfroid²
daniel.maurfroid@fr.thalesgroup.com

Olivier Muleritt³
olivier.muleritt@mbda-systems.com

Denis Aulagnier¹
denis.aulagnier@fr.thalesgroup.com

Vincenzo La Barba¹⁰
vincenzo.labarba@it.thalesgroup.com

Leonardo Tosi³
tosi@cesvitmicroelettronica.it

Pol Cuvelier⁶
Pol.Cuvelier@be.thalesgroup.com

Christian Gamrat⁸
christian.gamrat@cea.fr

Paul Gelineau²
Paul.gelineau@fr.thalesgroup.com

¹Université catholique de Louvain, Place du Levant 3, B-1348 Louvain-la-Neuve, Belgium
²THALES COMMUNICATIONS, Boulevard de Valmy 160, 92704 Colombes, France
³CESVIT MICROELETTRONICA, Via F. Frediani, 59100 Prato, Italy
⁴Tecnopolis CSATA, Str P. Casamassima km 3, 70010 Valenzano Bari, Italy
⁵MBDA, Avenue Réaumur 1, 92358 Le Plessis Robinson, France
⁶THALES COMMUNICATIONS, Rue des Frères Taymans 28, B-1480 Tubize, Belgium
⁷THALES - AEROSPACE DIVISION, Av. de la 1ere DFL 10, 29283 Brest, France
⁸CEA LIST, CEN Saclay, 91191 Gif Sur Yvette, France
⁹ELETTRONICA, Via Tiburtina Valeria km 13,700, 00131 Rome, Italy
¹⁰THALES ITALIA, Via E. Mattei 20, 66013 Chieti Scalo, Italy

Abstract—This paper evaluates the use of Dynamic Partial Reconfiguration (DPR) in professional electronics applications. It first introduces the specificities brought by this area. Then it details the new possibilities in the latest Xilinx Virtex components, the only big FPGA supporting DPR, and the status of the design flow for DPR. Since it is a key point for industrial applications, it highlights the design flow issues, its impact for DPR and how SystemC is used to supplement the current flow to implement Software Defined Radio.

This paper is based on experiments made on a set of seven applications for digital signal and image processing in the area of professional electronics. It evaluates the use of DPR on a set of criterions. It shows that these applications can take a significant benefit from this emerging technology. A high-speed reconfiguration manager used to interface the ICAP with an OPB bus, providing an 84 times improvement compared to the vendor solution, is presented. Detailed results on hardware virtualization and ICAP performances are also presented.

I. INTRODUCTION

Nowadays, the electronics market is mainly directed by consumer applications. They are characterized by a very short time to market, high volumes, autonomous and battery-powered applications, and have very few validations or qualification constraints.

However, professional electronics applications are quite different. They are sold in low volumes and they take longer time for development. They are often maintained, or even updated, during their lifetime that can reach several decades. They need to fulfill precise requirements and validation processes. Moreover, most of them are strongly coupled with other components inside complex systems. It is for example the case for the electronic sub-systems in a plane, or for an airport radar part of an Air Traffic Management system. They are also quite diverse. Indeed, a radar application, for example, performs in real-time a very large FFT of up to tens of thousands of points and must fit in a small volume in order to be integrated into a plane, while a Software Defined Radio (SDR) requires a high level of virtualization, with very low power consumption for handheld devices.

For systems with a high level of safety, further validations and certifications are performed. It is the case for systems that may affect the life of human beings. The ED80/DO254 for hardware and ED12B/DO178B for software, in civil aviation, are among the most restrictive standards [1]. They impose to strictly validate and demonstrate that the application requirements are fulfilled.
in any possible situation, and identify the possible failure modes.

This paper evaluates the use of DPR for professional applications. It is based on the research carried out as part of the RECOPS project that aims to study the use of DPR in military applications [2][3]. The evaluation is made on FPGA from the Virtex family from Xilinx. As they are the only big matrices available supporting DPR, they were selected as the FPGA platform for the project. Results show that the use of DPR in professional applications is a promising area.

In the remainder of this paper, section 2 exposes the DPR possibilities in the latest Xilinx Virtex components with its current design flow. Section 3 and 4, give the main interests for using DPR in professional electronics applications and the issues for their implementation using DPR. Section 5 presents the experiments carried out. Finally, section 6 discusses results obtained, followed by the conclusion in section 7.

II. DYNAMIC PARTIAL RECONFIGURATION

Current FPGAs are composed by a user programmable logic plane, configured by an underlying memory plane. The logic plane holds fine-grain customizable logic made of LUT and interconnection resources, but also optimized macro blocks like SRAM arrays, DSP accelerators, clock tree managers, I/O modules… For most of the components, the configuration memory is filled at startup by a bitstream through a reconfiguration interface. In order to do this, several interfaces are implemented; they are connected to the reconfiguration engine accessing the configuration memory. In Xilinx FPGA, the bitstream is a packet sequence, each packet containing a header and its data. The header holds commands and information for the configuration engine, so that very few external control signals are required during the reconfiguration process.

In some components of the high-end Virtex family from Xilinx, an internal interface (ICAP Internal Configuration Access Port) allows accessing the configuration engine directly from the user logic plane. Thanks to this interface, it is possible for an application to perform self-reconfiguration while it is running. Moreover, according to the vendor, the internal implementation of the configuration ensures that: (i) modifying a region of the component does not affect the configuration memory of other, unmodified, regions and (ii) when the content of the configuration memory is overwritten by the same content, its corresponding logic can operate normally without being affected. In order to perform a Dynamic Partial Reconfiguration (DPR), a partial bitstream is written into the ICAP. Since it is made of raw configuration data highly component-dependent, special tools are required for its generation.

A. NEW DPR POSSIBILITIES IN LATEST XILINX VIRTEX COMPONENT

The Virtex family encompasses the high-performances and biggest FPGA from Xilinx. Furthermore, DPR was introduced in this family with the Virtex II component. Its layout was organized in columns, defining an entire column of the component as the basic configuration granularity. Therefore, the DPR had to cope with severe hardware constraints that lessen its use in professional applications.

Since the Virtex 4, several improvements facilitate the use of DPR, making it a credible solution for some specific applications like Software Defined Radio (SDR) [4]. The improvements for DPR are on the layout architecture, the clock tree and the ICAP. The component is still organized in columns but the partial reconfiguration granularity is reduced to a frame, which is here only a part of a column. Therefore, the Partial Reconfigurable Region (PRR) can be almost any height and width. Clock regions are rectangular, allowing to be matched by a PRR. This leads to better timing performances and clock tree management. Moreover, the output frequency and phase shift of the Digital Clock Manager (DCM) can be modified using DPR.

Finally, the width of the ICAP port is extended to 32 bits and its speed is increased up to 100 MHz. This significantly speeds up the reconfiguration time, which is a main concern when using DPR. Indeed, the HW in the PRR cannot be used during the reconfiguration process. With all those improvements, the HW capabilities are far less a limitation for using DPR in real applications.

B. NEW TOOLS DEDICATED FOR RECONFIGURATION

The implementation of DPR in Virtex FPGA requires a non standard flow [5]. It is driven by constraints on the area and primitives location. It uses special dedicated resources, Bus Macro (BM) for communications, and uses the PlanAhead tool.

Area constraints make it possible to partition a design in a fixed part and a reconfigurable part. They are required to force a design to be Placed & Routed (P&R) in a predefined region composed of a fixed set of frames.

The BMs are slice-based pre-routed elements made of simple LUT. They enable the communication between the fixed and the dynamic part of the design. Indeed, since placement constraints cannot be directly applied to routing resources, they lock the starting or the ending point of the wires to ensure their correct connection after reconfiguration. They can be located on any edge of a PRR (left, right, top, bottom). They can be asynchronous or clocked, have a fixed direction (in or out), they are device dependent (Virtex-4, Virtex II, Virtex II pro,…). Their main drawbacks are that they consume a significant amount of resources and their automatic placement is not supported by the tools.
PlanAhead is a proprietary tool from Xilinx that is used to manage the DPR constraints and to drive the implementation process. It is a production tool with graphical interface allowing to handle constraints and area location of a PRR at component-level. It provides a hierarchical design view at netlist level and a resource view at component-level. Moreover, it makes a resources and bitstream's size estimation (that is not precise), performs some Design Rule Checking and exports the results for implementation.

Finally, a specific partial flow is used to P&R the different portions of the design. It is based on a modular flow available in ISE. It requires special patch updates that are currently available for selected customers and research centers. The flow is compatible with the embedded processors (PPC and Microblaze). At the end, it provides full and partial bitstreams that can be directly written into the ICAP. Notice that with this flow the modules do not have to follow the restrictive constraints of the XAPP290 [6] anymore.

C. THE NEXT GENERATION

The latest high-end FPGA available in Xilinx is the Virtex 5. It is fabricated in a 65 nm technology, and it introduces innovations like diagonal routing capabilities and 6-entry LUT. It has new configuration ports and can manage multiple configurations. Regarding DPR, the component will support the features available on Virtex 4. Since DPR is today highly component-dependent, specific tools and patches are required on top of the standard flow as for Virtex 4.

III. ADVANTAGE FOR USING DPR IN PROFESSIONAL ELECTRONICS

The DPR allows using more hardware than physically present in the FPGA. This can be used to reduce the size of the FPGA and its overall power consumption. This also permits to execute an algorithm with an optimized implementation depending on its parameters and data set. Furthermore, besides the usual speed and power research goals, DPR offers system-level advantages for professional electronics [2]. The advantages considered are:

a) Task speed: By shifting the interface between hardware and software toward (faster) hardware. More functions can be implemented in hardware without being limited by the size of the component.

b) Power reduction: By having less hardware instantiated and running, and by using a smaller FPGA.

c) Survivability: By allowing selection and operation in a degraded mode when a part of the system is damaged. It is necessary for applications running in harsh environments where environmental conditions can exceed the normal operating range.

d) Mission change: By allowing to configure the application for an entire mission without interrupting services. The real-time issue is not critical here. The application is configured for a long period. DPR provides an easy and safe way to strongly modify an entire system without having the complexity of implementing all the functionalities in one design. It is very useful and it facilitates the validation of applications interconnected in a complex system.

e) Environment change: During operation, the application can be developed specifically for several environments and switch dynamically. It is similar to mission change but the real-time issue is critical.

f) Algorithm adaptive change: By adapting dynamically an algorithm depending on the external conditions. It is a lower granularity than environment change.

g) On-line system test: A system in harsh environment can be damaged. For critical systems, it is necessary to know its level of functionality, for example, to decide to power on a redundant one.

h) Hardware virtualization: By having more hardware available than physically present in the FPGA. It allows to manage a set of hardware modules as a component library. It is used for example in SDR applications.

IV. IMPLEMENTATION OF DPR IN PROFESSIONAL APPLICATIONS

In order to implement DPR in real applications, some issues need to be carefully handled. The most important ones are the design flow, the constraints brought by the use of DPR on the application board, and the minimum development required for its integration.

A. THE DESIGN FLOW ISSUE

The deep validation requirements or even the certification in critical applications is not only performed on the final product, it also imposes the use of a validation methodology during the whole development process. In this scope, the design flow is a key point. For severe requirements, it must be certified [1]. The validation imposes that at each step of the development, from the first specifications to the final tests, one must be able to verify that the application meets the requirements. For this, it is necessary to have precise simulations and modeling capabilities and to have efficient tools, at least for productivity reasons.

The standard design flow for a digital system is based on a top-down approach. The main steps are:

a) System specification
b) Functional modeling and simulation
c) Hardware / software partitioning
d) Architecture definition
e) HW and SW development
f) Platform integration
g) Validation and qualification

A complete design is rarely done in one conception pass. Indeed, in order to meet the application requirements,
it is often necessary to make several iterations until reaching acceptable performances. The following paragraph focuses on the missing elements for DPR regarding this flow.

### B. Identification of Missing Elements for the Dynamic Reconfiguration Flow

For each step of the design flow, it is mandatory to have a simulation model of the DPR. For the first steps until the architecture definition, a model can be built with SystemC. For critical designs or for productivity issues, those steps are usually automated by tools. None of the existing tools supports DPR. Nevertheless, it is important to note that FPGA are generally not well supported. It is for example difficult to find SystemC models for the complex hard or soft IP in FPGA like the PowerPC 405 or the Microblaze.

For the hardware development step, it is necessary to have a behavioral model of the ICAP, the BM and the configuration process. Indeed, a hardware-level model is required to make functional simulation, ensure real-time constraints and for debug. Without a behavioral model of the hardware, it is only possible to simulate the modules independently and verify the design when committing the application to the final on-board tests. Then, it is possible to verify a DPR design only when performing the platform integration. Therefore, if something fails, it is not possible to reproduce the problem by simulation. Moreover, debugging tools like ChipScope from Xilinx [7] are not working in a reconfigurable region.

Finally, without complete behavioral models, the validation and qualification step can only be done on the hardware platform. Serious difficulties occur, when it is necessary to take into account the design flow for validation.

### C. The Hardware Platform and Constraints

The board here is the Printed Circuit Board hosting the FPGA. Its constraints are of two kinds. The first is that it requires a large amount of external FLASH memory, in order to permanently store the partial bitstreams, but also fast external memory like DDR in order to load bitstream and perform a reconfiguration at the maximum speed. The second kind of constraints is on the I/O position. For example, with DDR I/Os, the memory controller needs to be placed in front of the I/Os, constraining the placement of the PRR. Regarding those constraints, the standard development boards can easily be used to test DPR on a reference application.

### D. Developments Required for DPR Evaluation

The developments required for evaluation are the design of a reconfiguration controller and a scheduler. The controller is based on an interface between the OPB (On-Chip Peripheral Bus) bus available in the Virtex and the ICAP. It offers the flexibility of the standard bus that allows connecting any type of memory through a standard interface. Furthermore, the bus benefits of DMA services that provides sustained data rate to reach the maximum speed of the ICAP. Since the configuration manager is connected to a bus that can be connected to a Microblaze or a PowerPC, the scheduler can be implemented in software, which strongly reduces its development cost. Note that an OPB-ICAP interface is available from Xilinx, but not for all the components and the full utilization of the ICAP is not supported on the latest one.

### E. Use of SystemC for Dynamic Reconfiguration

SystemC/TLM can validate the HW/SW interface at a very early stage of the system specification step [8]. Due to its high-level modeling capabilities, SystemC is a good candidate to model the reconfiguration process. With SystemC/TLM it is possible to have all configuration modules available in a DPR platform with a reconfiguration manager controlled by software and to model the reconfiguration behavior [9].

Furthermore, in the SDR applications SystemC is a promising way to deal with the virtualization of the hardware components. The SCA (Software Communication Architecture) layer of the SDR can be modeled by SystemC/TLM to simulate a platform independent model that can be targeted to a platform specific model inside a same representation [10].

### V. Experiments

The DPR is evaluated in industrial conditions on a broad range of applications mainly in the field of defense applications. The approach is to use a common platform in order to easily share development experiences, make relevant comparisons and demonstrate the platform flexibility offered by the FPGA implementation. The broad range of applications is required by the diversity of applications and constraints encountered in professional electronics.

### A. The Evaluation Applications

The experiments are made on real applications or on a representative part of them. The applications are listed here:

- **a) Portable device for remote-control video capture and transfer**: This application realizes image acquisition and transfer with remote control video using wireless communication. The reconfigurable process performs axis motion control, image capture and data transmission in the same reconfigurable region.

- **b) Blanking management for naval Electronic Counter Measure /Electronic Support Measure systems**: this application generates control signals avoiding interferences between the ECM and ESM systems.
c) **Real-time image processing unit for missile applications:** it applies several operators on an image with hard real-time constraints. The operators are reconfigured dynamically.

d) **Front end processing for airborne radar:** it tests the impact of reconfiguration on the use of high-speed I/O links and the front-end data processing under hard real-time and jitter constraints.

e) **Short range radio modem:** it is used for local and private data communications and needs to support several data rates. The base band functions of the modem are implemented in a reconfigurable region.

f) **Software defined radio transmitter:** the waveforms are implemented in a reconfigurable region. It tests the waveform parameterization and change.

g) **Software defined radio receiver:** evaluates the hardware virtualization brought by the dynamic reconfiguration and demonstrates the partial reconfiguration of the FPGA by software from the SCA core framework.

Each demonstrator covers its own area of assessment, but together, they cover a large panel of activities and techniques. The tests are carried out to emphasize the different areas where the use of DPR can be interesting. It is not possible to detail them here. Nevertheless, we will highlight features of the software defined radio application.

**B. THE SOFTWARE DEFINED RADIO EXPERIMENTS**

The opportunity for SDR is to use the virtualization brought by DPR in order to be implemented in hardware, rather than in software. It is thus a very good candidate to take benefit of the dynamic reconfiguration. The experiments conducted with the SDR transmitter for multiband, multi-mode radio, are performed by switching between two different waveforms, the D8PSK and the 16QAM. The reconfiguration is controlled by the internal Power PC core of the FPGA. The mapping of data patterns to symbol, and symbols to in-phase and quadrature components, are done in the PRR. The pulse-shaping filter is also implemented in the PRR. The experiments with the receiver implement the SCA layer of the SDR in an FPGA platform using DPR.

**C. THE EVALUATION PLATFORM**

The platform targeted for all the experiments is the ML410 development board from Xilinx. It hosts a XC4V- FX60 with sufficient external storage resources for storing the partial bitstreams for all the applications. No other specific board development is needed by using the platform. Nevertheless, the boards ML403 and ML405 are also used for some experiments (due to the late availability of the ML410 board). The components used by all the experiments are the XC4V- FX12, FX20, FX60 and LX60.

**D. THE RECONFIGURATION CONTROLLER**

For hard real-time applications that need to change frequently the reconfigurable module, a high bandwidth through the ICAP port is required. It is for example the case for the image processing application, which applies sequentially a set of reconfigurable functions for each image in a narrow time frame. The theoretical speed of the ICAP is 100 MHz and its width can be programmed as an 8-bit or a 32-bit port allowing a bandwidth of 0.75 or 2.98 Gbit/s. The ICAP is usually connected to an OPB bus. This allows to use easily all the existing memory resources of the board by means of standard interfaces. The bus is driven by a microprocessor that can be a hard embedded PPC core or a soft MicroBlaze core. Unfortunately, when the experiments where carried out, the interface provided by the vendor was able to operate at no more than 40 MHz in the 8-bit mode, and was designed for the Virtex II component. Those performances are not sufficient to meet the real-time constraints of the image processing application. Therefore, an improved version based on the OPB-ICAP interface for Virtex II was developed. The improvements allow working with the ICAP in the 32-bit mode at 100 MHz. Moreover, it also provides DMA services allowing to use the OPB bus with the required bandwidth.

![](image)

**FIGURE I. RECONFIGURATION CONTROLLER**

Figure I shows a functional description of the reconfiguration controller implementing the OPB-ICAP interface. It is viewed as an OPB peripheral though an IPIF from the bus.

It is written in VHDL and generic parameters allow selecting the memory size as well as the ICAP mode of 8 or 32 bits. A control and status register allows writing and reading bits to command and control the reconfiguration controller. It implements a DMA able to steer data from the bus at a sustained rate. The parameters are first written in two registers, a DMA_START_ADDR register that holds the base address of the data segment and a DMA_BURST_SIZE register that defines its size (max 1024 words). Then a start bit in the CTRL / STATUS register is set to start the transfer. When it is finished, a status bit is set in the same register. The BRAM memory is used as a buffer. It was originally implemented in the
Virtex II controller to convert the data from a 32-bit stream from the bus to an 8-bit stream toward the ICAP. Nevertheless, its instantiation is not mandatory. Indeed, the ICAP can be controlled at data word level by an enable signal, thus a single 32-bit register can be used to split the 32-bit data in four 8-bit words. When using the DMA, the buffer is bypassed. The reconfiguration controller supports read-back, allowing to transfer the content of selected configuration frames to the bus. The DMA is not supported with this feature. Configured in the 32-bit mode, the reconfiguration manager occupies 973 slices and 1 BRAM. It corresponds to 3.7 % of the mid-range XC4VLX60 Virtex 4 component.

![Diagram](image)

**FIGURE II: MEMORY ORGANIZATION**

The memory organization for the image processing application is given in Figure 2. The external flash is used to permanently store the bitstreams. The DDR and ZBT (Zero Bus Turnaround) are used as fast memories. The bitstream size for the reconfigurable region is around 300 KB for the biggest. In order to obtain high-speed transfer, the partial bitstreams needs to be stored in a fast external memory. Indeed, the permanent flash memory is very slow; it can only deliver 8-bit at 10 MHz. Therefore, the ZBT SRAM or the DDR are used since they are the only memories on the board able to sustain the 100 MHz, 32-bit, throughputs. When the application starts, the bitstreams are first copied from the flash to the ZBT SRAM or the DDR depending on the application.

**VI. RESULTS AND DISCUSSION**

The results are presented here from a system-level approach with highlights on the virtualization in the SDR experiments. Finally, measures of the performances of the ICAP are detailed.

The characteristics of the seven applications are given in Table I. The first column gives the XC4V component used, then the number of reconfigurable regions and the relative size of the reconfigurable regions counted in slices. The fourth column gives the reconfiguration time followed by the maximum size of all the partial bitstreams targeted to a region. Note that not all the applications use the fast reconfiguration manager. The last column gives the real-time constraint for performing a reconfiguration.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a) Im. acq.</td>
<td>FX60 1</td>
<td>16</td>
<td>2.2</td>
<td>330</td>
<td>20 ms</td>
</tr>
<tr>
<td>b) Naval</td>
<td>FX12 1</td>
<td>28</td>
<td>195</td>
<td>90</td>
<td>1 s</td>
</tr>
<tr>
<td>c) Im. proc.</td>
<td>FX60 1</td>
<td>13</td>
<td>1.3</td>
<td>17</td>
<td>5 ms</td>
</tr>
<tr>
<td>d) Radar</td>
<td>FX20 1</td>
<td>33</td>
<td>0.47</td>
<td>166</td>
<td>1 ms</td>
</tr>
<tr>
<td>e) Modem</td>
<td>FX12 1</td>
<td>32</td>
<td>3.5</td>
<td>241</td>
<td>1 s</td>
</tr>
<tr>
<td>f) SDR Tx</td>
<td>FX12 1</td>
<td>9</td>
<td>31.6</td>
<td>38</td>
<td>1 s</td>
</tr>
<tr>
<td>g) SDR Rx</td>
<td>FX12 2</td>
<td>12</td>
<td>15.5</td>
<td>46</td>
<td>1 s</td>
</tr>
</tbody>
</table>

**A. SYSTEM-LEVEL ASSESSMENT**

Most of the advantages listed in §3 are tested by the experiences. The results are given in Table III. For each advantage, a rating between −2 and +2 evaluates its benefit in the application. An empty evaluation means that the advantage was not tested during the experiments by the application. A rating between parenthesis means that the advantage was not directly evaluated during the experiments but rather estimated. Results show that the most interesting advantages are: virtualization, environment and mission change. They allow changing the functionality of a system with relatively few real-time constraints. Moreover, in all the applications using this feature, the architecture of the system remains the same while an instance of a particular function is changed. This leads to add more functionalities without increasing the complexity of the data path of the application. The telecommunication applications change their waveforms simply by reconfiguring a region. The algorithm change is obtained by the virtualization advantage. It was only lightly tested by the experiments, since this feature is today rarely implemented by the applications due to its novelty. There was no increase of the tasks speed of the application, except for the SDR applications, where more accelerators can be instantiated in the FPGA instead of being executed on a DSP. For the other applications, the accelerators are always instantiated in hardware. The power reduction is then obtained by using a smaller FPGA when using DPR, reducing the leakage and the dynamic power consumption generated by unused hardware. The survivability possibility of the application is lower for the SDR application because the DPR brings new failure modes to the system. Since they are not yet well characterized, they have to be cautiously handled. The on-line system test is experienced by the radar and the image processing application where every reconfigurable module has its own communication lines.
TABLE II: RESOURCE CONSUMPTION

<table>
<thead>
<tr>
<th></th>
<th>Plain</th>
<th>Scrambler</th>
<th>DES</th>
<th>Static</th>
<th>Static + max PRR</th>
<th>Static + all modules</th>
<th>Avail</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Encr</td>
<td>Decr</td>
<td>Encr</td>
<td>Decr</td>
<td>Encr</td>
<td>Decr</td>
<td></td>
</tr>
<tr>
<td>Slice</td>
<td>53</td>
<td>53</td>
<td>69</td>
<td>69</td>
<td>379</td>
<td>379</td>
<td>2674</td>
</tr>
<tr>
<td>RAMB16</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>17</td>
</tr>
<tr>
<td>DSP48</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PPC405</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>Bitstream (KB)</td>
<td>31.7</td>
<td>31.7</td>
<td>33.7</td>
<td>33.4</td>
<td>46.3</td>
<td>46.1</td>
<td></td>
</tr>
</tbody>
</table>

Therefore, it is possible for the application to monitor the good operation of the functions and the reconfiguration process.

TABLE III: ADVANTAGES EVALUATION

<table>
<thead>
<tr>
<th></th>
<th>Task speed</th>
<th>Power reduction</th>
<th>Survivability</th>
<th>Mission change</th>
<th>Environment change</th>
<th>Algorithm change</th>
<th>On-line system test</th>
<th>Virtualization</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) Im. acq.</td>
<td>+1</td>
<td>(0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+2</td>
<td></td>
</tr>
<tr>
<td>b) Naval</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c) Im. proc.</td>
<td>0</td>
<td>+1</td>
<td>(0)</td>
<td>(0)</td>
<td>(+1)</td>
<td>(+1)</td>
<td>(+1)</td>
<td>+1</td>
</tr>
<tr>
<td>d) Radar</td>
<td>0</td>
<td>+2</td>
<td>+2</td>
<td></td>
<td></td>
<td></td>
<td>+2</td>
<td></td>
</tr>
<tr>
<td>e) Modem</td>
<td>0</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td></td>
</tr>
<tr>
<td>f) SDR Tx</td>
<td>+1</td>
<td>(+1)</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td></td>
</tr>
<tr>
<td>g) SDR Rx</td>
<td>(+1)</td>
<td>-2</td>
<td>(0)</td>
<td>+1</td>
<td>+1</td>
<td>+1</td>
<td>+2</td>
<td></td>
</tr>
</tbody>
</table>

Most the evaluated features bring the promised advantages to the application. Nevertheless, a drawback comes from the size of the design. Indeed, where few virtualization is used, there is a significant size overhead when using DPR compared to the static implementation.

B. THE HARDWARE VIRTUALIZATION

Specific hardware virtualization tests are carried out in the SDR receiver application. The experience compares implementations of crypto algorithms for an SDR application. The radio has two channels implemented and running at the same time, one requiring encryption and the other decryption. The algorithms can be changed dynamically and can be different for each channel. Three functions are considered, PLAIN, SCRAMBLER and DES. Plain returns the clear message (no encryption), scrambler performs basic bit scrambling and DES is a standard symmetric algorithm. The implementation is done with two PRR and a static region. Since the PRR are statically defined, they need to be sufficiently big to fit the biggest module and a significant amount of resources can be wasted. This problem is illustrated in Figure 3 for two PRR in three situations. In Figure 3 (a), the PRR are dimensioned to hold F1A and F2A. When reconfiguring with F1B and F2B in Figure 3 (b), the place left by F2B cannot be used by F1B. Therefore, the PRR reserved for F1 in Figure 3 (c) needs to be maximal.

![Figure III: PRR Reservation](image)

For industrial and security reasons, the bitstreams need to be encrypted. In the components used for the experiments, the ICAP is not usable when the bitstream encryption is enabled. The ICAP throughput is an issue for the designs using DPR in hard real-time applications and it is carefully measured by the experiments. The ICAP was successfully tested at 100 MHz in 32 and 8-bit modes. The write and read-back modes were tested. Note that for reaching this speed, we found that the data needs to be sent on the falling edge of the ICAP clock. This is maybe due to a clock skew between the user logic and the ICAP. Table IV summarizes the performances measured by connecting the ICAP to a basic GPIO, with the OPB-ICAP provided by...
Xilinx for Virtex II and with our reconfiguration manager. The partial bitstreams are stored in a DDR with a clock at 100 MHz. The Xilinx interface is always in 8-bit mode. The maximum theoretical bandwidth of the ICAP is 2.98 Gbit/s. It is not reached due to DMA overhead. Nevertheless, our custom OPB-ICAP is 84 times faster, compared to the vendor’s module. 4 times are due to the data word size and 21 times to the DMA accesses.

<table>
<thead>
<tr>
<th>ICAP mode</th>
<th>GPIO OPB-ICAP</th>
<th>Xilinx OPB-ICAP</th>
<th>Custom OPB-ICAP</th>
<th>Theor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bits</td>
<td>-</td>
<td>32.4 Mbit/s</td>
<td>0.56 Gbit/s</td>
<td>0.75  Gbit/s</td>
</tr>
<tr>
<td>32 bits</td>
<td>22.2 Mbit/s</td>
<td>-</td>
<td>2.8 Gbit/s</td>
<td>2.98  Gbit/s</td>
</tr>
</tbody>
</table>

VII. CONCLUSION

This paper evaluates the use of DPR in professional electronics applications. It introduces specificities related to this area and their impact on the use of DPR, especially regarding the design flow and the tools. Indeed, with the improvements introduced in the Virtex 4, the limitations are no longer due to the hardware capabilities but rather to the tools. Experiments carried out on a broad range of applications clearly show the interest for using DPR in professional electronics applications. Furthermore, the evaluation of the applications on a set of criterions illustrates how professional electronics applications can take benefit from DPR for several other advantages than the classical speed and power research goals. One of the most challenging application fields is the SDR. It can use DPR to achieve hardware virtualization and use many more hardware than physically present in the component. It requires implementing high-level models in SystemC in order to make a link between the software components and the reconfigurable hardware modules. Finally, a custom implementation of the ICAP interface shows that significant improvements can be obtained on the reconfiguration speed compared to the standard approach.

ACKNOWLEDGMENTS

This research is part of the RECOPS project. It is a EUROPA contract from the European Defense Agency, funded by French and Italian Ministries of Defense and the Walloon Region in Belgium

REFERENCES

[9] Y. Qu, J-P. Soininen, ”SystemC-based Design Methodology for Reconfigurable System-on-Chip”, EUROMICRO, 2005
Developing a Context-aware Architecture in DySCAS

DeJiu Chen
The Royal Institute of Technology (KTH)
Stockholm, Sweden
chen@md.kth.se

Richard Anthony
The University of Greenwich
London, UK
R.J.Anthony@gre.ac.uk

Martin Törngren
The Royal Institute of Technology (KTH)
Stockholm, Sweden
martin@md.kth.se

Gerrit de Boer
Robert Bosch GmbH
Hildesheim, Germany
Gerrit.deBoer@de.bosch.com

Viktor Friesen
Daimler AG
Stuttgart, Germany
viktor.friesen@daimler.com

Cecilia Ekelin
Volvo Technology AB
Göteborg, Sweden
cecllia.ekelin@volvo.com

Abstract—Software systems in vehicles are of rapidly increasing commercial importance for the automotive industry. Current vehicular embedded systems exhibit a fixed software structure and the possibilities for software upgrades are limited. Due to increasing expectations of context-awareness, dependability, flexibility, cost efficiency and time-to-market; we argue that it is now necessary to enable advanced features that are based on dynamic configuration.

To reach the goals of dynamically self-configuring automotive systems, we propose an architecture which is currently being developed in the EC funded DySCAS project. We describe the proposed architecture, with a focus on its support for adaptive behavior. To illustrate the type and extent of adaptability achievable, a set of generic use case classes are briefly described and the requirements they place on the architecture and in particular the adaptive aspects, are discussed.

The architecture is based on a distributed hierarchical control and decision strategy, which through the policy-based middleware functions and underlying technologies allows separation-of-concerns, on-line reconfiguration, seamless connectivity, plug-and-play capability, and also increased dependability and resource optimization. Decision points are concerned with both application and resource aspects and are dispersed throughout a distributed platform, allowing the embedding of numerous policies to make context-aware decisions whilst avoiding excessive complexity. The approach is scalable and provides additional freedom for engineers to define the dynamic configuration characteristics.

The developed architecture is so far on a functional level and has been validated against system requirements. Current work concentrates on defining the corresponding software architecture, considering automotive platforms and constraints.

I. INTRODUCTION

Embedded software systems have been widely used in modern automotive systems for a variety of purposes, ranging from driver assistance, to fuel efficiency and active safety. Advances in the embedded electronic systems currently account for a very large portion of all innovations in the automotive industry.

However, current state-of-the-art vehicle electronic systems, for which a static configuration is defined during the development process and remains stable over the complete lifetime of the vehicle, will not be sufficient due to increasing expectations of advanced functionality, context-awareness, dependability, flexibility, cost efficiency and time-to-market. Future scenarios for vehicle electronic systems typically include (1) simultaneous access to a number of mobile devices that build ad-hoc networks with the ability to share information and synchronize behaviours, (2) adaptive configuration with ability to dynamically determine which software components execute at which computation nodes and control their executional behaviour for the reasons of dependability, maintenance, and optimized resource utilization. These aspects are discussed in section IV.

Recent advances in technologies are beginning to provide basic mechanisms necessary for fulfilling the future scenarios described above, ranging from engineering support for configuration standardization and component specification, to online decision-making algorithms and mechanisms for quality of service and dynamic reconfiguration. For example, the AutoSAR initiative by the automotive industry provides an open standard defining standard interfaces and a middleware layer that facilitate the use of “off-the-shelf” software components from different suppliers and to allocate functions during design time [1]. The autonomic computing paradigm advocates self-managing behaviour in which applications modify their behaviour to suit their environment and context and is widely acknowledged to be a potential solution to the software complexity crisis [2], [3]. Control-theoretic approaches have been used to support load-balancing and quality of service [4]. State of the art middleware technologies also provide basic mechanisms for transparent communication [5] and configurational adaptability [6], [7].

This paper describes the approach taken by DySCAS (Dynamically Self-Configuring Automotive Systems), which is a European Commission funded project that started in June 2006 [8] aiming to advance the technologies required to build self-configuration aspects into automotive systems. In particular, the paper provides an overview of the DySCAS architecture and an elaboration of the policy-based computing techniques adopted for embedded reasoning and decision-making. Issues covered in the architecture overview
include the qualities of concern, the architectural reasoning, and the overall system design. The description of policy-based computing explains the control process and covers the information that is of particular concern, including the physical environmental conditions of the vehicle and the status of the platform and communication network.

The developed architecture corresponds to a functional architecture in that it specifies the middleware system in terms of subsystem blocks and the abstract data- and control-flows without considering implementation and technology details. The architecture has so far been validated against system requirements and partly implemented. Current work concentrates on further defining and elaborating the corresponding software architecture, considering automotive platforms and constraints. It should be noted that there need not be a one-to-one mapping from the functional architecture to the corresponding software implementation. The paper includes a discussion of several outstanding issues regarding the software implementation.

This paper includes five sections. Section II provides an overview of the DySCAS middleware architecture. Section III explains how policies have been used to achieve flexible and adaptive behaviour in the middleware. Section IV describes the generic use case classes and their requirements for adaptive behaviour in the middleware. Section V discusses some open issues and concludes the work.

II. OVERVIEW OF THE DYSACS ARCHITECTURE

A. Context-Awareness and Dynamic Configuration Management

The DySCAS middleware provides direct support for context-awareness, allowing an embedded software system to be aware of its design context and runtime environment. Such contextual or environmental concerns include: application specific properties and design constraints; the environmental conditions (such as sensor information) and events (such as details of newly discovered devices); the deployment of network and hardware resources; the availability of other external systems; as well as internal influences such as CPU load and hardware failures.

Based on such knowledge, the system adapts and manages its configuration dynamically and exhibits the following autonomies characteristics [2], [9]:

- **Self-defining:** the ability to derive knowledge of its components, status, ultimate capacity, and external connections in regard to a particular context.
- **Self-optimizing:** the ability to monitor its constituent parts and fine-tune workflow to achieve predetermined system goals.
- **Self-healing:** the ability to discover errors or other potential problems, and find alternate ways of using resources or reconfiguring the system to keep functioning smoothly.
- **Self-protecting:** the ability to detect, identify, and protect itself against malicious attacks and maintain overall system security and integrity.

The DySCAS middleware system is motivated by the industrial needs of advanced functionality, optimized resource utilization, performance, maintainability, dependability, and reusability. Specific requirements have been extracted from a number of use case classes which are briefly discussed in section IV.

The mechanisms of context-awareness and dynamic configuration can also be used to detect and handle unexpected events like overloading and component failures, and thereby through function migration and system reconfiguration ensure a high level of quality-of-service and fault-tolerance. Making network-based resources available to vehicle functions will allow software functionality and components to be automatically evaluated, updated, repaired, or removed via WLAN hotspots. All these features will contribute to significant added value and differentiating features.

These future scenarios impose very high demands on the configurational flexibility and scalability of systems. Clearly, such features cannot be easily fulfilled by state of practice vehicle electronic systems in which a static configuration of proprietary hardware and software components is applied. The configuration is fixed at development time and remains stable over the complete lifetime of vehicle. Under the circumstances, it is difficult and expensive to perform software maintenance for vehicles, for example to upgrade a device driver or to repair a software fault.

B. Reasoning behind the DySCAS Architecture

Flexibility has many aspects, implying the scalability of functions, components and performance, the interoperability of networked systems and heterogeneous devices, and the portability to other established technologies like commercial RTOS’ target platforms, and networks (e.g., LIN, MOST, and wireless networks). There are a number of challenges to fulfilling flexibility in a distributed system, several of which the DySCAS middleware system is intended to cope with.

One kind of challenge is concerned with the support for automated assessment of the impacts of potential changes on system functionality, performance, safety, and other quality concerns. For example, when dynamically adding a software component, it might be the case that the resources of concern (e.g., in terms of other application services or resources like bandwidth and memory) will not meet the component needs. For the DySCAS middleware system, it is important that the prerequisites and constraints of components in regards to communication, synchronization, allocation, and execution can be specified (e.g., as component meta-data) and maintained at runtime. For dynamic reconfiguration, a consistent global view of the current configuration and resource utilization is also necessary. For each request for
configurational change, the middleware system analyses the current configuration and thereby resolves subsequent operations on the basis of such a specification of components.

Another challenge related to the online decision-making is dealing with the complexity of embedded systems and the existence of multiple concerns. Instead of a single decision criterion and optimization approach, there are multiple and sometimes conflicting optimization needs and mechanisms. For example, the DySCAS middleware system is intended to support load balancing for optimized resource utilization and thereby to promote performance and reliability. Load balancing can be applied to different resources including processors, networks, and memory, or to the service capacity of application software and middleware. For example, for a component whose services are in high demand, one may reduce the response time by increasing the instances of this component in time (e.g., by increasing the bandwidth), or in space (e.g., by creating multiple copies of the component on several different nodes). While the first approach results in increased CPU workload of a single node, the second approach leads to more balanced CPU workload but increases communication overheads and memory usage. Clearly the policies or rules for load balancing need to consider such multi-criteria trade-offs as well as the operation and environment conditions.

While the above mentioned challenges deal with the decisions for dynamic configurational changes, other challenges have to do with the necessary communication and execution support. The communication support are often referred to as location transparency and access transparency (see e.g., [5]), which separate the functional aspects of interaction from those aspects that are location and platform specific. The support is necessary to meet many other common features of dynamic configurable systems, such as to migrate and reallocate components without being noticeable to other components or the end user (i.e., migration-transparency), to move a (mobile) resource in use without being noticeable to the components or the end user (i.e., relocation-transparency). The communication transparency also facilitates fault tolerance design, such as to allow transparent interactions with replicas (e.g., replication transparency). For dynamic configurations like software migration and upgrade, portability and transparent execution support are also of particular concern. The execution support here is related to the control and coordination of dynamic configurational operations (such as: to initialize a component, to plan for rollback, to transfer component states, to check the safety of configurations, and to adapt the schemes for task execution and communication) for adding or removing software components on different nodes.

One performance related challenge is the middleware overheads, both in time and in resource utilization (e.g., bus, CPU, and memory). Such overheads are unavoidable as the penalty for performing dynamic configuration and self management and for handling heterogeneous infrastructures, computer hardware, and devices. Instead, the objectives ought to be the "overall" system optimization, in which the online configurational flexibility is explicitly considered in the system traded-off together with other quality attributes like performance. The merits by the flexibility are for example related to system maintainability, modifiability, and QoS. Nevertheless, to minimize and assess the impacts of such overheads on the overall quality of services, the choice of control algorithms, the instantiation, mapping, and allocation of middleware services, as well as the planning and controlling of the configuration and management tasks, are all of particular concern. For dynamic operations, determinism can be achieved through mechanisms that reserve and initialize necessary resources in advance. It is also possible to facilitate dynamic changes by allowing different operation modes of applications.

There are also challenges with respect to acceptance. The move from quasi-static to systematic support for dynamic configurations is a large step for the automotive industry. The introduction of adaptive aspects of behaviour, and the ability to defer behaviour decisions beyond the point of systems deployment requires that the systems are fully verifiable. In addition, it is intended that DySCAS supports pre-fixed standard configurations (mode-like) so that a vehicle manufacturer is free to decide the extent to which he would like to introduce reconfigurability. This support for progressive introduction of the dynamic aspects is expected to be a major influence on the acceptance of DySCAS.

C. Middleware Services

Being an intermediary between application software and the underlying system platforms and infrastructures, the DySCAS middleware system constitutes a means for hiding the complexity of the underlying automotive target and provides an open interface for integrating external devices. The DySCAS middleware system has a layered architecture consisting of two groups of runtime services: (1) core services, and (2) optional services. Figure 1 provides an outline of the DySCAS functional architecture, including the major middleware services and interfaces. The dashed blocks represent optional services allocated at the DySCAS Instantiation Interface. The DySCAS middleware architecture is layered into three levels of controls following the concept of autonomous architecture as in [10]. The fundamental benefit of this layered strategy is the hierarchical decomposition of control tasks through which a larger reconfiguration problem is reduced to more elementary operations. For technical and practical reasons, DySCAS provides the freedom to run other standalone and legacy software programs along with DySCAS specific application
TABLE 1. An overview of DySCAS Core Services and some of their properties

<table>
<thead>
<tr>
<th>DySCAS Core Service</th>
<th>Overall System Roles</th>
<th>Related Autonomic Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autonomic Configuration Management Service</td>
<td>Analyzer for the overall impacts of requested dynamic changes; Planner of configuration tasks.</td>
<td>Self-configuring (mainly the online configuration reasoning support); Self-healing (only error-handling and fault removal).</td>
</tr>
<tr>
<td>Repository Service</td>
<td>Repository of files for decision policies, component images, and information relating to knowledge of configuration and diagnostics.</td>
<td>N/A</td>
</tr>
<tr>
<td>Dependability &amp; Quality Management Service</td>
<td>Dependability and security controller; Performance Optimizer.</td>
<td>Self-healing; Self-optimizing; Self-protecting (for external accesses and service requests).</td>
</tr>
<tr>
<td>Autonomic Configuration Handler</td>
<td>Coordinator of distributed dynamic configuration operations.</td>
<td>Self-configuring (through the execution synchronization support).</td>
</tr>
<tr>
<td>Resource Deployment Management Service</td>
<td>Monitor of target and external resources that also brings together distributed information and generates normalized figures of quality feedback; Executor of dynamic configuration operations.</td>
<td>Self-defining; Context awareness (in respect to external devices/systems); Self-configuring (mainly through sensing and execution support).</td>
</tr>
<tr>
<td>SW Load Management Service</td>
<td>Executor of dynamic load operations.</td>
<td>Self-configuring.</td>
</tr>
</tbody>
</table>

software programs. Such external application software programs are not ported to the middleware system, but to the underlying target platform or hardware devices.

For software programs that are ported to DySCAS, the middleware provides uniform access to the resources and portability support. The core middleware services are responsible for consolidating and managing the configuration of a distributed network system. They provide support for managing the variability of software components and hardware resources (e.g., the allowed cross combinations of application software, drivers, and devices) and maintaining a consistent view about the actual configuration and resource utilization. Based on such information, the core services also provide support for inferring the system-wide impacts of changes, planning and coordinating policy-based dynamic adaptation and reconfiguration actions, and implementing QoS and dependability control. Table 1 provides an overview of the core services and their main responsibilities.

The optional middleware services interact directly with the underlying system platforms and infrastructures. See also figure 1. They provide support in respect of portability and transparent communication, concurrency control, and membership management, etc. for the DySCAS core services as well as the application software programs, much as the support offered by other traditional middleware systems. Such services are optional since they may overlap with the services offered by the targeted system or infrastructure. In this case, the components will be placeholders/containers for the external services.

Of great importance to the DySCAS middleware system is the concept of interactions and synchronization between the system constituent units, (shown in figure 1 as middleware services), application software programs, target platforms and infrastructures. Such system constituent units interact.
and synchronize with each other to compose a system behaviour as a whole while carrying out some parts of the total system behaviours and maintaining some parts of the total system states.

Table 2 provides an overview of the functional dependencies between DySCAS components, which are defined following the client-server paradigm. For instance, the Autonomic Configuration Management Service providing support for the online assessment and planning of all dynamic configurations is initialized by reconfiguration requests from three other middleware services. While the Application Program Interface provides user requests, the Resource Deployment Management Service and the Dependability & Quality Management Service generate reconfiguration requests for the reasons of self-optimization, error-handling, and the handling of resource changes. All the decisions are based on the policies/rules loaded in the middleware services and the system state and configurational information. Historical information can also be retrieved from the Repository Service, which provides support for logging events for diagnostic purposes. After receiving the elementary configuration tasks from the Autonomic Configuration Management Service, the Autonomic Configuration Handler calls the Resource Deployment Management Service and the Dependability & Quality Management Service to negotiate and reserve the timing and resources for the subsequent primitive operations. The execution of these primitive operations is then performed by calling the underlying Resource Deployment Management Service and SW Load Management Service.

To guarantee the timing of service response and communication, a strategy for the partitioning, allocation, scheduling of middleware services as well as for the communication and synchronisation is currently under development. Basically, most of the core services of DySCAS could be implemented as individual software components and tasks. A normal deployment strategy is to follow the inherent resource and decomposition hierarchy in a distributed embedded system. Each individual resource domain, ranging from an individual ECU node at the lowest level, to a network realm (e.g., a group of ECU nodes sharing a CAN bus), and to an aggregation of network realms, is allowed to have its own set of core services that together form a monitoring and decision hierarchy in a cascade way. For example, to decide a migration of functionality in a network realm, the global Autonomic Configuration Management Service communicates and negotiates with the corresponding local instances to find an optimized solution. For dependability reasons, the services can also be implemented with redundant components or distributed. When dealing with aperiodic tasks when their behaviour is not known, share-driven or server-based scheduling [11] is considered useful.

### III. INCORPORATING POLICY-BASED COMPUTING IN DYSCAS

As has been discussed in the previous section, the middleware architecture needs to be flexibly reconfigurable to enable responsiveness to a range of run-time contexts and events. With respect to reconfiguration, the fundamental limitation of embedded systems is that the deployed code is static over the short time frame (this is true even if the deployed image is modular and it is possible to change parts of the image independently, as in the case of DySCAS).

After extensive analysis of the suitability of a range of self-management technologies, policy-based computing was selected as the means for providing the dynamic control logic. The analysis took into account the capabilities of the technologies, the constraints of the deployment environment, and requirements of the identified use-cases. Policies offer a means of keeping the high level business logic separate from the implementation mechanism so that the behaviour can be changed easily without re-deploying code (e.g. field upgrades of policy, and owner customization). A detailed review of the state of practice in policy-based computing has been provided in [12].

The particular characteristics of policy-based computing that make it suitable for DySCAS include: 1. The policy logic is used to specify high-level behaviour, in a standard and platform-independent way; 2. The policy logic is stored separately from the mechanism and is loaded at run-time initiation (or even during run-time), effectively as data rather than as application code; 3. Policy evaluation has low run-time resource requirements; 4. Policy configuration is sufficiently flexible that it is generally applicable to all of the identified use-cases.

Of particular importance for the specific application domain is the fact that policy-based computing does not require that the policy writers are experts in self-managing
systems. They only need to be able to express the desired behaviour of the system using the policy language grammar; and even this requirement will be eased with the eventual introduction of graphically-based policy-authoring tools (which are also being developed within the project).

Although conceptually there is an autonomic configuration service, the actual implementation of self-adaptive behavior in the DySCAS middleware is based on embedding policies at numerous decision points throughout the middleware, i.e. each of the many component services can be policy-configured. This facilitates a highly flexible architecture in which the control logic is dispersed and many different aspects of configuration and behavior can be changed easily. This approach also controls the extent of complexity in the adaptive aspect. This is because there are numerous relatively simple and low latency policy decisions instead of a more monolithic AI approach in which the decision making is concentrated into a more complex central component. The distribution of simple policies makes the policies less error prone and easier to validate and understand. Simpler policies have lower processing requirements which contributes to faster decisions which in turn contributes to the real-time aspects of the system. Simpler policies also require less storage space in the system repository and can be transferred between nodes with low latency.

The insertion of policy control into several services is a cross-cutting concern for DySCAS. To deal with this, a policy wrapper class will be used to preserve the separation of concerns and to enable a consistent approach throughout the entire system.

Several of the identified use-cases require dynamic decisions to be made which are simultaneously influenced by several contextual factors. Thus there is an identifiable need for an alternative decision-making technique, such as Utility Functions (UF), to be supported in addition to policies. UFs have the particular strength of enabling the various contextual factors to be weighted (possibly dynamically) to reflect their relative importance when choosing amongst several alternative paths. As with policies, utility functions have low structural complexity and low processing overheads.

In the automotive middleware ‘signals’ can have several forms: they can be readings from sensors providing raw and direct indications of environmental conditions; they can be pre-processed or aggregated data from several sensors, or from a single sensor averaged over time; they can be contextual notifications of events - such as indicating the detection of a new wireless device such as a mobile phone, or the detection of the failure of an electronic control unit (ECU); or they can be general context information such as the current version or current execution location of a particular software component; or otherwise represent the notion of ‘state’. The various signals are provided as inputs to the policy. The behavior of the system is thus consequent of the current execution environment and context, as well as the configuration of the policy logic.

The implementation approach is illustrated in figure 2. The middleware comprises many services, and each one can embed (possibly several) policies. The policies can be loaded at initialization time (or even in some scenarios it might be possible to load a policy at run-time). The context information is provided dynamically, as this reflects the current state of the system and its operating environment. The context information can come from many different sources including sensors, other services, state information stored in the repository (such as a table of attached external devices), and from interaction with the driver or vehicle occupants.

The collective middleware functionality is achieved through the cooperation of several software components. Thus the behavior exhibited by the system in response to a certain event can be influenced by multiple policies spread across these components. This means that even with individually very simple policies, the system as a whole functions in a rich behavior space.

The specific policy expression language used is AGILE [13], see also [14]. AGILE has a simple grammar which describes the allowed relationships between the nine different types of objects (rule, action, etc). The AGILE language is capable of expressing powerful adaptive behavior within short policy scripts. In AGILE terminology Templates are restricted versions of policy scripts which only provide configuration parameters. Policies additionally contain logic rules and action sequences. The typical mode of use is that a policy will be loaded into a component at run-time initiation. Templates can be subsequently loaded as required to configure the policy at a finer granularity. For example the policy may be specific to a particular vehicle type, or manufacturer, but there can be templates for each driver to facilitate customisation of certain aspects of behaviour.

The current AGILE implementation library was designed as a proof-of-concept research tool, and is not optimized for resource usage or portability, so a new lightweight version ‘AGILE-Lite’ is in development for use in DySCAS.

Figure 3 provides some details of the internal organization of the policy library and the way it is embedded into a software component.

![Diagram of policy implementation approach](image-url)
The typical sequence of events required to initiate policy control of a component is: 1. The host component is initiated by the middleware. 2. During the component initialization, the appropriate policy is loaded from the middleware repository. 3. A template may be applied if necessary to initialize the policy by setting the values of ‘internal variables’ (these are typically used as flags counters and threshold values which parameterize the policies rules). 4. Context information is input to the host component, from sensors or other components. 5. The host component passes appropriate context information (possibly pre-processed) into the policy via the environment variables which serve to decouple the policy from the actual sensors. 6. The host component invokes the policy logic, which applies its rules to the context information and returns a decision result to the host component. 7. The policy logic is internally reconfigurable (details of this aspect are beyond the scope of this paper – the interested reader is referred to [14]), however the policy can be instructed to store its adapted state in the form of a new template; this facilitates a means of persisting new configurations over the longer term. Typically steps 4–6 will be repeated periodically, at a suitable rate that matches the rate of significant context change in the system, or on an event-driven basis.

The project has identified four generic classes of use-case which each comprise several specific use-cases with related functionality. The use cases place requirements on the architecture and in particular on the ways and extents to which flexibility and adaptivity are built into the architecture. Generic use case 1 contains several use cases which are all concerned with the dynamic discovery and incorporation of devices (such as mobile phones or PDAs) and wireless hotspots (such as a home network or a wireless network used for toll collection at a bridge or toll road) that the vehicle comes into contact with dynamically. In this set of use cases policies are used to determine which devices can be connected based on the type of device, its ownership, the services it offers or requests, vehicle owner / driver preferences for security and service prioritization, and environmental context such as the amount of available processing and storage resources available. This last point is very important because the use cases are generally related to infotainment services which involves media processing and streaming and in addition to requiring substantial resources, also has a real-time aspect. Specific architectural impacts of this class of use case include the need for open and standardized communication interfaces, an API for device drivers, and a means of learning and storing dynamically created policy details reflecting driver preferences with respect to various devices and configurations. Generic use case 2 is concerned with integrating new software functionality. The scope covers both operating and application software, and can involve adding totally new components or applications, as well as upgrades of existing components and applications. The use-case might be initiated to perform an update because a problem has been detected with the current software configuration, or because new functionality has become available. Policy decisions will include deciding whether an upgrade is necessary, whether an upgrade is allowed (given the current set of component dependencies), which components to upgrade, and, subsequently, whether to rollback. Specific architectural impacts of this class of use case include the need for a modular binary image, and storage of current configuration details (component versions, dependencies), and of allowed configuration details, in a repository. Generic use case 3 deals with a number of use-cases requiring closed dynamic reconfiguration to provide self-healing and availability. For example, should a critical service fail (or its host ECU fails) the middleware will relocate the service. This may involve shutting down a less important service to free up resources. In this case policies make decisions such as what recovery action to take when a problem is detected, which services are to be retained, and which services can be shutdown. Specific architectural impacts of this class of use case include the need for a modular binary image, a means of prioritizing services, and storing current configurations and services’ resource requirements in an on-line repository, and a mechanism for process migration. Generic use case 4 is concerned with resource optimization. This includes dynamic selection amongst redundant ECUs for reliability and power saving (if unused devices are shutdown). Policy decisions will include which services should be active at each ECU.

Figure 3. The internal organization of the policy library, showing the interfaces to the host software component (the deployed code) and the externally held policies and templates.

IV. THE USE CASES AND THEIR INFLUENCE ON THE ARCHITECTURE

The envisioned future application scenarios for vehicle electronic systems include automatic connectivity with mobile devices that build ad-hoc networks with the built-in devices and utilize network-based services. External mobile consumer electronic devices (e.g. PDAs) need to be dynamically integrated with the in-vehicle entertainment/infotainment system for on-the-fly sharing of computing power, storage, and services (e.g., navigation information).

The project has identified four generic classes of use-case which each comprise several specific use-cases with related functionality. The use cases place requirements on the
and which services or ECUs can be shutdown. Specific architectural impacts of this class of use case include the need for instrumentation (power availability etc.) and the ability to shutdown ECUs.

Note that typically each use case will require the invocation of several middleware services, and thus the overall behavior of the system is resultant of the combination of the decisions made by several policies.

V. CONCLUSION AND OPEN ISSUES

This paper has described a context-aware middleware for embedded automotive systems; focusing on the dynamic configuration aspects. We have justified the need for adaptive behavior and explained the strategy to deploy many simple policies at decision points dispersed throughout the middleware to achieve maximum flexibility, whilst keeping the actual complexity of the dynamic configuration low.

The policy technology to be deployed and the way in which the policies are integrated with the host components has been outlined, and the four generic groups of use cases have been discussed, highlighting the types of policy decisions that are necessary.

There are several open issues which the project must tackle in the coming months, these include:

1. Acceptance. The automotive arena is a traditionally static domain, thus it is important to introduce dynamic behavior in such a way that the system remains predictable and robust despite its autonomy. This requires that significant effort be devoted to verification of adaptive behaviors and validation of overall system concepts. Furthermore, new concepts must be developed for testing and legislation approval, especially for when DySCAS is used for safety relevant systems. Moreover, since the DySCAS middleware is more powerful and complex than static middleware solutions, its implementation is related to additional costs in terms of extra bus capacities, CPU power, and memory consumption. It has to be proven within the project that these costs are fully justified by the benefits coming from DySCAS, in terms of reduced costs due to redundancy, increased flexibility and cost efficiency, and reduced time-to-market.

2. Migration. A migration path from existing static systems to dynamic DySCAS systems is given by the specification approach followed in DySCAS: Since the system specification is independent from the underlying self-configuration mechanisms, it is possible to restrict the self-configuration behavior of the system very strictly. This may be reached by degrading the algorithms for self-configuration to a predefined set of tested system configurations which are activated depending on the system environments/situations. In the extreme case, a DySCAS system may be degraded to a static system. With this, a vehicle manufacturer is free to decide to which extent he would like to introduce reconfigurability to the DySCAS system, paving the way to a migration from existing static systems to DySCAS systems.

3. Standardization. As with the ITEA project EAST-EEA, which may be seen as a starting point for the AUTOSAR initiative, it is intended that DySCAS will form the basis of standards describing self-adaptive configuration and behavior in automotive systems. AUTOSAR and DySCAS are each aware of the other’s objectives. It is mutually important for both projects, and highly advantageous for the beneficiaries (primarily the vehicle manufacturers, and thus the vehicle owners, drivers), if the results of DySCAS can be merged into the AUTOSAR roadmap.

ACKNOWLEDGMENT

The DySCAS project is funded within the 6th framework program “Information Society Technologies” of the European Commission. Project number: FP6-IST-2006-034904. Project partners are: Volvo Technology AB, DaimlerChrysler AG, Enea Services AB, Robert Bosch GmbH, The University of Greenwich, The University of Paderborn, The Royal Institute of Technology (KTH), Systemite AB, and Movimento. The project started in June 2006 and runs until November 2008. Further details are available at the project website [8].

REFERENCES

Abstract—Life shows impressive capabilities: evolution, adaptability, self development/self repair, fault tolerance. Such behaviours are complex and simulating them requires an important amount of processing power. Learning from the European POEtic project [1], this paper introduces the Perplexus Project which aims at creating a scalable, distributed, pervasive and bio-inspired platform to simulate these complex phenomena and study emergent behaviours.

I. INTRODUCTION

There exist several theories that relate to life, its origin and all its associated characteristics. It is however usually considered that life relies on three essential mechanisms that are Phylogenesis, Ontogenesis and Epigenesis (P, O and E in short):

1) Phylogenesis is the origin and evolution of a set of species. Evolution gears species towards a better adaptation of individuals to their environment; genetic algorithms are inspired from this very principle of life.

2) Ontogenesis describes the origin and the development of an organism from the fertilized egg to its mature form. Biological processes like healing and fault tolerance are qualified of ontogenetic.

3) Epigenesis refers to features that are not related to the underlying DNA sequence of an organism. Learning as of performed by neural networks is a process which scope remains limited to an individual lifetime and therefore is Epigenetic.

These three fundamental mechanisms have inspired researchers and led to the realization of bio-inspired artificial systems (be they software, hardware or mixed) that use up to two mechanisms. Some ANN implementations make use of Epigenesis (learning) and have been realized in several forms: dedicated digital circuits (such as proposed in [2] and [3]) or analog circuits [4]. Some other works combine 2 mechanisms like morphogenetic which is a PO combination proposed in [5]. In [6] an evolving ANN is introduced which can be considered as a PE combination (learning and evolution). Finally in [7] variable topology ANNs are presented and can be considered as OE combinations (growth and learning).

The POEtic chip [1] (realized within the confines of the POEtic European project [IST-2000-28027]), was the first chip implementing all three axis of life together, making it the most “living chip” ever created. The main objective of this project was to design a specific reconfigurable circuit well suited for the implementation of POE systems, taking advantage of their “natural”, intrinsic parallelism.

This chip, as depicted in figure 1, contains an embedded processor which runs genetic algorithms (Phylogenesis) and computes the fitness of individuals installed on an array of configurable processing elements named “molecules”. These molecules are similar to FPGA slices and therefore rely on Look-Up Tables (LUTs), registers and reconfigurable datapaths. Molecules are grouped together into “cells” that carry out certain functionalities and may undergo processes such as migration, replication, etc. Since evolution and development processes are unconstrained for the targeted applications, configurable routing uses multiplexers only (comparable to that of Xilinx XC6200 series FPGAs) for discarding any possibility of short-circuit occurrence. All data exchanges and reconfiguration/migration related operations are handled thanks to dynamic routing. This facilitates the mapping of Ontogenetic processes like cell replication that can be triggered by any cell of the design. This last feature is supported by the system that permits transfers of local bitstreams from one molecule to another.

The flexibility of this architecture enables the mapping of systems such as neural networks that range from simple topology-fixed perceptron ANN to more advanced biologically-plausible neural networks models such as the spiking neuron presented in [9].

Unfortunately, the amount of used resources for this last class of systems hardly permits the realization of large networks which limits the scalability of the platform. Furthermore, the scope of possible explorations remains limited to a single individual due to the lack of distributed/collaborative infrastructure for using multiple chips in a networked fashion.
II. THE PERPLEXUS PROJECT

The goal of the Perplexus Project is to develop a scalable hardware platform made of custom reconfigurable devices endowed with bio-inspired capabilities that will enable the simulation of large-scale complex systems and the study of emergent complex behaviors; this in a virtually unbounded wireless network of computing modules. Figure 2 gives an overview of the platform. It will be composed of numerous UBIquitous moDULES so called UBIDULES. These communicating modules will act as a distributed intelligent sensor network capable to compute parts of the global simulation using bio-inspired features and environmental natural parameters.

This paper is organized as follows:
- Section III introduces the Perplexus Ubidle hardware and software substrate which is used in this project.
- Section IV presents the programming framework which covers the different possible methods for specifying distributed bio-inspired applications.
- Section V shows an application that has been developed in order to demonstrate the features of the proposed solutions.
- Section VI draws some conclusions on the realized work and puts it in perspective with the upcoming tasks in the project.

III. HARDWARE-SOFTWARE MIXED PLATFORM

A. Generic Ubidle architecture

As described in the previous section the Ubidle is the tile of the PERPLEXUS architecture. Although it can be specialized by appending application-specific sensors and actuators the common Ubidle platform has the following characteristics:

- An XScale-class processor that manages the software components of the system, such as communications with other Ubidules, monitoring, but also phylogenetic processes. This processor is equipped with an open hardware subsystem which allows connecting any sort of USB device (sensors, actuators, Wifi/Bluetooth dongles, mass storage, etc.). The processor runs an embedded Linux operating system which facilitates Ubidle programming and network management tasks while ensuring portability at the same time.
- An Ubichip [10] [11] [12] composed of an array of processing elements which exhibit a neural networks friendliness for mapping Epigenetic processes. Ontogenetic behaviours such as growth through cell replication are also supported. This chip is based on the experience acquired during the POEtic project and therefore relies on the principles briefly presented in section I. A specific interface was also designed in order to facilitate the grouping of multiple Ubichips.

Figure 3 shows the common Ubidle platform and its abstract representation in the form of a software partition and a hardware partition.

B. The developed mobile platform

Depending on the targeted application, various sensors and actuators may be used for enabling interactions with the

environment. Most applications however imply mobility and therefore an Ubidule usually controls mobile robot motors, and receives data from sensors (video camera, proximity sensors, etc.). Although support for any wireless communication system may be added thanks to USB ports, we chose to use IEEE 802.11 technology (also known as WiFi) in B/G mode. To this purpose, a USB wireless network adapter is used on every Ubidule. The necessary support (drivers, etc.) was added to the Linux kernel we use; this enables remote operation of the device from a workstation.

Figure 4 depicts the prototype realized for validating the work presented in this paper. It is based on a low-cost mobile robot from Wany Robotics\textsuperscript{2}. On top of this robot a compact prototype motherboard was installed; this board along with the XScale module it hosts are provided by the Toradex company\textsuperscript{3}. It features USB ports and general purpose Inputs/Outputs that we use respectively for attaching USB devices and establishing communications with sensors and actuators.

C. The decentralized network infrastructure

The PERPLEXUS platform being made of many computing nodes (Ubidules) whose radio ranges are limited, the use of such protocols is mandatory for ensuring the reachability of every node of the network; as illustrated in figure 5.

There exist in the literature some implementations, they generally fall into two categories: reactive and proactive protocols.

- In reactive protocols, routes are calculated upon transmission request only, using a flood search principle (“local” broadcasts are repeated until target is reached and the route is established by returning the shortest available route). These protocols induce quite high latencies and a variable communication overhead depending on network activity and network stability. If routes become unavailable when used, another flood search is initiated to create a new one connecting transmitter and receiver.

- In Proactive protocols nodes regularly inform their neighbors for their presence. By doing this each node knows other nodes standing one hop away. Then each node broadcast its own neighborhood routing table on the network this allowing nodes standing two or more hops away to complete their own tables. Proactive protocols are more power consuming (for unused or low activity networks), and induce a constant communication overhead but provoke smaller latencies than reactive ones.

A bibliographic study put a light on the OLSR\textsuperscript{4} routing protocol (especially in \cite{13}). This proactive routing protocol regularly sends 3 messages types to create and maintain automatically network routes and is well suited for the kind of applications we target to, mainly in the robotic application case.

OLSR is also open source this allowing us to compile it for our XScale processor based target. Finally the chosen OLSR implementation offers the possibility to use plug-ins to provide additional services such as name/address translation.

\textsuperscript{2}http://www.wanyrobotics.com/
\textsuperscript{3}http://www.toradex.com/
\textsuperscript{4}OLSR : Optimised Link State Routing
In fact proactive protocols are the best solution to deliver messages with a minimal latency due to mostly up to date routes (in comparison to reactive protocols that are slower to establish a route).

For validating this solution we conducted several experiments which confirmed that proactive protocols such as OLSR perform better with respect to latency.

Figure 6 shows the experimental protocol we used for OLSR. This map of the premises shows four nodes, three being static and the last one (Ubidule 3) in motion along the track (illustrated by the plain arrow on figure 6).

Fig. 6: Mobile test protocol.

As suggested in figures 6 and 7, different network topologies are observed, changes from one to another occur whenever a node drops out of radio range of another.

Figure 8 shows the evolution of the bitrates received by the mobile node from the three other units; it can be clearly seen that a change in the network topology results in a break in the communication that lasts up to 5 seconds.

We consider these results satisfactory for the targeted applications; furthermore the flexibility of OLSR allowed us to use a nameservice (DNS-like) plug-in that proves useful in the following.

IV. PROGRAMMING METHODOLOGY AND FRAMEWORK

A. Agent Oriented Programming

The distributiveness and capabilities of the PERPLEXUS platform greatly rely on the chosen programming style. Object-oriented programming (OOP) has become popular during the past decades mainly thanks to object-oriented languages like C++ which provided many benefits over other less formally defined languages.

Agent-oriented programming (AOP) derives from the initial theory of agent orientation which was first proposed by Yoav Shoham [14]. This programming format infers OOP by endowing objects with additional characteristics; they are viewed as entities which exhibit behaviours, capabilities and are entitled to take decisions. Agent-orientation was initially defined for promoting a social view of computing and finds natural applications in areas such as artificial intelligence or social behaviours modeling. An AOP computation consists in making Agents interact with each other through messages of different natures: they may be informing, requesting, offering, accepting, and rejecting requests, services or any other type of information. Messages type is called a performative. AOP furthermore fixes constraints on the parameters defining the state of the Agent (beliefs, commitments, choices) and constraints on the type of messages exchanged between Agents. These constraints actually define the Agent oriented computational system which is then viewed as a set of communicating modules that exhibit a certain degree of awareness. These characteristics naturally geared the PERPLEXUS modeling framework towards AOP which fits perfectly the objectives of the platform and more specifically the applications.

B. JADE : the Agent platform

Numerous Agent Oriented Programming Platforms already exist, such as JADE [15], IXTA, FIPA OS, JAX or MADKIT [16]. Due to a real need of application portability, the major part of AOP platforms are JAVA coded, therefore ensuring portability to various hardware targets or even guaranteeing functionality in heterogeneous hardware platforms [17]. These solutions were evaluated and we finally chose JADE which is a reliable, user-friendly and standardized [18] solution.

JADE (JAVA Agent Development kit) [15] is an Agent Oriented environment that offers services and base classes to develop agent based applications. It additionally exists

Fig. 7: Time changing Topology.
JADE is also IEEE FIPA (Foundations of Intelligent Physical Agents) compliant. This compliancy gives a clear advantage for evolution, portability, reliability, and interoperability with other AOP platforms as the FIPA IEEE standard aims at creating a unified Agent Programming methodology. A SUN phoneME advance JAVA virtual machine was ported to the XScale processor, this provides support for the Light Extensible Agent Platform (LEAP), the light version of JADE. LEAP has almost the same functionality than JADE but is designed for handheld devices making it appropriate for our platform. Agents in a JADE Framework “live” in containers. They exist inside or outside of the original hardware hosting the JADE platform. JADE provides support for both scenarios:

1) **intra platform**, where agents are registered on the same platform and are considered to be local even if they are hosted on an other hardware (left part of figure 9).

2) **inter platform**, where agents are registered on their platform, services discovery allows Agents to find other agents to work with (right part of figure 9).

In both cases inter hardware messages are transmitted through TCP protocol.

![Fig. 9: JADE programming platform.](image)

A JADE platform (Figure 9) contains at least one container named “Main-Container”. This “Main-Container” contains at least three agents, the Agent Management System (AMS), the Directory Facilitator (DF) agent and the Agent Communication Channel (ACC) which is not viewed as an agent but more a set of services.

- The AMS agent represents the authority in a JADE platform. It manages the life cycle of all agents of the platform; it can create, suspend or kill agents. It updates a list containing the name of all agents present on the platform, providing a “white pages” service. Finally, it controls the communication channels between agents.
- The DF agent records the services provided by all the agents of the platform, it acts as a “yellow page” service.
- The ACC agent manages the communication between agents. The exchanged messages can be intraplatform and intracontainer (JADE events), intraplatform and extracon- tainer (RMI) or extraplatform (http protocol).

### C. Perplexus Programming Framework

Bio-inspiration and the three fundamentals of life being at the heart of the project, the proposed framework extends JADE default agents (AMS, DF and ACC) by defining agents whose purpose relate to both interfacing and bio-inspired (POE) mechanisms support.

Figure 10 schematically depicts the Ubidule programming which is regarded as a mixed hardware/software entity: the Ubichip and the Xscale microprocessor that runs the Linux operating system.

The framework specifies 7 agents belonging to 2 families:

- **Application agents**: Phylogenetic agent(s), Ontogenetic agent(s) and Epigenetic agent(s).
- **Infrastructure agents**: UbiCom agent(s), Interface agent(s), Network agent(s) and Spy agent(s).
These agents are either software, hardware or operate at the boundary between hardware and software as described in Table I. Ontogenetic and Epigenetic agents may either run in software or hardware.

<table>
<thead>
<tr>
<th>Agent Type</th>
<th>Software</th>
<th>Hardware</th>
<th>HW/SW Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phylogenetic agent(s)</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ontogenetic agent(s)</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Epigenetic agent(s)</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Ubicomm agent</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network agent</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Interface agent(s)</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spy agent</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TABLE I: Agents operating mode

V. VALIDATION APPLICATION

In order to demonstrate the features of the programming framework, we chose to develop a simple application that makes use of some infrastructure and application agents.

The chosen application relies on an obstacle race that takes place in a circular arena as depicted in Figure 11. Improving participant performances -which are expressed as lap time- is achieved by evolving the population over multiple generations (Phylogenesis). The individuals have an average lifetime that is set by the application; whenever an individual dies a new one resulting from the Phylogenetic process is grown (Ontogenesis) onto the same robotic host.

Each robot is equipped with three proximity infrared sensors (Front, Left and Right), each of which delivers a Boolean information whenever an obstacle is detected within a 15cm range. The robots are controlled via setting one speed value for each of the two driving wheels motors. These values are integers in the $[-7, +7]$ range, sign changes yield to changes in rotation direction. For the simplicity of the sensors information (3 bit information) and commands ($2 \times 4$ bits) we chose not to use a neural network but rather implement the behavior in a Look-Up-Table fashion (truth table). This application therefore makes use of two mechanisms: Phylogenesis (evolution) and Ontogenesis (growth) whereas Epigenesis is unused: the behavior of each robot is completely specified in its genome.

Next to the three POE agents, four additional Agents have been defined for interfacing and networking purposes:

- **I Agent**: The Interface agent provides a set of methods for issuing commands to the actuators or retrieving data from the sensors of the Ubidule.
- **U Agent**: The UbiCom agent provides software API-like access to the Ubichip.
- **S Agent**: The Spy agent provides information on the platform state (agents status/results, activity traces, bug notification).
- **N Agent**: The Network agent provides a collection of methods for network-related aspects: time-synchronizing of data among Ubidules, setting / getting clusters of Ubidules, obtaining the list of neighbors, etc. For it requires access to low-level network-topology information, it also implements the MANET functionalities.

Finally a Host Agent (H Agent) instantiated on a workstation allows controlling remotely the Perplexus platform (Start/Stop/Schedule actions).
The specific parameters of this bio-inspired application are as follows:

- The population is composed of three robots.
- The genome of a robot is composed of a $2 \times 8$ array giving the left and right motors speeds depending on the front, right and left sensors information. These data provided by the infrared sensors are translated into speeds information for the two motors of the robot that adapts its behavior if an obstacle is detected. The speed of a single motor is coded with a 4 bits signed value $[-7; 7]$.
- The fitness function reports the quality of the expected solution; in our application the fitness is represented by the lap time.
- The duration of a generation is expressed in laps for avoiding excessive rounding.
- The termination criteria of the application.

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Genome 1</th>
<th>Genome 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L Spd</td>
<td>R Spd</td>
</tr>
<tr>
<td>Front</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Left</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Right</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Front</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Left</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Right</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Front</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Left</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Right</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**TABLE II: Genome examples**

Table II shows two example genomes. The first three columns give the outputs of the three IR sensors (Front, Left and Right). The two following columns are the effective columns giving the outputs of the three IR sensors (Front, Left, Right). The two following columns are the effective genome for a first robot, giving the two motors speeds corresponding with the sensors output. As said previously speeds are code from -7 to 7:

- 0 stands for no move
- Values in the $[1; 7]$ range result in forward motions with increasing speed.
- Values in the $[-1; -7]$ range result in backward motions with increasing speed.

The following describes the 6 different phases leading to the execution of a generation for three robots and the creation of the next generation:

Phase 1: The Host agent runs the `sendApplicationParameters()` method, this method broadcasts some application dependent parameters to all the Network agents which in turn forward some of these parameters to their respective local Phylogenetic agent. Those parameters comprise the individual lifetime, genome description and transcription into individual characteristics, methods for Phylogenetic agent and termination criteria of the application.

Phase 2: The P agent of each Ubidule computes a random genome and sends this genome to the Ontogenetic agent. The O agent is responsible to give birth to the individual by instantiating the E agent which is a dummy E agent here since the behavior is in form of a LUT.

Phase 3: All the bio-inspired agents are initialised; Ubidules are ready to start the race. Each P agent informs the H agent that it is ready. When the host has received all the ready information messages, it runs the `startApplication()` method which broadcasts a start message to all the Ubidules. The race starts.

Phase 4: Each Ubidule runs for its entire lifetime. Every time the Ubidule crosses the start/finish line its fitness is broadcasted by the Phylogenetic agent to the other Ubidule Network agents through the local network agent.

Phase 5: When the generation is over, every Phylogenetic agent computes a new individual based on the previously received fitness information of all other Ubidules. The Phylogenetic agent requests the genome of the individuals that have been selected for the crossing process. The creation of the new generation then begins using the phylogenetic methods passed during application initialisation.

Phase 6: Once the new individual genomes have been calculated they can be deployed similarly to phase 2: the Phylogenetic agent passes the local individual genome to the Ontogenetic agent that instantiates the Epigenetic agent according to the genomic description.

The process composed by phases 1 to 6 is then repeated until the termination criteria is verified; which can be in this case a given fitness or a number of generations.

**VI. CONCLUSION**

Several experiments have been conducted in order to validate the different aspects of this platform such as the self-organizing network management mechanisms based on the standardized OLSR protocol and the services provided by the infrastructure agents.

Among these experiments the example application, although purely software so far, demonstrated the use and the viability of such a platform. We estimate that the imminent availability of many Ubiphieqipped Ubidules will allow conducting more extensive tests for more realistic applications. Within the project, three applications are planned for demonstrating the features of the platform: a collective robotic garbage sorting application, a biologically plausible Neural network simulation and a culture dissemination simulation.

**REFERENCES**


HLS-based Implementation of Real-time face detection parallel architecture

Nicolas Farrugia and Franck Mamalet and Sébastien Roux  
Orange Labs - IDEA Laboratory  
28 Chemin du Vieux Chêne  
38243 Meylan, France  
Email: firstname.lastname@orange-ftgroup.com

Fan Yang and Michel Paindavoine  
Laboratoire LE2I  
Université de Bourgogne  
21000 Dijon, France  
Email: fanyang at u-bourgogne.fr, paindav at u-bourgogne.fr

Abstract—We describe in this paper a High-Level Synthesis implementation of a parallel architecture for face detection. The chosen face detection method is the well-known Convolutional Face Finder (CFF) algorithm, which consists in a pipeline of convolutions operations. We rely on dataflow modelling of the algorithm and we use a High-Level Synthesis tool in order to specify the local dataflows of our Processing Element (PE), by describing in C language inter-PE communication, fine scheduling of the successive convolutions and memory distribution and bandwidth. We then build a parallel architecture composed of a PE ring and a FIFO memory, which constitutes a generic architecture capable of processing images of different sizes. A ring of 25 PE running at 60 MHz is able to process 27 VGA images per second.

I. INTRODUCTION

Face detection and analysis in images and video streams is an important research field and has many applications in security access control, image indexing, and person identification. New applications on power constrained devices are foreseen, like video coding in mobile videoconference and intelligent user interfaces.

Many algorithms for face detection have been proposed in the past twenty years [1]. The chosen face detection method is the Convolutional Face Finder (CFF), introduced by C.Garcia and M.Delakis in [2]. It leads to the best performance on standard face databases. The CFF is an image-based neural network approach that allows robust detection, in real world images, of multiple semi-frontal faces of variable size and appearance, rotated up to ± 20 degrees in image plane and turned up to ± 60 degrees. In [3], the authors have shown that the CFF algorithm can be implemented efficiently on embedded software platforms, while keeping good detection rate and a low false alarm rate. Many optimizations were done, leading to significant gains in terms of processing speed and memory requirements, thus enabling face detection on a mobile phone with 5 QCIF (176 × 144) frames per second and only 220 KBytes of memory [3]. However, face detection is most often the first step of a face analysis process and will require a faster system.

There are been a few attempts of hardware systems for face detection [4] [5] but the authors report lower detection rates and higher false alarm rates than with the CFF [2], and frame rates up to 50 QVGA (320 × 240) frames per second. So far no hardware implementation of the CFF algorithm has been reported.

We therefore aim at realizing the first fast and robust face detection optimized hardware by defining a parallel architecture for the CFF algorithm. In [6] we have described an optimized Algorithm Architecture Matching methodology, consisting in dataflow modelling of the algorithm, parallelism extraction and complexity analysis. Using this information we have performed a coarse-grain design space exploration, which enabled us to specify an efficient parallel architecture, consisting in a ring of PE where each PE processes the whole face detection algorithm on a small block of data and communicates a small amount of data with one of its neighbour. In this paper we present the implementation of such a PE capable of handling the successive convolutions of the CFF algorithm.

Hardware convolution implementation has been extensively studied in the past 20 years [7], and solutions have been proposed, using for example linear or systolic arrays [8] of processors elements. Recently, reconfigurable convolvers have emerged [9], capable of handling several mask sizes and variable datapath width. In [10], the authors propose an implementation with three levels of memory hierarchy which enables important power savings, by careful data reusing schemes. But these solutions are not flexible enough to handle the variability of the kernel size of the successive layers of the CFF algorithm, necessary to implement the whole convolutional network.

The remainder of the paper is organized as follows. In section 2, we present the CFF algorithm and previous works which have introduced a dataflow model of the algorithm and a theoretical architecture of PEs implementing efficiently this algorithm. We present in section 3 our High-level synthesis based implementation in order to accelerate and ease the development and validation of a PE. We also give processing times and synthesis results for this PE. In section 4, we present a parallel architecture composed of the previous PE, which enables us to process images of different sizes. We finally give the performances of this architecture, and we compare our face detection systems with the ones in the literature. We will then conclude this paper and give our perspectives in section 5.
II. THE CFF ALGORITHM: OVERVIEW AND PREVIOUS WORKS

A. Overview of the CFF Algorithm

The Convolutional Face Finder was presented in [2] and relies on Convolutional Neural Networks introduced by LeCun and al. [11].

In this paper, we will only consider the core of the face localization process as depicted in figure 1. The convolutional neural network used to implement the face detector has been previously optimized in [3], and consists in a set of two different kinds of layers:

- Layers CSi are called convolutional layers, and contain a certain number of planes. Each element in a plane receives input from a small neighbourhood (biological local receptive field) in the planes of the previous layer. Each plane can be considered as a feature map that has a fixed feature detector, which corresponds to a pure convolution with a mask applied over the planes in the previous layer. A bias is added to the results of each convolutional mask. Multiple planes are used in each layer so that multiple features can be detected. Once a feature has been detected, its exact location is less important. Hence, each convolutional layer CSi is typically done with horizontal and vertical steps of two pixels which corresponds to perform local averaging and subsampling operations. Then, the results are passed through a hyperbolic tangent function, used as an activation function. As a result, layer CSi also performs a reduction by two of the dimensionality of the input.

- Layers Ni are called classification layers and are applied after feature extraction and input dimensionality reduction of CSi layers. These layers correspond to a multilayer perceptron.

All parameters (convolution kernels, biases, neuron weights) have been learnt automatically using a modified version of the backpropagation algorithm with momentum [2]. In the remainder of this paper, we will only consider the face localization process when training has been completed.

The detail of CFF layers is given in figure 1:

- Layers CS1 performs convolutions with masks of dimension $6 \times 6$. It contains four feature maps and therefore performs four convolutions on the input image.
- Layer CS2 performs $4 \times 4$ convolutions and has 14 feature maps. Each of the four subsampled feature maps of CS1 is convolved by two different $4 \times 4$ masks, providing the eight first feature maps of CS2. The other six feature maps of CS2 are obtained by fusing the results of two $4 \times 4$ convolutions on each possible pair of feature maps of CS1.
- Layers N1 contains 14 sigmoid neurons. Each neuron is fully connected to exactly one feature map of layer CS2 whose size is $6 \times 7$.
- Layer N2 consists of a unique neuron fully connected to all the neurons of the layer N1. The output of this neuron is used to classify the input image as face (positive answer) or non-face (negative answer).

B. Previous works - Dataflow model description

In [6], we have established a Design Space Exploration methodology based on dataflow modelling and parallelism extraction of the CFF algorithm. This exploration has shown that the maximum parallelization efficiency is obtained by massive data parallelism exploitation with a ring of PE.

We detail in figure 2 the corresponding dataflow model for a given path in the CFF Algorithm, which involves four successive PEs ($N$, $N - 1$ $N - 2$ and $N - 3$) and seven successive temporal iterations for the calculation of two N1 layer output data.

Each PE processes the whole algorithm on a block of 8 lines of 12 pixels and transfers a small amount of data (overlapping parts) to one of its neighbours. The several slashed parts in figure 2 represent the data to be transferred between successive PEs.

To start with, CS1 $6 \times 6$ convolutions are applied on a $12 \times 8$ block of data with horizontal and vertical steps of two pixels, thus producing $2 \times 4$ output data. Then, PE $N$ has to send a $2 \times 2$ output data to PE $N + 1$ and receives a $2 \times 2$ block from PE $N - 1$. Using data from previous iteration, CS2 is applied on a $4 \times 6$ data block which produces 2 data to be used as input of N1 layer. These data are sent to its nearest neighbour and each PE gathers 5 data from 3 other PEs. With the 6 previous iterations, 2 N1 output data can be computed and so 2 face detections.

III. IMPLEMENTATION OF A PROCESSING ELEMENT

In this section, we detail a complete implementation of a PE able to process the whole CFF Algorithm. The design of this PE is guided by the results of the DSE/AAA methodology presented in [6]. The coarse-grain PE model previously used in this exploration doesn’t detail neither the internal dataflow needed to process the convolutions, nor the way the coefficients for the convolutions are handled.
In this paragraph, we will describe the local dataflows necessary to implement each CFF layer on the same PE. We will see that the requirements of the CFF algorithm in terms of dataflow management makes it difficult to do an optimized design in hand-held VHDL coding.

We will then present a High Level Synthesis based implementation which enables us to quickly evaluate several implementations of this complex dataflow. We then give simulation and synthesis results of this implementation.

A. Convolutions and Local Dataflows

In the design of a PE, we need to consider the local dataflow of the computation in terms of input data and also in terms of convolution kernel coefficients. Each PE processes CS1, CS2, N1 and N2 layer on an input block of $12 \times 8$ pixels, as described in figure 2. To illustrate the complexity of the local dataflow, we will focus on CS1 layer whose details are given in figure 3:

The slashed part on the left side (respectively on the right side) represents the first $6 \times 6$ input window (respectively the result of CS1 on this input window). Dotted borders represent the next horizontal and vertical input windows (respectively the next results) considering a horizontal and vertical step of 2 input pixels. There are 8 input windows to compute, depicted in figure 3 by crosses on their top-left corner. These windows comprise many overlapping data: 17% of the input image block is involved in the computation of 6 different windows.

Thus, the first coefficient of each kernel is applied to these data, and each coefficient is used 8 times. Besides, each window is processed with 4 different CS1 kernels. This complex local dataflow can also be represented sequentially as a 5-deep "for" loop nest written in C language, as depicted in figure 4. The inner operation in the loop nest is a Multiplication Accumulation (MAC) between a kernel coefficient, an input data and a previous accumulation. Furthermore, as presented in §II-A, CS2 and N1 layer have also very complex local dataflows.

Our main goal is to design a PE which processes the whole algorithm on a window of size $12 \times 8$ and communicates with its neighbours as described in the dataflow defined in the previous section (II.3). Concerning the local dataflows of the algorithms, there are three main issues to cope with:

1) PE memory organisation and bandwidth: each feature map has to be stored and read back in a parallel way, to enable local task parallelism (e.g. 2 CS2 feature maps computed in parallel).

2) Fine scheduling and address generation: a choice of scheduling has to be done to efficiently exploit data locality and convolution’s regularity, and address sequences have to be computed for each memory.

3) Inter-PE communication and temporal iterations: overlapping parts between PEs have to be transferred and data between successive iterations have to be stored and read back.

Each issue can be addressed with several implementation solutions. For example issues one and three involve a choice of the type of memories (FIFO or shared RAM, double port RAM, register banks...) and their size, as well as the definition of the links between PEs. Besides, there exists many fine scheduling choices to exploit different kinds of parallelism, and for each scheduling, complex address generation has to be computed for each memory.

The complexity and interdependency of these issues makes it hard to do a handheld RTL description for each possible implementation solution. Even if theoretical study using polyhedral models and loop transformation techniques [12] could be done in order to define an optimized fine scheduling of one loop nest, such a study on the whole algorithm is out-of-the-scope of this paper.

We present in the next paragraph a High-Level Synthesis (HLS) flow which enables us to explore in high-level language $C$ several fine schedulings, memory organizations and bandwidths and inter-PE communications, and to automatically
generate a synthesizable PE including datapath, control and address generation. Our objective is to obtain quickly an efficient and functional PE prototype which will be used to implement the parallel dataflow described in the previous section.

B. PE Design using a High-Level Synthesis Tool

1) High-level synthesis: We have chosen to use a HLS approach in order to accelerate exploration, implementation and validation of the PE. Many approaches for HLS have been proposed. There are commercial tools [13], [14], [15] which provide complete frameworks for hardware synthesis. Such tools are designed to accelerate and ease the development in a hardware project. The formalisms, formats or platforms used are thus proprietary and often very specific and locked. Therefore, these tools may not be well suited for research purposes. On the contrary, some academic tools are open-source [16] [17], and provide the user with more flexibility and visibility on the tool behavior. We present, in the next subsection, our implementation using UGH (User Guided HLS), an open-source HLS tool integrated in a larger framework for SoC design called Dysident, and was developed by LIP6 laboratory. This tool is adapted to our study for two main reasons. First, it allows to target an architecture by specifying the available resources before synthesis. This allows us to rely on the previous results of our manual PE implementation [6]. Secondly, UGH allows us to describe directly in C language all data Input/Output transfers, address calculation and dataflow management (e.g. ping pong strategies, modulo addressing, multiple data sources, etc.) which are required for our application [16].

2) UGH design flow: We will focus on the first part of UGH design flow (figure 5) which performs a Coarse-Grain Scheduling (CGS) with a behavioural C description of the algorithm and a Draft Data Path (DDP). The C description uses a subset of the C language whose main limitation is the forbidden use of pointers. Special datatypes are introduced to handle variable bit sizes (for example, unsigned integer coded on 4 bits). The DDP describes the available resources for the target design (e.g. ALUs, RAMs, multipliers, bit wise logic, ..) , and UGH-CGS binds and schedules the algorithm on the available resources, and generates a VHDL Finite State Machine (FSM), and a VHDL structural datapath which are both synthesizable.

Two particular features of UGH have been much used in our design:

- **Automatic parallelization, datapath and address generation**: UGH-CGS is able to detect parallelism at the instruction level and schedules it on the available resources which are specified in the DDP file: UGH-CGS is then able to automatically generate the corresponding datapaths and all necessary links (multiplexers and corresponding connexions) between the resources, and also all control signal such as memories addresses.

- **Flexible communication model**: Input and Outputs can be specified in a direct and flexible way, by simply writing specific "ugh_read" and "ugh_write" operations at any time of the source C code, and by defining the corresponding input or output port in the DDP.

Given that the C description exhibits some parallelism at the instruction level, the first feature makes it possible to quickly test and compare several implementation alternatives (for example, parallelism degree). The second feature enables us to describe our PE with the same dataflow than in our previous dataflow modelling with SynDEx, by specifying the data acquisition and the successive transfer between PEs as we described it before.

3) Data Paths and low level Parallelization: As explained in the previous paragraph, in order to enable UGH-CGS to parallelize computation on the available resources (DDP), a hand-held loop transformation and unrolling has to be done. For instance, we present, in figure 6, a loop nest permutation and unrolling that enables to exhibit an inner loop with 4 independent MACs using the same kernel coefficient. Then, specifying in the DDP file that 4 multipliers and 4 adders are available, UGH-CGS is able to automatically generate four appropriate datapaths and all necessary control signals; an example of such a datapath is shown in the bottom-left of figure 6.

Such transformations have been done for each CFF layer. Table I details for each CFF layer an example of loop nest permutation enabling UGH to parallelize the algorithm on 4 or 8 datapaths.

4) Memory issues - parallel accesses, data reuse and modulo addressing: UGH memory models are only single or dual port memory. In order to be able to parallelize the computation on several datapaths, multiple memory banks have to be introduced, thus increasing memory bandwidth (e.g. to read four distinct columns of input data for CS1, we use four memories). This can be easily managed with UGH by dispatching data on different C arrays. UGH generates one memory bank per C array and handles address generation for each memory. By this way we control the memory access parallelism, by specifying exactly the number and size of the memories needed for our system.

Furthermore, careful address calculation has to be per-
formed in order to handle seven successive iterations, which can be simply done in C by computing the necessary addresses using increments and modulos (for example, in CS1, four new lines are acquired and four previous lines are reused in the current iteration).

C. PE validation by simulation and synthesis

Using techniques described above, we have implemented a full PE able to handle CS1, CS2, N1 and N2 layers. UGH successfully generates the FSM and the Datapaths implementing the entire algorithm. We also have defined the appropriate I/O ports in the PE, in order to handle input data acquisition, initial coefficients load, CS1 and CS2 data transfers and face detection results restitution. We have validated the generated VHDL using the same test image in the initial behavioural C description of the PE and in a VHDL testbench.

The processing cycle times for a PE containing 4 datapaths are summed up in table II, where each layer of the CFF algorithm is detailed. We also point out in this table the efficiency of parallelization on the datapaths, by computing for each CFF layer the theoretical minimum number of cycles \( T_{th} \) achievable (the number of MACs divided by the number of datapaths) ; the efficiency is then equal to \( Eff = \frac{T_{th}}{T_{par}} \).

We obtain an overall parallelization efficiency of 0.7 for the complete CFF calculation. This efficiency is mainly due to the overhead implied by the FSM controlling the system, and remains high considering the complexity of the algorithm.

First synthesis results for a PE with 4 datapaths are given in table III for a Virtex 4 SX 35 FPGA. No major modifications have been made on the VHDL code generated by UGH. Implementation memory banks are balanced between Flips flops, Distributed RAM (RAM implemented in logic blocks [18]) and Block RAM (embedded static RAMs), in function of the memory bank size and the number of ports needed. DSP48 blocks are used to implement multipliers in the datapaths.

The maximum frequency of the PE is 60 MHz and is mainly due to unpipelined utilization of DSP48 blocks and the large number of memories which imply many multiplexing logic.

IV. Multi-PE Parallel System

In this section we present a parallel architecture based on the previously designed PE with 4 datapaths. In the first subsection, we describe this architecture and its application for processing a whole image of any size. In the second subsection, we give its performances, in terms of frame processed by second, for different image sizes, and we finally...
TABLE III
4 D ATAPATH PE: SYNTHESIS RESULTS ON VIRTEX 4 SX 35 FPGA

<table>
<thead>
<tr>
<th></th>
<th>Device occupancy / capacity (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slices</td>
<td>3312 / 15360 (21)</td>
</tr>
<tr>
<td>Number of Flip Flops</td>
<td>164 / 30720 (1)</td>
</tr>
<tr>
<td>Number of Block Rams</td>
<td>19 / 192 (9)</td>
</tr>
<tr>
<td>Number of DSP48</td>
<td>4 / 192 (12)</td>
</tr>
<tr>
<td>Maximum Frequency</td>
<td>60 MHz</td>
</tr>
</tbody>
</table>

**Fig. 7. Generic Ring + FIFO Architecture.**

compare these performances to other face detection systems in the last subsection.

**A. Generic Parallel Architecture**

In previous work [6], we have shown that a good tradeoff between efficiency and number $N_{PE}$ of PE is obtained when the input image is divided in $P = N_{PE}$ blocks of 8 rows of 12 pixels: each block is processed by one PE, and each PE is connected to two other PEs, thus building a ring architecture. Considering data overlapping, this allows to process an image of width $12 + (N_{PE} - 1) \times 8$ and of any height greater or equal to 36 (the minimum number of rows necessary to compute a face detection). We rely on this result to establish a generic architecture using a ring of PEs and a FIFO memory (figure 7).

We divide the input image in vertical strips of width $12 + (N_{PE} - 1) \times 8$ and process each strip by dividing it in blocks of 12x8 as described above. The FIFO is connected to the first and the last PE of the ring and is used to provide the overlapping data from the previous strip, and to write the overlapping data for the next one (grey blocks in the top of figure 7). We have successfully simulated a 4 PE ring with FIFO. Synthesis estimates show that a ring of 25 PE fits in a Virtex 5 LX 330 device, the largest FPGA available.

**B. Performances**

In this architecture, each PE works synchronously. At each vertical iteration, 4 new input lines are loaded in the PEs and if we consider that this load is done in pipeline with the algorithm computation, we can give the processing time of the algorithm on a complete image of size width $L$ and height $M$ using the following formulas:

- $T_{CFF}$: time to process the CFF algorithm on one $8 \times 12$ block
- Number of vertical iterations: $N_{it} = \frac{M - 4}{8}$
- Number of vertical strips per PE: $N_{str} = \frac{L - 4}{8 \times N_{PE}}$
- Processing time of a strip: $T_{str} = N_{it} \times T_{CFF}$
- Total Processing Time: $T_{tot} = N_{str} \times T_{str}$.

In order to detect faces of different sizes, an image pyramid is built to apply the CFF algorithm on each image (see [3]). This pyramid is constituted of a set of sub-images obtained from the initial image by applying a reduction factor of $1.2$ on each dimension. We can have an estimation of the total processing time of the entire pyramid by summing the processing times for each sub-image.

We sum up in figure 8 the performances (in frames per second) of this parallel architecture on several image sizes and for 1 to 64 PEs (to improve image readability both axes have logarithmic scales). We also represent on this figure the real-time threshold of 25 fps: we can see that we achieve real-time face detection with a ring of 8 PEs for images up to $640 \times 480$.

Table IV presents a comparison between the main hardware implementations of face detection found in literature. All these implementations are done on FPGA hardware and therefore run at relatively low clock frequencies (except [4] which is an ASIC implementation). We can see that our system compares well with the others in terms of frame rate, depending of the number of PEs implemented. An architecture consisting of 4 PEs fits in a Virtex 4 SX35 FPGA and performs face detection at 22 QVGA images per second which is close to real-time. When considering a larger system with 25 PEs, we can achieve real-time face detection with VGA images, and process 98 QVGA images per second, which leaves enough time to consider other face analysis tasks in real-time following face detection.
The main drawback of the other implementations is usually a significant loss in overall detection rates or even a lack of detection performance measure: most works don’t give results on detection accuracy of their hardware implementation. As a contrary, our system has the same detection performances than the original CFF software implementation [3], therefore it is very robust in terms of detection accuracy and has a very low false alarm rate [2].

V. CONCLUSIONS - PERSPECTIVES

We have implemented a parallel architecture for face detection composed of Processing Elements based on CFF algorithm. We have implemented our PEs using a High-Level Synthesis approach which enabled us to obtain a fully functional PE composed with 4 datapaths exploiting local parallelism of the algorithm. This PE has been successfully simulated and synthesized on a Virtex 4 SX35 FPGA, and occupies approximately 21 % of the device, and is capable of running at a maximum frequency of 60 MHz. We have then presented a ring of PE with a FIFO memory, which constitutes a generic parallel and scalable architecture able to process images of variable sizes. Such an architecture with 4 PEs can process up to 22 QVGA images per second. An architecture with 25 PE achieves real-time face detection of VGA images.

We are currently working on optimizing this PE in order to be able to implement more PE in our targeted FPGAs, and to increase its maximum clock frequency. In future works, we will investigate the generalization of such modelling and architectures for other algorithms based on convolutional neural networks.

ACKNOWLEDGMENT

The authors would like to thank Christophe Garcia for his support and help on the CFF algorithm. We are also very grateful to Clément Quinson for his help for starting up with UGH, and UGH team for their support and encouragement on this work.

REFERENCES

Algorithm-Architecture Affinity
– Parallelism Changes the Picture

Rasmus Abildgren*, Aleksandras Šaramentovas†, Paulius Ruzgys†, Peter Koch†, and Yannick Le Moullec†

*Center for Embedded Software Systems(CISS)
Aalborg University, Selma Lagerløfs Vej 300,
DK-9220 Aalborg East, Denmark
rab@es.aau.dk

†Center for Software Define Radio (CSDR)
Aalborg University, Fredriks Bajers Vej 12,
DK-9220 Aalborg East, Denmark
{aleksara,paulius, pk, ylm}@es.aau.dk

Abstract—Reducing the time-to-market factor is a challenge for many embedded systems designers. In that respect, hardware-software partitioning is a key issue which has been studied during the last two decades. In this paper we present an extension to recent works dealing with metrics for guiding the hardware-software partitioning step. This extension builds upon and complement our own work with metrics in the Design Trotter project, and is combined with the affinity metric approach. We show that the proposed extension improves the original affinity metric in terms of parallelism detection, and thus can help system designers to make wiser hardware-software partitioning decisions, which in turn reduces the time-to-market factor.

I. INTRODUCTION

In order to achieve more advanced and faster services in embedded systems, increasingly sophisticated algorithms are used. To keep abreast with the increased need for processing power, heterogeneous multiprocessor platforms are introduced, which includes GPPs, DSPs and FPGAs.

Introducing this variety of processing elements (PEs), not only increases the computational capacity of embedded systems but also adds various computational properties. To exploit this increased capacity and properties, the designer needs to find the best suited PEs for the different system functionalities. By considering these facts together with all the system constraints (Area, Time, Power, Price, Development Time), it becomes a non-trivial task to decide how the system functionality should be mapped on the architecture.

To handle this task system level design methodologies have been developed, including structured design space exploration (DSE). A suite of academic DSE frameworks, e.g. [1]–[3], as well as commercial tools have been proposed, in order to provide the design engineer with qualitative information for partitioning.

Exploring the design space with optimising for different constraints is known to be \( \mathcal{NP} \) hard [4]. The DSE in these frameworks is therefore carried out as heuristic simulations, which still can be a time-consuming but necessary task for state-of-the-art large scale products. Large companies can usually find these resources and keep up with their competitors.

However, small and medium enterprises (SMEs), which typically sell state-of-the-art products of much smaller volumes, must also stay on the competitive edge. They are also restricted by the time-to-market factor, and can also benefit from using system level design methodologies (SLD) and tools. Unfortunately, many SMEs can not afford tools and specialists like big companies, and therefore have problems with changing their design methodology into SLD methodology.

We have examined the design methodology of a high-tech company in Denmark and found that the design space exploration phase in their overall design trajectory is limited in the sense that their partitioning depends on prior design, designers intuition and experience, and in rare cases on ad hoc analysis. Danish Technological Institute, a consulting company helping many SMEs incorporating new research results, agrees on that picture in most SMEs [5].

As a consequence of sticking to ad hoc design methodologies, SMEs development often run into situations where redesigning part of the system is necessary and therefore increases the time-to-market.

In this paper we propose an extension to the existing affinity metric proposed in [6] for guiding the partitioning of the system specification, and help making the DSE faster and easier. The rest of the paper is organised as follows. In section II, the existing affinity metric is presented and examples for the need of an extension to the original metric are shown. In section III the new proposed metric for parallelism is presented. The benefits of the proposed parallelism metric are illustrated in section IV by means of a Reed-Solomon decoder case-study. Finally we conclude in section V.

II. AFFINITY METRIC

This section summarises the affinity metric proposed by D. Sciuto et.al. in [6], [7], and argues for the need of an extension of this metric. The affinity metric is designed to guide the design partitioning of system specification between general purpose processors, DSP processor, and FPGA/ASIC. The metric consists of a triplet of values \((A_{GPP}, A_{DSP}, A_{FPGA})\) indicating the match between the processing elements and the examined code. The individual values in the metric are calculated based on 14 other metrics which are designed to measure the source code for certain patterns highly correlated with architectural properties. The measurement is a static analysis of the source code and the metrics are defined as ratios between lines with specific properties, e.g., the ratio between lines with a condition and the total number of lines, or...
defined as the number of assignment of a special type related to the total number of assignments. The metrics measure properties such as data types, Harvard architecture patterns, MAC patterns, and bit manipulation.

To illustrate how the affinity metric works on a real life example, we have applied it onto c-code (Fig 3) calculating a matrix multiplication. The results of the different metrics are shown in table I:

<table>
<thead>
<tr>
<th></th>
<th>A_{GPP}</th>
<th>A_{DSP}</th>
<th>A_{FPGA}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.89</td>
<td>0.96</td>
<td>0.39</td>
</tr>
</tbody>
</table>

The normalised metric values indicate that the best architecture matching the algorithm is a DSP architecture, which the designer could easily rely on. An in-depth analysis of the code shows that besides the already extracted properties from the affinity metric, a high degree of inherent parallelism is present in the matrix multiplication algorithm. This is further discussed in section III. A high degree of inherent parallelism indicates that the algorithm is suited for parallel execution. This is one property of a FPGA architecture, and the original affinity metric does not consider it.

### III. Parallelism Metric

From the analysis of the matrix multiplication shown in Fig 3, we see that the inherent parallelism of an algorithm is an important parameter. Therefore it would be beneficial to measure the degree of inherent parallelism in the algorithm and use this in calculating the $A_{FPGA}$ value of the affinity metric.

One of the first metrics considering the parallelism is Amdahl’s speedup metric [8]. Here the potential execution speedup of an algorithm is defined as the ratio between the sequential execution, and the fully parallelised execution. What determines the fully parallelised execution is the critical path in the algorithm.

This is also the case for more recent parallelism metrics e.g. [9], [10], so let us consider the critical path by looking at precedence graphs.

**Definition 1:** Let $G = (N, E)$ represent the precedence graph of a method, $n_i$ where $N$ represents the set of nodes $n$ and $E$ is the set of edges $e_{i,j}$. A node $n$ can have a source node and a destination node. If the node does not have a source node, it is defined as a start node, and if the node does not have a destination, it is a sink node. If a dependency between two nodes; the parent node, $n_i$ and the child node, $n_j$, exists, it is connected with an edge $e_{i,j}$. The node, $n_j$, cannot execute before it has obtained data from its parent(s).

Using definition 1, we can now express the critical path of algorithm using the following definition:

**Definition 2:** The critical path, $CP$, is a set of nodes $n_{start}, ..., n_i, ..., n_{sink}$ and associated edges $e_{start,h}, ..., e_{i,j}, ..., e_{k,sink}$ forming a path, $p$, from a start node, $n_{start}$, to a sink node, $n_{sink}$, for which the sum of costs are a maximum:

$$ CP = \max \text{cost}(\{n_{start}, e_{start,h}, n_h, ..., n_i, e_{i,j}, n_j, ..., n_k, e_{k,sink}, n_{sink}\}) \tag{1} $$

A way to measure the inherent parallelism that uses the critical path is the $\gamma$ metric developed in our previous work [9] which is defined as:

$$ \gamma = \frac{N}{CP} \tag{2} $$

where we consider the nodes to be atomic, meaning that $N$ represents the total number of operations in the precedence graph.

The metric described in (2) expresses the level of inherent parallelism of the algorithm by calculating the ratio between the number of operations in the algorithm, and the number of operations in the critical path. In this case, where we consider all nodes as basic operations, $N$ is equivalent with the total number of nodes $N$. This metric is organised such that with no inherent parallelism its gives the value 1. The metric value increases along with the inherent parallelism.

The affinity metric [7] on the other hand is in comparison a normalised measure, where zero indicates the worst match and one indicates a perfect match between the algorithm and the architectural property. Using the $\gamma$ for expressing the inherent parallelism will lead to non-comparable results. A metric expressing the parallelism together with the affinity metric should have the same normalised properties. To suit these properties we can rewrite the $\gamma$ metric into a normalised metric:

$$ \gamma' = 1 - \frac{CP}{N} \tag{3} $$

The affinity metric is based on textural analysis of the source code and therefore does not refer to the number of operations, critical path or any of the terms used above for $\gamma$ and $\gamma'$. Instead it operates with source lines which contain certain patterns.

In order to cope with the parallelism measure inside this source line based framework, we propose a new metric, $\theta$, inspired by the $\gamma'$ metric. $\theta$ is defined as:

$$ \theta = 1 - \frac{S_{CP}}{S_m} \tag{4} $$

where $S_{CP}$ is the number of source lines included in the critical path and $S_m$ is the total number of source lines in the code. To emphasise the weight of the critical path, a loop unrolling is need to be performed before measuring $S_m$ and $S_{CP}$ of the $\theta$ metric.

This way of expressing the parallelism is not equivalent with $\gamma'$ since every source line in a high level language will usually lead to more than one atomic operation. The danger is that the number of atomic operations highly depends on the programmers coding style. A compact code will result in
more operations per source line than a fragmented code with many intermediately/temporary variables which come close to one operation per code line. It is therefore impossible to obtain the same precision, as the modified and normalised $\gamma'$ metric.

To examine their differences, extreme cases, i.e. a purely sequential and a fully parallel execution as well as two random cases have been considered. The two random execution graphs are shown in Fig 1 and Fig 2. Comparing the $\gamma'$ metric and the $\theta$ metric on these cases provides us with the results shown in the four first lines of table II. We here consider $N = 40$ in the precedence graphs, where a source line on average corresponds to four nodes. The sequential execution gives, as expected, the same result for both metrics i.e., 0. The fully parallel execution however, gives a slightly different result for the two metrics, $\gamma' = 0.975$ and $\theta = 0.9$. None of them reach the value 1 for a full parallel execution, because of the way $CP$ is defined. But we notice that $\theta$ gives a lower score than the $\gamma'$ metric. This is due to the smaller number of code lines compared with the number of nodes, which influences the ratio. For the random case there are larger differences ($0.65$ vs. $0.56$) and ($0.7$ vs. $0.75$).

![Fig. 1. Precedence graph of random 1 algorithm.](image1)

![Fig. 2. Precedence graph of random 2 algorithm.](image2)

![Fig. 3. Matrix multiplication example.](image3)

Even though the $\theta$ metric and the $\gamma'$ metric do not give similar results, $\theta$ still gives a good indication of the algorithms affinity to a parallel architecture. Let us discuss this issue by re-considering the matrix multiplication case given by:

$$C = AB \quad (5)$$

where $C \in \mathbb{R}^{X \times Z}$, $A \in \mathbb{R}^{X \times Y}$, $B \in \mathbb{R}^{Y \times Z}$ are matrixes where $X, Y, Z$ denotes the dimensions. Here the dimensions are $X = Y = Z = 10$. The c-code taken from the DSPstone project [11] is shown in Fig 3, and we see that the kernel of the algorithm consists of multiplications, memory reads and writes together with some indexing controls. A precedence graph of the kernel of the algorithm is shown in Fig 4. The results of the examination of the algorithm with the two metrics are also shown in table II. From this we see that there is an insignificant difference between the two metrics (i.e., 0.999 and 0.989), which is due to the high number of nodes and unrolled source lines. From these cases it appears that the newly proposed metric $\theta$ serves its purpose of indicating parallelism.
A. Reed-Solomon Decoder

Reed-Solomon codes are a forward error correction codes used in many modern communication systems. The decoder is able to detect and correct some bit errors which have occurred during the transmission. It is an algorithm which involves many conditional branches in order to detect and repair errors.

The algorithm has been examined with the affinity metric, and the results are shown in table III. The table shows the original affinity metric values for GPP, DSP and FPGA architectures and the affinity metric for FPGA with our new extension (added as an extra parameter for FPGA metric before normalisation as in [7]). We see that the Reed-Solomon decoder has the highest score (0.795) on a DSP architecture with the original affinity metric, however, the score for FPGA architecture increases significantly (from 0.205 to 0.806) when including our extension, and thereby gets the highest score. To verify the results, the algorithm has been implemented on a Analog Devices TigerSHARK ADSP-TS201 DSP and a Xilinx Virtex II FPGA, in high-level languages (C and Handel-C, respectively). The latency for decoding one block was measured on both platforms. The FPGA implementation was done in two steps: first, a version without exploiting the parallelism, which corresponds to the original affinity metric interpretation, and second, a version exploiting the inherent parallelism. These latencies are also shown in table III.

Inspecting the results shows that the best performance is obtained by the parallelised FPGA implementation, with a latency of 244µs. We can then deduce that using the original affinity value for FPGA in this case will not disclose the architectures potential for the Reed-Solomon algorithm. Without considering the parallelism, the designer would make an inefficient partitioning choice.

Using the extended metric that we propose gives a better indication of the affinity between algorithm and FPGA architecture, thus helps the designer to make wiser partitioning decisions.

V. Conclusion

In this paper we have proposed an extension of the affinity metric [6], in order to improve the capability to measure the algorithm-architecture affinity for FPGA. The extension consists of a new metric derived from some of our previous work [9]. This new metric provides a mean for measuring the inherent parallelism of the algorithm inside the source code. We have shown that adding this new metric to the original affinity metric improves its score for FPGA matching.

References

Abstract—In the automotive industry there is a continuing trend to distributed applications. The reasons are that local systems are coordinated to perform a global vehicle function and that some algorithms are becoming more complex and must be distributed on several electronic control units, which must be coordinated. The required coordination in the time-domain can partially be achieved by synchronization. A well suited basis for synchronization is a Time-Triggered Architecture, which includes time-triggered communication and task activation. In order to achieve a fault-tolerant communication with redundant communication channels and nodes in the network, which are transparent for the application a fault tolerance layer is necessary. These three components of a Time-Triggered Architecture, i.e. time-triggered operating system, time-triggered communication and the fault tolerance layer, have an influence on the behaviour of the distributed application. This could result in non-satisfied requirements, which were fulfilled previously by the non – distributed application. These influences are presented in this paper. In order guarantee the detection of the problems during distribution these influences are modelled with a formal language such that formal verification of the distributed application is possible. The modelling takes the complexity of the verification problem into account and contains abstraction levels and further optimizations.

Index Terms—Architecture, distributed algorithms, software verification and validation, formal language, network fault tolerance, network operating system

I. INTRODUCTION

The trend to vehicle electronics has started for round about 40 years and can be split in several phases.

In the first phase, which is called substitution phase, the mechanical control was replaced by electronics. This started at 1967 with the introduction of electronic fuel injection.

In the adaptation phase, which is the second phase of the vehicle electronics, electronic systems were adapted to the vehicles in order to realise new functions, which could not be realised with mechanical components, e. g. Lambda control, idle-running speed control. With the introduction of micro controllers this trend increases and a lot of new vehicle functions were developed, which could not be realised with mechanical components.

The third level of vehicle electronics is called integration phase, where the single functions of the electronic components are combined to perform an overall vehicle function. Examples are global chassis control and global power train management. For the global chassis control single components like ESP, dumping control and active steering can be combined to increase driving dynamics. This requires that the single electronic control units are performed in a coordinated manner. This can be achieved partially by synchronization.

In the meantime the transition to the next phase of vehicle electronics takes place, which is called innovation phase. During this phase inventions from other domains like railway and aerospace industry are more and more innovative and payable solutions for vehicles. One interesting development in the aerospace domain is the so called X by Wire technology. With steer by wire and brake by wire systems the active safety of a vehicle can be increased. This implies, that the environment of the car must be observed. Dangerous situations must be recognised and the driver should be supported at his driving manoeuvre. Due to the fast change of the vehicle environment as a result of a possible high speed and complex objects of the environment a high amount of data has to be processed. Additional there are high demands on response times, where the system has to react on a certain driving situation. This leads to the problem, that the high amount of data can not be processed with micro controllers, which are in use today. One approach is to use a distributed system in order to meet the performance requirements of the application.

In [1] a distributed system is defined as "[...] a collection of autonomous computers linked by a network, with software designed to produce an integrated computing facility".

This definition clearly refers to three components of a distributed system:
- autonomous computers
- communication network
- software, in order to coordinate the nodes for the commonly performed task
These three components are considered in this paper. In order to achieve synchronisation a Time-Triggered Architecture is used as basis of the distribution.

In the following the Time-Triggered Architecture is briefly introduced. Based on this technology an application can be distributed on different nodes. The influences of this distribution are described in chapter 3. For further analysis of these influences a formal model of the corresponding components is explained in chapter 4 including a short description of the formal language. This paper ends with a conclusion and gives an outlook on future activities in this area.

II. TIME-TRIGGERED ARCHITECTURE

In this work a Time-Triggered Architecture similar to [2] is considered. The base of a Time-Triggered Architecture is a time-triggered communication. This means, that each message can only be sent at a certain point of time. This is realized with a corresponding communication protocol within a communication controller of a node in the network and the communication schedule. The required fault tolerance can be achieved by redundant communication channels and notes.

In order to achieve transparency, the application should access a standardised interface, which is equal in all nodes and stable for several years. The middleware, which is considered in this work, is derived from the OSEKtime standard. The FTCOM layer contains management services for the handling of redundant messages. The applications need not know, how many times a message was transmitted redundantly.

According to [3] the time-triggered activation of tasks also belongs to a Time-Triggered Architecture. With it the fulfillment of hard deadlines can be proven easily. Such a real time operating system can be synchronised with the communication by the FTCOM layer. Due to this synchronisation of the operating system with the communication, the required synchrony of the application tasks on the different nodes can be achieved.

As a summary the components of a TTA node are shown in the following figure signed in hatched.

![Fig. 1. Components of a TTA node (hatched)](image)

The Time-Triggered Architecture with its components fulfils the requirements for synchrony, fault tolerance and transparency.

III. INFLUENCE OF DISTRIBUTION

The distribution of software partitions has influences on the overall functionality of the application. These are explained within the following subsections.

A. Problem of execution order

If tasks are executed on a single electronic control unit, they are performed one after the other. This is also true for quasi parallel programs, because there is in fact only one micro controller.

After the distribution on several electronic control units, it is possible to execute the tasks also physically in parallel. The affect on the functional level can be explained with a simple example.

Two tasks are executed simultaneously on different processors. Their overall functionality consists of adding its previously calculated value with the value of the other task. Formally this can be described in the following way.

\[ T_{1,n} = T_{1,n-1} + T_{2,n-1} \quad \text{with } T_{1,0} = 1 \quad \text{(Initial value)} \quad n \geq 0 \]  

\[ T_{2,n} = T_{2,n-1} + T_{1,n-1} \quad \text{with } T_{2,0} = 1 \quad \text{(Initial value)} \quad n \geq 0 \]

By applying the commutative law one could assume, that \( T_1 \) and \( T_2 \) always calculate the same results. For simultaneous execution this is given as shown on the following figure.

![Fig. 2 Results of parallel execution of Task 1 and Task 2](image)

The execution generates an order of numbers, which represents the function \( f(x) = 2^x \). If both tasks are executed one after the other, the following order of numbers is generated.

![Fig. 3 Results of serial execution of Task 1 and Task 2](image)

This order of numbers represents the Fibonacci numbers. With this simple example it is proven, that the distribution of software partitions affects the overall behaviour of the application.
B. Communication delay

Communication delays, which are of interest with regard to the behaviour of the application, are the communication delay during normal operation and the start-up – phase.

Delay during start-up phase

Typically in the automotive industry different electronic control units are delivered from different suppliers. They are using their individual hardware platforms and different standard software components, which provide services for the start-up. These differences between electronic control units can lead to different start-up times in the network.

Assuming there are three nodes in the network – node A, B and C – and node C performs the functionality:

\[
\text{Output (C)} = \text{Output(A)} + \text{Output(B)} \quad (3)
\]

Node A and B are counting different events, where only one event can occur at the same time. The number of events is output by each node.

\[
\text{Output (A)} = \# \text{ event A} \quad (4)
\]

\[
\text{Output (B)} = \# \text{ event B} \quad (5)
\]

Both counters are initialised with zero.

The nodes A and B are started in different orders. The results of simulation are shown on the following tables.

**TABLE I**

<table>
<thead>
<tr>
<th>Start-Up Scenario A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output (A)</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

**TABLE II**

<table>
<thead>
<tr>
<th>Start-Up Scenario B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output (A)</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Assuming the functional requirement would state"between two successive execution-steps the output value of node C must not differ more than 1". If node A and B are started at the same time, i.e. begin to send messages, this property can be violated. In a sequential start-up order like shown in the second table the property is hold.

Delay during normal operation

After the start-up phase the normal operation begins corresponding to the pre-defined communication schedule. This can influence the behaviour of the application such that control problems can not be solved any longer.

To describe this kind of problem Stavros Tripakis has introduced different classes of control problems in [4]. This includes the following classes:

- **CC**, set of all control problems, which can be solved with a single controller
- **DC**, set of all control problems, which can be solved with two controller without communication
- **DCUC**, set of all control problems, which can be solved with two controllers and communication with unbounded delay
- **DCC_k**, set of all control problems, which can be solved with two controllers and communication with k – bounded delay

In his technical report Stavros Tripakis prove the following relation:

\[
CC \supset DCC_0 \supset DCC_1 \supset DCC_2 \supset \ldots \supset DCUC \supset DC \quad (4)
\]

With \( \supset \) the real subset in meant. It can be recognized, that there are problems, which can be solved with a k – bounded communication delay but not with a k+1 bounded one. This can be found out with a formal analysis.

C. Influence of transparency

In order to support the required transparency, the considered middleware provides two mechanisms. These realise the usual access to a standardised interface (API, Application Programming Interface) and mechanisms to handle fault tolerance. By this standardised interface certain information about the status of the communication are provided. By a formal analysis it can be checked if the functional requirements are also fulfilled, if a communication fault is present.

For systems, which cannot support the vehicle in reaching a safe state and therefore must have a fail operational behaviour, sufficient redundancies must be provided. This can be supported by redundant message transfer. As consequence, the redundant messages must be merged at the receiver side. This is done with a corresponding agreement algorithm of the considered FTCOM layer. The degree of redundancy is transparent for the application, i.e. the application need not to know, how many times a message is transmitted.

The functionalities of the considered FTCOM layer have an influence to the behaviour of the application. In the time domain, differences for the behaviour of the application appear, because the FTCOM tasks can be activated at different times. But this mainly depends on the task schedule of the operating system. In the value domain there can be differences between the sent signal value and the received one. Besides this the selected agreement algorithms influences also the availability of the received signals.

A safety relevant application should fulfil the formal requirements even if a single fault is present in the system. In order to prove this by formal verification, modelling of the relevant part of the considered FTCOM layer is necessary.
IV. MODELING

In the following modelling of the components of a TTA node is described. This contains modelling elements, communication (delay during start-up and normal operation), fault tolerance layer and operating system. The modelling is done with the formal language SCADE which is well known in the aerospace industry [5]. The main reason for using SCADE is that the synchronous approach of this language together with the synchronous approach of a TTA reduces the complexity of the verification problem due to the minimal size of the Cartesian product [6] of the different automata. A further benefit of this language compared to others is, that SCADE enables modelling the control- and data flow of the application as well as the behaviour of the TTA components. Thus a formal verification of the distributed application can be performed.

Formal verification checks, if a formally described algorithm fulfils its corresponding requirements. Compared to other verification methods like simulation or testing formal verification guarantees completeness with regard to the state space. In the context of this work formal verification of the distributed application should enable to detect the problems, which can arise during the distribution of an applications or to prove their absence completely.

In contrast to other work in this area like [7] and [8] where the formal verification of the clock synchronisation and interaction of clock synchronisation and group membership is described, this work aims at verifying the property of the distributed application.

A. Modelling elements

SCADE provides a logical time semantic, which abstracts from the physical time. Every action is performed synchronously at a certain time instant in zero time. To use the logical time concept of SCADE for modelling the components of a TTA node two operators are revealed as useful.

The FBY (=Followed By) Operator generates a time delay for a defined number of instances of the logical time scale. The semantic is defined as followed:

\[
\text{FBY}(E,n,\text{Init})_{1..n} = \text{Init}
\]

\[
\text{FBY}(E,n,\text{Init})_{n+1} = E_{n}
\]

Fig. 4 Symbol and semantic of the FBY operator

The conduct (=conditional activation) operator in SCADE can be used to model time-triggered activations where the time is related to the logical time scale. The following example shows how the conduct operator is executed.

Within the subsystems Init_msg_A_CHA und Init_msg_A_CHB all signals of message msg_A_CHA and msg_A_CHB are initialised with a defined value. The subsystem startup_A generates the Boolean value false for the duration of the start-up phase. After this time there is a transition to the normal operating mode, i.e. the output

These two operators are mainly used in the following to model the time behaviour of the TTA components.

B. Modelling the communication

For modelling the communication the delays during start-up and normal operation have to be described in a formal way.

The delays which occur during normal operation are known, because they can be derived from the communication schedule, which is defined at the design time. For the start-up phase the delay, which is needed by a node in order to integrate in the network, can vary. Nevertheless there are upper bounds for the start-up time of a node, which are given by the automotive industry.

With the FBY and conduct operator both kinds of delays, i.e. delay during start-up and during normal operation can be formally described. The fundamental behaviour of a time-triggered communication is shown on the following figure.
changes the value to \textit{true}. This behaviour is modelled with the \textit{FBY} operator in the following way.

\begin{equation}
\text{FBY(} \text{true, Startup\_Time, false}\text{)}(1\ldots\text{Startup\_Time}) = \text{false} \\
\text{FBY(} \text{true, Startup\_Time, false}\text{)}(\text{m}>\text{Startup\_Time}) = \text{true}
\end{equation}

After this transition sending according to the global communication schedule is enabled for the node by an AND connection. The messages of this host are communicated at the time slots, which are generated from the subsystem \textit{Comm\_Schedule}, i.e. messages are passed for example from input \textit{msg\_A\_CHA} to the output \textit{msg\_A\_del\_CHA} of subsystem \textit{Comm\_delay\_msg\_A\_CHA}.

The messages are passed to all other nodes, what is denoted as broadcast communication.

If no value is transmitted on the channel, the old value is remained and all signal status of the corresponding message are set to the value \textit{TT\_E\_FTCOM\_MSG\_NOT\_RECEIVED}. This corresponds to the behaviour of common communication protocols, where the old value is stored as long as a new valid message is received. The only difference is that due to efficiency reasons the status value of the FTCOM layer is directly used for the status of the communication, such that no projection of a communication fault to a FTCOM status needs to be applied.

Modelling of time must be done with natural number similar like in other synchronous languages [9]. These must be derived from the ratio of the time to be modelled and the time of the base clock.

For the purpose of modelling the communication delay it must be defined, what kind of granularity is used for the time. For the considered time-triggered access scheme this can be both a TDMA round and a single TDMA slot. The latter one is more exact, but increases the complexity of the verification problem.

\subsection*{C. Fault tolerance layer}

The objective of the fault tolerance layer is the handling of redundantly transmitted messages transparently for the application. The application does not know, how many times a signal is transmitted.

For redundant messages it can be distinguished between redundancy in the value- and time domain.

Redundancy in the value domain means the transmission of signals on both channels. The transmission of messages more often than required to meet timing constraints means redundancy in the time domain.

In order to tolerate a fault of a channel or a node of the distributed system, at most two messages in the time domain and two messages in the value domain must be handled by the fault tolerance layer.

\subsubsection*{Sender side}

For sending redundant signals these are copied corresponding to the degree of redundancy, before they are packed to messages in the subsequent interaction layer. Due to simplicity the modelling of the copying function is not shown here.

A signal can also be sent redundantly in the time domain. This need not to be regarded at the configuration of the fault tolerance layer, but can be realised with the communication schedule.

\subsubsection*{Receiver side}

The principal structure of the fault tolerance layer at the receiver side is shown on the following figure.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig7.png}
\caption{Structure of the fault tolerance layer at the receiver side.}
\end{figure}

All redundant incoming signals including their status information are evaluated and merged with an agreement algorithm. In order to be regarded by the agreement algorithm a signal must have the status value \textit{TT\_E\_FTCOM\_OK}. As a result of the agreement the status \textit{TT\_E\_FTCOM\_RDA\_FAILED} is output, if the agreement could not be performed successfully or the status \textit{TT\_E\_FTCOM\_OK} is output, if the agreement could be performed successfully. In the latter case the signal value passed to the application is also valid. The value of the agreement status is visible at the output \textit{RDA\_status}.

For the agreement in the value domain all signal values are available at the same time. The agreement algorithm can be performed immediately.

The agreement in the time domain can be modelled in a similar way with the selected modelling language, because the used \textit{condact} operator keeps it output value as long as it is activated again. This corresponds to the behaviour of an autonomous communication controller, which stores the received message in the CNI interface as long as a new valid message is received. With the used \textit{FBY} operator, which can output the previously received message (redundant in the time domain), it can be achieved that all messages can be evaluated at the same time.
At modelling the agreement algorithms the problem arises, that there are no data types in SCADE to model an array. Since the number of redundantly received messages for a time-triggered communication is exactly known, also other data types like the base data type of the received signals can be used. As a consequence for each number of inputs of redundantly transmitted messages a separate model must be created. With the requirement to tolerate the fault of a channel or a node, up to four models must be created. 

The following agreement algorithms are modelled in the subsequent sections of this paper

- Majority Voting
- Average

**Majority Voting**

At majority voting the majority of the received redundant signals must be equal. Therefore for a number of

- $n=1$ inputs, there is no majority decision possible
- $n=2$ inputs, there is no majority decision possible
- $n \geq 3$ inputs, there is a majority decision possible

A possible implementation of majority voting is proposed in the OSEKtime standard with Pseudo Code [10].

**Declaration:**

```
int counter; int values[];
```

**Initialisation:**

```
counter = POSITION_ONE;
```

**Next Value:**

```
values[counter]=value;
counter = counter+1;
```

**Finish Computation:**

```
if counter > POSITION_ONE:
  operating on values [POSITION_ONE .. counter-1] do:
    sort values;
    find largest group of identical values;
    find second largest group of identical values;
    if size of largest group > size of second largest group
      or there is only one group of values:
        result = value of largest group;
        RDA status is VALID
      else:
        result = NO_RESULT;
        RDA status is INVALID
    else:
      RDA status is INVALID
```

This algorithm does not take into account diversity of software and hardware components. This would result in the problem that exact equality is often not fulfilled. Therefore one has to define domains of tolerance, where received signals are interpreted as equal. The corresponding model is shown on the following figure.

![Majority Voting Algorithm for 3 input signals.](image)

**Average**

Also for this algorithm there is given an example in the OSEKtime standard. The corresponding algorithm represented in Pseudo Code has the following functionality.

**Declaration:**

```
int counter; int sum;
```

**Initialisation:**

```
counter = 0;
sum = 0;
```

**Next Value:**

```
counter = counter+1;
```

First the algorithm checks, if signal $b$ is close to $a$. If this condition is fulfilled, the value of $a$ is passed to the output. Else it is checked, if $c$ is close to $a$ and if so the value $a$ is passed to the output. Otherwise it is still checked, if $c$ is close to $b$. In this case the value of $b$ is passed. If no of these conditions is fulfilled, the value NO_RESULT would be output. If these conditions is fulfilled, the value NO_RESULT is output and the signal status is set to the value TT_E_FTCOM_RDA_FAILED. Otherwise it is set to TT_E_FTCOM_OK.

This algorithm can be easily extended to four input signals but this would be too complex to be presented in this paper. Majority voting for four signals checks, if three of the four received input signals are close to each other. If this is not given, the value NO_RESULT would be output. In general the presented majority voting checks, if from $n$ received messages

- $n \mod 2 = 0$, or
- $n \mod 2 = 1$.

With this general rule the algorithm can be arbitrarily extended.

**Average**

Also for this algorithm there is given an example in the OSEKtime standard. The corresponding algorithm represented in Pseudo Code has the following functionality.

**Declaration:**

```
int counter; int sum;
```

**Initialisation:**

```
counter = 0;
sum = 0;
```

**Next Value:**

```
counter = counter+1;
```
sum = sum + value;

**Finish Computation:**

if counter > 0:
    result = sum / counter;
    RDA status is VALID
else:
    RDA status is INVALID

Apart from the missing end criteria in the program section Next Value, this algorithm can also be modelled in SCADE.

Prior to the average calculation it must be determined, how many valid signals are received. If there are only three valid signals, an average calculation with four values would deliver wrong results.

If a signal has the status TT_E_FTCOM_OK, its value is taken into account for the average calculation and also the number of the signals to be regarded is incremented accordingly. Differently to the OSEKtime standard, a division is also performed if no valid message is received. In this case there no division by the number of received message, because this would result in a division by zero. Instead there is a division with a default value, which can never occur in reality. Furthermore the signal status is set to TT_E_FTCOM_RDA_FAILED.

The algorithm shown above calculates the average value about two signals. The dots should show where the algorithm has to be extended, if there are more input signals to be taken into account.

**D. Modelling a time-triggered operating system**

The general structure of an operating system, which performs time-triggered task activation, is shown on the following figure as a model.

The task schedule, which is at the top of the figure, activates the tasks according to the pre-defined task schedule. The task of the FTCOM layer and the application are activated with a `condact` operator.

For sending and receiving of messages different FTCOM tasks are used in this work. In order to activate the tasks the scheduler is connected with the tasks by a Boolean signal. For the activation the scheduler outputs the Boolean value `true` and with it the `condact` operator or task respectively is activated. The scheduler must not activate more than one task at the same time.

**Modelling the scheduler**

The content of the scheduler is shown on the following figure.

The subsystem `one_to_n_counter` is a counter, which counts from one to `Number_of_OS_Slots` and then starts again with one. The output of the subsystem `one_to_n_counter` is connected with the `compare_to` subsystems. These check, if the value of the counter equals to the value of the constant, which is connected to the input, e.g. `Trigger1_value`. In this case the value `true` is output and `false` otherwise. The outputs of the `compare_to` subsystems are connected to the output interface of the scheduler and trigger the corresponding tasks.
Modelling the task schedule

The outputs of the scheduler are activated one after the other, i.e. gets the Boolean value true. According to this sequence the tasks must be allocated. For this purpose every trigger output of the scheduler is connected to the activation input of the corresponding conduct operator or task respectively. The position of a task within the schedule table corresponds to the position of the trigger output of the scheduler, i.e. the output at the top of the scheduler is activated first and gets the value true and with it the connected conduct operator is activated first.

The connection between modelled scheduler and schedule table of the operating system is shown on the following figure.

![Connection between scheduler and schedule table.](image)

Since the conduct operator in SCADE has only one input, multiple activations of a task within a schedule round has to be modelled within the subsystem of the scheduler. For this purpose two or more trigger outputs of the scheduler have to be linked with a logical OR operator.

Idle tasks can be modelled with the termination of the corresponding scheduler output.

In this work it is assumed that the task schedule is generated from another tool and is transferred to the model.

V. CONCLUSION AND OUTLOOK

We have presented an approach, which can be used to formally verify a distributed application in order to detect the problems, which can arise during the distribution of an applications or to prove their absence completely. As a base of distribution a Time-Triggered Architecture was used, which fulfil the requirements for synchrony, transparency and fault tolerance. The influence of the components of a TTA node to the behaviour of the application was analysed. In order to describe them a formal description language was selected and the modelling was performed.

The presented approach was applied to an automotive application, which generates the feedback at the steering wheel of a steer by wire system. This includes formal verification of static and dynamic properties in the value domain.

In order to perform a more accurate analysis with formal verification, a more detailed modelling is necessary. This could include the interruption of tasks by interrupts or other tasks.

Furthermore there will be a mixing of different networks, which are connected by gateways. A typical variant is the connection of a CAN network with the FlexRay communication with a gateway. This is of special interest, because it represents a mixture of a synchronous and asynchronous communication within a system. This can also be given within a single electronic control unit, if for example an event driven operating system is executed on a node, which communicates via a time-triggered bus.

The approach presented in this paper, could be extended in the depth as well as in the breadth of application. Thereby the complexity of the verification problem should be considered very well. This could result in appropriate abstraction techniques as well as more efficient tools for verification. This is seen as one key criterion for the acceptance of formal methods in the automotive industry.

REFERENCES

Wake-Up Interval Optimization for Sensor Networks with Rendez-vous Schemes

Olivier Sentieys*, Olivier Berder*, Philippe Quémerais*, Mickaël Cartron†

*IRISA / University of Rennes 1
6 Rue de Kerampont
BP 80518
22305 Lannion Cedex, France
Email : Firstname.Lastname@irisa.fr

†CEA / LIST
Saclay
91191 Gif-sur-Yvette France
Email : mickael.cartron@cea.fr

Abstract—The aim of this paper is to maximize the lifetime of a sensor network by correctly configuring the wake-up interval of a pseudo-asynchronous wake-up media access control technique. The evaluation of the overall energy consumption is done by a realistic simulation based on a global framework simulating simple nodes with a microcontroller and a communication system dedicated to sensor networks. The processing of this simulation program includes the MAC and physical layers, the link layer, the network and the application layers, allowing a good estimation of cross-layer interactions. Indeed, the wake-up interval that is the main concern of the paper highly depends on the application paradigm.

I. INTRODUCTION

Media access control (MAC) layer design is a very active domain of research on sensor networks since the realization of energy-efficient designs with high lifetimes still requires innovation. The performance of MAC protocols is difficult to evaluate because it is tightly linked to the application. Developers still face problems to get realistic energy consumption evaluation. Some works proposed new methods [1], [2] to give accurate consumption models based on microprocessor platforms running the widely used tinyOS event-driven operating system. These tools capture the detailed energy requirements of the CPU, radio, and every other peripheral. A framework has been proposed in [3] for cross-layer design of the link layer components. However, the paper does not particularly target “real world systems” and the power consumption estimation is highly theoretical, contrary to [1] and [2].

Contrary to other kinds of wireless networks, for example high throughput networks, too simplistic application models lead to very bad energy estimation. However, the impact of application paradigms is rarely precisely considered, causing wrong evaluation, since it is now widely known that cross-layer optimization is necessary in order to get the highest lifetime sensor networks [4]. The analysis of the consumption in a typical non optimal functioning mode (figure 1) shows that the communication component is the main cause of energy consumption and that improvement of the communication part offers most opportunities for energy savings.

Fig. 1. Consumption repartition for non optimal wake-up interval tuning. Total average power per node : 0.36 mW.

For that reason, new kinds of methods and tools for cross-layer design are required for maximizing the energy efficiency and the lifetime of these wireless networks. The aim of this paper is to define optimal low-level platform parameters for a given sensor network application, from the energy per successfully transmitted useful bit point of view. This work focuses on a pseudo-asynchronous MAC layer, that allows turning off the reception. The communication occurs when the transmitter and the receiver synchronize for what is called...
a rendez-vous. This rendez-vous is initiated by the node that plays the role of the receiver for the communication, and this occurs periodically according to a wake-up interval.

In section II, the performance models used are described, from the channel to the link layer. Then, the power consumption estimation is developed in section III. The applications tested in this work are presented in section IV and some results concerning the tuning of the wake-up interval are given in section V.

II. PERFORMANCE MODELLING

A. Application model

The network is constituted of \( N \) nodes, in a particular geographic configuration. The distance between a source node \( i \) and a destination node \( j \) is noted \( D_{i,j} \). What is often difficult to evaluate and which is however crucial is the typical volume of data transmitted for a particular application. Our sensor network prototype is able to run two kinds of situations. First, for a limited quantity of nodes, it is possible to realize real condition experiments on our physical platforms. However, it is not possible to realize large networks of hundreds or thousands of nodes, so a simulator for the prototype allows to extend the study to larger scale networks.

B. Channel

The channel model used is a gaussian additive noise. Considering that the signal power before the transmission is \( P_{\text{level}} \) and the gain of the transmitter antenna is \( G_t \), the radiated power is \( P_{\text{rad}}(\text{dB}) = P_{\text{level}}(\text{dB}) - G_t(\text{dBi}) \). At a distance of 1 meter from the antenna, the signal power can be modelled by \( P_{1m} = P_{\text{rad}} \cdot (\lambda/2\pi)^2 \), with \( \lambda \) the carrier wavelength. For a distance \( d \) between the transmitter and the receiver, the power loss lowers the signal strength which can be expressed by \( P_d = P_{1m}/\alpha \), with \( \alpha \) the attenuation coefficient [5]. At the receiver, the signal observed is then weakened by the receiver antenna. The signal-to-noise ratio (SNR) can be deduced by \( \text{SNR}(\text{dB}) = P_d(\text{dBm}) - G_r(\text{dBi}) - P_n(\text{dBm}) \), with \( G_r \) the gain of the receiver antenna and \( P_n \) the noise power. In order to minimize the energy consumption, one should note that \( P_{\text{level}} \) is chosen with respect to the distance between 2 nodes. On Fig. 2, the minimum energy consumed for a transmission between 2 nodes is represented by the bold continuous line.

C. Physical layer model

The physical layer is modelled by a BPSK modulation system which gives an expression of the BER (binary error rate) that can be observed in the bitstream. This expression is \( \text{BER} = \frac{1}{2} \cdot \text{erfc}(\sqrt{\text{SNR}}) \).

D. Link layer model

The error control strategy is applied at the link layer and different solutions have been evaluated, all based on block coding. The quality criterion that is of main interest is the tolerance to binary errors. A detection-only strategy will not tolerate any binary error, while a correction strategy associated with a simple checksum for detection will tolerate one error. Considering that the packet length is \( pl \) and if the tolerance is noted \( tol \), then the retransmission rate for a communication from node \( i \) to node \( j \) for a tolerance \( tol \) is

\[
R_{i,j}(tol) = 1 - \left( \sum_{k=0}^{tol} \frac{C^k_{pl} \cdot \text{BER}^k (1 - \text{BER})^{pl-k}}{k!} \right),
\]

where the binomial coefficient \( C^k_{pl} = \binom{pl}{k} = \frac{pl!}{k!(pl-k)!} \).

E. Collision probability model

In order to evaluate the collision probability, the MAC level has to be accurately described. Our MAC and physical schemes are highly inspired by the RICER (Receiver Initiated CyclEd Receivers) version of the MAPL Technique (MAC And Physical LAyer Power) [6]. This MAC/Physical layer is a pseudo-asynchronous technique (also called cycled receiver) that realizes rendez-vous between wireless nodes, that means that nodes establish rendez-vous on demand. Despite the ability to work asynchronously, the software initiates the rendez-vous at approximately constant periods, that is why this method is considered as pseudo-asynchronous. This MAPLAP/RICER MAC scheme can be used with very simple communication hardware. The media access timing for a successful rendez-vous is showed on figure 3, and the unsuccessful wake up is illustrated on figure 4.

In this scheme, two kinds of collision are possible. The first one, the wake up collision, occurs during the first phase of the communication when a node wakes up before the end of the
The probability that \( k \) nodes want to respond to the waking up node \( j \) is then

\[
p_{j,k} = \exp(-\mu_{j}) \frac{\mu_{j}^k}{k!}.
\]  

(7)

In the end, the probability for a node \( i \) to collide while answering to the waking up node \( j \) is

\[
p_{\text{col2}}(j) = \sum_{k=2}^{\infty} p_{j,k} \frac{k-1}{k}.
\]  

(8)

### III. ENERGY CONSUMPTION MODELING

The total power consumption \( E_{\text{wu}} \) is the sum of 3 terms, \( E_{\text{algo}} \), \( E_{\text{an}} \), and \( E_{\text{amp}} \). \( E_{\text{algo}} \) is the energy consumed by the microcontroller for the link and MAC layers control, \( E_{\text{an}} \) is the part of the energy consumed by the shaping of the signal, and \( E_{\text{amp}} \) is the additional part that depends on the output power.

Our sensor network software is based on the embedded system Contiki, and more precisely on the Protothread library [7] which allows to realize event-driven systems. It has been showed that asynchronous processing, typical of sensor networks applications, perfectly fits event-driven programming. Another famous event-driven system is TinyOS, which is widely used on the sensor networks community, but the Protothread library has been chosen instead because of its greater flexibility, the compactness of code and the ease of development in C language. The main kernel of the program is an infinite loop that sequentially gives the processor to each process. After each loop of the main kernel, it is possible to record an execution trace which depends on external events and which can be identified and classified. The approach used for the power estimation is accurate, because it is based on the analysis of real code and includes the consumption of the communication component and the microcontroller. It is also simple, because after having identified the consumption of each software task of each typical execution trace, the total power only depends on the number of these typical traces.

The evaluation of the term \( E_{\text{algo}} \) is done by measuring the processing time of the different tasks of the loop traces. The terms \( E_{\text{an}} \) and \( E_{\text{amp}} \) are calculated with the time spent by the communication component in the receive mode or transmit mode, which depends on the transmission power \( P_{\text{level}} \). With \( P_{\text{t}} \) the power consumption of the communication component in receive mode, \( P_{\text{t}}(P_{\text{level}}) \) and \( P_{\text{t}}(\text{min}) \) respectively the power consumptions of the communication component in transmit mode for an output power of \( P_{\text{level}} \) and for a minimal power transmission, \( T_{\text{t}} \) the time spent in receive mode and \( T_{\text{t}} \) the time spent in transmit mode, \( E_{\text{an}} \) and \( E_{\text{amp}} \) are

\[
E_{\text{an}} = (P_{\text{t}} + P_{\text{t}}(\text{min}))(T_{\text{r}} + T_{\text{t}})
\]  

(9)

\[
E_{\text{amp}} = (P_{\text{t}}(P_{\text{level}}) - P_{\text{t}}(\text{min}))T_{\text{t}}.
\]  

(10)
IV. Classes of Applications

The configuration that has been selected is a set of 16 nodes positioned as a square of 4 by 4, with a fixed step between neighbors. The base station is assumed to be located in a corner of the network. The application modelling the activity in the network is defined by data volume transmitted during the observation time $T_{\text{obs}}$ and it has been declined in three versions.

The first application model that has been evaluated is a random and non realistic configuration defined by a data volume matrix $V_{i,j}(\text{random})$. In this model, the communications repartition is a continuous uniform distribution.

For the second application scenario, it has been considered that the sensor network is completely static. The environment is periodically sampled and each sample is routed to the base station in a multi-hop way. The initial routing calculation is done only once at the setup. The data volume matrix modelling this behavior is $V_{i,j}(\text{static})$. A typical example of this class of application is a large temperature sensing system, where each node transmits regularly its information to a base station.

The third application scenario is used for low mobility dynamic sensor networks, as a network for livestock survey, for example. In this case, the updating of the routing information is done at a period $T_{\text{update}}$. The update mechanism is rather complex and it is constituted of 4 phases: during the first phase, the base station initiates an update order, with a packet that is flooded in the whole network, indicating the location of the base station at the same time. Each node then initiates a neighbors discovery, which is the broadcasting of a packet to every node considered as a neighbor; this is the second phase. The answers to the neighbor discovery constitute the third phase. Then, each node sends an identification packet by multi-hop routing to the base station that has the privilege of knowing every node in the network. This update mechanism uses a flooding mode during the first phase and a broadcasting mode during the third phase and these two modes intensely use the channel during short periods of time, thus increasing the probability of collision. The data volume matrix modelling this behavior is noted $V_{i,j}(\text{rdum})$.

V. Wake-up interval and Results

The influence of the wake up interval is particularly crucial for the MAC layer studied in this work. If the wake-up interval for a particular node is rather short, then the global energy consumption is high because the density of rendez-vous is high, with many wake up cycles, most of them unsuccessful. A longer wake-up time lowers the average power of the network. The choice of a very long wake-up interval should intuitively allow trading reactivity for longer lifetimes. However, the global energy increases again for long wake-up intervals. This phenomenon can be analyzed more precisely by observing the global energy repartition for all nodes. As shown by figure 5, for short wake-up intervals the main part of the energy is used by the communication component in transmit mode, since wake-up frames are transmitted at high rate. For longer wake-up intervals, the energy is mostly consumed by the receive mode of the communication component, because source nodes willing to transmit spend a long time waiting for the destination node to wake-up. In our system, this waiting is done with an active microcontroller, and in that case, the microcontroller consumption part grows proportionally to the receive mode of the communication component. Figure 6 illustrates the repartition of the consumption for every cause in the system at the optimal wake-up interval.

Figure 7 shows the impact of the tuning of the wake-up interval on the global network lifetime for the three applicative scenarios presented in section IV. The optimal tuning to get the highest lifetime is different for each application. These results prove that high level constraints have to be considered for fine tuning of low level MAC parameters. This work of accurate simulation of the power consumption of sensor networks allows to make good decisions to get that optimal tuning.
VI. CONCLUSION

A generic framework has been described including the channel, physical, MAC, link, and application layers, to have a realistic overview of the way the energy is used in the system. The MAC strategy has a particular impact on the performance of the whole system, since the consumption of the radio part is the most important, and it has been precisely modelled according to a promising pseudo-asynchronous MAPLAP RICER strategy, making it possible to use very simple hardware in a power efficient way. It has been supposed that the system is based on the Protothread library with a static scheduler. The power evaluation is done by the analysis of the execution traces of the scheduler, and the particular traces used during communications phases have been isolated, characterized, and injected into the consumption model. This method is a realistic, pragmatic and accurate way to evaluate the energy consumption of a communication process. Finally, all elements are gathered into the expression of the energy per successfully transmitted useful bit, allowing to take decision for unknown parameters. Let us point the fact that realistic network and application models are crucial, because the impact on the final energy per successfully transmitted useful bit figure is very high like illustrated by simulations results, for a fixed amount of data exchanged.

It has been proved that the fine tuning of the MAC layer has an important impact on the whole energy consumption, but this tuning must be done with the knowledge of the application characteristics to get the highest possible lifetimes.

REFERENCES


Self-Organization in Networked Reconfigurable Systems for Computer Vision Applications

Klaus Drechsler, Jan Schulte, Dominik Murr, Felix Mühlbauer and Christophe Bobda
Self-Organizing Embedded Systems Group, Department of Computer Science
Kaiserslautern University of Technology
67663 Kaiserslautern, Germany
drechsler@acm.org

Abstract—In this work we present a methodology based on self-organization to manage resources in networked embedded systems based on reconfigurable hardware. To enable self-organization in such systems, we have implemented a flexible monitoring service running on embedded Linux. Furthermore we explain our target application, a smart camera system, and give an insight into our effort to accelerate the open computer vision library from Intel with reconfigurable hardware.

I. INTRODUCTION

Increasing complexity due to the rapid progress in information technology is making systems more and more difficult to integrate and control. Due to the large amount of possible configurations and alternative design decisions, the integration of components from different manufacturers in a working system cannot be done only at design time anymore. Systems must be designed to cope with unexpected run-time environmental changes and interactions. They must be able to organize themselves to adapt to changes and avoid non-desirable or destructive behaviors. In heterogeneous environments devices with different abilities are available. An example would be a sensor network consisting of powerful nodes called motes doing all the computational work and collecting environmental information from less powerful sensor nodes, which of course could also do some in-network processing of the collected data.

The ongoing progress in chip technology provides us with steadily increasing computational power whereas the accompanying miniaturization shrinks the device sizes to unprecedented measures at rock bottom prices. It is obvious that even the weakest node will have enough computational power to share its resources with other nodes in the near future by offering its resources as services to them. In this work we try to answer the question how nodes in wireless ad-hoc networks may utilize reconfigurable hardware and share their resources with other nodes to optimize load and dependability, thus adapting to a changing environment. To answer this question we investigate self-organization principles to implement a distributed resource management in these kind of systems. Self-organization is naturally found in biological systems and can be defined as a process in which pattern at the global level of a system emerges solely from numerous interactions among the lower-level components of the system. Moreover, the rules specifying interactions among the system’s components are executed using only local information, without reference to the global pattern [1]. The translation of this definition into embedded systems has led to concepts like autonomic computing [2]. There an autonomic system is defined as a collection of so called autonomic elements, which are individual components that contain resources and deliver services to humans and other autonomic elements. Figure 1 shows the building blocks of an autonomic element [2]. It consists of an autonomic manager which controls one or more managed elements. The autonomic manager gathers information about the managed element and its surrounding environment by using a monitor and constructs and executes plans based on an analysis of this information. Prior learned knowledge can be used in these steps to construct more intelligent plans, thus behaving more intelligent. Not all of these blocks are necessary to build an autonomic element. It depends heavily on the application to decide which block should be implemented and how.

In this work we target a system made of a set of autonomous computing elements, each of which is a reconfigurable processor. The use of reconfigurable processors provides the required performance, while enforcing the flexibility by means

![Autonomic manager](image-url)
of reconfiguration.

Figure 2 gives an overview of our system. In this section only the architecture is described. The communication is described in section V. The monitor gathers information from the local system and offers this information to neighbour nodes and in turn it also gathers information from them. The type of information a monitor can offer and collect is manifold and ranges from system information like CPU load, memory usage, used/unused reconfigurable area to positioning information and offered services. A services a node may offer is for example its computational power. This information together with information from a knowledge database (integrated to the monitor) that is built upon foregoing experiences, is used by a resource manager that implements the \textit{analyze} and \textit{plan} blocks, to distribute tasks and optimize resource utilization or power consumption in the network. In order to enforce flexibility in such systems and allow single nodes to adapt their behavior to disturbances, we use reconfigurable processing units, in this case FPGAs. Those devices pose a decent tradeoff between computation power, energy consumption and flexibility. They can be configured and reconfigured to provide hardware acceleration for highly specialized services. Reconfiguration can be partial, while keeping the rest of the system working. Some are even capable of initiating their reconfiguration themselves keeping the part of their logical circuits that hosts the local operating system up and running while only a small partition that hosts specialized accelerators is being reconfigured[3]. The \textit{execute} block implements something like a reconfiguration management unit. It is able to take a piece of hardware (we call it hardware module or hardware accelerator) either from a local storage or from any other node in the network and reconfigure itself to execute a computation-intensive task.

To investigate the principles of self-organization in networked embedded systems, we have developed a flexible and extensible monitor to track the state of the surrounding neighbours, which serves as the basis for a powerful and flexible approach to manage the available resources in networked embedded systems with reconfigurable hardware.

The concept of a framework is shown that enables nodes to (re)distribute tasks in a marketplace-like manner, which delivers the basis technology to elevate implementations of collective task completion to a new level. Here a node that recognizes the need for a certain task to be done formulates it as a query to itself and its neighbours. Every node that offers the execution of a task replies to a query with its cost for fulfilling the job. The inquirer is now able to choose whether it is more appropriate to maybe reconfigure and do the task by itself or to delegate.

Both worlds, namely self-organization in embedded systems and embedded systems built around reconfigurable hardware, lead to highly adaptive distributed systems.

The remaining part of this paper is structured as follows. The next section gives an introduction into the technological basis we have used to develop and test our system (section II). It follows a description of our targeted application where we use partial reconfiguration for image processing (section III). Next we give an overview of the implemented monitor (section IV) and an explanation of a conceptual framework for a distributed resource management (section V). Finally we conclude this article.

\section{II. TECHNOLOGICAL BASIS}

The essential resources that must be available on each node in a distributed cooperative system are a minimal local computation power, a resource management, the ability to communicate and a system to distribute or gather jobs in the network. To develop and test our concepts we use a Virtex-II Pro FPGA from Xilinx sitting on the XUP development board [4], a Xilinx ML403 board [5] as well as the hydraXC-50 module [6] which are both equipped with a Virtex-4 FX12 FPGA.

Besides the configurable logic cells that allow custom hardware accelerators, the Virtex-II Pro and Virtex-4 both supply a certain amount of basic, hard-wired circuits (primitives) to extend the device’s speed and effectiveness. These are e.g. multipliers, block RAM and especially a module named ICAP: Internal Configuration Access Port. This module is the key ability for a node to change its own reconfigurable logic. Additionally Xilinx FPGAs since the Virtex-II series are capable of partial reconfiguration. That means that single hardware modules can be exchanged while vital parts of the cell, like the memory controller or the network interface controller, keep on operating uninterruptedly.

The hydraXC module has been used in conjunction with a MicroBlaze soft-core CPU and the uClinux operating system to rapidly develop the monitoring system as described in the next section. The Virtex-4 FPGA on this module is capable of partial reconfiguration and thereby able to exchange hardware modules with other devices in a network.

Linux was also used as general purpose operating system on the other development boards. Linux serves our needs in several ways. It encapsulates the difficulties with accessing hardware and thus facilitates resource deployment through its
abstract driver interfaces. For example the ICAP is incorporated into the Linux system with John Williams’ device driver [3]. Especially the fully developed networking abilities are a convenient way to realize communication. The innumerable quantity of applications for any kind of need and furthermore adaption and development of software in high-level programming languages pose a big plus. The use of a standard Linux kernel and applications also speeds up development for another reason: the large community that exists and the vast database of solutions to standard problems.

Linux provides us with a fully featured TCP/IP-stack that we have used as a basis for developing and prototyping our applications. We use tools like telnet to control devices remotely and wget to transfer data. It is clear that certain application fields have to adapt the used communication protocols to fulfill their needs, e.g. energy awareness.

Using partial reconfigurable hardware devices, we are in the position to equip our devices with a virtually adaptive behaviour. To keep development easy at the beginning our nodes provide only one region of fixed size that can be reconfigured separately from the rest of the system as shown in figure 3.

With this, we are able to realize a hardware-software co-design for optimized performance. Tasks that can efficiently be computed in hardware may be executed in a specialized hardware module. Other tasks that are not available as a hardware module or are more efficiently solved in software are then executed on the local CPU. We use the ICAP to allow the CPU to reconfigure the device itself to utilize free reconfigurable space or to replace unused hardware modules. The individual behavior of a device is determined by the controlling application, which decides whether to replace a local hardware accelerator or to employ a neighbouring node according to a set of parameters, stored locally in a database. These decisions are based on the analysis of the surrounding neighbourhood and constructing plans based on this information as explained in the introduction.

III. TARGETED APPLICATION

Our targeted application is a distributed smart camera system that is able to track an object optically over a large area where several cameras have to cooperate to keep the target within sight. A local node is now equipped with a video camera and has a certain computation power to extract features of object recognition from the taken pictures. Information about what the camera nodes detect will be send to the user’s terminal. To test partial reconfiguratin we built up a sample application to test our system that captures the video signal from a camera, digitizes it and applies a filter to the stream to provide it for further processing. The whole task is solved in hardware on the FPGA with the filter being reconfigurable at runtime. Figure 3 shows the physical layout of our sample system. There the small boxed area is reconfigurable, whereas the bigger region encloses fixed logic like the ethernet controller or the SD-RAM controller. Both are connected via fixed interfaces, so called BusMacros. The lighter lines on the picture are wire connections between logic blocks.

In the domain of computer vision the free available and open source library OpenCV by Intel [7] offers a wide spectrum of functions and is improved every day by its big community. It is based on a library of low-level functions like matrix multiplication or histograms. One of our goals is to speed-up these low-level functions by dedicated hardware accelerators to increase the overall performance of the library on FPGA based embedded systems. Therefore we developed a novel architecture for streaming data processing on FPGA.

In general streaming processing needs a data source, processing units (PU) and a data sink. The data source supplies pixels one by one and sends them to the first PU. PUs (which are performing the calculation) in any order and number are cascaded like a chain and finally the last PU is connected to a data sink.
To have the possibility to create a pool of PUs it is obvious to define a generic interface which all PUs have to satisfy. Our interface was named Streaming Data Interface (SDI) and is a set of data and control signals. We call the data source and sink module interconnection modules (IM) which have to link PUs to other modules where actually the data is located. Two cases can be distinguished: Data is accessible directly over point-to-point communication or via a bus (see figure 4). Examples for the first case are video-in module, dual ported RAM or a processor (p.ex. FSL of MicroBlaze = possibility to extend instruction set) and for the second, memory (usually main memory where processor stores its data) or any memory-mapped peripheral.

In hardware-software-co-design the data flow analysis plays an important role because the communication between modules can heavily influence the performance of the system. Our architecture offers a wide spectrum of possibilities to interconnect modules and to let data flow.

IMs which are connected to a bus are similar to the well known DMA controllers. They can access memory independently from the main processor and thus the IM’s PUs can work in parallel to the processor. As usual to bus connected modules the configuration registers are accessible over memory-mapped IOs. The processor has to write the source and destination address of the image data into the appropriated registers and finally set a start bit. After processing of a complete data block is finished an interrupt is raised to notify the operation system.

Since our system ought to be flexible and expandable the loss of camera nodes should be compensated to a certain degree in terms of surveilled area as well as computation power for image analysis. This will be achieved by distributing the work the failing node contributed to other units. Also new camera nodes should be able to be added as the task of the network and therefore the demand for computation power changes.

IV. MONITORING SYSTEM

The monitoring system consists of three parts. The main component of the system is the monitor daemon, which permanently monitors the neighbourhood. The second part is an interface, which we have implemented in c++ and java for simplifying access to the daemon. The third and last component is a java application used to log into a daemon to display its knowledge about its neighbourhood.

A. Daemon

The monitor daemon runs in an uClinux environment as a background service. Other applications may access the daemon by using a simple interface. The task of the daemon is to permanently collect information from and about other devices in the neighbourhood and simultaneously broadcast its own state. Information a device may gather are:

1) Devices in the neighbourhood,
2) their state (e.g. CPU load, memory usage, position, etc.),
3) and what capabilities they have (e.g. services they offer).

1) Device Discovery - Identifying surrounding devices: The daemon periodically broadcasts a ping message to inform other devices that it is still alive. This is done by every device in the network. When the daemon receives a ping message from another device A, it updates its list of known neighbours. If device A is not yet in the list, it is added to the list and a counter for that device is set. If device A is already in the list, the counter is reset to a standard value. Therefore the daemon keeps a list with all known neighbours with a counter value for each node. The daemon now periodically walks through the list in a specified interval (e.g. every 10s) and decrements each counter. When a counter of a device, let’s say device B, has reached a given limit, a ping message is sent to node B. If device B doesn’t respond with a ping message within a given time, device B is removed from the list of known neighbours. In this case an event can be generated to inform connected applications of this situation, thus supporting fault tolerance.

2) Service Discovery - Gathering a node’s capabilities: As well as the ping messages, each device in the network periodically broadcasts a list of information that it is offering. E.g. device A broadcasts that it is capable of sending a list of its services. Another device B may be able to give information about it’s state, like CPU load, memory usage, etc. Then device B periodically sends a message that other device can retrieve the system state of device B. Therefore each node in the network knows, which information it can obtain from its surrounding nodes. While ping messages are sent in quite a short interval, the lists are broadcasted less often, since we don’t expect them to change very often in stable environments. It is thinkable that the frequency of broadcasting these lists is gradually adapted when a node detects an unstable environment (e.g. moving nodes).

3) Being communicative - Obtaining information: Obtaining information is done in a publish/subscribe manner. If a device A is interested in receiving information from a device B, e.g. the CPU load, device A subscribes to this information. Device B then periodically broadcasts the requested information. If another device C is interested in the same information, device C also subscribes to this information, however, device B recognizes the subscriptions being the same and only sends the information once every period. This mechanism minimizes network traffic.

Basically the daemon consists of two interfaces, a node-to-node (NTN) and a node-to-application (NTA) interface, a tiny database and a set of plug-ins. The daemons communicate with each other using the NTN interface. Figure 5 shows three nodes, connected via a broadcast medium. Node A has four applications that use the NTA interface to gather information from the daemon. One of these applications is a notebook, which can be used to observe the system-behaviour during runtime.

If an application wants to get information from a daemon, it uses the c++ or java interface (as described in the next section) to connect to the NTA interface and collect the desired information. Using two interfaces, the complexity of the protocol for node-to-node communication can be kept
small. The database is needed for storing information from neighbours and allows simple search queries. So far, we’ve explained how distribution of information is done in the network, but where do the information come from? For this purpose the daemons use a plug-in mechanism to enhance the scope of the monitor. For instance there may be a plug-in that provides information about the available services, or a plug-in to catch system information from the underlying operating system. Additional plug-ins can easily be integrated in the system to enhance the daemons functionality.

B. The application Interface

The interface provides access to the daemon for information interchange. This is basically a c++ library, which can be included by other applications, but also a lightweight java version has been implemented. The interface provides a basic set of functions to retrieve information from the daemon and to configure the way it monitors the surrounding environment. This for instance makes it possible to retrieve the list of known neighbours by a single function call. The communication between the interface and the daemon service is done by using UDP sockets. This allows connections to daemons that are more than one hop away by using a routing service or a lower level routing protocol. The complexity of retrieving information from nodes that are far away can hereby be taken out of the monitor daemon.

Getting information from the monitor daemon using the c++/java interface is done in a polling way. If an application is interested in information (e.g. the services node A offers) it must explicitly call a function to retrieve the information. However, we have also implemented an event-driven mechanism to push critical information to connected applications. This might be the disappearance of a neighbour device A, which causes the system to reorganize itself to adapt to the changing environment. Therefore the daemon system may send crucial system events to any connected application.

C. Monitoring-Application

The last component of the monitor system is a java based application. Using this application it is possible to log into a monitor-daemon and display the daemon’s knowledge of its environment. Figure 6 shows a screenshot of the graphical user interface. It can be seen that the application is connected to a device with IP address 192.168.1.4. The nodes roundabout are those the device (with IP 192.168.1.4) has detected. The small window shows some local information of the device the application is connected to. arctime of By graphically displaying the neighbouring devices with their information, we have developed a simple tool to observe the system-behaviour during runtime.

V. LOCAL MARKETPLACES, GLOBAL SYMBIOSIS

We developed a simple concept to deploy our new features: LMGS - Local Marketplaces, Global Symbiosis [8]. The work is distributed according to principles of supply and demand within the network.

The simple idea is that every device does exactly what it deems to be the best according to its stored parameters. The minimal software to actively ‘take part in the game’ consists of two elements. A customer which issues requests to the network to have a certain job done and a purveyor that answers requests for jobs with the costs it charged if it would be commissioned. The effort a device makes to create the answer can vary widely. According to its computation power, knowledge and storage capacity this can reach from a simple return of standard values to a complex measuring where the load, the utilization of the node’s components or the probability for the effectiveness of anticipated reconfiguration might be taken into account.

In sensor networks for example communication is the most
expensive action so a distribution of work should be chosen that minimizes overall traffic, maybe by executing a lot of tasks locally through reconfiguring the node.

A. Requests

Requests issued by the customer component of a node consist of a tuple as follows:

\[ \text{Request} = \{ \text{Source}, \text{Target}, \text{Requester}, \text{DataVolume}, \text{Task}, \text{MaxHops} \} \]

Where Source contains the data source for the task, Target the data sink and Requester the device which posted the inquiry. DataVolume holds the quantum of data to be processed with Task. MaxHops specifies the maximum number of times a request may be relayed by a node’s neighbors.

Source, Target and Requester may be partly or completely identical. Given they are not, we included mechanisms to distribute jobs within a channel between data source and target. The values can be one-to-one identifiers of nodes or might as well contain wildcards to address groups of nodes so that for example, modifications that have to be applied to a set of devices can be launched by a single command.

The DataVolume will be taken into account when an offer is generated for the request. It usually influences the decision whether it is more appropriate to solve a task in software or in hardware and if the data may be processed remotely or if the communication costs for that would be too high.

In order to be able to specify a task or a whole group of tasks, we suggest an hierarchical nomenclature which comprises every single service that can be rendered in the network under one root node.

As Task in the request structure is not limited to one atomic job it may be a list of tasks. These lists may contain jobs that are to be processed sequentially or in parallel reaching from a single data-source to a single target, to complex data flows with multiple sources and targets.

Finally the restriction to MaxHops ensures that inquiries are not simply flooded through the net but stay local working off jobs at a kind of in situ marketplace when sensible. This of course only if the local neighbors provide appropriate solutions to tasks at a reasonable price. The composition of this ‘price’ will be presented in section V-B. The idea behind that is obvious: since we are able to reconfigure devices to serve virtually any need, even small localized groups are highly adaptable and will be able to cope with most challenges with optimal efficiency. Thus data will in general be kept in a spatially narrow cloud and communication is minimized. Later in this section and in part V-B we will explain how this is accomplishable with the claim of super-regional cooperation.

To publish and find services to fill in a valid request basically three mechanisms can be deployed. First one central directory server knows which device offers which services and has to be prompted for every job to work off. If it fails the whole network is paralyzed. New services have to be entered, causing additional traffic.

Second, searches for certain services are flooded through the whole network. This is very flexible, but rather inefficient. In this work, we developed a third approach, a hybrid solution between the two previously mentioned ones. Here data is flooded only within the closest neighborhood. Only if the answers from them are insufficient, say because none of the next nodes wants to execute the requested task, the job is advertised again, this time with a higher number of maximum relays. Additionally devices keep a more or less extensive list of other, remote hosts that satisfy a certain request. In this manner not only local offers will be taken into account but also more distant ones which possibly better suit the current situation. The maintenance of this list is closely related to the structure of offers replied to a request which will be covered in section V-B.

Generally, requests will be posed to the direct neighbors and to the issuer itself. A neighbor that finds the MaxHops greater than one reduces that counter and sends the request to all it’s neighbours. Especially the answer from the node itself is interesting in this regard: with the possibility to reconfigure, scenarios can be managed where communication cost is very high and nodes have to cut back on transporting lots of data through the net.

B. Offers

The response to a concrete query contains two elements: the cost-vector that the replying device is estimating for supplying the service and the local list of known providers that are also capable of satisfying this particular request:

\[ \text{Offer} = \{ \text{CostVector, List\_of\_known\_providers} \} \]

Costs mean figures given as multipliers of a base cost. For example the transmission of one byte of data over a wireless bluetooth connection will be a lot more expensive than over a wired ethernet link.

We identified three dimensions of costly actions: time consuming, energy consuming and space consuming. Thus a device’s purveyor calculates the cost vector for a task \( A \) according to

\[
C_A = (Z_K E_K P_K)^T W
\]

where \( Z_K \) denotes the cost for the local effort concerning time, \( E_K \) for energy and \( P_K \) for required space. \( W \) contains the willingness of the device to spend part of the specific resource to locally execute task \( A \) as a diagonal matrix. A battery driven device might for example want to lower its willingness to accept very energy consuming jobs as it runs low on battery power. The final cost-vector \( C_A \) is passed to the issuer as part of the offer.

The list of additional service providers is being built up through logging of messages indicating the commission of a node for a particular task or through deliberate writing. On the one hand devices that relay an acceptance message (basically a request with a specially formulated task) to a device store the target device and task together with an expiration time. On the other hand devices can advertise their capabilities by giving
C. Negotiation Example

The flow of a negotiation bases on sending requests, retrieving responses, determining the most appropriate solution and commissioning a purveyor (figure 7). When evaluating the replies, the contained lists of alternative, maybe remote, providers may be taken into account and selected ones may be prompted for a bid. This enables the system to incorporate both: decentralized, distributed computing as well as central services like the storage of gathered and processed data.

When enough answers came back in or after an amount of time the customer determines the optimal partner to commission. The weighted cost including communication is therefore calculated according to

\[ A : CC_A = \sum_i C_{Ai} \cdot G_{Ai} = \sum_i C_{Ai} \cdot G_{Ai} \]

Manipulating the components of \( G_A \) allows the device to emphasize a rather fast, energy-efficient or space preserving execution of a task. Depending on the computation-power and willingness the node might accept the earliest offer or may run a multi-goal optimization on the plenty of data retrieved to find the best tradeoff.

VI. CONCLUSIONS

In this work we have shown an approach for a distributed resource management based on self-organization which utilizes (partially) reconfigurable hardware to optimize the resource utilization in constrainted networks. Our work also contributes to dependability in networked embedded systems as disappearing nodes are easily detected, so that tasks can be reassigned to nodes willing to utilize their spare resources. To implement our approach we have developed a monitor system that collects information from the surrounding neighbourhood that can be used for device discovery, service discovery and the detection of disappearing nodes due to failures. For example, this system can be deployed to prototype self-organizing routing algorithms, load balancing, resource management etc. Furthermore we have described our targeted application, which is a distributed intelligent camera system which consists of nodes built upon reconfigurable hardware. Our approach for a self-organizing resource management is based upon local marketplaces where work is distributed according to the principles of supply and demand within the network to optimize the resource utilization, especially that of the reconfigurable hardware.

REFERENCES


Framework for Fast Performance Evaluation of Flexible Models Applied to Interconnect Delay

Alexis Vander Biest*, Dragomir Milojevic *, Aliénor Richard *, and Frédéric Robert*

* BEAMS Department - Faculté des Sciences Appliquées
Université Libre de Bruxelles, AV. F. D. Roosevelt 50, 1050 Brussels, Belgium

Email: {avdbiest,dmilojevic,arichard,frrobert}@ulb.ac.be

Abstract—In this paper we present a framework for the construction and evaluation of flexible models for fast performance analysis. Compared to state-of-the-art performance prediction systems, our tool allow the user to perform model/input sensitivity while improving flexibility and reusability of the models. All models and simulation inputs are described in XML with a strict grammar and enable direct XML simulation scripting or use through a C++ interface. To illustrate the functionalities, we implemented a chip frequency model extracted from the literature. Through our three case studies, we evaluate process variations impact on delay uncertainty, discuss optimal repeater insertion in wires and explore the limitations of the widespread Elmore delay.

I. INTRODUCTION

Nowadays designing chips is not anymore all about running CAD tools and programming, a lot of the design time is now spent on modeling. Predictability of the performances before implementation becomes crucial in order to meet the specifications. Not only power consumption, performance, maximum silicon area and reliability constraints are required to be fulfilled but also economical related performance like time-to-market, production and development cost, yield and so on.

In such a context, efficient and early design space exploration becomes mandatory to avoid to further implement a sterile solution that could have been discarded thanks to performance prediction [1]. Therefore designers use models that allow them to represent the most relevant aspects of the design. While complex simulation tools provide very accurate estimations they lead to very prohibitive evaluation times to reach that level of performances. This paper focuses on a different approach that uses much simpler models (mostly analytical relation based) more suited to carry out large number of evaluations at the earliest stages of the design process. We present our contribution as a framework providing support for the specification and execution of flexible performance models relying on XML for model description and input/output data. The remainder of the paper is divided as follows: section II reviews some existing modeling systems for performance prediction.

In section III we describe the hierarchical model structure of our framework while highlighting its features and present in section IV our implementation choices and XML based models grammar. Section V presents several model/input sensitivity studies and compare the results with the literature. Finally we conclude and describe future work in section VI.

II. STATE OF THE ART

A. A bit of model taxonomy

Before all, we will define how we use the term "model" trough this paper since they are the cornerstone to any performance prediction system. Without entering in deep model taxonomy considerations (the interested reader can find further information in [2][3][4]) we will define a model as a simplified representation of a "real" and measurable phenomenon in order to capture its main effects [5]. We assume that a model is characterized by at least the following criteria:

- A clearly defined interface with the inputs and outputs
- An execution capability
- Assumptions that specify which are the limits of use of the model

Many performance prediction systems have already been defined in the literature and their complete reviewing goes out of the scope of this paper. However we can distinguish two different categories:

- HW/SW co-design tools that are able to represent both the application and the architecture and represent their interaction to estimate the performances of the final system. We can cite among others Metropolis [6], Polis [7] and Design Trotter [8].
- Analytical based modeling systems are much simpler and therefore faster to evaluate. Strictly speaking, they cannot be considered as real tools since they only gather analytical relations to make a more complete models. Among all existing systems, those devoted to processor performance estimation developed during the last ten years deserve some special attention (like GENESYS [9], RIPE [10], BACPAC [11]). They simultaneously permit the evaluation of important parameters like power consumption, maximum clock frequency and wire related performance metrics.

If analytical relations based systems are interesting because of their evaluation speed, models are most of the time hardcoded hence remain hidden to the user so that no model sensitivity study can be carried out and no model can be added. To cope with these limitations, a very interesting model-centric framework called GTX [12] has been developed by the GSRC.
B. Modeling with GTX

GTX is a flexible modeling tool that allows the user to define its own models and make them coexist with existing ones. GTX defines three objects: parameters, rules, rule chains.

- **Parameters** represent all the physical data that have to be considered in the model (for instance power consumption, wire resistivity, technology node...).
- **Rules** represent the model and describe how the parameters are related. For instance the dynamic power consumption depends on the capacitance to switch, the operational clock frequency and the supply voltage as follows:

\[
P_{\text{dyn}} = C_{\text{switch}} \times f_{\text{clock}} \times V_{\text{dd}}^2
\]

Differential types of rules can be used such as hard-coded, ASCII closed-formed expressions or tables.

- **Rule chains** are formed by the gathering of rules allowing the user to build larger models based on several sub-models. For instance if we have at disposal a rule of the total capacitance based on technological parameters, we could use it as input for the dynamic power.

Rule chains are the cornerstone of GTX giving the framework a lot of flexibility and enabling model-sensitivity analysis. However it suffers from some limitations:

- Each rule is defined in a such a way that it defines one parameter as the output, the other ones being implicitly defined as inputs: GTX then automatically composes the resulting rule chain. Depending on the context of use, we however may want to turn some inputs of the model into outputs and inversely. For instance, all the microprocessor systems presented above estimate the clock frequency based on the technological parameters but it may sometimes be much more useful for a designer to express that same clock frequency as a model input rather than an output so that he can fix constraints on it and find the resulting possibilities on the technological parameters.

- Parameter sensitivity is supported in GTX at the expense of evaluating several times the same rule chain while varying each input parameter value around its nominal value. Its leads to multiply the number of experiments hence the time required to carry them out.

- GTX relies on ASCII textual files to specify rules and their meta-information. Mechanisms to manage rules storage, parameter and rule name uniqueness and a grammar to specify studies were never implemented.

In this paper we present principles that we implement in our framework to respectively overcome the above limitations:

- By introducing a three-level hierarchical model description we remove input/output reversibility limitations both at rules and rule chains levels.
- Closed-formed expressions may be represented by trees composed out of basic operations allowing both dynamic model execution and under certain conditions much faster evaluation of output parameters extrema based on input constraints values.

- The underlying framework grammar for models and parameters is entirely specified by XML schemas to offer automatic verification along with powerful element constraint mechanisms.

Oppositely to GTX appearing as a standalone tool with its GUI, our framework comes as an easy-to-use C++ library that was originally meant to suit our own needs but may finally be interesting to share. The main functionalities of this library are the following:

- Construction and grammatical verification of user defined models
- Composition of larger models based on the gathering of smaller models
- Single execution of models and parameter sweeps
- Fast evaluation of output parameters extrema for closed-formed expressions
- Easy model sensitivity analysis

In the coming section we detail the general principles listed above.

III. FRAMEWORK DESCRIPTION

A. Hierarchy

In our representation, models are described in a hierarchical way: each level of abstraction exchange information through parameters i.e. the data on which models operate. The parameters are characterized by a unique name and value.

The three types of entities used for our model description are the following:

- **Generic rules** are equivalent to rules in GTX except that they do not assume any underlying semantics of execution but only requires execution capability.
- A **Relation** is a logical link between n parameters meaning that they are not independent so that we can only fix the n-1 parameters values among the n possible. A relation encompasses generic rules associated to parameters in order to make the relation executable.
- A **behaviour** is a complete model with several input parameters and output parameters. It is based on a network of relations.
The improved flexibility and context adaptability of our models precisely relies on the introduction of the relation and behaviour levels in our model representation. Let us now have a closer look at these three levels.

B. Generic rules

A generic rule is an executable model as it has been defined in II-A for a given output and its corresponding input parameters. The generic rule for equation 1 is represented by (2).

\[
P_{\text{dyn}} = GR_1(C_{\text{switch}}, f_{\text{clock}}, V_{\text{dd}}^2)
\]  

Examples of generic rules are readily available like wire delay models based on technological inputs [13] (inter-wire dielectric value, wire resistivity, wire pitch...), any hard-coded model or any algorithm linking the output parameter value with its input values. Each particular model class inherits from the properties of the generic rule definition so that it is easy to derive new model classes. However, we provide special support for analytical closed-formed rules in order to enlarge the framework features with:

- A well-defined representation of closed-formed expressions
- An efficient parametric sensitivity estimation method based on input parameters values constraining.

1) Analytical rules tree representation: To allow the user to choose among rules interactively at run-time, we need to find an adequate way to represent closed-formed expressions. In order to turn a textual closed-formed expression into a dynamically executable model, we used the Dijkstra shunting yard algorithm [14] to translate it into a tree. Each vertex represents a parameter, each directed edge a basic operation pointing to the vertex that is the output parameter of this basic relation.

\[
D = \left( \log(A) + \frac{C}{2} \right) \ast B
\]  

For instance, the closed-formed expression (3) is represented by Fig.2. As we can see, intermediate parameters are added resulting from some basic operations: they have no significant meaning but are necessary to ensure the computability of basic rules. Only input and output parameters are terminal vertices hence are the only visible parameters outside the rule. The basic operations supported actually are multiplication, division, addition, subtraction, logarithm and power operator. With this basic set of operations and the use of parenthesis to force execution priority, we can already represent a lot of closed-formed models.

2) Fast extrema evaluation: Parametric sensitivity analysis can become highly time consuming when the number of parameters and values per parameter to sweep increases. If the purpose of this study is to find out the statistical distribution of the output parameter, the time devoted to the study cannot be reduced as all solutions need to be explored. But when it comes to find out extremum values, our representation can save a lot of time. Indeed if we consider any of the previously defined basic operations, it is easy to find out the minimum and maximum output values knowing the input value bounds. For instance, if we proceed to a parameter sensitivity analysis for two parameters \(a\) and \(b\) by fixing their respective bounds between \([\text{min}_a, \text{max}_a]\) and \([\text{min}_b, \text{max}_b]\) then we can compute the minimum and maximum values for the \(\ast\) operator as being respectively \([\text{min}_a + \text{min}_b, \text{max}_a + \text{max}_b]\). Using a depth-first algorithm, we can easily compute the output extrema of the leaves and propagate it to the root. Hence only a sole tree evaluation is needed to get output extrema values (algorithm complexity of \(O(1)\)) compared to hard-coded rules for which the spanning of all possible input values within the defined bounds would require much more time (algorithm complexity of \(O(M \ast N)\) where \(M\) is the number of relations and \(N\) the number of values per parameter to explore).

There is only one limitation to that method: to make the constraint propagation and find the valid extrema, parameters may only appear once in the whole behaviour. In other words, it means that our framework is only able to find the extremum of first degree parameters in expressions: otherwise it would have required mathematical solving functionalities. For behaviours combining first degree and higher degree parameters, it is still possible to propagate constraints on the sole first degree parameters while sweeping other parameters values so that we could still obtain a performance gain compared to the full parameter values sweeping solution.

C. Relations

A relation is a logical dependence between a set of parameters meaning that they are related in a certain way and that their value cannot be fixed arbitrarily. The exact dependence is precisely specified by one generic rule defining a parameter as being the output and the others becoming inputs. So if a relation links \(n\) parameters together, each parameter may be associated with a generic rule (regardless of its exact nature) so that the value of the other \(n-1\) parameters is sufficient to determine the output value. This mechanism of association relates to the reversibility property of some models: based on the knowledge of one generic rule associated with one specific parameter, it may be possible to derive the generic rules for

![Fig.2. Tree representation of expression](image-url)
the other parameters (manually or automatically depending on the equation parameters degree). For instance if we take back equation 1 we can easily associate a new generic rule to $C_{dyn}$, $f_{clock}$ and $V_{dd}$ defining (4).

$$Rel(P_{dyn}, C_{switch}, f_{clock}, V_{dd}) = \{GR_1, GR_2, GR_3, GR_4\}$$

$$GR_1 : P_{dyn} = C_{switch} * f_{clock} * V_{dd}^2$$

$$GR_2 : C_{switch} = \frac{P_{dyn}}{f_{clock} * V_{dd}^2}$$

$$GR_3 : f_{clock} = \frac{P_{dyn}}{C_{switch} * V_{dd}^2}$$

$$GR_4 : V_{dd} = \sqrt[1]{\frac{P_{dyn}}{C_{switch} * f_{clock}}} \quad (4)$$

It is important to note that not all generic rules may be inverted (like algorithmic based rules): in that case the corresponding association may remain empty.

D. behaviours

A behaviour gathers relations to make a complete model without any restriction on the number of outputs and inputs. Starting from a collection of relations, an adequate representation of the relations has to be used in order to make the behaviour executable.

1) Graph representation of behaviours: Again we are using a graph-based representation of the relations:

- Each relation is represented by a multi-edge
- Each parameter is represented by a vertex
- The parameters (identified by their name) shared by several relations form in turn a whole graph expressing their mutual dependence.

Here is a simple example of a behaviour composed out of four relations and expressed by the system of four equations (5).

$$Rel_1(A, C, D, E) \quad Rel_2(E, B, H) \quad Rel_3(D, F, G) \quad Rel_4(G, E, H) \quad (5)$$

The number of degrees of freedom of the resulting behaviour equals $\#parameters - \#relations$ and represents the number of parameter values we have to fix for the system of equations to be completely determined.

Translated in our non-oriented graph representation, this behaviour becomes the graph shown in Fig.3 (all the vertices linked to a same Rel vertex belong to the same relation).

2) Behaviour association: In order for the behaviour to be executable, we have to specify which are the input and output parameters i.e. to associate each relation with an output parameter (in other words to turn the non-oriented graph into an oriented graph). If the maximum combination number equals $\prod_{i=1}^{\#relations} \#parametersInRelation_i$, all these solutions are not leading to directly executable behaviour. To illustrate that, let us take back our previous system of four equations (relations) and choose a rule association for each of them (6).

$$E = GR_{Rel_1}(A, C, D) \quad E = GR_{Rel_2}(B, H) \quad$$

$$D = GR_{Rel_3}(F, G) \quad G = GR_{Rel_4}(E, H) \quad (6)$$

Looking at these associations we can notice two important things about the way the equation system (6) is formed:

- First, $E$ is the result of two different rules which has no sense at all as it can only have a single value.
- Second, if we try to solve this system by a simple substitution of rules $GR_{Rel_1}$, $GR_{Rel_3}$ and $GR_{Rel_4}$ we get the following result:

$$E = GR_{Rel_1}(A, C, GR_{Rel_3}(F, GR_{Rel_4}(E, H))) \quad (7)$$

As we can see, this equation is not explicit because the parameter $E$ appears in both members. This means that the equation cannot be solved without using a numerical solver. Coming back to our behaviour graph representation, these two considerations involve two rules that an oriented graph should follow in order to be valid and executable:

- Each vertex of the oriented graph should have one and only one edge pointing to it.
- The graph must be acyclic (in other words, starting from a vertex, there should be no path of consecutive oriented edges that could lead the same vertex).

Fig.4 shows two valid graphs for our studied behaviour: compared to non-oriented graphs, each relation gets one oriented edge and output parameters are surrounded by boxes while input parameters are surrounded by dashed boxes. As we can see a same non oriented graph may lead to oriented graphs with different number of outputs: these different behaviour orientations can in turn be used to solve different problems while still using the same underlying models. To find all the rule association combinations that satisfies the two previous
conditions for the behaviour, we developed a simple algorithm based on a backtracking method. Starting from the graph, the main steps are the following:

1) Gather all the relations into a unexplored relations list using a depth-first search algorithm
2) Check the unexplored relations list:
   - If the unexplored relations list isn’t empty select the next relation of the unexplored relations list. We go on to point 3.
   - If it is empty, it means that we have explored the whole graph without violating any of our two rules for each of them : this behaviour is valid and all relation associations are memorized. We restore the context (lists states) of the next association to check and go back to point 2.
3) Test the next available association for current relation:
   - If the relation adjunction to the graph does satisfy both non acyclic graph and one output per vertex conditions, move the relation from the unexplored relations list to the explored relations list and go back to point 2 to associate the next relation.
   - If it doesn’t, retry another association for the relation as long as there are any, otherwise go to point 2 to try another relation.

This algorithm explores all possible associations while eliminating all associations that don’t satisfy both rules for the graph. To keep track of the state of the graph (associations of the relations and lists) along the algorithm execution, we are using an incremental context saving mechanism so that only elements that have been modified since the last valid possibility need to be restored saving both memory and computation time. When a partial association of the graph violates one of the conditions, all subsequent associations will not be explored since they will lead to non-valid solutions.

IV. IMPLEMENTATION

A. Core engine

The core of the framework has been developed in C++ under Eclipse 3.2.1 using a GCC 4.0.1 compiler. The implemented object oriented structure perfectly mimics the hierarchical structure (see section III-A) composed out of behaviours, relations and generic rules. During implementation, a particular focus has been put on the maintainability and reusability. This can be illustrated throughout the different objects composing our framework:

- **Generic Rules** are the core model evaluation mechanism. Two very important types of generic rules have been implemented yet: *analytical rules* and *table-based rules*. *Analytical rules* are represented by trees composed out of smaller operations called analytical elements (see section III-B.1). Each of these elements derive from a generic *analyticalElement* class making it very easy to add further basic operations beyond those already implemented. Only the associativity, the representation symbol, the precedence and the method to evaluate this operation are required to build a new type of analytical element.

With this single modification, an expression including this new basic operation is ready to be parsed and directly executable since no modification of the shunting-yard algorithm is needed. If an invalid mathematical operation occurs for a set of input values (negative logarithmic base, division by 0 and so on) the current evaluation is simply invalidated and we move on to the next evaluation. *Table-based rules* allow the user to define output parameters values for each combination of n input parameters values. The data are represented in n-dimensional dynamically built tables. If one input parameter value does not correspond to any of the predefined values, the current evaluation is tagged as invalid.

- Relations can simultaneously contain different types of rules for the evaluation of each associated parameter since all types derive from a common genericRules class.
- Behaviours completely encapsulate and hide relations to the user so that they can be evaluated by calling a single method regardless of the number of output/input parameters, relations and estimations.

B. XML grammar for model support

In order to make our models reusable, it is crucial that we define a strict grammar to represent them, allow the user to store and save models of its own and provide the framework with a standard interface to initialize the C++ object structure. The easiest way to do that is to define our own formatted input and output files and parse them in C++. However such approach is highly time-consuming for more complex structures. To cope with all that problems we have chosen to use a XML shema based grammar to represent all our input and output data. The advantages of such a technology are numerous:

- The grammar is specified thanks to XML schema so that it is completely independent of the C++ framework. Given one XML document we can easily verify if it is valid against the grammar thanks to an external validating parser.
- Apart from the structure of the document, using XML schema also enables data constraining mechanisms (value uniqueness and key references) which -to a certain extend- relieves the programmer from verifying the data correctness.
- Standard interfaces (namely SAX and DOM) have been defined for object-oriented languages to manipulate XML documents easily and extract data from the structure.
- Transforming XML files into browsable content is very easy and offers very promising opportunities for the possible perspective of building an online model repository.

The use of XML has only one major drawback: producing human-readable documents implies a lot of structural information to be present in the document. Hence XML scores very poorly when it comes to estimate the ratio of meaningful data per total file size: data rarely represents more than one half of the file memory weight.

The structure of our schemas again matches our previous
hierarchical decomposition into behaviours, relations and generic rules. Each entity is described by a separate XML schema and are hierarchically recomposed. Since XML schemas granularity matches class granularity, it is very simple to turn a model description into a C++ object structure and inversely.

XML schemas are parsed thanks to the XercesC 2.7 API [15] using SAX2 to extract data. Despite being one of the wide spread and most complete XML parser for C++, Xerces does not implement all XML features and technologies. For instance Xinclude which enables easy data inclusion from one XML document into another one is currently not supported. This feature is however particularly well suited to our purpose: relations might be defined in separate documents and a behaviour could then just consist of an enumeration of relation’s XML documents. Apart from making behaviour building much more easy than relation cut and paste operations it further avoids redundancy of information thus reduces error rate. To cope with that lack, we have implemented our own Xinclude processor thanks to a simple Perl script that replaces all occurrences of Xinclude tags by their linked XML file content while preserving the validity of the resulting file.

C. Detailed functionalities

The available functionalities are the following:

- Behaviour building based on an XML file description
- Setting up a value estimation simulation. It consists in detailing the input parameters values (unique values, listed values or sweep mode from a start value to a stop value with a defined step)
- Performing a value estimation for all parameters values combinations specified before
- Setting up a sensitivity which consists in specifying the value constraints for each behaviour parameter.
- Performing a sensitivity analysis
- Modifying a sensitivity analysis
- Modifying rules in a relation/behaviour to perform model sensitivity analysis
- Changing the orientation of an existing behaviour.

All these features can be invoked either through the C++ interface or may directly be read from an XML file containing the above functionalities written by order of execution. This last solution is a very convenient for script simulations since it does not require to recompile the code several times.

V. CASE STUDY

In this section we will illustrate the functionalities of our framework by implementing and evaluating a model of the maximum clock frequency attainable by the chip.

A. Model assumptions and underlying relations

In our frequency model we suppose that:

- Each stage is represented by a gate driving a wire with a fan-out 4 load. We used the delay mode (8) proposed by [16] where \( R_{gate} \) is the driver resistance, \( C_{diff} \) the diffusion capacitance, \( C_{wire} \) the total wire resistance, \( C_{load} \) the load capacitance and \( R_{wire} \) the total wire resistance.
- \( C_{wire} \) accounts for both substrate and side-to-side capacitances using a four wire neighbors model [17] but neglects fringing capacitances.

\[
\text{Delay}_{stage} = R_{gate} \cdot (C_{diff} + C_{wire} + C_{load})
+ R_{wire} \cdot (0.5 \cdot C_{wire} + C_{load})
\]  

(8)

B. Estimation results

Several experiments have been performed on the above model to show how our framework can be used in the context of input parameter/model sensitivity analysis. To do so we have implemented this model in our framework and we obtained one behaviour composed out of 14 relations. All the technological input data are extracted from the ITRS 2003 Roadmap for the 90nm node [18].

a) Model sensitivity: We compared our model with a widespread and commonly used delay model called the Elmore delay model[19]. Its major advantage over other models relies in its high simplicity that enables fast delay computation in CAD tools[20]. Unfortunately it always underestimates the stage delay which is very problematic for critical path timing analysis.

\[
\text{ElmoreDelay}_{stage} = R_{gate} \cdot (C_{wire})
\]  

(9)

To estimate the accuracy of both models, we have performed several simulations with the Horowitz model (8) and again with the Elmore delay (9) (by simply changing the generic rule associated to the stage delay). During the sweep simulation, we made pitch and wire dimensions vary from one to three times the minimal size based on design rules. The result show that stage delay underestimation is comprised between 29.8% and 51.6% with an average of 38.5%. If the proportions between wire dimension and wire pitch vary beyond four, the error increases. This can be easily explained by looking at both model equations: Elmore delay only takes into account the term \( R_{gate} \cdot C_{wire} \) that is believed to be the most important contributions in (8). This assumption becomes wrong when \( C_{wire} \) decreases too much and when other terms cannot be neglected anymore: that’s exactly what happens when wire spacing increases for instance.

In conclusion, Elmore model remains very useful to make rough estimations of stage delay if the wire dimensions and pitch vary in the same proportion. In that case a 40% correction factor could be applied for worth case timing analysis; otherwise more complex models should be used.

b) Delay sensitivity to process variations: In performance estimation, it is important to take process variations into account to test the robustness of one design choice. In this experiment we used our constraints mechanism (see section
<table>
<thead>
<tr>
<th>Technological parameters</th>
<th>Miller effect</th>
<th>Stage delay</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Lower value</td>
<td>Upper value</td>
<td></td>
</tr>
<tr>
<td>±5%</td>
<td>1</td>
<td>−6.8%</td>
<td>21.1%</td>
<td></td>
</tr>
<tr>
<td>±5%</td>
<td>0.5 → 1.5</td>
<td>−24.6%</td>
<td>55.2%</td>
<td></td>
</tr>
<tr>
<td>±10%</td>
<td>1</td>
<td>−13.4%</td>
<td>48.1%</td>
<td></td>
</tr>
<tr>
<td>±10%</td>
<td>0.5 → 1.5</td>
<td>−30%</td>
<td>76.4%</td>
<td></td>
</tr>
<tr>
<td>nominal</td>
<td>0.5 → 1.5</td>
<td>−19.1%</td>
<td>19.1%</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 5. Process variations impact on stage delay

III-B.1) to specify bounds for process variations on the inter-wire isolation material relative dielectric constant, the gate oxide relative dielectric constant, the transistor threshold voltage and the supply voltage. 5% and 10% variation around these technological parameters nominal values produce delay stage variations that are computed and normalized afterwards around the stage delay nominal value. Additionally Miller effect[13] is taken into account to represent neighbors wire switching parasitic capacitances by multiplying side-to-side capacitances by a correction factor. While many authors assume worst case Miller effect coefficient value to vary between 0 and 2[17], we used a less pessimistic 0.5 → 1.5 range around 1. The nominal stage delay value equals 4.97ns and table5 represents the results.

The results clearly show that even a 5% process variation highly impacts the stage delay and introduce almost as much uncertainty as the Miller effect. The combination of 10% input variations with Miller effect even produces a 100% dispersion around the nominal value. This kind of simulation is very precious to designers and process engineers wanted to quickly quantify the impact of process variations on delay.

c) Optimization of the stage number: Given a fixed silicon surface, let us now examine how the maximum clock frequency evolves with the number of stages. This problem is very similar to the optimization of the buffer size/number in long wires[21]. Since we know that wire delays are proportional to the square of the length, splitting wires into smaller segments decreases the total delay and tend to make it linearly -instead of quadratically- dependent on the length. However intermediate buffers have their own delay so that there is an optimum where the adjunction of one repeater adds a bigger delay than the gain achieved through further wire splitting.

To find the optimum number of stages, we performed a sweep simulation by varying the value from 1 to 40 for a 1cm² chip. Fig.6 shows that the optimal number of stages equals 13 for our model. This is a very interesting conclusion since authors indeed consider that the average number of stages for one path is comprised between 10 and 14 [13]: our result confirms the delay optimality of this value. Furthermore Fig.6 highlights that long wires with few drivers are much more penalizing in terms of delay than smaller wires to cover the same distance on the chip. One could also be surprised at first sight that the curve is very flat particularly beyond the optimum value. It can be explained quite simply by taking back (8). When the term $R_{gate} \times C_{wire}$ dominates, the delay of one stage becomes proportional to the wire length since $R_{gate}$ does not depend on the length. So when it is multiplied by the number of stages (proportional to the wire length of one stage), it tends to keep the total delay almost constant. For our experiment we are in that case because we chose moderate sized drivers (10 times the minimum sized inverter) which implies a high value of $R_{gate}$. That’s not the case for classic optimal buffer insertion that leads to high output current drivers (450 times the minimum sized inverter) that makes speed prevail over power consumption. This can also be seen in our framework: when the driver size increases, the optimum value for the number of stages increases and the curve becomes much more steeper around the optimum.

C. Framework performances

To estimate the computation performance of the framework we have carried out several tests on our frequency model. All experiments were made on a 2.0 GHz Intel CoreDuo2 running under MacOSX.

First we made the framework initialize the behaviour based on the XML input file. It took 61ms to parse the file and to build the object structure.

Second we performed sweeps across parameters values while enabling XML file generation feature for the result values. To compute 20,000 entire behaviour evaluations the framework took 5.95s which leads to 3390 behaviour evaluations per second. This measure is however dependent on the complexity of the underlying model: to get rid of that dependence we have estimated the number of basic operations involved in this operation. Our 14 relations model contains a mix of 45 different basic operations so that the framework performs on average 152k basic operations per second. This measure represents a fairly constant and reliable value because each relation (and each enclosed basic operation) is executed only once per behaviour evaluation whatever the number of shared expressions.

Third we have carried out the same test without the XML
output file generation: this is useful for anyone wanting to interface the framework with its own C++ code that only requires to exploit raw data coming from evaluations. The framework performed 1.78M basic operations per second showing that the generation of XML output files makes the performance decrease more than 10 times (only because of the file writing and storage overhead).

The good evaluation speed per operation makes possible the use of our framework for model and input parameters exploration.

VI. CONCLUSION

In this paper we presented our framework and detailed its features for model estimation and construction. The XML grammar combined to the C++ interface makes it very convenient to use it as a standalone tool or combined with the user own application. We presented three different case studies to demonstrate both input parameters and model sensitivity analysis abilities.

In the future we would like to provide the program with enhanced analysis functionalities to help the user to manage the huge amount of data that is produced. Apart from adding statistical analysis functionalities (average and variance) we also plan to add XPath support to the framework to perform queries on the result file (maximum value for one parameter, number of solutions where parameter value is above a threshold and so on). This feature is already available using external XML editor tools but should now be embedded into the framework core.

REFERENCES


Speed-up run-time reconfiguration implementation on FPGAs

Florent Berthelot\textsuperscript{1}, Dominique Houzet\textsuperscript{2}, Fabienne Nouvel\textsuperscript{1}
\textsuperscript{1}CNRS UMR 6164 IETR/INSA Rennes, 20 av des Buttes de Coesmes, 35043 Rennes, France
\textsuperscript{2}GIPSA-lab INPG, 46 avenue Flix Viallet, 38031 Grenoble cedex, France
florent.berthelot@ens.insa-rennes.fr

Abstract—Reconfigurable computing is certainly one of the most important emerging research topics over the last few years, in the field of digital processing architectures. The introduction of run-time reconfiguration (RTR) on FPGAs requires appropriate design flows and methodologies to fully exploit this new functionality. For that purpose we present an automatic design generation methodology for heterogeneous architectures based on Network on Chip (NoC) and FPGAs that eases and speed-up RTR implementation. We focus on how to take into account specificities of partially reconfigurable components during the design generation steps. This method automatically generates designs for both fixed and partially reconfigurable parts of a FPGA with automatic management of the reconfiguration process. Furthermore this automatic design generation enables configuration pre-fetching techniques to minimize configuration latency and buffer merging techniques to minimize memory requirements of the generated design. This concept has been applied to different wireless access schemes, based on a combination of OFDM and CDMA techniques. The implementation example illustrates the benefits of the proposed design methodology.

Keywords: System-level design flow, Reconfigurable computing, Run-time reconfigurable FPGA-based system.

I. INTRODUCTION

Applications such as multimedia, encryption or wireless communication require highly repetitive parallel computations and lead to incorporate hardware components into the designs to meet stringent performance and power requirements. On the other hand architecture flexibility is the key point for developing new multi-standard and multi-application systems. Application-specific integrated circuits (ASICs) are a partial solution for these needs. They can reach high performance, computation density or power efficiency but to the detriment of architecture flexibility as the computation structure is fixed. Furthermore, the non-recurring engineering costs (NREs) of ASICs have been increasing dramatically and made them not feasible or desirable for all applications especially in case of bug-fixes, updates and functionality evolutions. Flexibility is the processor’s architecture paradigm. The algorithm pattern is computed temporally and sequentially in the time by few execution units from a program instruction stream. This programmability potentially occurs at each clock cycle and is applied to a general computation structure with a limited parallelism computation capacity. The data-path can be programmed to store data towards fixed memories or register elements, but can not be truly reconfigured. This kind of architecture suffers from the memory controller bottleneck and power efficiency. These architectures have a very coarse-grained reconfiguration, reported as system level. Reconfigurable architectures can provide an efficient platform for satisfying the performance, flexibility and power requirements of many embedded systems. These kinds of architectures are characterized by some specific features. They are based on a spatial computation scheme with high parallelism capacity distributed over the chip. Control of operator behaviour is distributed instead of being centralized by a global program memory. Multiple reconfigurable architectures have been developed \cite{1} \cite{2}, they can be classified by their reconfiguration granularity. FPGAs have a logic level of reconfiguration. Communication networks and functional units are bit-level configurable. Their highly flexible structure allows to implement almost any application. A volatile configuration memory allows to configure the data-path and the array of configurable logic blocks.

Architecture granularity elevation allows either specialization and performance or the improvement of architecture flexibility as for the processors. The introduction of dynamically reconfigurable systems (DRS) has opened a new dimension of using chip area. Recently run-time reconfiguration (RTR) of FPGA parts has led to the concept of virtual hardware \cite{3}. RTR allows more sections of an application to be mapped into hardware through a hardware updating process. A larger part of an application can be accelerated in hardware in contrast to a software computation. Partial reconfiguration ability of recent FPGAs allows updating only a specified part of the chip while the other areas remain operational and unaffected by the reconfiguration. These systems have the potential to provide hardware with flexibility similar to that of software, while leading to better performances.

However, cutting-edge applications require heterogeneous resources to efficiently deal with the large variety of signal and multimedia processing. This is achieved by mixing various processing granularities offered by general purpose processors (GPPs), digital signal processors (DSPs) and reconfigurable architectures. These processing units need to communicate through a flexible media managing heterogeneous communications. A Network on Chip (NoC) is an emerging solution. This paper presents our methodology allowing a fast integration and use of run-time reconfigurable components, namely FPGAs. This implies a convenient methodology and
conception flow which allows a fast and easy integration of run-time reconfiguration onto applications and an automatic management of reconfiguration process. Design space exploration will benefit from this improvement as operations of the application algorithm graph could be mapped either on DSP/GPP devices or statically/dynamically on FPGA devices thanks to run-time reconfiguration. The rest of the paper is organised as follows. Following the introduction, Section II gives an overview of related approaches, especially those which are focused on dealing with run-time reconfiguration from a high-level design specification. Section III then addresses the automatic design generation targeting run-time reconfigurable architectures. This design flow and methodology has been applied for the implementation of a transmitter system for future wireless networks for 4G air interface presented in section IV where two implementation examples based on a Network On Chip (NoC) and point to point communication scheme are presented. Finally section V provides a conclusion of this work and gives some indication of future work.

II. RELATED WORK

Numerous researches are focused on reconfigurable architectures and the way to exploit efficiently their potentials. Higher level of abstraction, design space exploration, hardware/software partitioning and co-design, rapid prototyping, virtual component design and integration for heterogeneous architectures; all these topics are strong trends in architectures for digital signal processing. Silva and Ferreira [5] present a hardware framework for run-time reconfiguration. The proposed architecture is based on a general-purpose CPU which is tightly connected to a dynamically reconfigurable fabric (DRF). A tool named BitLinker allows the relocation and assembly of bitstreams of individual components and is used to automate these steps which are very dependent on the underlying hardware’s organisation. This approach is architecture-centred and application mapping steps on this specific platform is not addressed. This issue requires a higher level of abstraction for design specification. PaDReH [6] is a framework for the design and implementation of run-time reconfigurable systems and deals only with the DRS hardware design flow. The use of SystemC language enables a higher level abstraction for design and validation. After a translation to the RTL level a space-time scheduling and hardware partitioning is performed allowing the generation of a run-time reconfiguration controller. Bitstreams generation is based on Xilinx [7] Modular Design Flow. Craven and Athanas [8] present a high-level synthesis (HLS) framework to create HDL. The hardware/software partitioning is performed by the designer. Reconfiguration simulations and reconfiguration controller generation is allowed from a modified version of the Impulse C ANSI C-based design language supporting HLS from C to RTL HDL. Nevertheless, final implementation steps are not addressed. The EPICURE project [9] is a more comprehensive methodology framework based on an abstraction tool that estimates implementation characteristics from a C-level description of a task and a partitioning refinement tool that realizes the dynamic allocation and the scheduling of tasks according to available resources in the dynamically reconfigurable processing unit (DRPU). An intelligent interface (ICURE) between the software unit and the DRPU acts as a hardware abstraction layer and manages the reconfiguration process. Overlapping between computations and reconfigurations is not supported. Most of these above methodologies frameworks assume a model of external configuration control, mandating the use of a host processor or are tightly coupled to a specific architecture [9] [5]. Hence custom heterogeneous architectures are not supported. Few of them address methods for specifying runtime reconfiguration from a high-level down to the consideration of specific features of partially reconfigurable components for the implementation. Our proposed methodology deals with heterogeneous architectures composed of processors, FPGA or any specific circuits around a Network on Chip. The implementation of run-time reconfiguration on hardware components, especially FPGAs, is automated and eased by the use of a high-level application and architecture specification. That is handled by the SynDEx tool which allows the definition of both application and hardware from a high level and realizes an automated Hardware/software mapping and scheduling.

III. AUTOMATIC DESIGN FLOW TARGETING RUN-TIME RECONFIGURABLE ARCHITECTURES

Figure 1 depicts our overall methodology flow based on Syndex tool [10]. Each macrocode executive generated by Syndex is translated toward a high-level language (HDL or C/C++) for each HW or SW component. This translation produces an automatic dead-lock free code. Macro-code directives are replaced by a corresponding code from libraries (C/C++ for software components, VHDL for hardware components). Many libraries have been developed for heterogeneous platforms and we have extended SynDEx capacities to handle runtime reconfigurable components. In our methodology, the algorithm graph application description is realized at the functional level representing coarse-grained operations. This description is handled during the design flow through the use of libraries containing Ips definitions for code generation. These coarse-grained IPs are supposed to be developed to fully
exploit parallelism of their final implementation targets.

A. Generic computation structure

After selection of candidates among all operators of the algorithm graph for partial reconfiguration or parametrization, we have a set of operators that must be implemented into a run-time reconfigurable device. To achieve the design generation a generic computation structure is employed. This structure is based on buffer merging technique and functionality abstraction of operators. The aim is to obtain a single computation structure able to perform through run-time reconfiguration or parametrization the same functionalities as a static solution composed of several operators. Encapsulation of operators through a standard interface allows us to obtain a generic interface access. This encapsulation eases IP integration process with this design methodology and provides functionality abstraction to operators. This last point is helpful to easily manage reconfigurable operators with run-time reconfiguration or configuration for parametrized operators. This encapsulation is suitable for coarse-grained operators with data-flow computation. It is a conventional interface composed of an input data stream, an output data stream along with 'enable' and 'ready' signals to control computation. For the case of parametrized operators a special input is used to select the configuration of the operator. As operators can work on various data widths, the resulting operator interface has to be scaled for the worst case.

B. Design generation for run-time reconfiguration management

In order to perform reconfiguration of the dynamic part we have chosen to divide this process in two sub-parts: a Configuration manager and a Protocol builder. The 'Configuration manager' is automatically generated from our libraries according to the sequencing of operations expressed in the macro-code. A 'Configuration manager' is attached to each parametrizable operator or dynamic operator. The configuration manager is in charge of operation selection which must be executed by the configurable operator by sending configuration requests. These requests are sent only when an operation has completed its computation and if a different operation has to be executed after. So reconfigurations are performed as soon as the current operation is completed to enable configuration prefetching as described before. This functionality provides also information on the current state of the operator. This is useful to start operator computations (with signal 'enable') only when the configuration process is ended. Figure 2 shows a simple example based on two operations (j and k) which are executed successively. Labels M and P show where functionalities 'Configuration manager' and 'Protocol builder' respectively are implemented. Case a) shows the design generated for a non-reconfigurable component. The two operators are physically implemented and are static. Case b) is based on a parametrizable operator, the selection of configurations is managed by the configuration manager. There is no configuration latency, the operator is immediately available for computation. The signal Config Select is basically a request of configuration, it results in the selection of a set of parameter among all which are internally stored. The third case c) is based on a dynamically reconfigurable operator which implements successively the two operations thanks to run-time reconfiguration. The reconfigurable part provides a virtual hardware, so at a time only one operator is physically implemented on this dynamic part. The configuration requests are sent to the protocol builder which is in charge to construct a valid reconfiguration stream in agreement with the used protocol mode (e.g. selectmap).

During reconfiguration process (parametrization or partial reconfiguration) other communications or computations are allowed. Encapsulation of operators with a standard interface allows to reconfigure only the area containing the operator without altering the design around. Buffers and functionalities involved in the overall control of the dynamic area remain on a static part of the circuit. This partitioning allows to reduce the size of the bitstream which must be loaded and decreases the time needed to reconfigure.

![Diagram](image.png)

**Fig. 2.** Architecture comparison between a fixed/parameterized or dynamic computation based structure
IV. RESULTS

Our design flow and methodology have been applied for the implementation of a transmitter for future wireless networks in a 4G-based air interface [11]. In an advanced wireless application, SDR does not just transmit. It receives informations from the channels networks, probes the propagation channel and configures the systems to achieve best performance and respond to various constraints such as bit error rate (BER) or power consumption. Hence we have considered a configurable transmitter which can switch between three transmission schemes. The basic transmission scheme is a multi-carrier modulation based on Orthogonal Frequency Division Multiplexing (OFDM). OFDM is used in many communications systems such as: ADSL, Wireless LAN 802.11, DAB/DVB or PLC. The first scheme corresponds to the most simple transmitter configuration named OFDM. The second configuration uses a Multi-Carrier Code Division Multiple Access (MC-CDMA) technique [11]. This multiple access scheme combines OFDM with spreading allowing the support of multiple users at the same time. The third configuration uses a Spread-Spectrum Multi-Carrier Multiple-Access with frequency hopping pattern (FHSS-MC-MA), as used in 802.11b (WiFi). It is a Spread Spectrum Modulation technique where signal is repeatedly switching frequencies during radio communication, minimizing probability of jamming/detection. SynDEx algorithm graph, depicted by Figure 3 A), shows the numeric computation blocks of this configurable multimode transmitter. These three transmission schemes use channel coding and perform a forward correction error (FEC), corresponding to the Channel coding block. The DSP can change the FEC to select a Reed-Solomon encoder or a convolutional encoder. Next a modulation block performs an adaptive modulation between QPSK, QAM-16 or QAM-64 modulation. For MC-CDMA and FH-SS-MC-MA schemes a Spreading block implements a Fast Hadamard Transform (FHT). This block is inactive for OFDM scheme. A chip mapping (Chip mapping block) is done in order to take into account the frequency diversity offered by OFDM modulation. This block performs either an interleaving on OFMD symbols for MC-CDMA, whereas the interleaving in FH-SS-MC-MA scheme is a frequency hopping pattern (FH) to allow user data to take advantage of the diversity in time and frequency. The OFDM modulation is performed by an Inverse Fast Fourier Transform thanks to IFFT block which also implements zero-padding process. For complexity and power consumption this IFFT can be implemented in radix-2 (IFFT-R2) or radix-4 (IFFT-R4) mode. Configuration selection (conditional entry Config) and data generation are handle by the Data Gen block, whereas CNA If block represents the interface to the CNA device of the platform.

A. Implementation on a prototyping platform

This board is composed of one DSP C6701 from Texas Instrument and one partially reconfigurable FPGA Xc2v2000 from Xilinx (10700 CLB slices). Communications between DSP and FPGA are ensured by SHB (Sundance High-speed Bus) and CP (Communication Port) communication medium from Sundance technology. We have chosen to divide the FPGA in four vertices. One is static (Interface) and represents pre-developed logic for FPGA interfacing. The three remaining (FPGA_Dyn, FPGA_Dyn_1 and FPGA_Dyn_2) are runtime reconfigurable vertices. Internal communications between these parts are ensured by the LI media. Table I details the configurations and complexities of the reconfigurable transmitter computational blocks (depending on the transmission schemes). These complexities are obtained on a Xilinx VirtexII FPGA, where each slice includes two 4-input function generators (LUT). Some of these functions can be implemented thanks to available Xilinx IPs [12].

1) Functions mapping: From the characterization of computational blocks we can determine a possible implementation of the transmitter over the prototyping board. Table II summarizes this mapping. IFFT block will be implemented thanks to a parameterizable IP from Xilinx, and mapped...
Transmission schemes: OFDM (A), MC-CDMA (B), FH-SS-MC-MA (C)

### TABLE I
**Configurations and complexities**

<table>
<thead>
<tr>
<th>Transmitter configuration</th>
<th>Computational blocks</th>
<th>Channel coding</th>
<th>Modulation</th>
<th>Spreading</th>
<th>Chip mapping</th>
<th>IFFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>PAD Map: 12 slices.</td>
<td>C</td>
<td></td>
<td></td>
<td>C Parameterizable operator: 1600 slices</td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- Sampling frequency = 20Mhz
- Number of users = 32
- FFT = 256 points
- OFDM symbol duration = 12.8 us
- Frame duration = 1.3 ms

### TABLE II
**Functional blocks mapping**

<table>
<thead>
<tr>
<th>Functional Block</th>
<th>FPGA_Interface</th>
<th>FPGA_Dyn</th>
<th>FPGA_Dyn_1</th>
<th>FPGA_Dyn_2</th>
<th>DSP C87</th>
<th>CNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data_Gen (1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNA_if (1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFFT (2)</td>
<td>Reed Solomon Encoder(1)</td>
<td>Convolutional Encoder (1)</td>
<td>Chip Mapping (3)</td>
<td>FHT (1) Modulation (2)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- (1) : Static operator
- (2) : Parameterizable operator
- (3) : dynamic operator
- Design automatically generated

The code, both for fixed and dynamic parts, has been automatically generated with SynDEEx thanks to the libraries. However, the generation of bitstreams needs a specific flow from Xilinx called modular design [7]. Modular Design is based on a design partitioning in functional modules which are separately implemented and allocated to a specific FPGA’s area. Each module is synthesized to produce a netlist and then placed and routed separately. Reconfigurable modules communicate with the other ones, both fixed and reconfigurable, through a special bus macro (3-state buffers) which is static. They guarantee that each time a partial reconfiguration is performed the routing channels between modules remain unchanged. Partial bitstreams are created from the individual modules designs. The Virtex II integrates the ICAP FPGA primitive. ICAP is an acronym for Internal Configuration Access Port providing a direct access to the FPGA configuration memory and so enables a self partial reconfiguration. Figure 4 shows the resulting design of the reconfigurable transmitter which is compliant with the modular design flow for partial reconfiguration. The non reconfigurable part of the FPGA is composed of four areas resulting of the design generation of the four architecture graph vertices. Reconfigurable vertices architectures are detailed. Each are composed of the general computation/communication controller with buffers. Parameterizable and run-time reconfigurable operators have their own configuration manager. The Dynamic operator configuration manager can address reconfiguration requests to the protocol builder. The protocol builder performs partial reconfigurations thanks to the ICAP primitive and bitstreams stored in external memory (Interleaving and Frequency hopping bitstreams). Only the left FPGA side is a run-time reconfigurable area and implements the dynamic operator. Parameterizable and run-time reconfigurable operators have their own configuration manager. The Dynamic operator configuration manager can address reconfiguration requests to the protocol builder. The protocol builder performs partial reconfigurations thanks to the ICAP primitive and bitstreams stored in external memory (Interleaving and Frequency hopping bitstreams). Dynamic operator encapsulated signals are accessed through bus macros as circumscribed by the Modular Design flow. The size of the reconfigurable area has to be scaled to the most demanding function in logical resources, here FH function (246 slices, 2 BlockRam). Besides the shape of the reconfigurable area is constrained by the Modular Design and leads to allocating a greater area size than really necessary. In this case the area takes the full FPGA height and 12 slices width (1300 slices). This area is the only run-time reconfigured, other areas remain unchanged and are defined...
once during the full FPGA configuration.

3) Numerical results of implementation: Reconfiguration operates at 50Mhz. The first and full configuration of the device takes 16 ms while the partial reconfiguration process of chip mapping functionality (operator Op Dyn) is about 2ms. That is of the order of several data frames, hence partial reconfiguration is suitable for a transmission scheme switching, as in the case of chip mapping functionality which is changed for MC-CDMA and FH-SS-MC-MA schemes. On the other hand, partial reconfiguration is too time consuming to be used for intra transmission scheme reconfiguration, it is the case if the channel coding and IFFT are implemented on the same dynamic operator. As shown in Table III, FPGA resources usage needed to implement the operator logic controls are more important with a dynamic reconfiguration implementation scheme (107+550=657 slices) as for a static and manual implementation (200 slices). The overhead is about of 450 slices to allow run-time reconfiguration of chip mapping functionality. This overhead is due to the generic VHDL structure generation, based on the macro code description. However this gap is decreasing with a greater number of configurations implemented on the dynamic operator. The aim is to take advantage of the virtual hardware ability of the architecture. However the flexibility given by this methodology and the automatic VHDL generation can overcome this hardware resource overhead. For instance we can add a Turbo convolutional encoder for the channel coding block (1133 slices, 6 BlockRam - IP Xilinx 3GPP Compliant Turbo Convolutional Codec v1.0). As the size of the reconfigurable part is fixed by the designer, any design able to be satisfied with this area constraint can be implemented.

B. Implementation based on a Network On Chip

Network on Chip (NoC) is a new concept developed since several years and intended to improve the flexibility of IP communications and the reuse of intellectual properties (IP) blocks. NoCs provide designers with a systematic, flexible and scalable framework to manage communications between a large set of IP blocks. It can also reduce IP connection wires and optimises their usage. The dynamic reconfigurability of communication paths responds to the fluctuating processing needs of embedded systems. Dataflow IPs can be connected either through point to point links or through a NoC. Tools have been proposed in order to design and customize NoCs according to their application needs. We have developed both a NoC and its corresponding tool. This NoC is one possible target of the presented methodology. This NoC is adapted and optimised to allow the plug and the management of dynamically reconfigurable IPs. Reconfigurability is one important source of flexibility when combined with a flexible communication mechanism.

1) MicroSpider NoC presentation: Our NoC [13] is built with routing nodes using a wormhole packet switching technique to carry messages. Operators are linked to the routing nodes through Network Interfaces (NI) with a FIFO-like protocol. Our NoC is customizable through an associated CAD tool [13]. Our CAD Tool is a decision and synthesis tool to help designers to obtain the had-hoc NoC depending on the application and implementation constraints. It is able to configure the various functionality of our NoC. Finally, this
TABLE III
STATIC-DYNAMIC IMPLEMENTATION COMPARISON

<table>
<thead>
<tr>
<th>Chip mapping implementation</th>
<th>Manual (all static)</th>
<th>Automatic (SynDEX) with run-time reconfiguration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Controls</td>
<td>IPs (Interleaving+FH)</td>
</tr>
<tr>
<td>CLB Slices :</td>
<td>200</td>
<td>186 + 246 = 432</td>
</tr>
<tr>
<td>Block RAM (18Kbits) :</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>FPGA area :</td>
<td>1.8 %</td>
<td>4 %</td>
</tr>
<tr>
<td>Switching latency :</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

2) Specific features: The number of ports of a router is configurable. By this way the network topology is flexible and there is no unnecessary port created. We have opted for source routing instead of a distributed one to avoid full rule table distribution and to facilitate possible run-time routing configuration. Finally, the routing technique can be simplified if the NoC topology is regular. However, it can be useful to have a routing technique that can be independent from the topology. So, a choice between two routing techniques is available. The first technique is called "dimension-ordered routing". It can be used only in the case of a regular n-dimension mesh topology. In dimension-ordered routing, each packet is routed in one dimension at a time. The second technique, called "street-sign" allows the NoC to be independent from its topology. But it imposes at source interface some tables containing paths to the destinations. Virtual channels (VC): Virtual channels are used to carry best effort (BE) and guaranteed traffic (GT). The designer can choose the number of virtual channels he wants in the NoC as well as the arbitration technique. Virtual channels make possible the separation between different traffics according to their priority despite the use of a common physical medium. In our case, priority level corresponds to its VC number. For each virtual channel, it is possible to configure independently from other virtual channels: the buffer depth, the routing technique, the arbitration technique, and the flow control technique. Network interfaces: Network interfaces are flexible and configurable to be adapted with the connected processing elements. They implement the network protocol. NIs connect IP blocks to the NoC. For NoC standardisation reasons, we made the choice of the OCP interface [14] for the communication between NI and IPs. NI uses a table to transform OCP addresses map in packet header routing information according to the NoC configuration. The network interface architecture is strongly related to SynDEx scheduling technique that requires a virtual channel controller and buffering capabilities.

3) Implementation: The operations from the application example (Figure 3 A)) are data-flow operations. Data-flow operators have FIFO like protocols. We have added specific features to our NoC in order to optimise the data-flow traffic and the plug of IPs. We have also standardized the interfaces of the NoC. A subset of OCP interface standard has been selected and implemented. We have implemented the previous application on a six-node NoC (Figure 5) integrated in the same FPGA, with one reconfigurable area per NoC routing node, and one node dedicated to a bridge to the external C6701 DSP. Each of the five remaining nodes can be the target of any application task. Several tasks can be grouped on the same node either to be dynamically reconfigured, parametrized or fixed and simply scheduled in time. We have evaluated latency and throughput of the unloaded NoC links. These figures are introduced in SynDEsX heuristic. Table IV shows the functions mapping on this NoC. As SynDEsX schedules transfers in time, we use virtual channels in order to guarantee priority...
of first scheduled transfers. Thus the latency is deterministic and accurate. The M4 code generated by our methodology provides all the scheduling of treatments and communications as well as the source and target of each communication. These informations are extracted and translated to a C code [15] for a Xilinx Picoblaze micro-controller in charge of the dynamic operators and parametrizable operators. Figure 6 details a NoC node structure. There is one Picoblaze for each NoC interface linked to dynamically reconfigurable operators. The Picoblaze controls the scheduling of operators, the size, the target and the starting of data transfers from the running operator. A NoC IP is the grouping of the operators, the picoblaze processor and the operators control and OCP adaptation logic. The scheduling of communications is managed with virtual channels in the NoC interfaces. They are configured by the Picoblaze. Implementation results are presented in table V. The NoC cost is similar to the point to point solution with all the advantages of flexibility and scalability. With this solution there is no need to design a dedicated architecture graph for each new application. One general purpose 2-D mesh can be selected for the architecture graph. Also, the coupling of a NoC with dynamically reconfigurable operators allow a new level of flexibility and optimisation.

V. CONCLUSION

We have designed and presented a flexible hardware platform for partial dynamic reconfiguration based on a heterogeneous NoC targeted to FPGA. We have integrated the dynamic reconfiguration manager directly in the NoC nodes in order to make the reconfigurable parts manageable like software tasks on a processor. We have used a methodology flow to manage automatically the partially reconfigurable parts of the FPGA. It allows to map applications over heterogeneous architectures and fully exploit advantages given by partially reconfigurable components. This design flow has the main advantage to target software components as well as hardware components to implement complex applications from a high-level functional description. This methodology is independent of the final implementation of the run-time reconfiguration which is device dependent and achieved with back-end tools. This modelling can be applied on various components of different granularities. This methodology can easily be used to introduce dynamic reconfiguration on pre-developed fixed design as well as for fast IP block integration on fixed or reconfigurable architectures. This top-down design approach makes it possible to accurately evaluate system implementation, according to functions complexity and architecture properties. Besides, the benefits of this approach fit into the SoftWare Radio requirements for efficient design methods, and adds more flexibility and adaptation capacities through partial run-time reconfiguration on FPGA-based systems.

REFERENCES


<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
<th>N5</th>
<th>N6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turbo encoder (1)</td>
<td>Reed Solomon encoder(3)</td>
<td>Modulation (2)</td>
<td>Spreading (1)</td>
<td>Chip mapping (3)</td>
<td>IFFT (2)</td>
</tr>
</tbody>
</table>

TABLE IV

1: STATIC OPERATOR, 2: PARAMETERIZABLE OPERATOR, 3: DYNAMIC OPERATOR APPLICATION FUNCTION MAPPING ON THE NOC

<table>
<thead>
<tr>
<th>IP</th>
<th>Nb Slices</th>
<th>Freq (MHz)</th>
<th>Nb BRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicoBlaze</td>
<td>110</td>
<td>200</td>
<td>1</td>
</tr>
<tr>
<td>NoC Node</td>
<td>430</td>
<td>200</td>
<td>2</td>
</tr>
</tbody>
</table>

TABLE V

NOC NODE RESOURCES USAGE


An ultra-low voltage high gain operational transconductance amplifier for biomedical applications

Farid Bautista*, Sergio O. Martínez*, Graciano Dieck* and Olivier Rossetto†

∗Electrical and Computer Engineering
Instituto Tecnológico y de Estudios Superiores de Monterrey, Monterrey México
Email: {farid, smart, graciano.dieck.assad}@itesm.mx
†LPSC, UJF Grenoble, CNRS/IN2P3, IPNG, Grenoble, France
Email: rossetto@lpsc.in2p3.fr

Abstract—A novel differential-input single-output Operational Transconductance Amplifier (OTA) is presented in this paper. The topology proposed consists of an input stage based on a folded cascaded amplifier, and an output stage based on a current source amplifier and a bulk-driven current mirror. The simulations show that the amplifier has a 1.94 µW power dissipation, 92 dB open-loop DC gain, a unit gain-bandwidth of 390 kHz, a low noise between 537 Hz to 390 kHz and operates at 0.5V rail-to-rail supply voltage. It was designed for a 0.35 µm CMOS process. The OTA’s performance satisfies the required parameters for its implementation in biomedical portable devices.

I. INTRODUCTION

THE TRENDS of scaling down the channel length in CMOS technology and emergence of portable devices such as Ambulatory Brain Computer Interface (ABCI) systems, insulin pumps, hearing aids and mobile communications require to develop circuits that work at ultra low voltage power supply. Moreover, low power dissipation is essential in these systems to have longer battery lifetime and is even more critical in wireless and batteryless systems.

Most of the biomedical portable devices monitor patients all day long. Therefore, such devices must have low power dissipation; typical values in novel devices are around 40 µW up to 60 µW. In addition, the amplification stage in their analog-front-end must have enough gain and low-noise in the base band to amplify the biomedical signal (0.05 Hz to 500 Hz and 5 µV-5 mV).

Different techniques have been developed for low-voltage operation such as, charge-pump, low \( V_{TH} \), bulk-forward biased, bulk-driven and devices working in weak inversion. However, charge pump technique is not a true low voltage and the fabrication cost with low \( V_{TH} \) process is high. Amplifiers with bulk as input and supply voltages down to 0.8V have been reported in [1], [2], [3] and [4]; and recently a bulk-driven amplifier working in subthreshold region with 0.5V power supply has been reported in [5]. In addition, a gate-driven amplifier with 0.5V rail-to-rail, 62 dB DC gain and 75 µW power dissipation is also proposed in [5].

Differential pairs are commonly used as input stages, in an ultra-low voltage OTAs the tail current must be removed in order to have more voltage headroom [6]. However, a Common Mode Feedback (CMFB) must be added to improve the Common Mode (CM) operation and CM rejection ratio (CMRR). The configurations reported in the literature are based on current source amplifiers (CSA). This paper proposes a novel topology working with ±0.25V supply voltage. The first stage is similar to the folded cascode (FC) OTA and the output stage is a CSA to achieve greater output swing.

II. SUBTHRESHOLD OPERATION

When the \( V_{GS} \) in the MOS transistor is less than the threshold voltage (\( V_T \)), the device works in subthreshold region or weak inversion region. In subthreshold region, the \( I_D \) curve changes from quadratic behavior to exponential behavior. The current \( I_{DS} \) in weak inversion region is given by [7]:

\[
I_{DS} = I_{DO} \frac{W}{L} e^{V_G/nU_T} (e^{-V_S/U_T} - e^{-V_D/U_T})
\]

where \( I_{DO} \) is the characteristic current, \( n \) is the slope factor, \( U_T \) is the thermal voltage \( (kT/q) \), approximately 25 mV at room temperature. The above equation is also applicable for p-channel transistors by changing the signs of \( V_G, V_S \) and \( V_D \).

By definition, the gate transconductance can be found from equation 1:

\[
g_{md} = \frac{\partial I_{DS}}{\partial V_{GS}} = \frac{1}{nU_T} I_{DS}
\]

The small signal source conductance can be found from equation 1 as:

\[
g_{ds} = \frac{\partial I_{DS}}{\partial V_{DS}} = \frac{1}{U_T} I_{DO} \frac{W}{L} e^{V_G/nU_T} e^{-V_S/U_T} e^{-V_D/U_T}
\]
When \( V_{DS} \) is less than \( 3U_T \) the linearity is poor, on the other hand when \( V_{DS} \gg 3U_T \) the conductance is almost constant [8].

III. OTA DESIGN

The proposed OTA consist in two stages. A configuration similar to classical FC was chosen for the first stage and a CSA like for the output stage. The whole circuit is shown in the figure 1.

A. First OTA stage

The main difference between the classical FC configuration and the proposed configuration is that the former has a stack of 4 transistors at its output (figure 2), to obtain a high output resistance. The proposed topology only has a stack of 3 transistors. Assuming that all the devices are working in saturation (\( V_{DS_{sat}} \geq 0.1V \)), and using 4 transistors at ultra-low voltage supply produce an extremely reduced output swing. Hence, to increase the output swing and have relatively higher gains, 3 transistors are stacked. Nevertheless, an extra stage is needed to obtain better output swing.

A differential NMOS pair is used as input (M1a and M1b), the bulks of the transistors are forward biased (tied to \( V_{DD} \)) to reduce the \( V_{TH} \) and increase the inversion level [5]. In order to maintain the input transistors on, the required \( V_{cm,in} \) is \( V_{DD}/2 \). The \( V_{cm,o} \) is set to \( V_{DD}/2 \) in order to have maximum output swing.

Transistors M3a and M3b constitute two symmetric common gate amplifiers (CGA). The current sources \( M_{2a} \) and \( M_{2b} \) provide the current bias to the CGA and also are the current sources for differential input. Both CGAs have active current sources composed by \( M_{4a} \) and \( M_{4b} \). Bulks of \( M_2 \) to \( M_4 \) are connected to the gate in order to reduce the \( V_{TH} \).

Considering that \( I_{tail} \) has been removed from the design in order to provide voltage headroom, devices \( M_{5a}, M_{5b} \) and capacitors \( C_{c1a}, C_{c1b} \) are added to perform CM operation and achieve good CMRR. In addition, the output common level \( (V_{cm,o}) \) is fixed through \( M_5 \) bulks.

The DC gain of this stage is given by:

\[
A_{v1} = \frac{g_{m1}(g_{ds3} + g_{m3} + g_{mb3})}{g_{ds1}(g_{ds1} + g_{ds2}) + g_{ds4}(g_{ds3} + g_{m3} + g_{mb3})}
\]  

(4)

By decreasing the current in the CGA, the output resistance increases, enhancing the gain of this stage, \( A_{v1} \). However, the unit gain-bandwidth (GBW) is reduced. The simulation shows a 49.6dB gain for the first stage.

The input referred thermal and flicker noise obtained for this stage is:

\[
\overline{n}_n^2 = 2\overline{n}_1^2 + 2\left(\frac{g_{m2}}{g_{m1}}\right)^2\overline{n}_2^2 + 2\left(\frac{g_{m4}}{g_{m1}}\right)^2\overline{n}_4^2
\]  

(5)
\[ V_{n_i}^2 = \frac{8kT}{3g_{m1}} + \frac{KF}{C_{ox}^2 W_1 L_1 f} \]  \hspace{1cm} (6)

where \( k = 1.38 \times 10^{-23} \text{J/K} \) is the Boltzman constant, \( T \) is the temperature, \( KF \) is the flicker noise process dependent constant, \( C_{ox} \) is the oxide capacitance, \( f \) is the frequency, \( W \) is the channel width and \( L \) is the channel length. The first expression of the equation 6 represents the thermal noise while the second term corresponds to the flicker noise.

In the design \( g_{m1} \) is set larger than \( g_{m2} \) and \( g_{m4} \), in order to reduce the total input referred noise. Only the noise of the first stage is computed, since the gain is high enough to overcome the equivalent noise of the output stage.

### B. Output OTA stage design

A second stage is in cascade with the first stage and is used to achieve higher gains while maximizing the output swing.

For single output, devices \( M_{7a} \) and \( M_{7b} \) are used to make the conversion from differential to single output. Bulks of both devices are tied together performing a bulk driven mirror. Gates of both transistors are connected to \( V_{SS} \) to keep them on. The bulk of the devices \( M_b \) fixes the output common mode operation to \( V_{DD}/2 \). Figure 3 shows the diagram of the bulk-driven mirror (BDCM) working at subthreshold region.

\( I_{OUT} \) is obtained knowing that \( V_{GB7a} = V_{GB7b} \), then:

\[ I_{OUT} = I_{IN}(W/L)_{7b}(e^{V_{SB7a}/UT} - e^{V_{DB7a}/UT}) \]

\[ \frac{W/L}{7a(e^{V_{SB7a}/UT} - 1)} \]  \hspace{1cm} (7)

If the aspect ratios of the transistors \( M_7 \) are the same, equation 7 becomes:

\[ I_{OUT} = I_{IN}(e^{V_{SB7}/UT} - e^{V_{DB7}/UT}) \]

\[ (e^{V_{SB7}/UT} - 1) \]  \hspace{1cm} (8)

Also, it is easy to see that if \( V_{DB} \) is positive \( I_{OUT} \) is slightly greater than \( I_{IN} \).

Figure 4 shows the behavior of the BDCM when a sweep of \( V_{IN} \) changes the input current.

The gain expression of this stage is given by:

\[ A_{v2} = \frac{g_{m6}}{g_{ds6} + g_{ds7}} \left( 1 + \frac{g_{mb7}}{g_{ds6} + g_{ds7} + g_{mb7}} \right) \]  \hspace{1cm} (9)

the obtained numerical value of this stage was \( 42.4dB \). The whole amplifier gain is \( 92dB \).

### IV. SIMULATION

For the simulation of the proposed OTA, BSIM3 models of 0.35\( \mu m \) CMOS process were used. The simulation was done under typical conditions, the circuit was fed with a source of \( \pm 0.25V \) and loaded with a resistance and a capacitance of 5\( M\Omega \) and 2\( pF \). The transistor’s aspect ratios and the capacitor’s values are summarized in table I.

<table>
<thead>
<tr>
<th>TABLE I</th>
<th>TRANSISTORS ASPECT RATIOS AND CAPACITORS VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Stage</td>
<td>Output Stage</td>
</tr>
<tr>
<td>Transistor</td>
<td>Length</td>
</tr>
<tr>
<td>( M_1 )</td>
<td>500( \mu m )</td>
</tr>
<tr>
<td>( M_2 )</td>
<td>450( \mu m )</td>
</tr>
<tr>
<td>( M_3 )</td>
<td>250( \mu m )</td>
</tr>
<tr>
<td>( M_4 )</td>
<td>75( \mu m )</td>
</tr>
<tr>
<td>( M_5 )</td>
<td>22.2( \mu m )</td>
</tr>
<tr>
<td>Capacitors</td>
<td> </td>
</tr>
<tr>
<td>( C)</td>
<td>1( P )</td>
</tr>
<tr>
<td>( C_{e2} )</td>
<td>5( P )</td>
</tr>
<tr>
<td>( C_{e1} )</td>
<td>20( P )</td>
</tr>
</tbody>
</table>

The OTA has an open-loop DC gain of 92\( dB \), a GBW of 390.1\( KHz \), a PM of 57\(^o\) (figure 5) and dissipate 1.94\( \mu W \). The circuit has better CMRR at low frequencies (see table II).
TABLE II
SUMMARIZED RESULTS

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Open-loop DC gain</td>
<td>92 dB</td>
<td>72 dB</td>
<td>55 dB</td>
</tr>
<tr>
<td>Gain Band Width</td>
<td>390.1 KHz</td>
<td>15 MHz</td>
<td>8.72 MHz</td>
</tr>
<tr>
<td>Phase Margin</td>
<td>57°</td>
<td>60°</td>
<td>61°</td>
</tr>
<tr>
<td>CMRR @10 Hz</td>
<td>48 dB</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>CMRR @100 Hz</td>
<td>45 dB</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Input ref. Noise @10 KHz</td>
<td>42 nV/√Hz</td>
<td>120nV/√Hz</td>
<td>N/A</td>
</tr>
<tr>
<td>Input ref. Noise @4 KHz</td>
<td>50 nV/√Hz</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Offset</td>
<td>2.6 mV</td>
<td>2 mV</td>
<td>N/A</td>
</tr>
<tr>
<td>Power Consumption</td>
<td>1.94 μW</td>
<td>100 μW</td>
<td>77 μW</td>
</tr>
</tbody>
</table>

Fig. 5. Open loop frequency response of the OTA

The noise behavior is shown in figure 6, where the flicker noise decreases from $583 nV/\sqrt{Hz}$ at 1 Hz to $60 nV/\sqrt{Hz}$ at 537 Hz. This later frequency is approximately the noise frequency corner.

Fig. 6. Input referred noise of the OTA

V. DISCUSSION

Comparing the simulation results with previous reported work [5] [6], our proposed design has larger open-loop DC gain and dissipate less power (1.94 μW). The input-referred noise is less than the reported in [5]. The circuit present the smallest GBW. However, since several biomedical signals have frequencies much less than 390.1 KHz, we were not concerned in achieve a high GBW.

The proposed OTA has the advantage of high DC gain and low-noise while dissipating low-power.

The overall results are summarized in table II.

VI. CONCLUSIONS

A differential-input single-output OTA that operates with ±0.25V rail-to-rail has been proposed in this paper. The OTA achieves high gain (92 dB), low noise in the 538 Hz–390.1 KHz frequency band ($\leq 60 nV/\sqrt{Hz}$) and a CMRR of 48 dB@10 Hz. The CMRR can be improved by adding a CM network like the one proposed in [5].

The OTA’s performance satisfies the required parameters for its implementation in biomedical portable devices.

REFERENCES

Automatic Code generation for Interconnected distributed RAM in the AAA Methodology: H264 Motion Estimation Case Study

A. Maccari¹, F. Urban², M. Raulet¹, J-F. Nezan¹ (1) IETR CNRS UMR 6164 Image and Remote Sensing Group INSA, 20, av des buttes de coesmes Mraulet,jnezan@insa-rennes.fr

(2) THOMSON RD FRANCE, Video Compression Lab 1 av. de belle fontaine, CS 17616 35576 Cesson Sévigné CEDEX, France fabrice.urban@thomson.net

Abstract
New video compression standard achieves high compression rates at the cost of a high encoder complexity. High definition context magnifies the difficulty of a real-time embedded implementation. Programmable multicomponent architectures can provide suitable target solutions combining flexibility and computation power. New architectures are designed to use up to 70 processors in a single equipment to address such an application. The result is that a fast and automatic prototyping methodology cannot be ignored for an optimized application development. This paper presents the methodology to automatically generate a distributed implementation of an application over such a multiprocessor platform from a high level application description. The methodology is adapted to handle interconnected distributed RAM media and illustrated in the context of a H264 coding application. The encoder performances are greatly linked to the motion estimation which can reach 60% of the computation load. A multiprocessor implantation of a motion estimation operation is presented.

I. INTRODUCTION
New embedded multimedia systems require more and more computation power. They are increasingly complex to design and have a shorter time to market. Computation limits of systems (i.e. video processing, telecommunication physical layer) are often overcome thanks to specific circuits. Nevertheless, this solution is not compatible with short time designs or the system’s growing need for reprogramming and future capacity improvements. An alternative can be provided by programmable software (DSP, RISC, CISC processors) or programmable hardware (FPGA) components since they are more flexible. The parallel aspect of multicomponent architectures and possibly its heterogeneity (different component types) raise new problems in terms of application distribution: handmade data transfers and synchronizations quickly become very complex and result in lost time and potential deadlocks. A suitable design process solution consists of using a rapid prototyping methodology. Results are obtained here with the AAA methodology and the SynDEx tool that supports it. The aim is then to go from a high level description of the application to its real-time implementation on target architecture as automatically as possible. The aim is to avoid disruptions in the design process from a validated system at simulation level (mono-processor) to its implementation on a heterogeneous multicomponent target.

The AAA methodology relies on two description models which must match the application behaviour and the architecture. The model used for the application description is a DFG (Data Flow Graphs) which have proven to be an efficient representation model. The model used for the architecture allows describing heterogeneous multicomponent architectures (processors and media) in a single graph. Two types of media can be described in SynDEx: RAM (Random Access Memory) and SAM (Single Access Memory) models. Nevertheless, many new target platforms are made of several processors with their own private memory, and a specific hardware is used to realise data transfers between those memories. This memory model can be described as Interconnected Distributed RAM.

In this article, we will focus on the way to use interconnected distributed RAM in SynDEx. An automatic code generation is provided for automatic code generation in this context. Results are illustrated with an H264 motion estimation algorithm onto one computer and five DSP.

The paper is organized as follows: section 2 introduces the SynDEx tool and the AAA methodology. The prototyping platforms and the executive kernels are described in section 3. Application implementations according to the AAA methodology are explained in section 4. Finally conclusions are given in section 5.

II. SYNDEX OVERVIEW
SynDEx is a rapid prototyping system level CAD tool developed in INRIA Rocquencourt, France. It is based on the AAA methodology (Adequation Algorithm Architecture) for distributed real-time processing. Its goal is to go from a high level description of the application (Algorithm) to its real-time distributed implementation on a multicomponent platform (Architecture). Figure 1 describes the design process using SynDEx from the description graphs to the compilable source codes which will be loaded in the component targets.
A. Adequation Algorithm Architecture methodology

The aim of AAA methodology is to find the best matching between an algorithm specifying the application to perform and a multicomponent architecture. Besides, real-time constraints for embedded targets must be satisfied. AAA methodology is based on graph theory to model the software application and the hardware architecture. Both, software and hardware are described by two distinct graphs (Fig. 1).

1) Algorithm graph

The software algorithm is described by a data flow graph in which each vertex corresponds to an operation of the algorithm and each edge represents a data transfer between operations. Thus two operations which are not in data-dependence relation may be executed simultaneously by two different processors. Thus, DFG shows the potential parallelism of an algorithm.

2) Architecture graph

The architecture is defined in a second graph where each vertex is a processor and each edge a communication media. Thus, the graph shows the architecture potential parallelism. In the Architecture graph each processor is modelled by one operator and as many communicators as connected media, in Figure 1 processors are connected with two communicators Com1 and Com2 of different types SAM and RAM. This approach parallelises computations and data transfers. Operators can execute operations which are parts of the algorithm while communicators execute communication operations when data transfers are required. Operator and communicator are connected together by a shared memory inside a same processor.

3) Adequation

“Adequation” means efficient mapping, and consists in exploring graphs and exploiting their parallelisms through the implementation solution with optimisation heuristics. Those heuristics aim at minimizing the total execution time of the algorithm running on heterogeneous multicomponent architecture, taking the execution time of operations and the communication bandwidth between processors into account.

B. Executive generation

1) Generic executives

SynDEx deals with both optimised distribution (allocating parts of the algorithm on components) and scheduling (giving a total order for the operations distributed onto a component) for a defined algorithm on a specific architecture. A generic Synchronised Distributed Executive (SynDEx) is generated from the Adequation in the form of several source files, one for each processor. Those files are in macro-code (i.e. independent of the processor language). They include memory allocations, calculation and communication scheduler, and inter-processor synchronizations. The generated executives can be seen as a static offline operating system.

2) Executive kernels

These macro-coded files will automatically be transformed into compilable language (i.e. C code for PC and DSP, VHDL for FPGA). This is done through the M4 macro processor using code generation kernels (library files). M4 replaces all the macro-calls by their corresponding definitions given in the executive kernels, which depend on target processor and/or communication medium.

Several kernels have already been developed for processors such as GPP (General Purpose Processors), Texas Instruments TMS320C6x (C62x, C64x) and Virtex FPGA families [4]. The executives are thus automatically generated; the user concentrates only on application related code as complex multiprocessor programming is handled by the tools. Memory allocations, operation scheduling, inter-processor communications and synchronisations are automatically inserted. Adequation and code generation avoid deadlocks in the communication scheme and ensure processing safety. It allows eliminating part of the tests, decreasing the development lifecycle.

C. Communication models

We have seen that the AAA methodology allows in an architecture graph to connect processors with several communicators of different types. Two different communication models are possible: the Single Access Memory (SAM), and the Random Access Memory (RAM).

The SAM model is a FIFO in which data are added in the queue by the producer, if not full, and removed from the queue by the receiver, if not empty. Synchronization signals between the two processors are assumed to be completely hardware; they are not handled by SynDEx.

The RAM model is a shared memory allocated for several processors where data can be read in a different order they have been written. All the inter-processors synchronization signals, describes in Figure 2, are handled by SynDEx.

This scheme is effective but tends not to match hardware architectures anymore: many new platforms do not hold one large memory shared among all processors but each processor owns its private memory. In this case, a specific hardware component manages data transfers between private memories. Next sections describe a new approach of the
shared memory closer to this hardware reality and the modified RAM model called interconnected distributed RAM model.

![Fig. 2 Basic synchronization sequence between two processors](image)

### III. INTERCONNECTED DISTRIBUTED RAM

AAA RAM model is used for adequation and generic executives’ generation in SynDEx. To match the reality of interconnected distributed RAM, kernels have been developed to reinterpret these generic executives in order to adapt them to this context.

In comparison with executives for a shared memory among processors, our final executive has to manage data transfers between private memories. It includes memory management (data allocation and transfers) and semaphore management (synchronizations).

#### A. Memory management

Each processor private memory is divided into several parts: one is still used for local data when others are divided into banks. Banks are specifically designed for one buffer transfer between processors, managing them properly is the preliminary step before synchronizing transfers.

![Fig. 3 Example of memory management](image)

A bank aims to contain a buffer ready to be sent or received between several processors. Figure 3 illustrates three of them: point-to-point transfers are realized for Buffer2 and Buffer3 and a broadcast configuration is illustrated with Buffer1. Thus, the memory mapping is identical on each processor private memory on each side of the medium.

The fact that a memory space is allocated on each processor ensures the data to be sent as soon as the buffer is ready. So the transfer is no longer dependent from a unique shared memory to be available, used by any other processor transfer. Thus, there is time loss by polling. Broadcasting data to several destinations is now possible and the synchronisation becomes easier avoiding the data to be crushed.

#### B. Semaphore management

Figure 2 shows that at least two semaphores are needed to synchronize a transfer in a AAA RAM model:

- “RAM empty”: the data has been read and the shared memory is free to be filled again.
- “RAM full”: the data has not been read yet and the memory can not be filled.

These two semaphores are always followed by an action; Write follows a RAM full signal and Read follows a RAM empty, so it is possible to simplify the sequence supplanting this two semaphores by a unique semaphore linked to the buffer reducing the amount of transfer on the medium. A semaphore number is placed in the first address of each buffer (Fig. 4).

For example in Figure 3, the Processor3 would be pending to Semaphore3 associate to Buffer3 from Processor1. Processor1 will post in one unique transfer the semaphore and the data (avoiding the medium to be busy a second time). Receiving the data, Processor3 reads the first address of the buffer and posts the semaphore launching the reading action.

![Fig. 4 Example of semaphore allocations](image)

#### C. Synchronisation sequence

From the two previous points we can now detailed a full transfer sequence (in bank 3) illustrated in Figure 5.

Let assume that Buffer3 from Processor3 is empty and that the operator of Processor1 has data ready to be sent. The communicator of Processor1 will fill the Buffer3 with the Semaphore3 at its first address, followed by data. The full buffer will be sent to the Buffer3 of Processor3. After what, the communicator will be pending for new data from the operator and Semaphore3 from the Processor3 to restart a new transfer.

The communicator of Processor3 will receive the buffer, copy the data into another place in local memory and inform...
its operator that new data have come. Next, it will post back in direction of Processor1 the first line of the buffer, which contains the Semaphore3, in order to restart the full cycle.

This model matches the hardware reality and reduces the use of the medium: transfer synchronisations are simplified, a better parallelisation of communication and computation tasks is allowed, and the latency of the scheme is reduced at the cost of a larger allocation of memory resource.

![Enhanced RAM model](image)

### IV. DEVELOPED KERNEL

The platform used in this work is a desktop computer with a multi-DSP board. The VP3-pmC multi-DSP platform from Vitec multimedia (Fig. 6) has been chosen as target to develop a specific SynDEx medium kernel in order to validate the model.

The VP3-pmC is a powerful parallel programmable processing platform dedicated to professional and industrial video applications like: AVC / H.264 real time encoders, MPEG-2 to H.264 / AVC transcoders, image processing... The platform contains 5 DSP TMS320DM642 running at 720-MHz clock rate, each of them with a SDRAM of 32 Mbytes and a FPGA hardware co-processor to manage the communications between all the components implanted on the platform and the communications with the PCI-Host. Up to 14 boards of this kind can be plugged together allowing 70 DSPs to compute and communicate different parts of an algorithm in parallel.

![Data transfer controllers on the platform](image)

#### A. Data transfers on the platform

All transfers between DSPs on the 64bits bus are managed by five DMA controllers inside the FPGA. Transfers between DSPs and PC are also managed by the FPGA. They allow each DSP to be aware of data recorded in their respective SDRAM thanks to an interruption uploading the first address of the data in memory. DSPs are also informed by an interruption from the FPGA when a transfer is finished.

#### B. Preliminary test

We have tested on different Architecture an application named “TestCom” which aims to test the reliability of a medium executing a great quantity of transfers from one to 1000 integers in direction of all the processors defined in the SynDex architecture graph. Stars and rings architecture have been used to test the communications between the five DSPs of the board and the PCI host.

#### C. Motion vector estimation application

A motion estimation application has been prototyped on a heterogeneous multiprocessor platform to illustrate previously described work. Motion estimation is a key operation in video compression, and is also the most computationally intensive part of a video encoder, reaching up to 60% of the computational load of an MPEG4 AVC encoder. Its main challenging properties are variable block size, quarter-pixel accuracy and multiple reference frames [1].

The application is composed of an acquisition module, a motion estimator and a display module to visualise motion fields. The acquisition module is a raw video file reading on the PC, and the display module is in charge of computing an image from motion fields and display on the computer screen.

To handle large displacements in high definition video sequences, a multi-resolution technique called HME [3] has been adopted. The motion is first estimated on a coarse resolution frame, and then successively refined through increasing resolution levels. The full resolution level performs variable block size motion estimation and quarter-pixel refinement. Table 1 summarises execution times on a single DSP at 720 MHz. Hierarchical level operations comprise the construction of reduced resolution frames and computation of motion fields. The full resolution level is the most complex.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchical levels</td>
<td>14 ms</td>
</tr>
<tr>
<td>Full res. Level</td>
<td>60 ms</td>
</tr>
</tbody>
</table>

Table 1: Motion estimation timings for a 720p frame (1280x720)

The total time of motion estimation on a single DSP is 74 ms (85 ms with communication time), which is too long for real-time. To reduce execution time, the application can be parallelized and distributed on a multiprocessor platform.

To reach 30 frames per second, 4 processors are needed. Firstly, hierarchical levels and full resolution level operations can be pipelined, and secondly, the full resolution level operation can be performed in three stripes (or slices). The complete application also comprises video acquisition
(file reading on hard drive) and motion fields visualisation on the computer. Thus, for each frame 1MB luminance image is sent to the DSP board and 250KB motion vector data is received. Figure 7 shows the distribution and scheduling performed by the prototyping tool. The prototype computes motion fields of a 720p sequence (1280x720 pixels) at 33 frames per second on 4 DSPs (compared to 11 fps on a single DSP). An acceleration factor of three is achieved.

Fig. 7: Parallel multiprocessor distribution and scheduling

The application is easily parallelised on the multiprocessor platform thanks to the prototyping tool. Temporal and spatial parallelism was used with pipeline and stripe abilities of the tool.

Complexity of this application gives us the opportunity to achieve full tests of our kernel. The algorithm has first been validated on a PC before been implemented on the Vitec’s board with an increased number of DSP involved in the computation, from one to five. Executives for each processor (PC and DSPs) have been automatically generated by SynDEx and the kernels according to the architecture.

V. CONCLUSION

Interconnected distributed RAM is a topology commonly adopted for complex multiprocessor architectures instead of a shared memory where one unique large memory is connected to all processors. The consequence of interconnected distributed RAM based architectures is a better use of hardware resources (bus). This involves more possibilities for algorithm distribution on processors.

AAA methodology models RAM and SAM communication links. The RAM model is suitable for shared memory based architectures and SAM for FIFO transfers based ones. In this paper we have presented a new approach to handle interconnected distributed RAM in the AAA methodology. We define the rules of memory management and a simplest scheme of transfer synchronisation in order to minimize the use of the bus and to improve the implementation of complex algorithm onto multicomponent systems.

The SynDEx kernel, (i.e. automatic C code generation) designed for this communication model is now currently used in the laboratory on several VP3-pmC boards for video application and image processing such as AVC decoder, LAR coder or motion vector estimation. The kernel allows automatic code generation and leads both to an optimal use of our platform resources, and to reduce development times.

Because this model is not incorporated in the adequation step, the adequation have to be constrained to reach good accelerations. Future work would be to include the interconnected distributed RAM model in the adequation phase of SynDEx in order to reduce the development cycle.

REFERENCES

On Modeling the RapidIO communication link using the AAA Methodology

Sylvain Moreau*, Slaheddine Aridhi**, Mickaël Raulet*, Jean-François Nezan*

* IETR CNRS UMR 6164
Image and Remote Sensing Group, INSA of Rennes
symoreau@ens.insa-rennes.fr, mraulet@insa-rennes.fr, jnezan@insa-rennes.fr

** CIV Texas Instruments France
saridhi@ti.com

Abstract—Embedded real-time applications in communication systems have significant time constraints, thus requiring high processing power. One time manual scheduling developed for single-processor applications is not suited for multiprocessor architectures: manual data transfers and synchronizations quickly become very complex, leading to wasted time and potential deadlocks. We aim to develop a fast and automatic prototyping process optimized for parallel architectures particularly suited to static executives. The process is based on AAA methodology which improves algorithm implementation on multiprocessor architectures by finding the best match between algorithms and architectures. This paper presents an automatic code generation approach for architectures using the RapidIO medium. A new communication model which optimizes data transfers in the generated code is depicted. Finally, we demonstrate the effectiveness of the approach with a Locally Adaptive Resolution (LAR) coding (PC-driven) application deployed on a multi-DSP platform.

I. INTRODUCTION

The recent evolution of digital communication systems (voice, data and video) has been dramatic. Over the last two decades, low data-rate systems (such as dial-up modems, first and second generation cellular systems, 802.11 Wireless local area networks) have been replaced or augmented by systems capable of data rates of several Mbps, supporting multimedia applications (such as DSL, cable modems, 802.11b/a/g/n wireless local area networks, 3G, WiMax and ultra-wideband personal area networks).

As communication systems have evolved, the resulting increase in data rates has necessitated a higher system algorithmic complexity. A more complex system requires greater flexibility in order to function with different protocols in different environments. Additionally, there is an increased need for the system to support multiple interfaces and multicomponent devices. Consequently, this requires the optimization of device parameters over varying constraints, such as performance, area and power. Achieving this device optimization requires a good understanding of the application complexity and the choice of an appropriate architecture to support this application.

The task of mapping an algorithm to a multicomponent architecture is not straightforward. When performed manually, the result is inevitably a sub-optimal solution. This raises the need for new methodologies, which allow the exploration of several solutions, to achieve a more optimal result.

This article describes the Algorithm-Architecture Adequation (AAA) methodology for fast prototyping. Subsequently, details are presented on the use of this methodology to model the Serial RapidIO (SRIO) medium in SynDex to attain the theoretical SRIO bandwidth while exploring complex topologies. The article concludes with the validation results of the SynDEx-modified RapidIO kernel performed with Locally Adaptive Resolution (LAR) coding application.

II. AAA FAST PROTOTYPING METHODOLOGY

The aim of the Algorithm-Architecture Adequation (AAA) methodology is to find the best match (or adequation) between an algorithm which specifies an application and a multicomponent architecture [1]. This methodology is based on graph theory and is used to model the software application and the hardware architecture. Both the software and the hardware are described by distinct graphs. AAA methodology transforms these two graphs in order to find an optimized implementation that satisfies real-time constraints. Once an optimized implementation is derived, a generic script is automatically generated for each processor.
SynDEx (Synchronized Distributed Executive) is a system-level CAD software designed at the INRIA Rocquencourt Research Unit (France). This tool supports the AAA methodology for both rapid prototyping and optimized implementation of distributed real-time embedded applications onto multicomponent architectures [2).

SynDEx is a framework (Fig.1) for exploring various implementation solutions using heuristics. The heuristics aim to minimize the total execution time of an algorithm running on a multicomponent architecture, based on the execution time of individual operations and data transfers between these operations. SynDEx also provides a simulated timing graph for the distributed application. Thus, SynDEx may be used as a virtual prototyping tool.

A. Adequation

An application may be modeled by a Data Flow Graph (DFG) which is an oriented hyper-graph. Each vertex corresponds to an algorithm operation and each edge represents a data transfer between operations. Thus a DFG shows the potential parallelism of an algorithm [1].

An architecture may be modeled by a non-oriented hyper-graph, in which each vertex is a processor (component) and each hyper-edge represents a communication medium. In this model, a processor consists of one operator and as many communicators as there are connected media. An operator executes operations which are a part of the algorithm and a communicator executes a communication operation when a data transfer is required. In this way, a multicomponent architecture may be represented by a network of Finite State Machines (FSM) interconnected with communication media (FIFO, or shared memories). In Fig.2, each processor contains three FSM (in this figure, red indicates calculation functionality, while blue indicates communication functionality).

B. Generic Synchronized Distributed scripts

Once an optimized implementation has been determined, generic executives are automatically generated for each processor. An executive consists of a list containing M4 macro-instructions which specify memory allocation, communication sequence(s) and one computation sequence. Executives are generic because they are independent of the programming language.

Fig.3 describes two generic executives in a Petri network representation. In this representation, each executive is made up of two threads (the operator or processing thread, and the communicator or communication thread). Processor1 executes the operation OP1. The result of OP1 is sent from Processor1 to Processor2 via a FIFO link. Processor2 then executes the OP2 operation. Send and Receive operations are handled by the communication threads in each executive, and are synchronized with respect to each other by the FIFO. Synchronizations between threads are realized with semaphores. The result is the automatic generation of two
synchronized distributed executives without deadlocks.

C. Translation: Automatic Executive Generation

Generic executives automatically generated by SynDEx tool are translated into the appropriate language, as specified by the different target operators (Matlab/C for cosimulation, C or assembler for DSP, VHDL for FPGA etc.) [3]. Translation is handled by M4 macro-processor.

Inputs/outputs of the above translation operation are illustrated in Fig.1. The macro-processor transforms the list of M4 macro-instructions contained in generic executive into code for a specific processor target. Each macro-instruction is then replaced by its definition, as given in the corresponding library (also called kernel) which is dependent on a processor target and/or a communication medium.

Fig.4 illustrates the typical kernels used to translate the macro-instructions. One of these kernels is generic and does not depend on the algorithm. This “Generic Kernel” supports the architecture specifications such as memory allocation, sequence synchronization and inter-operator transfers. In addition to the Generic Kernel, “Application Dependent” kernels are also required, and may be further divided depending on their function. They are used to describe the algorithm specifications such as function-calls. “Architecture Dependent” Kernels include the translation of both processor and communication media, and are divided into two subtypes. “Processor Type Dependent” Kernels contain macro-instructions relating to computational tasks such as memory allocation or interrupt handling; “Media Type Dependent” Kernels contain macro-instructions relating to data transfer and synchronization.

In the prototyping context described in [3], most of the kernels are developed in C language since this permits reuse for any C-programmable device. Indeed, kernels are quite similar for the host (PC) and the embedded processors (DSP). Additionally, the difference between an executive generated from C language code and one generated from assembly instructions is minor.

In AAA methodology, two different models are used to represent the communication media: the SAM (Sequential Access Memory) and the RAM (Random Access Memory). The SAM model defines a basic FIFO-like protocol between two processors or virtual processors. The two processors are considered to be synchronized by means of hardware signals. The data is sequentially pushed by the producer if FIFO is not full and is used by the receiver if FIFO is not empty. This model requires data to be received in the same order as it is sent. Conversely, the RAM model works with an undefined number of processors through a shared memory with controlled access. The read and write order need not be respected in the RAM model; the data transfer management is consequently more complex.

The majority of communication kernels proposed are designed using the SAM model. Specifically, libraries for traditional media such as TCP and FIFOs have been created for multi-PC and multicomponent transfers.

III. RAPIDIO AUTOMATIC CODE GENERATION

To date, processors from different manufacturers have not been developed to communicate with each other. The result is that multi-component platforms for embedded applications require additional logic to make data transfers possible. This additional logic differs from one platform to another, so that the modification of communication kernels is always necessary in the design process.

To address this issue, the RapidIO [4] standard has been developed by a group of embedded platform development leaders and is now an open communication standard for embedded multicomponent platforms. A RapidIO kernel allows automatic code generation and will address all next generation embedded platforms with this medium. SRIO is not a simple FIFO or shared memory but a complete protocol designed to add scalable connectivity and control among processing components in embedded systems.

The goal of this paper is to model the SRIO medium in SynDEx to attain the theoretical SRIO bandwidth while exploring complex topologies.

A. Equipment and Topology

The TC16482DSK with mezzanine platform was chosen to validate the automatic executive generation. Each of two boards contains a single DSP: the TMS320TCI6482 recently developed by Texas Instruments operating at 1.0 GHz. This DSP has a C64x+ core based on TI’s advanced Very Long Instruction Word (VLIW) architecture, which can perform up to eight 32-bit instructions each cycle.

The two DSPs are connected to each other through a
RapidIO link (4 x 3.125 Gbps). This stand-alone TI platform is connected to an Ethernet network, which allows TCP/IP (100/1000 Mbps) communications between the DSPs and any computer. Fig.5 illustrates the topology of this platform.

![Fig. 5: Architecture example](image)

**B. RapidIO Kernel Development**

The complexity of the RapidIO technology necessitates certain choices in the kernel development, in order to optimize the algorithm latency and respect real time constraints.

For example, as an endpoint device, the peripheral accepts packets based on the destinationID (DestID). Two options exist for packet acceptance and both are mode selectable. The first option is to only accept packets whose DestIDs match that of the local deviceID. The second option is a system multicast operation. During the initialization step, the master allocates the slave identifiers, then synchronizes the different processors using handshakes before beginning the processing stage [5].

SRIO transactions are based on request and response packets. These packets are the communication element between endpoint devices in the system. A master or initiator generates a request packet which is transmitted to a target. The target then generates a response packet, which is then sent back to the initiator to complete the transaction.

For this kernel, the packet type NWRITE was chosen to optimize the transfer throughputs, as it does not require an acknowledgement. In this way, the available bandwidth is maximized for payload data. Additionally, a 4-lane single-port configuration of the SRIO was chosen, rather than four single-lane ports in order to simplify the peripheral management while still maximizing the bandwidth. Furthermore, streaming transfer mode was employed, permitting the simple RapidIO transfer payload limitation of 4kbytes to be bypassed. This strategy uses the DMA to successively send requests to the SRIO peripheral that initiates the data transfers. Therefore, to complete data transmissions between processors, the DMA and SRIO modules run in parallel with the CPU, which releases computing cycles for data processing. A simplification of the SRIO communication model in SynDEx is thus possible when receiving data in a communication thread.

**C. Model Optimization**

The existing communication model in SynDEx has been improved to fit the RapidIO technology, consistent with the assumptions presented in this paper.

The SynDEx SAM communication model, presented Fig.3, and the SynDEx RAM model are both based on a classic view of communications in multiprocessor platforms, where memory is shared (Fig.6).

![Fig. 6: Classic communication model](image)

Conversely, RapidIO is designed to be an externally driven slave module, capable of acting as a master in the DSP system. This capability means that an external device can push data to the DSP as needed, without generating an interrupt to the CPU and without relying on the DSP EDMA. Thus, the shared memory used to transfer the data between processors in the SynDEx communication model can be removed (Fig.7). This new model allows buffers to be exchanged directly from one local memory to another. The result is a reduction of both memory resources and transfer latency.

![Fig. 7: DirectIO memory model](image)

Consequently, the SynDEx Petri network (Fig.3) was updated, leading to the new representation presented Fig.8. As can be seen by the new Petri network representation, the producer (which performs OP1 operation in the processing thread of Processor1) executes the data computations before the communicator (responsible for operations in the Communication Thread of Processor1) executes the Send operation. The data is directly sent into the local memory of the second processor. Then, the processing thread of the Processor2 is informed that a new data stream has arrived and can directly be consumed in the OP2 operation.

![Fig. 8: Petri network of the new model developed](image)
This new memory model has been integrated as a communication model outside the SynDEx tool. It has been implemented as part of the macrocode translation, and is presented here in the RapidIO kernel. To use this new model, a SynDEx user just has to create a classic SAM medium in its architecture graph and called SRIO. The macrocode translation using the SRIO kernel will automatically create communications based on the new model.

For the implementation of the RapidIO model, the direct I/O transfer mode was selected. With direct I/O, the RapidIO packet contains the specific address where the data is stored or read in the destination device. Direct I/O requires that a RapidIO source device maintains a local table listing memory addresses within the destination device. Once a table is created, the RapidIO source controller uses this data to select the destination address, which is then inserted into the packet header. The RapidIO destination peripheral extracts the destination address from the received packet header and transfers the payload to memory via the DMA. Additionally, to send data from memory to an external Processing Element (PE) or read data from an external PE, the CPU provides vital information to the RapidIO peripheral about the transfer such as DSP memory address, target device ID, target destination address, packet priority, and so on.

Inter-processor synchronizations are based on doorbell messages. The doorbell operation is used by a PE to send a very short message to another PE through the RapidIO peripheral. The doorbell transaction contains an information field which holds the semaphore number of the buffer requiring synchronization and does not contain a data payload. The local processor reads the queue to determine the transmitting PE and the information field, and determine the subsequent action required. The doorbell functionality is user-defined; in the model, this packet type is used to initiate DSP core (CPU) interrupts. In this way, the interrupt service routine fetches the info field of the received doorbell, then sends the appropriate semaphore, allowing synchronization of the transfer, and then clears the pending interrupt flag.

With this new model, the number of semaphores, the generated code length and the memory required are all minimized compared to the previous SynDEx communication model. The RapidIO kernel enables the automatic code generation of the implementation of this optimized model. Furthermore, as with other SynDEx kernels, deadlocks are avoided with this generated code.

IV. LAR IMAGE CODER PROTOTYPING

The validation of the SynDEx-modified RapidIO kernel was performed with Locally Adaptive Resolution (LAR) coded application.

LAR is a compression and decompression image processing algorithm developed in IETR laboratory [6]. This algorithm adapts the local resolution (pixel size) according to the luminance uniformity. Thus, for uniform luminance, the image resolution will be low (block 8×8); a high resolution image (block 2×2) will be produced when the activity is high. The encoder has already been implemented in SynDEx and optimized for DSP platforms [7]. The SynDEx algorithm graph is made of 79 operations.

In Video applications, the processing time is critical and must be less than 40ms per frame (video latency constraint). For this reason, CIF images (352*288 pixels) have been chosen.

Once the latency of every operation has been measured and integrated in SynDEx, the tool gives an estimated scheduling as well as an estimate of the total processing latency of the application mapped onto multicomponent architecture. SynDEx estimates the algorithm latency at 24.1 ms for an architecture represented by one PC and one DSP. When another DSP is added to the platform the latency estimated goes down to 18.0 ms. The gain factor estimated is 1.34 but could reach 2 when the application DFG have more potential parallelism. A second reason why the gain factor is not very high comes from the TCP link. This communication medium, which is much slower than the RapidIO link, causes a bottleneck.

Such a complex application combines TCP and SRIO bidirectional transfers to verify that no deadlock left. To implement the LAR algorithm onto the TCI6482DSK with mezzanine platform linked to a PC, SynDEx schedules 6 TCP transfers and 24 SRIO transfers.

A first functional check of the still image LAR coder was realized on a PC and runs in 17 ms per CIF image. As the generated code is totally portable on DSPs (due to use of C language), this coder has been automatically implemented onto multiprocessors (PC + multi-DSPs). A new SynDEx graph describing the PC and the two DSPs as well as the TCP and RapidIO links is created. The use of SynDEx and the new kernel allows the automatic code generation. In this system, the PC checks the functional description, and then transmits the data corresponding to the image to be coded. The PC then receives the data from the platform in order to display the reconstructed image. 24.2 ms are needed to encode an image with one DSP, and 18.7 ms with two DSPs. Therefore, the gain factor is 1.29 given that the theoretical maximum rate given by SynDEx could be 1.34.

To execute the application onto a parallel architecture made up of a PC and two DSPs, SynDEx estimates the CPU cycles needed at 18 000 000. The execution time really measured is 18.7 ms at 1 GHz i.e. 18 700 000 cycles. So, the difference between the estimated latency and the execution reaches
SynDEx supposes that the transfers are entirely achieved in parallel of the computations. Actually, the CPU needs a period of time to setup the data transaction. By adding these setup times to the estimation, the gap between the estimation and the execution decreases to 3%. Another difference between estimations and real timings lies in the use of a Real-Time Operating System (RTOS: DSP/BIOS) which is not taken into account in the SynDEx adequation (inline scheduling of computation and communication tasks and relating context changes).

V. CONCLUSION AND PERSPECTIVES

The design approach proposed in this paper covers major steps, from simulation to integration in digital signal application development. Compared with a manual approach, the use of the AAA methodology ensures easy design reuse, fast prototyping, efficiency and portability. Therefore, this concept improves design security, flexibility and reduces time to market. Several complex tasks are performed automatically, such as distribution/scheduling, code generation of data transfers and synchronizations. Once the kernels have been adapted to the platform, the development of a new application is limited to the algorithm description.

The kernel developed automatically implements an optimum use of the RapidIO peripheral. It has been specifically created to be compatible with the TI’s TMS320TCl6482 RapidIO technology. The RapidIO peripheral is an open communication standard, which enables the model extrapolation. Consequently, the RapidIO communication model could be extended to a DSP farm or a heterogeneous architecture including DSPs and FPGA.

REFERENCES


Configurable Instruction Architecture for Q/IQ on the Altera Nios II Processor

A. Ben Atitallah\(^{(1,2)}\), P. Kadionik\(^{(2)}\), N. Masmoudi\(^{(1)}\), H. Levi\(^{(2)}\)

\(^{(1)}\) Laboratory of Electronics and Information Technology
National Engineers School of Sfax (E.N.I.S.), BP W 3038 Sfax - TUNISIA
Nouri.Masmoudi@enis.rnu.tn

\(^{(2)}\) IMS Laboratory - ENSEIRB - University Bordeaux I - CNRS UMR 5818,
351 Cours de la Libération, 33 405 Talence Cedex, FRANCE
{benatita;kadionik,levi}@enseirb.fr

Abstract—In this paper, we present a hardware implementation of the Quantization (Q) and Inverse Quantization (IQ) in FPGA using the Nios II custom instruction architecture. The instruction for the Q/IQ is described in VHDL language and implemented in a Stratix II EP2S60 FPGA. The performances of these instructions have been tested using the H.263 video encoder.

I. INTRODUCTION

In the video coding area, all significant coders employ the quantization. In this paper, we focus on the hardware implementation of the quantization within the H.263 standard frameworks. The H.263 video coding standard targets low bit rate telecommunication applications. The main components of the video coding algorithm are motion estimation, Discrete Cosine Transform (DCT) transformation, quantization and coding.

Early implementations of the H.263 video encoder algorithm involve a full-search motion estimation and DCT algorithm those typically required 95% of the total number of computations. However, many efficient motion estimation and DCT developed algorithms reduce the computational complexity by up to two orders of magnitude [2-7]. When a fast motion estimation algorithm and a hardware DCT/IDCT implementation are used, the Quantization (Q) and Inverse Quantization (IQ) steps become significant in terms of computation [8].

In order to optimize and achieve best performance in the implementation of the H.263 video encoder, we have used hardware implementation in FPGA of the Q/IQ. The Nios II is a RISC processor optimized for an implementation into an Altera’s FPGA. The Nios II has a highly flexible and reconfigurable architecture that allows the designer to add user defined and custom hardware instruction. This instruction is used for hardware implementation of the Q/IQ.

This paper is organized as follows: section 2 describes the direct and inverse quantization method. Section 3 presents the Nios II softcore processor architecture. The hardware implementation of the Q/IQ using Nios II custom instruction is presented in section 4. The evaluation of the H.263 video encoder and the experiment results are presented in section 5. Section 6 concludes the paper.

II. QUANTIZATION AND INVERSE QUANTIZATION

The quantization is the stage where the information losses are introduced in order to achieve compression. The quantization equations are not standardized in the H.263 recommendation. The International Telecommunication Union (ITU) has suggested two quantizers in their Test Model 8 (TMN8) [9] corresponding to INTRA and INTER modes and are given by (1):

\[
LEVEL = \begin{cases} 
\frac{|COF|}{2QP}, & \text{INTRA} \\
\frac{|COF| - QP}{2}, & \text{INTER} 
\end{cases}
\]  \hspace{1cm} (1)

The INTRA DC coefficient is uniformly quantized with a quantized step of 8. The quantization parameter \(QP\) may be an integer value from 1 to 31. \(COF\) stands for a transform coefficient to be quantized. \(LEVEL\) is the quantized version of the transform coefficient that must be in the interval [-127,127]. These equations are useful as a reference and not only because they are commonly used as a reference model, but also because studies performed by the ITU during the standardization process [10] indicated that the quantization equations in (1) were nearly optimal subject to the constraints of uniform scalar quantization with a dead zone. The basic inverse quantization reconstruction rule for all non-zero quantized coefficients is defined by equation 2 which gives the relationship between coefficient levels (LEVEL), quantization parameter (QP) and reconstructed coefficients (REC)
After calculation of $|REC|$, the sign is added to obtain REC:

$$REC = \text{sign}(\text{LEVEL})|REC|$$  \hspace{1cm} (3)

### III. SOFTCORE PROCESSOR

A softcore processor is a microprocessor fully described in software, usually in VHDL/Verilog language, which can be synthesized into a programmable hardware circuit such as a FPGA. Softcore processor implemented in a FPGA can be easily customized to the needs of a specific target application. The two major FPGA manufacturers provide commercial softcore processors. Altera offers its Nios and Nios II processors [11], while Xilinx has Microblaze processor [12]. In this paper, we focus on the Nios II processor.

#### A. Nios II softcore processor

The Nios II is a softcore processor optimized for implementation in Altera FPGA. The Nios II processor (FAST version) is a 32-bit scalar RISC with Harvard architecture, 6 stage pipeline, 1-way direct-mapped 64KB data cache, 1-way direct-mapped 64KB instruction cache and can execute up to 150 MIPS [13].

The main interest of this softcore processor is its extensibility and flexibility. Indeed, users can add custom logic instruction and custom peripherals connected to a dedicated bus (Avalon Bus). Thus, users can define their instructions and processor peripherals to optimize the system for a specific application.

#### B. NIOS II custom instruction logic

With Nios II custom instructions [14], system designers are able to take full advantages of the flexibility of a FPGA to meet system performance requirements. Custom instructions allow system designers to add up to 256 custom functionalities into the Nios II processor ALU (Arithmetic Logic Unit).

### IV. HARDWARE IMPLEMENTATION OF THE Q/IQ

The Quantization and Inverse Quantization equations are a regular formula. In order to reduce time necessary to the Q/IQ processing and improve performance of the H.263 video encoder, we have used multi-cycle Nios II custom instruction logic to implement these equations. The custom instruction interface for the Q/IQ is presented in Fig. 2. This interface has 32-bit input and output data. The Dataa signal has the COF value in the quantization case (LEVEL in the inverse quantization case). On the other hand, the Datab signal has the QP coefficient for the Q/IQ. The Result signal provides the LEVEL value in the quantization case ($REC$ in the inverse quantization case).

#### A. Custom instruction for Q/IQ

Fig. 3 and 4 present the architecture for the hardware implementation of the quantization equation. Fig. 6 describes the hardware architecture for the implementation of the inverse quantification equation. The inputs (Dataa and Datab) and the output (Result) are synchronized by the system clock. The computation result is obtained in two clock cycles.

Both quantization equations have $(2xQP)$ as a divisor. This enables the calculation of the table containing values $1/2xQP$ for each possible QP value. This table makes possible the quantization with multiplication. Because the calculation is performed with a limited precision: the quantization using multiplication does not always yield exactly as accurate
results as the quantization using division. The computed coefficient for \(1/2 \times QP\) is sent to the hardware through the \textit{Datab} signal. This signal is consisted by two parts: a part contains the value of coefficient calculated on 24 bits and another part has the value of the QP on 8 bits. The computed value by the instruction is shifted on the right 16 times in order to obtain the final result. The comparator checks if this result belongs to the interval \([-127,127]\) if not the final result will be -127 or 127.

According to the Fig. 5, we can see that the instruction for the IQ uses a multiplexer. This multiplexer allows to select the output which depends on the first bit of the \textit{Datab} signal (\textit{Datab}(0) = ‘0’ then QP is odd, \textit{Datab}(0) = ‘1’ then QP is even).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig3.png}
\caption{Instruction for the INTRA quantization}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig4.png}
\caption{Instruction for the INTER quantization}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig5.png}
\caption{Instruction for the inverse quantization}
\end{figure}

\textbf{B. FPGA Implementation results}

Table 1 shows the implementation results of the custom instruction for the Q/IQ in the Altera Stratix EP2S60 FPGA circuit which is based on 1.2V, 90 nm technology with a density that reaches 48352 Adaptive Look-Up Tables (ALUTs), 310 KB of Embedded System Blocs (ESBs), 288 DSP blocks and 493 Input/Output Blocks (IOBs).

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
 & \textbf{Q\_INTRA} & \textbf{Q\_INTER} & \textbf{IQ} \\
\hline
\textbf{ALUTs} & 79 (<1\%) & 89 (<1\%) & 160 (<1\%) \\
\hline
\textbf{ESBs} & 0\% & 0\% & 0\% \\
\hline
\textbf{DSPs} & 2 (<1\%) & 2 (<1\%) & 8 (3\%) \\
\hline
\textbf{IOBs} & 100 (20\%) & 100 (20\%) & 100 (20\%) \\
\hline
\textbf{Freq (MHz)} & 120 & 120 & 120 \\
\hline
\end{tabular}
\caption{The implementation results in a Stratix II FPGA}
\end{table}

\textbf{V. EXPERIMENT RESULTS}

In our experiments, the Altera NIOS II development board was chosen [15]. The core of the board is the Altera Stratix II EP2S60F672C3 FPGA. The prototyping board operates at 120 MHz. Once the instruction for the Q/IQ described in VHDL at RTL level are fitted into the FPGA, we have determined the coding time of H.263 video encoder before and after timing optimization. The video encoder was executed under µClinux and performed on the NIOS II softcore processor. For experiments, the QCIF test sequences coded at 10 frames/s with different values of the quantization parameter (QP=8, 13, 31). We focus on the following video test sequences: “Carphone”, “News”, “Claire”, and “Miss America”. Our simulations have shown the average clock cycles of 5700 to code an 8x8 block using a full software solution for the Q/IQ. On the other hand, the average clock cycles are 1200 using a hardware solution. From these results, we can conclude that the hardware solution is 5 times faster than the software solution.

Fig. 6 presents a breakdown of the execution time before and after optimization of the H.263 encoder. The percentage distribution was very similar for all four sequences, so only the results for the Miss America sequence with QP=13 are shown here. However, we can note that the HW/SW implementation of the H.263 provides better results in coding speed compared to the software based solution.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig6.png}
\caption{CPU time percentage according to the processing before and after optimization}
\end{figure}

The results presented in table 2 illustrate the compression quality and the coding speed before and after optimization of the H.263 video encoder. For the quality measurements, we have used the PSNR (Peak Signal to Noise Ratio) which is used to evaluate the quality of a reconstructed image sequence of an image quantitatively and The Structural SIMilarity (SSIM). The SSIM [16] is a method for measuring the similarity between two images based on the characteristics of the human visual system (HVS).

From this table, we can conclude that the video encoder achieved a maximum frame encoding speed of 15 frames/s using hardware implementation of the Q/IQ with a marginal
loss in quality of reconstructed image compared to the software implementation of the Q/IQ.

Table 2. Experimental results for HW/SW implementation of the H.263 video encoder

<table>
<thead>
<tr>
<th>QP</th>
<th>PSNR-Y (dB)</th>
<th>SSIM</th>
<th>Bit Rate (kb/s)</th>
<th>Coding speed (fps)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q/IQ_SW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miss America</td>
<td>8</td>
<td>37.90</td>
<td>0.9943</td>
<td>19.58</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>35.87</td>
<td>0.9909</td>
<td>10.71</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>32.58</td>
<td>0.9803</td>
<td>5.40</td>
</tr>
<tr>
<td>Akiyo</td>
<td>8</td>
<td>35.52</td>
<td>0.996</td>
<td>21.87</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>32.87</td>
<td>0.9926</td>
<td>11.35</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>29</td>
<td>0.9817</td>
<td>4.51</td>
</tr>
<tr>
<td>News</td>
<td>8</td>
<td>33.76</td>
<td>0.9948</td>
<td>53.41</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>30.95</td>
<td>0.99</td>
<td>29.27</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>26.14</td>
<td>0.9692</td>
<td>10.34</td>
</tr>
<tr>
<td>Carphone</td>
<td>8</td>
<td>34.39</td>
<td>0.9963</td>
<td>53.89</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>31.81</td>
<td>0.9938</td>
<td>27.98</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>27.64</td>
<td>0.9834</td>
<td>10.41</td>
</tr>
<tr>
<td></td>
<td>Q/IQ_HW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miss America</td>
<td>8</td>
<td>37.90</td>
<td>0.9943</td>
<td>19.58</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>35.72</td>
<td>0.9902</td>
<td>10.22</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>32.51</td>
<td>0.98</td>
<td>5.35</td>
</tr>
<tr>
<td>Akiyo</td>
<td>8</td>
<td>35.52</td>
<td>0.996</td>
<td>21.87</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>32.71</td>
<td>0.9923</td>
<td>11.01</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>28.91</td>
<td>0.9812</td>
<td>4.42</td>
</tr>
<tr>
<td>News</td>
<td>8</td>
<td>33.76</td>
<td>0.9948</td>
<td>53.41</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>30.79</td>
<td>0.9896</td>
<td>28.25</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>26.08</td>
<td>0.9687</td>
<td>10.13</td>
</tr>
<tr>
<td>Carphone</td>
<td>8</td>
<td>34.39</td>
<td>0.9966</td>
<td>53.89</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>31.65</td>
<td>0.9935</td>
<td>26.75</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>27.57</td>
<td>0.9831</td>
<td>10.30</td>
</tr>
</tbody>
</table>

VI. CONCLUSION

Softcore processors play a key role in building embedded systems on a programmable chip. In this paper, we have presented an efficient hardware implementation of the direct and inverse quantization equation for video encoder using the custom instruction architecture of the Nios II processor.

REFERENCES

A TRIPLE PATH NONLINEAR PHASE FREQUENCY DETECTOR AND RESISTOR SCALAR SCHEMES USED IN PLLS FOR FAST PULL-IN

Minglang Lin, Ahmet T Erdogan, Tughrul Arslan and Adrian Stoica

School of Engineering and Electronics
University of Edinburgh, Edinburgh, EH9 3JL
Email: {m.lin, A.Erdogan, T.Arslan}@ed.ac.uk, Adrian.Stoica@jpl.nasa.gov

Abstract—A fast pull-in and locking PLL with a triple-path nonlinear phase frequency detector (TPNPFD) scheme is presented. The proposed scheme can reduce the pull-in time significantly and speed-up the lock-in process. The charge pump current and the loop filter capacitors are always chosen to $1/k$ of those in [2] regardless of the crossover frequency. Moreover two kinds of resistor scalar schemes are also presented in this novel architecture. Thus the area and power should be substantially reduced in this proposed scheme.

I. INTRODUCTION

Frequency synthesizers (FSs) based on PLLs with a fast pull-in and shortest acquisition time feature are essential blocks in frequency hopping communication systems where fast attaining frequency lock is critical. However, there is a trade-off between lock-in time and output phase noise in these FSs [1]. For achieving lock in a given time constraint, the wider loop bandwidth would make it quicker, but it should induce more phase noise. On the other hand, to minimize the phase noise, the loop bandwidth had better keep as low as possible, but a longer settling time will be expected. To achieve a fast lock-in PLL with a better output phase noise performance, nonlinear phase frequency detector (PFD) schemes are proposed in [2], [3], [4] and [5] and a frequency gear-shifting scheme is presented in [6]. These nonlinear PFD schemes are based on increasing the charge-pump current and/or decreasing the time constant of loop filter to adaptively widen the crossover frequency for speeding up the lock-in process. The adaptive bandwidth modification in frequency gear-shifting schemes is by controlling the pre-scalar counter of the reference signal and the divider of the voltage controlled oscillator (VCO) signal to obtain the widest available bandwidth and to achieve a shortest settling time. Either in nonlinear PFD schemes or in frequency gear-shifting schemes, the widest loop bandwidth needs to be limited under 1/10 of the reference frequency for loops stability consideration [1].

A linear system model of a type-2, 3rd order PLL is shown in Figure 1. The open loop gain function of Figure 1 is given as

$$G(s) \triangleq \frac{\theta_o}{\dot{\theta}_e} = I_P \frac{K_V}{N} \frac{sC_1R_1 + \frac{1}{sC_2}}{s^2C_1C_2R_1 + s(C_1 + C_2)}$$

where $I_P$ is the charge-pump current, $K_V$ is the gain of VCO and $N$ is the frequency division modulus. The time constants which determine the zero and pole frequencies of $G(s)$ can be defined as

$$\tau_Z = R_1C_1,$n

and

$$\tau_P = \frac{R_1C_2}{C_1 + C_2}.$n

Figure 1. Linear system model of type2, 3rd order PLL.

- 256 -
The open loop gain can be calculated in terms of frequency $\omega$, $\tau_Z$, $\tau_P$ and design parameters $I_P, K_V$ and $N$.

$$G(j\omega) = \frac{1}{N} I_P \frac{K_V}{\omega^2} \frac{1}{(C_1 + C_2)} \sqrt{\left(\frac{\omega\tau_Z}{\omega\tau_P}\right)^2 + 1},$$

(4)

and the phase function of $G(s)$ can be defined as

$$\phi(j\omega) = 180^\circ + \tan^{-1}(\omega\tau_Z) - \tan^{-1}(\omega\tau_P).$$

(5)

Therefore the phase margin can be obtained when $|G(j\omega_C)| = 1$,

$$\phi_M(j\omega_C) = \tan^{-1}(\omega\tau_Z) - \tan^{-1}(\omega\tau_P)$$

where $\omega_C$ is a crossover frequency derived from both (4) and (6) for a maximum phase margin

$$\omega_C = \frac{1}{\sqrt{\tau_Z \tau_P}} = \frac{I_P}{N} \frac{K_V}{R_1 C_1} \frac{R_2 C_1}{(C_1 + C_2)}$$

(7)

The relationship between settling time and phase margin was shown in [4] which have provided that the fastest step response of the closed loop function of a 3rd order PLL can be obtained when the phase margin equals to $51^\circ$. From (6), the phase margin of the 3rd order PLL is equal to $53^\circ$, near by $51^\circ$, when $\tau_Z$ is chosen as 9 times of $\tau_P$. In other words, if the ratio of $C_1 / C_2$ has been chosen as eight, a 3rd order PLL system with shortest lock-in time will be obtained.

The triple path nonlinear PFD (TPNPFD) is composed of a frequency detector (FD), a fast lock-in phase detector (FLPD) and a traditional tri-state PFD. The system architecture of TPNPFD and the loop filter parameters selection for fast pull-in with a high spectral purity performance will be described in Section II. Section III gives the principles of two different resistor scalar schemes. The circuit design principle and the implementation of the triple path nonlinear PFD will be explained in Section IV. Section V will illustrate the simulation results.

II. SYSTEM ARCHITECTURE

If the crossover frequency $\omega_C$ were redefined as $k\omega_C$, the step response time would be reduced by approximately one-kth of the original one. From both (6) and (7), there are three options for widening the crossover frequency from $\omega_C$ to $k\omega_C$ while keeping the phase margin unchanged by varying the parameters: $I_P$, $R_k$, $C_1$ and $C_2$ in Figure 1.

Option one: The charge-pump current is set to $k^2 I_P$ and the loop-filter resistor is set to $R_k / k$.

Option two: The charge-pump current and the loop-filter capacitors are set to $kI_P$, $C_1 / k$ and $C_2 / k$ respectively.

Option three: The charge-pump current is kept fixed, the loop resistor is changed to $kR_k$ and the loop-filter capacitors are scaled down to $C_1 / k^2$ and $C_2 / k^2$.

Although all these three options can be used to achieve the same result, for lower power consideration, options two and three are better than option one.

As well known, the online loop filter capacitors occupy significant area of PLL chips, therefore, if these capacitors in conventional PLLs could be always set to $C_1 / k$ and $C_2 / k$ while PLLs operate in $k\omega_C$ (wider-BW) mode and in $\omega_C$ (narrower-BW) mode, the chip area will be reduced significantly. In wider-BW mode, the bandwidth of the 3rd order PLL is $k\omega_C$, the time constants of pole and zero are

$$\tau_P = R_k C_1 C_2 / (k(C_1 + C_2))$$

and

$$\tau_Z = R_1 C_1 / k,$$

therefore the phase margin $\phi_M(jk\omega_C)$ is still kept the same as (6). However in narrower-BW mode, the bandwidth will have to be set back to $\omega_C$ for lower phase noise. If the loop filter capacitors are still set as $C_1 / k$ and $C_2 / k$ then the loop filter resistor and the charge-pump current will need to be changed to $kR_k$ and $I_p / k$ for keeping the same phase margin as in wider-BW mode.

Our novel TPNPFD PLL architecture is shown in Figure 2a where the switch set (sw1, sw2) is set to be (on, off) while frequency detector (FD) is activated, (off, on) while the system operates in the wider-BW mode and (off, off) when the output phase noise needs to be minimized. Since a resistor $kR_1$ is needed when $C_1 / k$ and $C_2 / k$ capacitors are adopted in narrower-BW mode, this resistor can be easily implemented by serially connecting a resistor $R_1$ and a resistor $(k-1)R_1$. However, more chip area would be needed for this simple serial resistor connection. A capacitance scalar scheme was presented in [7], where a larger capacitance can be obtained by sampling the input current flowing into a smaller capacitor, amplifying the sampled current and feeding back the amplified current to the input end. Therefore in order to remove the area needed for resistor $(k-1)R_1$, a resistor scalar scheme based on the similar concept of [7] and another resistor scalar scheme by using an extra scaling current will be explained in the following section.
Figure 2. Novel triple paths nonlinear PLL architecture. (a) Resistors serial connection, (b) Resistor scalar scheme based on [7], (c) Resistor scalar scheme by using an scaling current

III. RESISTOR SCALAR SCHEMES

The resistor scalar scheme based on [7] and shown in Figure 3a is adopted in Figure 2b to remove resistor $(k - 1)R_1$ in Figure 2a. The circuit implementation and the equivalence small signal model analysis of Figure 3a are shown in Figure 3b and Figure 3c respectively.

By using the typical circuit analysis, the small signal admittance of Figure 3c can be given as below:

$$\frac{i_m}{v_m} = g_{oM7} + g_{oM8} + sC_{p2} + \frac{1}{g_m + kR_1 + sC_{p1} \frac{k}{g_m} + 1 + Rg_m}$$

(8)

where $g_{oM7}$ and $g_{oM8}$ are the output impedance of transistors $M_7$ and $M_8$, $g_m$ is the transconductance of transistor $M_1$, $C_{p1}$ and $C_{p2}$ are the total parasitic capacitance at nodes 1 and 2. For proper operation, the pole $C_{p1}R_1 / (1 + Rg_m)$ and the zero $C_{p1}k / g_m$ can be located at far higher frequencies than the crossover frequency and typically can be neglected. On the other hand, since the cascode structure is adopted in Figure 3b, $g_{oM7}$, $g_{oM8}$ and $C_{p2}$ will be significantly smaller than $1 / ((k / g_m) + kR_1)$ and because of the low impedance of $1 / g_m$, $k / g_m$ will be smaller than $kR_1$. Thus the equivalence impedance can be approximated as

$$\frac{v_m}{i_m} = g_{oM7} + g_{oM8} + sC_{p2} + 1 / ((k / g_m) + kR_1) \approx kR_1$$

(9)

The resistor scalar scheme by using an extra scaling current technique shown in Figure 4 will be adopted in Figure 2c to replace $(k - 1)R_1$ in Figure 2a. There are two current sources, the charge-pump current $I_p$ and the scaling current $(k - 1)I'_p$. The open loop gain function and the phase function of Figure 4 are given as below.
The new zero is still located at but the new pole is moved to . The phase margin can be easily found from the Bode plot of . The Bode diagrams of \( \frac{1}{Z} = \frac{1}{\tau} \) \( \omega = \omega \) \( \phi(j\omega) = 180^\circ + \tan^{-1}(\omega\tau_z) - \tan^{-1}(\omega\tau_p / k) \) (11)

The new zero is still located at \( \omega_Z = 1 / \tau_Z \) but the new pole is moved to \( k\omega_p = k / \tau_p \). The phase margin can be easily found from the Bode plot of \( G(s) \). The Bode diagrams of Figure 2a and Figure 2b while operating in \( \omega_C \) mode are denoted as curve GO_Wc shown in Figure 5. The Bode plot of Figure 2c operating in \( \omega_C \) mode is denoted as curve GON_Wc. Curve GO_kWc denotes that all of figures in Figure 2 operating in \( \omega_C \) mode. The phase margin of both curve GO_Wc and GO_kWc are equal to 53°. The gain and phase of curve GON_Wc equal to 1.0476, very close to one, and −114° at frequency \( \omega_C \). Therefore the phase margin of GON_Wc could be denoted as 66°, slightly larger than 53°, when \( k \) is set to 3. Compared to [2] and [5], the locking time, shown in Figure 9, of Figure 2c is still the shortest one. Therefore, this resistor scalar scheme will be a nice option to save chip area of PLLs without increasing the complexity of circuits.

IV. CIRCUIT DESIGN

The basic operation of TPNPFD can be explained in Figure 6. Initially, there is a frequency difference between \( f_{\text{REF}} \) and \( f_{\text{DIV}} \) if phase error \( |\theta| \) is larger than \( 2\pi / N \) in Figure 6. Therefore, the FD and sw1 in Figure 2a need to be activated and to be switched on to make current \( \pm mkI_I \) directly inject into \( C_1 / k \) and \( C_2 / k \), where \( m \) is a positive integer, e.g., \( m \) will be set to 2 when \( 1 < f_{\text{REF}} / f_{\text{DIV}} \leq 2 \) or \( 1 < f_{\text{DIV}} / f_{\text{REF}} \leq 2 \), and so on. Thus the voltage of the capacitor \( C_2 / k \) controlling VCO should be more quickly adjusted to the desired one. Secondly, when the frequency of \( f_{\text{DIV}} \) has been adjusted to be very close to the one of \( f_{\text{REF}} \) but \( |\theta| \) is on the range of \( \{2\pi / N, 2\pi \} \), where \( N \) is the number of the delay stage used in VCO, the loop bandwidth of the 3rd order PLL system will be changed to \( k\omega_C \) for fast lock-in. Therefore, the block FLPD needs to be enabled, the charge-pump current and sw1 in Figure 2a, Figure 2b and Figure 2c will be set to \( kI_I \) and off respectively. The sw2 in Figure 2a and in Figure 2b will be set to on and off respectively. Finally the loop bandwidth will be switched back to \( \omega_C \) for lower output phase noise requirement when \( |\theta| \) is smaller than \( 2\pi / N \), so (sw1, sw2) in Figure 2a and Figure 2b will be set as (off, off) and (off, on) respectively, the charge-pump current will
be decreased to \( I_p / k \), and only block PFD left operates in this mode.

The topologies in Figure 2b and Figure 2c demonstrate a similar function of Figure 2a. The only difference is a different resistor scalar technique used in Figure 2b and Figure 2c respectively. Figure 2b scales up resistor \( R_1 \) to \( kR_1 \) for replacing \((k-1)R_1\) used in Figure 2a by using a current sampling, amplifying and feeding back way. The phase margin of Figure 2b is almost the same as that of Figure 2a. However Figure 2c injects an extra scaling current into the loop filter resistor \( R_1 \) to generate a new zero and pole pair in narrower-BW mode. Although the phase margin \( \phi_M(\omega_c) \) of Figure 2c is a bit larger than that in Figure 2a and b, the pull-in and lock-in performance is still better than that in [2] and [5].

\[
\begin{align*}
\text{Frequency error (FD activated)} & \quad \theta_1 \\
\text{Narrow-BW (PFD activated)} & \quad -\frac{2\pi}{N} \\
\text{Wider-BW (FLPD activated)} & \quad \frac{2\pi}{N} \\
\end{align*}
\]

Figure 6. Basic operation of triple loops nonlinear PFD

The simplified logic circuits of block PFD, block FLPD and block FD are illustrated in Figure 7a. The fine tuning current \( I_p / k \), coarse tuning current \( kI_p \) and the acquisition current \( mkI_p \) in Figure 2 are controlled by \((U_{PFD}, D_{PFD}), (U_{FLPD}, D_{FLPD})\) and \((U_{FD}, D_{FD})\) in Figure 7a, respectively.

When \( f_{REF} / f_{DIV} > 1 \), there are more than one rising edges of \( f_{REF} \) occurring between two rising edges of \( f_{DIV} \), node \( U_{1X} \) will be set to one at the following rising edge of \( f_{REF} \) until the rising edge of \( f_{DIV} \) comes. Node \( REF_Q \) will be set to one at the rising edge of \( f_{DIV} \) if the signal at \( U_{1X} \) is high. Node \( U_{1X}, U_{2X} \), etc. are used to indicate that the frequency of \( f_{REF} \) is how many times of the one of \( f_{DIV} \), for example \( U_{1X} \) will be set to high if \( f_{REF} / f_{DIV} > 1 \), \( U_{2X} \) will be set to high too if \( f_{REF} / f_{DIV} > 2 \), and so on. Thus node \( U_{FD} \) will be set to high when \( REF_Q \) or one of \( U_{1X} \) is high, where \( x \) is a positive integer. Therefore, block FD will be activated and block FLPD will be disabled by \( DA_{FLPD} \) generated from block FD until the ratio of \( f_{REF} / f_{DIV} \) is very close to one. The timing diagram of \( U_{FD} \) is shown in Figure 7b for different ratios of \( f_{REF} / f_{DIV} \), and vice versa \( D_{FD} \) will operate while \( f_{DIV} / f_{REF} > 1 \).

\[
\begin{align*}
\text{Timing diagram of block FD, (c) Timing diagram of block FLPD and block PFD}
\end{align*}
\]

\[
\begin{align*}
\text{Figure 7. Simplified logic circuit of TLNPFDD. (a) Schematic}
\end{align*}
\]
After the frequency of $f_{DIV}$ has been adjusted to be near by the one of $f_{REF}$. Block FD stops working and block FLPD will start to work if the phase error $|\theta_e|$ between $f_{REF}$ and $f_{DIV}$ is larger than $2\pi / N$ and smaller than $2\pi$. There are two extra signals $f_{DIV,\,lead}$ and $f_{DIV,\,lag}$ from VCO used in block FLPD to judge whether the phase error $|\theta_e|$ is on the range of $\{-2\pi / N, 2\pi / N\}$ or not, where $f_{DIV,\,lead}$ has a phase $2\pi / N$ lead over $f_{DIV}$ and there is a phase lag of $2\pi / N$ for $f_{DIV,\,lag}$. The $U_{FLPD}$ depicted in Figure 7 will be set to high when $2\pi / N < |\theta_e| < 2\pi$ and will be reset to low when $|\theta_e| \leq 2\pi / N$. Finally the block PFD will start to operate after both FD and FLPD stop working, $U_{PFD}$ will be firstly changed to high as the rising edge of $f_{REF}$ leads the one of $f_{DIV}$, and vice versa, $U_{PFD}$ will be firstly changed to high, then both $U_{PFD}$ and $D_{PFD}$ are simultaneously reset back to low when the following rising edge of $f_{DIV}$ or $f_{REF}$ appears. The timing diagram of $U_{PFD}$ is shown in Figure 7c.

V. SIMULATION RESULTS

The PLL with our novel TPNPFD architecture is built and simulated in time domain with Simulink and HSpice. All design parameters used in [2], [5] and our TPNPFD are shown in Table 1. The charge-pump current and the loop filter capacitors used in our TPNPFD scheme are always smaller than the one in DAPD scheme while PLLs operate either in wider-BW mode or in narrower-BW mode. The pull-in and lock-in simulation results of frequency down jump waveform by using the discriminator-aided PFD (DAPD), a capacitor scalar scheme nonlinear PFD (CNPFD) and our TPNPFD with $53^\circ$ phase margin (TPNPFDpM53) and with $66^\circ$ phase margin (TPNPFDpM66) are depicted in Figure 8. It can be found that the pull-in time in TPNPFD is much shorter than the one in DAPD and in CNPFD. The only difference between TPNPFDpM66 and TPNPFDpM53 just happens at the lock-in process. Clearly, the lock-in time of the PLL adopting our TPNPFD is manifestly shorter than the other two.

Lines DAPD and CNPFD are the result of the PFDs with a DAPD scheme [2] and a capacitor scalar scheme [5], lines TPNPFDpMxx are the results of our triple path nonlinear PFD with different phase margins.

A more better performance is demonstrated that a PLL with our TPNPFD scheme whose pull-in time and lock-in time are the shortest.

Figure 9 shows that the charge-pump current of the block PFD in DAPD, in CNPFD and in our TPNPFD architecture with $53^\circ$ phase margin and $66^\circ$ phase margin, the lock-in time in DAPD and in CNPFD is always longer than that in TPNPFD. Although the lock-in time in TPNPFDpM66 is a bit larger than in TPNPFDpM53, however the circuit complexity of TPNPFDpM66 is significantly less than the one of TPNPFDpM53. Therefore the performance of TPNPFD

![Figure 8](image302x541.png)

**Figure 8.** Simulation results of Frequency jump from 1GHz to 450MHz.

![Table 1](image340x442.png)

**Table 1.** Design parameters of different kinds of nonlinear PFD.
with the resistor scalar scheme by using an extra scaling current is still comparable with the one of previous nonlinear PFDs.

VI. CONCLUSIONS

In this paper, a novel triple path nonlinear PFD is proposed to reduce the pull-in time and the settling time of the PLL based frequency synthesizer which is very important for frequency hopping communication system. Compared to previous fast-locking PLLs with different nonlinear PFD schemes, the acquisition time of the PLL with our TNPPFD is significantly decreased while the system stability remains unchanged. Moreover, two resistor scalar schemes are proposed and adopted with our triple path nonlinear PFD, so that the chip area needed by the loop filter resistor and capacitors will be minimized. Therefore, due to both less area and lower charge-pump current, our novel architecture is more attractive for SoC implementations.

Figure 9. Charge-pump current in Wc mode with DAPD, CNPFD, TPNPFDPM53 and TPNPFDPM66

REFERENCES


OFDM/OQAM transceiver implementation

Chawki Sahnine
France Telecom R&D – Orange Labs, Meylan, France
chawki.sahnine@orange-ftgroup.com

Jean-Philippe Javaudin, Gabriel Degoulet, Bruno Jahan
France Telecom R&D – Orange Labs, Cesson-Sévigné, France
{jeanphilippe.javaudin|gabriel.degoulet|bruno.jahan}@orange-ftgroup.com

Abstract— For several decades the OFDM modulation is widely used in radio and wired communications systems. OFDM-based systems have proved their simplicity and robustness to transmissions over multipath channels. Nevertheless this modulation still suffers from drawbacks such as non negligible out-of-band radiations or spectral efficiency loss due to the insertion of a cyclic-prefix (CP-OFDM). An advanced modulation scheme, called OFDM/OQAM with IOTA pulse shaping has been identified as one potential method to enhance the performance of these systems. In this paper, we introduce the OFDM/OQAM modulation from both theoretical and practical point of view. It is shown how OFDM/OQAM can benefits from the implementation simplicity of CP-OFDM.

Index Terms— OFDM, OQAM, IOTA pulse shaping, prototyping.

I. INTRODUCTION

Orthogonal Frequency Division Multiplexing [1] has gained considerable attention in recent years. This technique, specially OFDM quadrature amplitude sub-carrier modulation with guard interval insertion and rectangular prototype function (CP-OFDM), is widely adopted in different communication applications, such as WIMAX technology for Wireless Metropolitan Area Network (WMAN), Wireless Local Area Network (WLAN) standards IEEE 802.11a/g/n or digital broadcasting systems.

An advanced modulation scheme denoted OFDM/OQAM (Offset QAM) with IOTA pulse shaping, is an alternative to classical CP-OFDM modulation, which does not require the use of a guard interval to obtain the same robustness to the multipath effects, and hence leads to a gain in spectral efficiency. This technique is often considered as a potential candidate for Beyond 3G wireless access technologies and was proposed in the 3G Partnership Project Long Term Evolution (3GPP-LTE) and for future Wireless Regional Area Networks (WRAN) IEEE 802.22 standard as an alternative air interface.

In this paper, we first introduce in Section II the mathematical approach of OFDM/OQAM modulation and the impact on the OFDM transceiver architecture compared to CP-OFDM. Then, we shall present an OFDM/OQAM transceiver implementation in Section III and the simulation results in section IV. We conclude in Section V.

II. THE OFDM/OQAM MODULATION

In conventional cyclic prefix OFDM (CP-OFDM) modulation, we transmit complex symbols $s_{m,n}$ over the sub-carriers. The orthogonality in complex domain among these sub-carriers guarantees simple reception. CP-OFDM is robust to frequency-selectivity but the insertion of the cyclic prefix leads to a loss in spectral efficiency. Moreover it suffers from a bad localization in the frequency domain due to the rectangular prototype function. In the following we describe OFDM/OQAM which is an alternative to classical CP-OFDM.

A. OFDM/OQAM Theory

Contrary to CP-OFDM, OFDM/OQAM modulation [2], [3] does not require the use of a cyclic prefix, which leads to a gain in spectral efficiency. Without adding the cyclic prefix redundancy, similar performance can be reached by modulating each sub-carrier by a prototype function. To obtain a sufficient robustness to the channel variations, this prototype function must be very well localized in both time and frequency domains. The localization in time limits inter-symbol interference ISI and the localization in frequency limits inter-carrier interference ICI.

The orthogonality among the sub-carriers must also be maintained after the modulation. It is possible using near optimally localized functions that guarantee orthogonality, but as explained below orthogonality on only real values. An OFDM modulation using these functions is denoted OFDM/OQAM. In OFDM/OQAM, each sub-carrier carries a real-valued symbol $a_{m,n}$ which corresponds to either the real part or the imaginary part of a complex OFDM symbol $s_{m,n}$, where $m$ is the frequency index, and $n$ is the time index. In the following we note $T_0$ the duration of the complex OFDM symbol $s_{m,n}$ with zero-length cyclic prefix, then $T_0 = T_{0}/2$ is the duration of the real OFDM/OQAM symbol $a_{m,n}$. By denoting the inter-carrier spacing we have $T_0 \cdot \Delta F = 1/2$. This means that the density of the sub-carriers in the time-frequency plane is twice greater in OFDM/OQAM than in classical OFDM, with a zero-length cyclic prefix. Table I shows the real and imaginary part repartition of the OQAM symbols. As mentioned above, for comparison purpose with CP-OFDM we often select the same inter-carrier spacing $\Delta F$ for both schemes, thus the symbol duration of OFDM/OQAM is twice smaller than for CP-OFDM.
As the information carried by two real-valued data correspond to the one carried by one complex-valued data, OFDM/OQAM has the same spectral efficiency as classical OFDM with no guard interval.

### TABLE I

<table>
<thead>
<tr>
<th>nT₀ - T₀/2</th>
<th>(2m-1)ΔF</th>
<th>(2m)ΔF</th>
<th>(2m+1)ΔF</th>
</tr>
</thead>
<tbody>
<tr>
<td>nT₀</td>
<td>j s₂₂m,n</td>
<td>s₂₂m,l,n</td>
<td>s₂₂m+1,l,n</td>
</tr>
<tr>
<td>nT₀ + T₀/2</td>
<td>j s₂₂m,n</td>
<td>s₂₂m,n</td>
<td>j s₂₂m+1,n</td>
</tr>
</tbody>
</table>

The OFDM/OQAM transmitted signal can be expressed as follows

\[ s(t) = \sum_{n=-\infty}^{\infty} \sum_{m=0}^{M-1} a_{m,n} e^{j2\pi n f t} f(t - n \tau_0) , \]

(1) where \( M \) is the number of sub-carriers, \( a_{m,n} \) is real-valued symbol transmitted on the \( m^{th} \) sub-carrier at the \( n^{th} \) symbol; \( f(t) \) denotes the real-valued prototype function. Equation (1) can be rewritten in a simpler manner

\[ s(t) = \sum_{n=-\infty}^{\infty} \sum_{m=0}^{M-1} a_{m,n} f_{m,n}(t) , \]

(2) where \( f_{m,n}(t) \) are the shifted versions of \( f(t) \) in time and frequency. Therefore the orthogonality condition among the sub-carriers is

\[ \text{Re} \left( \int f_{m,n}(t) f_{m',n'}^*(t) \, dt \right) = \delta_{m,m'} \delta_{n,n'} , \]

(3)

In case of no channel, the demodulated symbol over the \( m^{th} \) sub-carrier at the \( n^{th} \) instant is

\[ r_{m,n} = \int s(t) f_{m,n}^*(t) \, dt = a_{m,n} + \sum_{(m',n') \neq (m,n)} a_{m',n'} \int f_{m',n'} f_{m,n}^* \, dt + n_{m,n} \]

(4)

According to (3), the right part of (4) is pure imaginary. Thus, by simply considering the real part of \( r_{m,n} \) we can perfectly recover the transmitted symbol.

When passing through radio channel, and after adding noise contribution \( n(t) \), the received signal can be expressed as

\[ s(t) = \sum_{n=-\infty}^{\infty} \sum_{m=0}^{M-1} h_{m,n} a_{m,n} f_{m,n}(t) + n(t) , \]

(5)

In this case, equation (4) becomes

\[ r_{m,n} = \int s(t) f_{m,n}^*(t) \, dt \]

\[ = h_{m,n} a_{m,n} + \sum_{(m',n') \neq (m,n)} h_{m',n'} a_{m',n'} \int f_{m',n'} f_{m,n}^* \, dt + n_{m,n} \]

As the OFDM/OQAM prototype function \( f(t) \) is chosen to be well localized both in time and frequency domains, the intrinsic interference term from (6) only depends on a restricted set of time-frequency positions \((m',n')\) around the considered symbol. Assuming that \( h_{m,n} \) is constant over the summation zone, we can write

\[ r_{m,n} = \int s(t) f_{m,n}^*(t) \, dt \]

\[ = h_{m,n} a_{m,n} + \sum_{(m',n') \neq (m,n)} h_{m',n'} a_{m',n'} \int f_{m',n'} f_{m,n}^* \, dt + n_{m,n} \]

(7)

with a good approximation, after equalization, a simple recuperation of the real part is sufficient. A particular prototype function called IOTA (Isotropic Orthogonal Transform Algorithm) satisfying equation (3) is evaluated in this paper. Regarding the implementation it can be seen in equation (1) that the signal is as for CP-OFDM composed by a sum of functions that are shifted in time and frequency (and a phase term \( f^{m,n} \) also appears) Consequently it is possible as in CP-OFDM to use a FFT algorithm to simplify the computation of the overall signal. The additional complexity lies in the filtering by the prototype function \( f \), as explained below.

### B. Impact of OFDM/OQAM filter on the OFDM transceiver architecture

Filtering algorithm increases the complexity of an advanced OFDM modulator compared to the CP-OFDM modulator with rectangular prototype function. As we said before, IOTA function guarantee only orthogonality on real values and hence the complex QAM data stream must be separated into two real data, real part and imaginary part. So for the same throughput, an OFDM/OQAM modulator must process data twice as fast than classical CP-OFDM modulator. It is important to notice again that the density of the time-frequency frame equals 2. From architecture point of view, that means that for each M-point IFFT operation, only M/2 complex samples are emitted after the filtering.

The fact that the most common OFDM scheme transmits QAM symbols thanks to the use of a basic Inverse Fast Fourier Transform (IFFT) at the transmitter and a FFT at the receiver, allows focusing the description of our prototype only on what distinguishes this type of modulation from the conventional OFDM. The main differences are in the modulation scheme, i.e. pre-modulation, IOTA filter and also the processes dedicated to channel estimation. For the demodulation side, Iota
pulse "de-shaping" and data "de-weighting" are strictly equivalent to the modulation. Concerning channel estimation, nothing more is required than for conventional OFDM.

Fig. 1 shows the OFDM/OQAM/IOTA signal generation chain. In order to have orthogonality in real domain, each real data is pre-modulated by \( m_{\text{re}} \) (represented by the real QAM mapping block in Fig. 1) followed by the inverse Fourier transform (IFFT) of the length \( M \) vector. The last step is the IOTA polyphase filtering. An interface block can be needed between the IFFT module and the filter module. Indeed, the filtering algorithm needs the first and \((M/2)^{\text{th}}\) samples of the \( M \)-point IFFT output at the same time. If the decimation in time FFT algorithm is used, the output is in normal order and hence we need to store the first \( M/2-1 \) IFFT samples output before beginning the filtering step. Instead, if the decimation in frequency FFT algorithm is used, the output is in bit-reversed order and no memory is needed. Moreover, the choice of the IFFT architecture, i.e. either pipelined FFT architecture or memory-based FFT architecture, has also a great influence on the whole architecture [4].

During filtering, \( 2L \) IOTA functions (where \( L \) is truncation length of IOTA prototype function) are superimposed in time domain (each one of them being shifted of \( \tau \), in the continuous domain or \( M/2 \) samples in the discrete domain) and each sample of the emitted signal is the result of the summation of \( 2L \) \( M \)-point IFFTs filtered by the IOTA coefficients.

The filtering algorithm can be described in the following way [5]: at the exit of the IFFT, the polyphase filter receive two complex samples by clock cycle, \( C_{k,0} \) and \( C_{k,1+M/2} \) (with \( 0 \leq k \leq M/2-1 \)), which represent the \( M/2 \) first and the \( M/2 \) last samples of the IFFT result and are weighted by \( L \) different IOTA coefficients, using \( 2L \) multipliers and added to the samples of the \( 2L-1 \) previous filtered \( M \)-point IFFTs. Finally, \( M/2 \) complex samples are emitted by OFDM/OQAM/IOTA symbol. Hence, the filtering needs \( 2L-1 \) memory of \( M/2 \) complex coefficients to store the \( 2L-1 \) previous filtered \( M \)-point IFFTs, \( 2L \) multipliers and \( 2L \) adders for the real part, and as much for the imaginary part.

Fig. 2 shows the architecture carrying out the filtering algorithm with \( L=2 \). As we can see, by using \( 2L-1 \) FIFOs, we obtain an efficient architecture which reproduces the algorithm data-path with very little control logic.

III. OFDM/OQAM IMPLEMENTATION

A. SystemC implementation

SystemC makes it possible to optimize design development time and to refine system implementation. An OFDM/OQAM/IOTA modulator and demodulator have been implemented with SystemC to verify the filtering algorithm. Fig. 3 shows the block diagram of OFDM/OQAM/IOTA modulator/demodulator implemented. Each block has been implemented with SystemC transaction level modeling.

The Source block sends only real data (i.e. after the QAM complex data separation in two real data). The comparator block allows verifying orthogonality of the OFDM/OQAM modulation by comparing the real data sent by the Source block and the real data received by the Sink block. SystemC allows to us to verify quickly the real domain orthogonality of the modulation and the simple implementation efficiency of the polyphase filter at the transmitter and the receiver side and hence their duality.

B. Prototyping with IOTA filter

In this section, a real time demonstrator with IOTA filter is described. The functional blocks are depicted in Fig. 4. The special processes for OFDM/OQAM/IOTA are in pink color. Table II describes the Frame structure based on the 3GPP-LTE frame.

Below there some examples of data rate:
- QPSK, 1/2 : 1.88 Mbits/s
- 16QAM, 3/4 : 5.64 Mbits/s
- 64QAM, 11/12 : 10.53 Mbits/s

In this demonstrator, it should be noted that the coding scheme is a duo-binary turbo code with 8-states as used in DVB-RCT. The size of each bloc is 188 bytes, like a MPEG-TS packet. The implementation of pulse shaping is carried out with the previous IOTA polyphase filter. The first implementation consists in two different "daughter" boards.
one for the modulation side one other for the reception side.

The global modulation implementation (including coding, interleaving) requires in EP1S80 Stratix FPGA:
- 9288 Logic Elements (12%)
- 1.07M Memory Bits (14%) (including interleaver)
- 52 Multipliers (9 bits) (30%) (filters, weighting, IFFT, …)

The complete reception implementation (including channel estimation, decoding, de-interleaving) requires:
- 25782 Logic Elements (33%)
- 1.3 M Memory Bits (18%) (including de-interleaver, …)
- 102 Multipliers (9 bits) (30%) (Filters, de-weighting, FFT, channel estimation, …).

The main characteristics of FTR&D platform are based around CompactPCI 64bits/66MHz bus, bi-processors system board with QNX for the real-time Operating System and up to 3 peripheral mother boards which one PMC daughter board. Each PMC daughter board integrates 3 FPGAs (Stratix S80, 20 millions gates, maximum clock frequency around 250MHz), up to 3 interface (I/O) boards for adaptation to external constraints/system. Even though we are considering such a platform as very interesting, especially because of its availability and its suitability, the goal is not to proselyte this hardware platform but really is to make use of it to prove the feasibility of the algorithms proposed by partners or by ourselves.

![Fig. 4 Block diagram of the real time OFDM/IOTA modem](image)

**TABLE II**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampling frequency</td>
<td>3.84 MHz</td>
</tr>
<tr>
<td>Frequency band</td>
<td>2.26 MHz</td>
</tr>
<tr>
<td>Frame duration</td>
<td>4ms (60 symbols)</td>
</tr>
<tr>
<td>Symbol duration</td>
<td>66.66 µs</td>
</tr>
<tr>
<td>Inter-carrier spacing</td>
<td>7.5 kHz</td>
</tr>
<tr>
<td>FFT size</td>
<td>512</td>
</tr>
<tr>
<td>Modulation</td>
<td>QPSK, 16QAM, 64QAM</td>
</tr>
<tr>
<td>Code Rates</td>
<td>1/2, 7/12, 2/3, 3/4, 5/6, 11/12</td>
</tr>
<tr>
<td>Turbo code block size</td>
<td>188 bytes</td>
</tr>
</tbody>
</table>

**IV. SIMULATION RESULTS**

The first results presented in Fig. 5 and Fig. 6 show that the performances in terms of SNR are, with a Gaussian channel, equivalent to conventional OFDM.

In addition, a time synchronization (by correlation in time domain on a pattern generated in frequency domain) is implemented in the demonstrator and is acceptable to synchronization of OFDM/QAM/IOTA signal even if the frequency synchronization is slightly shifted (+/- 1kHz). A study is in progress to complete a time and frequency synchronization which demonstrates that the synchronization with OFDM/QAM system is not really more difficult than the conventional OFDM.

The France Telecom Demonstrator aims at assessment of technology in terms of feasibility, complexity (silicon surface or equivalent one), checking and testing in real time the performances of complete transmission chain.

**V. CONCLUSIONS**

The OFDM/QAM modulation is a promising evolution of the very used CP-OFDM modulation. The advances in micro-electronic technology make possible the realization of such complex and efficient algorithm allowing improving power and bandwidth efficiency of future wireless communication standard.

The presented demonstrator shows the real feasibility of OFDM/QAM/IOTA system and will allow evaluating a complete performance of system and will make possible the implementation of new prototype functions which improve the performance while also simplify channel estimation.

![Fig 5 performances in terms of SNR with QPSK mapping](image)
REFERENCES


Performance estimation of communication processors using design of experiments

David Monnereau, Dominique Bolot, Dominique Ragot
Thales Communication - France
{first_name.last_name}@fr.thalesgroup.com

Nathalie Julien, Eric Senn
LESTER – University of South Brittany - France
{first_name.last_name}@univ-ubs.fr

Abstract – Communication processors are real systems on chips (SoCs) enabling the interfacing of different communication standards such as Ethernet, PCI, serial, parallel, etc. The broadening application domains of these processors (Wi-Fi access points, set-top boxes, routers/switches, etc.) has caused an increase and a diversification of the offer, being harder to control.

This study proposes a methodology for testing and analyzing the performances of such processors according to their architectures and the application being executed. By performances, we mean CPU utilization, execution time and throughputs. The methodology is based on the use of a full factorial design of experiment, taking into account software parameters as well as hardware parameters. This work will then be used by engineers to help them choose wisely the processor which will suit best their product needs and give them arguments to explain their choice to their clients and/or superiors.

I. INTRODUCTION

When hardware electronic engineers are designing new telecommunication or networking devices, they struggle, searching the best way to design the desired product in the time and constraints required by the client. In fact, the complexity of these electronic systems is increasing in a tremendous way as they include more and more functionalities at higher data rates. Moreover, the time-to-market is getting shorter and shorter which poses real design problems to the designers. In order to stay in competition, it is important to respect the time at which a product should arrive on the market. Indeed, this window makes the whole difference between the first one and the others. For more specific projects, where there is no direct competition, it is mostly important to submit to the client’s time and functional constraints in order to have the opportunity to develop more projects for this same client and keep a privileged relation.

The complex systems this document will deal with are possessing a communication processor. On their own, these processors are already real complex systems on chips (SoC). They possess one or several cores with all their characteristics (L1 cache, pipeline, registers, FPUs, etc.), but also a set of different interface controllers, on-chip specific engines (e.g., encryption systems, TCP Offload functions), external memory controllers, all of these organized around an interconnection system (e.g., a bus with an arbiter, several buses with bridges). The problem with such components is that their architecture and functionalities may vary significantly from a manufacturer to another, thus, resulting in great overall performance differences. Designers cannot anymore simply compare basic characteristics such as clock frequencies, bus frequencies or cache sizes to know which processor will fit best their needs.

So, among the different communication SoC architectures and features that are available to the designer, some questions arise: how can the engineers proceed to efficiently compare the processors between them in order to choose the right one for their application and constraints? How can a designer evaluate the benefit of a special feature (e.g., new cache type, enhanced Ethernet controller)? How could a designer estimate the performance a processor could offer? Once the processor has processed the basic communications functions, what resources are left to add custom applications?

To answer these questions, this paper proposes a methodology based on experimentations. By varying some processor hardware parameters while executing a generic benchmark, performance measurements are made. These measurements are iterative as parameters from the benchmark will also vary. The way the different parameters are modified and the number of tests are defined using a full factorial design of experiment [1]. This way, the impact of hardware and software parameters is observed as well as the interactions between the parameters. The question which arises from this approach is how to determine the impact of parameters which don’t vary, such as the number of processing elements, the core type, etc. To answer this, we suggest that the tests are conducted on several processors with similar architectures and designed to answer the same kind of problems (see Fig. 1).

![Fig. 1. Proposed methodology](image-url)
The remainder of this paper presents related works in Section 2. The targeted processors and the generic benchmark are introduced in Section 3 and 4. Section 5 presents the chosen design of experiment and the principle of the presented methodology. A review of selected related work is given in section 6. Section 7 summarizes and concludes this paper.

II. RELATED WORK

The research in the area of modelling and analysis of processors or systems based on a processor is abounding. However, time constraints and actual working methodologies do not permit industrial designers to apply many available solutions. That’s why, cycle accurate simulations or transaction level modelling are either too time consuming or not accurate enough.

However, some interesting work has been done on processors using design of experiment. For example, PJ Joseph et al. [1] have been working on extracting a processor’s most significant parameters by simulating an application’s execution from the benchmark SPEC CPU2000 on a custom simulator and a D-optimal two level design of experiments. Two values was assigned to each parameter, a maximum and a minimum. Then, the configuration of the parameters was decided by using a D-optimal algorithm and the benchmark was executed. The procedure builds linear models relating a processor’s performance response, in terms of cycles per instructions, to the micro-architectural parameters. The constructed linear models can be used to predict the response at other parameter settings. However, there is one model per benchmark tested, so the impact of software parameters are not directly studied.

B.C. Lee et al. [3], working on the same problem as PJ Joseph, have decided to regroup some dependant parameters together, to vary them simultaneously. Each parameter can take more than two distinctive values. Instead of choosing a D-optimal design, it has been chosen to select a sample of 4000 different configurations randomly. This approach takes into account parameters related to the software being executed.

Whereas these two approaches were studying general purpose processors, we will be using the design of experiment method to study communication processors.

III. TARGETED PROCESSORS

Communication processors have been developed for more than 15 years and have been used in embedded systems at the lower-end of networks for data and control plane processing applications, such as routers, voice gateways, wireless infrastructure (gateways and base stations), DSLAMs and telephony equipment [4].

For some time, the bottleneck in communications and network systems used to be at the bandwidth level only. Today, with the advances in the technologies of the physical mediums used (such as optical cables) providing large bandwidth, and with the emergence of a diversity of new and complex protocols and services requiring significant processing power, the bottleneck of communications networks has moved to the network nodes, where several data flows must be managed.

To cope with the demanded flexibility and short development time, an idea would be to use a general purpose processor (GPP). The problem with such a processor is that it cannot provide, on its own, enough performance to process the data at wire rate. For example, with a 2.5 Gbps link with packet sizes of 64 bytes, there are 4.9 million packets per second. A single processor, which can execute one instruction per clock cycle at a frequency of 500 MHz, will be able to execute only 102 instructions per packet (IPP). This is not sufficient to process the communication protocol needs as well as the effective application to execute (compression/decompression, encryption, virus detection, etc.). Indeed, the average required processing capability is estimated at 380 instructions per packet for data plane algorithms. For control plane algorithms, 2 660 instructions per packet are needed on average. Consequently, in order to stay flexible and execute more instructions per packet, communication processors have been designed.

Not Network Processors

Communication processors are often confused with network processors as they can both be used in the same domains, that is to say telecommunications and networks. But, whereas communication processors are used in lower-end networking equipment, network processors are usually used in mid- to high-level networking equipment. They are optimized to perform a variety of functions, including frame classification, filtering, forwarding, marking, policing and traffic shaping. They are placed in the data path of network switching devices [5].

As the family of network processors has a great diversity of members, some processors can be very close to communications processors in their functionalities, whereas others are fundamentally different.

The study of network processors is beyond the scope of this paper.

For the time being, to limit the extent of the study, a generic characterization of the targeted processors is proposed (see Fig. 2). These processors possess one or two cores with data and instructions L1 caches, eventually L2 cache, a global memory, Ethernet and PCI-like interfaces as they represent the biggest part of inter- and extra-system communications, DMA controller and an interconnection system (shared bus, buses with bridges, crossbar, etc.). The hardware acceleration engines are left aside in our study since they are usually proprietary and would complicate in a tremendous way the tests. Our aim is to keep the tests as general and quickly portable as possible.

For example, the PowerQUICC II Pro from Freescale Semiconductors [6] is a communication processor which has one e300 PowerPC core, 32 KB of data and instruction L1 caches, no L2, 2 Gigabit Ethernet interfaces, one PCI interface 1x64 or 2x32, one DMA controller and a shared bus as interconnecting system. Other manufacturers, such as AMCC and its 440xx family [7], offers processors with similar architectures.
The aim of our work is to study these processors and elaborate models to estimate performances. The idea is to make some hardware parameters vary to observe the change in the performances and find the parameters which have the greatest impact. The problem is that on real platforms, not every configuration is possible. In fact, some parameters will be fixed and we won’t be able to touch them to observe their impact. That’s why we are willing to work on several platforms and execute a series of identical tests on each.

For example, if a processor has only one floating point unit but another one has two, the idea is to try and keep the maximum parameters identical to isolate the impact of the number of floating point units on the performances. This will also be done by choosing the right software to execute. In fact, the performances are highly dependent on the software being executed. This is what we are going to see now.

IV. THE BENCHMARK

When we speak about a processor’s performances it is important to know what kind of tests are made and especially what applications or benchmarks are executed while measuring the performances. It is not because one processor is good in a domain that this same processor will be good in another domain. For example, if a processor A has better performances for a scientific application than a processor B, this doesn’t mean that processor A will have better performances for a networking application. We will now see how this is taken into account in our approach.

First, let’s talk about the operating system (OS). In fact this entity has a big impact on the observed performances. Choosing the same OS for every platform should guarantee us a certain fairness in the conclusion we will obtain. This way, even if it will be difficult to give absolute conclusions for the tests, relative comparison between processors should be fairly accurate. The Linux operating system has been chosen since a Board Support Package (BSP) is usually available for the different processors or platforms that are targeted. Moreover, Linux has an important community working to maintain it and to make it evolve. So it won’t be difficult to make a Linux run on our platforms.

Now, let’s talk about the benchmarks. As the aim of our study is to find the best processor for a given application or type of application, it is necessary to identify what are the differences between application domains. Moreover, as we are willing to be able to estimate a processor’s performances without making any measurements, only based on the experience and results obtained with the previous processors studied, a way to characterize the application we would like to make an estimation for must be defined.

Another point which must be clarified is the meaning of performances, in other words, what performances are we willing to observe.

There are three main performances:

- *The CPU utilization*, that is to say, the time the processing units are in use during the execution of an application,
- *The Execution time*,
- *The throughputs* the system can achieve for a given application.

*The CPU utilization*

In applications which have some communication activities, it is possible that the processor spends some time waiting for data to arrive, or, in applications when some tasks are periodic, if the processor has finished its tasks it must also wait until a task becomes active again. This metric is very important with the type of devices we are studying. In fact, the main job these processors must do is managing communication packets and protocols (IP forwarding, for example) from an interface to another interface. The ideal situation is when no processing resources are used to perform these tasks. But in reality, this is not the case, and some processing power must be used (Fig. 3). One question would be, how much processing power is left once the communication protocols have been managed? What resources are left for the execution of a custom application? Manufacturers try to answer these questions by enhancing the interface controllers, the processor’s core or the interconnection system. But how to identify the benefit of such or such architecture choice?
Execution time

This performance metric is also important. In fact, it will enable the comparison of different configurations of a given processor (cache size, configuration of the interconnection, etc.). It we also enable a fair comparison of processors between them. How do processors behave for the same application?

Throughputs

Some applications have severe constraints according to throughputs. The system must be able to cope with a given incoming and/or outgoing throughput as well as other charges. So it is important to know the limits a system can achieve.

Many benchmarks are freely, or not, available on the Internet (SPEC [8], MiBench [9], CommBench [10], etc.) but none of them answer our needs. Some are very specific for a kind of application others are more general, but none are flexible. This last characteristic is the most important as we are willing to find key software parameters that have an impact on the performances and could enable us to define a general way to characterize any application in different domains (networking, multimedia, radio protocols).

To overcome this lack of availability, a generic benchmark has been developed which has been designed to be executed in a Linux environment. The hypothesis made is that any application can be divided in a limited number of tasks, each one executing an “atomic” operation, chosen from a limited list (an Ethernet transfer, a floating point operation, an integer operation, etc.). This assumption is done to enable anybody to characterize an application at a coarse grain level. In Table I, all the parameters which have been retained are listed. These parameters are applicable for one task, so when there are, for example, 5 tasks, this list is repeated 5 times with different values for each task. Fig. 4. represents some of the main parameters for a given task.

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the task</td>
</tr>
<tr>
<td>Period</td>
<td>The task’s activation’s period</td>
</tr>
<tr>
<td>Deadline</td>
<td>The task’s deadline</td>
</tr>
<tr>
<td>Start time</td>
<td>The time after which the task when activated</td>
</tr>
<tr>
<td>Start event</td>
<td>The event that will activate the task, if no period has been given</td>
</tr>
<tr>
<td>End event</td>
<td>The event the task will send when finished</td>
</tr>
<tr>
<td>Data size</td>
<td>The amount of data in memory on which the operations must be applied</td>
</tr>
<tr>
<td>Op name</td>
<td>The operation associated with the task</td>
</tr>
<tr>
<td>Op load</td>
<td>The number of times the operation must be executed when the task has been activated</td>
</tr>
<tr>
<td>Duration</td>
<td>When no load is given, the benchmark calibrates the number of iteration to fit the given duration</td>
</tr>
<tr>
<td>Sched policy</td>
<td>The policy for this task</td>
</tr>
<tr>
<td>Sched priority</td>
<td>The priority of this task</td>
</tr>
<tr>
<td>Affinity mask</td>
<td>When the processor has 2 cores, select the one on which to execute this task in priority</td>
</tr>
<tr>
<td>Max runs</td>
<td>If we are willing to execute the task only a given amount of times</td>
</tr>
</tbody>
</table>

All these information are written in a description file apart from the benchmark itself. This way, it is not necessary to compile the benchmark each time a new test must be done. The execution principle of the benchmark is simple. It reads this description file and creates one thread per task, with all the different parameters given. Then it associates an operation per thread. These operations are included in the benchmark during the compilation. Before the compilation, any operation can be written, but our aim is to keep these operations as general as possible and limited in number. If the op_load field is none null, the thread will execute the operation the number of times mentioned. But, when a duration has been given, a calibration must be done to determine the number of times the thread must execute an operation to fit the wished time.

Another point is the way the data is accessed. In order to see the impact of the memory and caches, it is important that the data size entered represent a real amount of data. For example, if we are willing to process 1MB of data, we must access the equivalent of 1MB in memory and not 1000 times the same 1KB, which would stay in cache. As this depends on the number of operations done and the size of data each operations access, it is important to define a stride, that is to say, how the thread will access data : every single byte or every 10 bytes ? If with the number of operations to be done, all the data can be accessed at least once, the thread will loop back to the beginning of the data. But if all the data cannot be accessed, the thread will have to skip some data to cover all the address space. Thus, cache effects are observable. Fig. 5. resumes this situation.

Once the initialization and calibration done, all the threads are started and they behave according to what is described in the description file.

If it is necessary to send an event in order to start the first thread of the application, it is possible to say to the benchmark to send a given event at the beginning, by the intermediate of a field in the description file.
The benchmark stops, either when all threads have been executed the given number of runs, or when a global time has been reached, again, given by a parameter, in the description file.

An example of an application is given on Fig. 6. This application is voluntary simple. Several parameters can be varied to identify the impact of each one on the performances.

Now that we have seen what kind of hardware was targeted and how the software is characterized, we will present the way these elements interact and how the design of experiment is applied.

V. THE METHODOLOGY

As we have seen before, an important amount of parameters are taken into account in this work. We have seen what kinds of applications are targeted and how they are characterized. We have also seen the kind of processors studied, however we haven’t seen how the tests were conducted.

The design of experiments

A design of experiments is a method to help conducting experimentations the best way. That is to say, when we have a problem involving several parameters, it is not always easy to find out which are the parameters the most significant to act on to improve performances, and to know what are the impact of parameters interactions on the performances. The method of the design of experiments tries to answer this problem by giving a way to determine quickly these information. Without the design of experiments, the general approach is to tune a parameter giving it different values, leaving every other parameters fixed. The impact of this parameter on the performances will be immediately observable. However, the interactions between parameters would be impossible to determine, unless, testing all the possible configurations of the system. Let’s take a system which has 10 parameters, each one having 4 different possible values. If we wanted to test all the possible configurations, we would have $4^{10} = 1'024$ possible tests to conduct. This would be unfeasible for solving our problem. Using the design of experiments, all the parameters are varied for each test, enabling the interactions between parameters to be quickly found. Then, the accuracy of the results and conclusions will depend on how and how many tests are executed. In fact, due to time, costs or other constraints, it is not always possible to do a full factorial design, that’s why other designs can be used such as fractional factorial designs, Plackett-Burman designs, randomized designs, or other mathematical generated designs, such as D-optimal designs.

In a two-level design, each parameter can take only two values, a minimum and a maximum. These extremums are chosen arbitrary, according to the use of the parameter. They are not necessary real extremums from their possible ranges, but can be extremums from their utility range, that is to say the range in which the values are generally used.

The benchmark stops, either when all threads have been executed the given number of runs, or when a global time has been reached, again, given by a parameter, in the description file.

An example of an application is given on Fig. 6. This application is voluntary simple. Several parameters can be varied to identify the impact of each one on the performances.

Now that we have seen what kind of hardware was targeted and how the software is characterized, we will present the way these elements interact and how the design of experiment is applied.

V. THE METHODOLOGY

As we have seen before, an important amount of parameters are taken into account in this work. We have seen what kinds of applications are targeted and how they are characterized. We have also seen the kind of processors studied, however we haven’t seen how the tests were conducted.

The design of experiments

A design of experiments is a method to help conducting experimentations the best way. That is to say, when we have a problem involving several parameters, it is not always easy to find out which are the parameters the most significant to act on to improve performances, and to know what are the impact of parameters interactions on the performances. The method of the design of experiments tries to answer this problem by giving a way to determine quickly these information. Without the design of experiments, the general approach is to tune a parameter giving it different values, leaving every other parameters fixed. The impact of this parameter on the performances will be immediately observable. However, the interactions between parameters would be impossible to determine, unless, testing all the possible configurations of the system. Let’s take a system which has 10 parameters, each one having 4 different possible values. If we wanted to test all the possible configurations, we would have $4^{10} = 1'024$ possible tests to conduct. This would be unfeasible for solving our problem. Using the design of experiments, all the parameters are varied for each test, enabling the interactions between parameters to be quickly found. Then, the accuracy of the results and conclusions will depend on how and how many tests are executed. In fact, due to time, costs or other constraints, it is not always possible to do a full factorial design, that’s why other designs can be used such as fractional factorial designs, Plackett-Burman designs, randomized designs, or other mathematical generated designs, such as D-optimal designs.

In a two-level design, each parameter can take only two values, a minimum and a maximum. These extremums are chosen arbitrary, according to the use of the parameter. They are not necessary real extremums from their possible ranges, but can be extremums from their utility range, that is to say the range in which the values are generally used.

The benchmark stops, either when all threads have been executed the given number of runs, or when a global time has been reached, again, given by a parameter, in the description file.

An example of an application is given on Fig. 6. This application is voluntary simple. Several parameters can be varied to identify the impact of each one on the performances.

Now that we have seen what kind of hardware was targeted and how the software is characterized, we will present the way these elements interact and how the design of experiment is applied.

V. THE METHODOLOGY

As we have seen before, an important amount of parameters are taken into account in this work. We have seen what kinds of applications are targeted and how they are characterized. We have also seen the kind of processors studied, however we haven’t seen how the tests were conducted.

The design of experiments

A design of experiments is a method to help conducting experimentations the best way. That is to say, when we have a problem involving several parameters, it is not always easy to find out which are the parameters the most significant to act on to improve performances, and to know what are the impact of parameters interactions on the performances. The method of the design of experiments tries to answer this problem by giving a way to determine quickly these information. Without the design of experiments, the general approach is to tune a parameter giving it different values, leaving every other parameters fixed. The impact of this parameter on the performances will be immediately observable. However, the interactions between parameters would be impossible to determine, unless, testing all the possible configurations of the system. Let’s take a system which has 10 parameters, each one having 4 different possible values. If we wanted to test all the possible configurations, we would have $4^{10} = 1'024$ possible tests to conduct. This would be unfeasible for solving our problem. Using the design of experiments, all the parameters are varied for each test, enabling the interactions between parameters to be quickly found. Then, the accuracy of the results and conclusions will depend on how and how many tests are executed. In fact, due to time, costs or other constraints, it is not always possible to do a full factorial design, that’s why other designs can be used such as fractional factorial designs, Plackett-Burman designs, randomized designs, or other mathematical generated designs, such as D-optimal designs.

In a two-level design, each parameter can take only two values, a minimum and a maximum. These extremums are chosen arbitrary, according to the use of the parameter. They are not necessary real extremums from their possible ranges, but can be extremums from their utility range, that is to say the range in which the values are generally used.

The benchmark stops, either when all threads have been executed the given number of runs, or when a global time has been reached, again, given by a parameter, in the description file.

An example of an application is given on Fig. 6. This application is voluntary simple. Several parameters can be varied to identify the impact of each one on the performances.

Now that we have seen what kind of hardware was targeted and how the software is characterized, we will present the way these elements interact and how the design of experiment is applied.

V. THE METHODOLOGY

As we have seen before, an important amount of parameters are taken into account in this work. We have seen what kinds of applications are targeted and how they are characterized. We have also seen the kind of processors studied, however we haven’t seen how the tests were conducted.

The design of experiments

A design of experiments is a method to help conducting experimentations the best way. That is to say, when we have a problem involving several parameters, it is not always easy to find out which are the parameters the most significant to act on to improve performances, and to know what are the impact of parameters interactions on the performances. The method of the design of experiments tries to answer this problem by giving a way to determine quickly these information. Without the design of experiments, the general approach is to tune a parameter giving it different values, leaving every other parameters fixed. The impact of this parameter on the performances will be immediately observable. However, the interactions between parameters would be impossible to determine, unless, testing all the possible configurations of the system. Let’s take a system which has 10 parameters, each one having 4 different possible values. If we wanted to test all the possible configurations, we would have $4^{10} = 1'024$ possible tests to conduct. This would be unfeasible for solving our problem. Using the design of experiments, all the parameters are varied for each test, enabling the interactions between parameters to be quickly found. Then, the accuracy of the results and conclusions will depend on how and how many tests are executed. In fact, due to time, costs or other constraints, it is not always possible to do a full factorial design, that’s why other designs can be used such as fractional factorial designs, Plackett-Burman designs, randomized designs, or other mathematical generated designs, such as D-optimal designs.

In a two-level design, each parameter can take only two values, a minimum and a maximum. These extremums are chosen arbitrary, according to the use of the parameter. They are not necessary real extremums from their possible ranges, but can be extremums from their utility range, that is to say the range in which the values are generally used.
fundamental characteristics for each domain, thus stressing the hardware differently to find the most significant parameters, and the elements which are dimensioning the system. The descriptions of the applications are done in the description file, readable by the benchmark. If necessary, some new operations must be defined if they are thought to be essential and elementary bricks of applications.

The characterization step is very important and should be done only once. In fact, once the applications have been characterized, they won’t be modified for the tests on the different platforms. It might be necessary to redefine new applications if the targeted platform has too many differences with the previous studied ones, or if the applications seem to be obsolete, or do not stress the platform enough anymore.

Then, the next step is to define the two values each hardware and software parameter will take during the tests. As said previously, it is important to choose suitably these values in order to have representative results to elaborate accurate models. Software values are modified in the description file. A simple tool has been developed to enable the modification of one parameter at a time in a description file. For the hardware parameters that can be modified dynamically, that is to say while the operating system is running, a module has been developed to access hardware configuration registers, which are not directly accessible with Linux. Most of the parameters can be modified automatically this way, without needing to change them constantly by hand. However, some parameters cannot be changed dynamically. This is the case, for example, of the frequencies for the PowerQUICC II Pro. For such parameters, a change must be done in the boot loader or kernel source code, so a reboot is necessary to upload the modified boot load or kernel.

The modification of the different parameters, hardware and software, is done by using a script file, enabling the automation of most of the tests. For each modified parameter, the benchmark is executed on the platform and results are generated. As the results generated by the benchmark are not all useful for our purpose, a formatting is necessary to keep only the information we are interested in (CPU utilization, execution time, throughputs). These results are formatted in a file with all the values of the parameters in order to be directly exploitable.

These tests are then repeated on each platform we want to study and elaborate models for. Fig. 7 resumes the methodology, notably the way the tests are conducted.

VI. PRELIMINARY RESULTS

For the time being, tests that have been done, were done mainly to set up the methodology, the different tools needed and the tests.

The first series of tests consisted in executing the benchmark with only one periodic thread, doing a simple integer operation, on the PowerQUICC II Pro. The data_size and op_load parameters where varied as well as hardware parameters. TABLE II Presents the main parameters. Other parameters were also tested, but further study must be done before presenting them here. Data_size and op_load have already been seen, Dcache and Icache stand for data and instruction caches, Pipedep is the number of transactions the system bus can start at once (1 by default), Apark is the indication where the bus must park the address, that is to say, which master will be able to have the bus immediately, without a demand process, core and TSEC priorities are the priorities given respectively to the core and to the Ethernet controller on the bus.

Fig. 8. presents the 19 main effects on the CPU utilization response, taking into account the effects of single parameters, and the effects of interactions of levels two and three. The numbers on the left refer to the parameters (11 \( \rightarrow \) op_load, 10 \( \rightarrow \) dcache, 10-11 \( \rightarrow \) Effect on the response of the interaction of 10 and 11, etc.). We can see, on this Figure, that there are three main parameters that impact the response here: the number of operations, the size of the data cache and the size of the data accessed. This is coherent according to the test that has been done. The high values obtained probably means that the system is overflowed and cannot cope with the number of operations to execute on the data. It could be an idea to reduce the op_load high value to observe the behavior of the system. Other parameters also have an impact on the response: the frequency of the bus system, the apark and pinedep parameters. This is interesting because, whereas the effect of the first parameters were predictable, those ones were not so forthcoming. In fact, even if their effect is reduced, it can enable us to find configurations for the system that can improve its overall performances. The instruction cache parameter doesn’t appear here, meaning that the benchmark probably fits completely in the cache.

Even if this test is very simple, and is not really representative, it enabled us to validate part of the different
TABLE II
VALUES OF THE PARAMETERS USED FOR THE FIRST TESTS

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>LOW VALUE</th>
<th>HIGH VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data_size</td>
<td>20000</td>
<td>2000000</td>
</tr>
<tr>
<td>Op_load</td>
<td>250000</td>
<td>2500000</td>
</tr>
<tr>
<td>Dcache</td>
<td>4KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Icache</td>
<td>4KB</td>
<td>32KB</td>
</tr>
<tr>
<td>System Bus Freq (CSBFreq)</td>
<td>264MHz</td>
<td>330MHz</td>
</tr>
<tr>
<td>PipeDep</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Apark</td>
<td>Disabled</td>
<td>Last owner</td>
</tr>
<tr>
<td>Core Priority</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>TSEC Priority</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

Fig. 8. First-, second- and third-level interactions effects table

tools used. The next step would be to characterize accurately representative applications and the range of the parameters.

VII. CONCLUSION AND PERSPECTIVES

We have presented here a methodology to study communication processors, aiming the extraction of the significant parameters of the system in order to elaborate models. This study is based on measurements directly on real platforms. The applications targeted were presented as well as the way to conduct the experiments using a full factorial two-level design of experiment.

This research work being at its beginning, more experimentations must be done to validate the approach. It is indeed intended to make the characterizations of representative applications and give realistic values for the different parameters involved.

ACKNOWLEDGMENTS

The authors are grateful to Dominique Ragot for having developed and made available its generic benchmark and for his implication in this research work.

REFERENCES


[8] www.spec.org


ECG Signal Artifacts Suppression in MRI Environment by Means of LMS filtering

Hervé Berviller, Vincent Frick, Lingchuan Zhou
Joris Pascal, Jean-Philippe Blondé
Institut d’Électronique du Solide et des Systèmes (InESS)
ULP / CNRS – UMR 7163
Strasbourg, France
vincent.frick@iness.c-strasbourg.fr

Julien Oster, Jacques Felblinger
Imagerie Adaptative Diagnostique et Interventionnelle (IADI)
UHP / INSERM – ERI 13
Nancy, France

Abstract—This paper presents the signal processing strategy of a future embedded system aiming at improving images quality of a MRI cardiac examination1. The processing consists of correcting the ECG signal by means of an adaptive filtering to cancel the interferences related to the MRI magnetic fields. It uses the magnetics gradients measured by the presented high resolution ASIC. The filter has been implemented together with a QRS complex detection algorithm on an FPGA target. The promising experimental results validate the principle and open new perspectives for an ASIC integration of the whole system.

I. INTRODUCTION

Magnetic Resonance Imaging (MRI) systems have sudden major technological evolutions over the last few years (fast imaging, parallel imaging, etc.). Nevertheless, MRI is still limited by physical constants such as relaxation time of the tissues. Therefore, the acquiring time cannot be indefinitely reduced without spatial resolution or signal to noise ratio losses. Hence, most of the MRI sequences are sensitive to physiological movements of the patient and his organs (breathing, heart beats, etc.). These movements are likely to cause artifacts on the image.

This problem is particularly sensitive during a cardiac examination, where the image acquisition must be perfectly synchronized with the heart beats. To do that, it is necessary to obtain the electrocardiogram signal (ECG) during the MRI sequence. However, the various magnetic fields affect the patient's bioelectric signals. This induces problems, not only for the patient's cardiac activity monitoring, but also for the acquiring sequence.

The objective is thus to apply a treatment to the collected ECG signal in order to extract it from its disturbances as well as some useful information, i.e. the heart beat phase (called complex QRS). The proposed technique is based on the use of an adaptive filtering to, in real time, cancel interferences by the means of the disturbance signals themselves, i.e. the magnetic fields signals.

The work presented in this paper is a part of a more ambitious project integrating magnetic sensors, ECG measurement and signal correction on the same ASIC chip. This paper presents both the ASIC circuit for gradient measurement we developed and the ECG adaptive filtering implemented on a FPGA target. This kind of circuit has been mainly selected because of its flexibility which is particularly interesting during a prototyping phase.

The general context and related problematic are discussed in section II. Section III is dedicated to the magnetic field measurement ASIC. Section IV presents the adaptive filter advantages and its applications about ECG filtering, while section V focus on methodology and architecture needed by the FPGA implementation. Experimental results are presented in section VI.

II. GENERAL CONTEXT OF THE SUGGESTED SYSTEM

MRI principle requires a very strong static magnetic field \( B_0 \) (1.5 or 3 Tesla in current clinical use). Additional coils generate magnetic pulses need to be superimposed to \( B_0 \). These pulses, commonly known as the gradients, are required for MRI signal locating and image construction. The main disadvantage of this technique lies in the interaction between these magnetic fields and the measured bioelectric signals measured, and particularly the ECG.

Figure 1 illustrates the disturbances caused by the MRI environment on the ECG due to the strong static magnetic field and the gradients.

During a heart imagery, the ECG is of course used to monitor the patient's cardiac activity, but it is also very useful for the images acquisitions even. Indeed, as mentioned in the introduction, a better synchronization between the heart movements and the frame grabbing makes it possible to improve the image quality. This synchronization signal can be extracted from the ECG as illustrated in figure 2.

---

1 This work was supported by the French National Network for Health Technology (RNTS), the French Ministry of Economy, Finance and Industry and the French Ministry of Research.
The raw ECG from figure 1 is clearly not usable for image synchronization. The goal of this work thus consists of discriminating between the artifacts caused by the MRI and the ECG itself in order to extract the synchronization informations.

To achieve this task in real time, we have to be able to measure the disturbances inside the imager. So it is necessary to use a magnetic fields sensor able to provide relevant information. Such an ASIC dedicated sensor is currently developed in our laboratory and gives interesting results [2]. It's principle is presented in the following section. Starting from results of various projects, the employed methodology is presented in figure 3.

III. Magnetic Field Measurement ASIC

Given the MRI principle, the challenge consists of accurately monitoring magnetic pulses of a few tens of milliteslas (the gradients) that are superimposed over a strong static field $B_0$ as high as 1.5 T (usual value in current clinical use) without saturating the sensor's signal. This section presents the fully analog instrumental chain developed to perform high resolution measurement of the MRI magnetic pulses.

1) Sensor choice: integration and linearity considerations

Silicon transducers appear to be the best suited magnetic devices for such an environment because they are not subject to saturation even when submitted to strong static field. Furthermore they don't require any post processing and can easily be associated to conventional signal conditioning electronics [1]. The use of a standard CMOS process is also quite important for the future versions of the system where the magnetic instrumental chain will be embedded together, on the same chip, with the digital ECG artifact suppression module, which is presented in the next section of this paper.

Figure 4 shows a "Greek-cross" shaped horizontal Hall device (HHD) together with its constant current biasing circuitry as proposed in [2]. The common expression for $S_i$, the biasing current related sensitivity of such a device, is recalled in the following equation [1]:

$$ S_i = \frac{r_n G}{q n e} \tag{1} $$

where $t_e$ is the effective thickness of the HHD’s active zone, i.e. an N-well with a doping concentration $n$. The term $r_n$ represents the scattering factor of silicon ($r_n \approx 1.15$) and $q$ the elementary charge. The parameter $G$ is a geometry dependent correction factor [1].

As far as the linearity is concerned, the current lines inside the HHD are deviated by an angle $\theta_H$, called the Hall angle, according to the strength of the magnetic field. $G$ is a function of this angle and thus induces non-linearity of $S_i$ [1]. Yet, the non-linearity can be neglected when the HHD is submitted to fields $B_{puls}$ varying in a range of a few tens of milliteslas [1]. Furthermore the high field $B_0$ has no influence on the linearity either because it only has a static component. In fact, even though it indeed induces a larger $\theta_H$ variation, this later remains constant. Of course, this is true under the condition that the HHD is not moved within the MRI environment during magnetic field measurement, i.e. the HHD is always submitted to the same static component.

2) Structure of the magnetic instrumental chain ASIC

Figure 5 shows the general structure of the instrumental chain ASIC, whose detailed architecture and operating has been presented in previous works [2].
The HHD is biased with a constant current $I_{pol}$ generated within a dedicated front-end featuring a low noise operational amplifier. The Hall voltage $V_{HH}$ at the output of the HHD is such that $V_{HH} = Sx_{pol}B_T$, where $B_T$ is the total magnetic field ($B_T = B_0 + B_{pol}$). The static component of $V_{HH}$ due to $B_0$ is first removed via the static field compensation bloc. The resulting signal is then amplified and filtered.

![Figure 5. Diagram of the magnetic instrumental chain ASIC](image)

Low noise performances are achieved by using spinning current of the HHD and chopper stabilization techniques throughout the instrumental chain [3-4]. The measured resolution of the chain is 20 $\mu$T over a 1.6 kHz bandwidth extending from 5 Hz to 1.6 kHz.

IV. ECG SIGNAL PROCESSING

The corrected ECG will be used to synchronize the image acquiring sequence, thus it is imperative the signal treatment to be compatible with a real time operation. In addition, since the system must be constantly adapted to the disturbances variations, the use of an adaptive filtering seems particularly indicated. Many researches concerning the choice of the most efficient algorithm have been done for several years by one of the partners team of this project [5].

The goal in this section is not to evaluate these various algorithms, but to validate the suggested concept, by implementing one of them which is compatible with a real time operation. Therefore, we set our choice on the Least Mean Square (LMS) algorithm given its relative simplicity, performances and potential in terms of FPGA and ASIC implementation.

A. Adaptive filtering

An adaptive filter is a filter whose coefficients are modified, according to a given law, before each arrival of a new value at the filter entry. These modifications have to follow the system evolutions in its environment as fast as possible. Adaptive filtering is generally associated with real time operations.

![Figure 6. Adaptive filter principle](image)

**Figure 6** presents an adaptive filter used in interferences cancellation, where:

- $d(n)$ is a signal containing the sum of the useful information and the interferences to be canceled. For our application, it is the perturbed electrocardiogram.
- $x(n)$ is a stripped useful information signal, obtained by a sensor close to the interferences: it is the magnetic disturbances measurement which are emitted by the MRI. It is necessary that the magnetic disturbances are correlated with the interferences present in the ECG to be able to successfully apply an adaptive filter.
- $y(n)$ is the signal generated by the filter whose coefficients are updated in order to achieve the transfer function between the MRI gradients and the ECG. From the magnetic disturbances measurement, the filter generates an estimate of the interferences present in the ECG.
- $e(n)$ is the perturbed ECG signal which one withdrew the filter's estimate MRI interferences, it is thus the corrected ECG signal.

The next phase thus consists of choosing the law of coefficients update which will allow to obtain an exploitable signal.

B. LMS Algorithm

LMS Filter is the simplest and most popular adaptive filter. LMS Algorithm is based on the least square method. By taking again the notations of figure 6:

The error signal is given by the following relation:

$$ e(n) = d(n) - y(n) $$

where $d$ represents the perturbed signal, and $y(n)$ represents the artifacts estimations.

$$ y(n) = \sum_{i=0}^{M-1} w_i(n).x(n-i) = W_M(n)^T.X_M(n) $$

$M$ is the filter's coefficient number. The $x(n)$ signal corresponds to a magnetic disturbance generated by the MRI. $w_i(n)$ are the filter's coefficients.

The goal is to minimize the mean square error $E[e^2(n)]$.

For adaptive filtering, the filter's coefficients are determined by an iterative method. Finally, for each time step $n$ one recomputes the filter's coefficients:

$$ W_M(n+1) = W_M(n) + \Delta W_M(n) $$

or too

$$ w_i(n+1) = w_i(n) + 2.\mu.e(n).x(n-i) $$

for $i=0,1,\ldots,M-1$

The adaptation step (or adaptation gain) $\mu$ regulates both the speed and stability of the algorithm.
C. Application to the ECG

During an MRI examination, three coils produce the magnetic field gradients allowing to obtain the images. Each one emits in one of the three space directions (X, Y and Z).

Since the electrocardiogram is perturbed by three distinct sources, the filtering structure should comprise three LMS filter. Each path takes one space direction into account (figure 7). In this example, the used disturbances will be the gradients control signals.

![Figure 7. Used ECG adaptive filter structure](image)

V. FPGA IMPLEMENTATION

A. Methodology

The choice of the implementation target mainly concerns our work organization and technical-economic constraints. Adaptive filtering was developed on FPGA in order to validate its feasibility at lower cost and with the related flexibility of reconfigurable circuits. This phase validated, the migration towards ASIC should be easier.

Integrating such a filter on an FPGA target requires some algorithm-architecture adequacy considerations. In the following paragraphs we will discuss the choice of the most efficient architecture to this target as well as retained strategy in the design of the circuit.

B. Architecture

As wrote before one of the characteristics of an adaptive filter resides in the variations of its coefficients. Its architecture, in its most traditional form, thus comprises a module of weight update, as illustrated in figure 8 in the case of a M coefficients filter [6].

![Figure 8. Classical adaptive filter structure](image)

This structure requires at least 2M multipliers 2M adders and 2M unit delays. Taking into account the number of limited resources of a FPGA and the number of filters to be implemented (one for each gradient), it is necessary to look for structures requiring less resources. One of them [7], allow reduction of the number of multipliers by factor 2, by using the same multiplier for the weight update and the sample calculation. But this reduction is unfortunately not sufficient for our application.

The adopted structure is that proposed in [8], for which there are only one MAC shared by all the coefficients to perform the calculation of the exit samples.

The MAC’s input is controlled by a multiplexer that distributes the delayed samples. The coefficients are stored in a RAM and the whole architecture is managed by a controller (figure 9).

![Figure 9. Multiplexed MAC filter architecture](image)

C. Adopted strategy

We choose a FPGA implementation in order to validate the feasibility of the whole system. To exploit as well as possible flexibility related to the use of reconfigurable circuits, it was decided to design a system which is skeletal by the user. The system (developed using Matlab-Simulink® software and Altera® DSP Builder Libraries) allows the automatic generation of the whole filter only starting from the filter's coefficient number.

This flexibility, allows users far away from the low level considerations to be able to refine the final characteristics of the system starting from the MRI environment collected signals. These final characteristics can be, for example, the coefficient number, the adaptation step µ, or the extremes characteristics of the input signal.

D. Constraints

In addition to the above mentioned surface problem, the other constraint related to the selected target relates to the numeral figures representation.

Since the FPGA uses a fixed-point representation, it is necessary to dimension the buses consequently. To do that, we have to define a sufficiently large bus width for the integer part so that there will be no saturation. The bus width
of the decimal part, is rather important so that the loss precision will not be significant. The total bus width should not exceed 51 bits which is the maximum value permitted by the software.

Given the ECG characteristics it appears that a 12 bits bus width for the integer part was sufficient to avoid saturation problems.

The bus width determination of the decimal part is more delicate. Indeed, it is necessary to determine if the precision loss induced is or not significant. For that several filters were carried out with various decimal part bus widths. A 24 bits bus width was finally selected since the accuracy loss was in this case not significant compared to the floating point simulations.

E. Implementation results

The filter has been implemented on a Altera Stratix EP1S80 80 MHz DSP Development Kit and designed using Matlab-Simulink® software and Altera DSP Builder libraries.

![Figure 10. Figure :8 coefficient number vs. used resources](image)

An efficient ECG filtering requires approximately hundred coefficients. As predicted, the classical structure requires too much Logic Elements (LE) resources and cannot be implemented in the circuit. The maximum capacities of the circuit are reached starting from about thirty coefficients. The possibilities are much wider with the adopted structure. For example, a two hundred coefficients filter consumes only half of the available resources in term of LE, as shown in the figure 10. The main drawback of this architecture relates to the operating frequency, since it must be M time faster than the basic structure. However, taking into account the value of the sampling rate of ECG signal (typically 1kHz), we are far from the maximum operation frequencies of the circuit.

VI. EXPERIMENTAL RESULTS

A complete filter including 100 coefficients was implemented on the FPGA device. The simulation results obtained under Simulink® environment are shown in figure 11. It features the ECG signal before (during the first 5 seconds) and after application of the gradients. The top view is a measurement of the raw ECG signal, i.e. before correction. The middle and bottom view are the resulting ECG signal after applying respectively the conventional SVD reference method [5] and the proposed LMS method. Both results are very similar, which validates the proposed architecture.

In order to illustrate in a more relevant way the filtering efficiency, a complete QRS complex detection has been implemented at the output of the LMS filter. At the system output one should only recover the QRS impulses of the non-perturbed signal. The used input signal is a recorded real perturbed ECG signal consisting of 4 heart beats, continuously replayed. The resulting output signals with and without filtering are respectively shown in figure 13 and figure 12.

![Figure 11. ECG signals without correction (top view) and with correction (SVD: middle view; LMS: bottom view)](image)

![Figure 12. Experimental QRS complex without correction](image)

![Figure 13. Experimental QRS complex with correction](image)
VII. Conclusion

An ECG signal artifacts suppression system based on an MRI environment dedicated CMOS magnetic field monitor and adaptive an LMS correction filter as been presented and implemented. The experimental results validate the principle and are particularly encouraging for a future real-time embedded system. The use of an FPGA target for prototyping has two advantages. Firstly, starting from various ECG benchmarks, it allows to refine the final characteristics of the desired system. Secondly it facilitate the transfer steps toward full integration system.

The next step will consist of testing the algorithm implementation directly in the MRI environment.

References

Abstract— Considering the performance increase provided by redundant operators such as adders and multipliers, it appears interesting to generalize the use of those operators in high computational digital circuit design. However, using redundant arithmetic in conjunction with classical arithmetic is a complex task for circuit designers who might not have necessarily the required arithmetic knowledge. Therefore, it becomes mandatory to develop optimization CAD tools which automate the use of redundant arithmetic in circuit design. This paper presents such an optimization tool which, based on non redundant representations, introduces automatically redundant operators thanks to pattern matching techniques. To illustrate this, the optimizations of a filter and a Discrete Cosine Transform (DCT) generators are presented.

Index Terms—automatic, optimization, redundant arithmetic, pattern matching, CAD tool

I. INTRODUCTION

REDUNDANT operators such as adders and multipliers [1], [2] have very good performances in terms of timing and area [3]. Using those operators in VLSI circuit design can thus appear advantageous, enabling architecture optimizations and consequently further improvements as for circuits’ performances. To illustrate this, we can consider the handmade implementations of a DCT macro bloc [4] and a distance computation unit [5] using those architectures, which both result in a significantly increase of the timing performances with a small area overhead.

Mixing classical and redundant arithmetics in an explicit way can nevertheless appear quite tedious to non initiated designers, for whom, furthermore, the rapid pace of technological evolution puts a great “time to market” pressure. That is why redundant arithmetic is not very often used for high computational digital circuits. Such a pressure on design cycle combined with strict performance constraints make more and more useful the automation of the introduction of redundant arithmetic in high computational digital circuit design, bringing it more accessible. Research works on automatic arithmetic optimization tools have therefore existed for some time, in several areas such as high level synthesis [6], [7] and logical synthesis [8]–[11]. Our work, based on pattern matching techniques, consists in studying the chains of arithmetical operators, and proposing general rules for optimization. High computational digital circuits involving signal, image and control processing indeed include arithmetic data paths composed of such kind of chains of arithmetical operators.

This paper presents an optimization tool which, starting from a representation of a circuit in classical arithmetic, introduces redundant operators through the analysis of the arithmetical operators chains by applying local transformations on the circuit. Furthermore, several kind of optimizations are presented as we aim at proposing a flexible technique. The automatic optimizations of a filter and a DCT generators show the results obtained with this tool. The remainder of this paper is organized as follow: Section 2 contains a global description of the redundant arithmetic and its main advantage. In Section 3, we describe our redundant optimization tool. Section 4 presents the results of our experiments. Finally, in Section 5, possible improvements are listed and then we conclude.

II. REDUNDANT ARITHMETIC

A. Number representations

Redundant arithmetic involves two number representations [12]:

- **Carry-Save representation**: A digit is defined by $cs_i = cs_i^0 + cs_i^1$ with $cs_i \in \{0, 1, 2\}$ so that a number is considered as the sum of two terms: $CS = CS^0 + CS^1$
- **Borrow-Save representation**: A digit is defined by $bs_i = bs_i^0 - bs_i^1$ with $bs_i \in \{-1, 0, 1\}$ so that a number is considered as the subraction of two terms: $BS = BS^0 - BS^1$

The abbreviations CS and BS are commonly used for Carry-Save and Borrow-Save representations, as well as NR and R for classical (Non Redundant) and Redundant representations.

B. Mixed arithmetic

The arithmetic used is called **mixed arithmetic**, defined as the combination between classical and redundant representations. The sole use of redundant arithmetic is indeed not conceivable for several reasons:

- Firstly, we must preserve the NR representations of the inputs/outputs of the circuits
- Secondly, we have to deal with operators which are not arithmetic operators such as multiplexors, boolean operators, etc ...

Data Path Optimization using Redundant Arithmetic and Pattern Matching

Sophie Belloeil, Roselyne Chotin-Avot, Habib Mehrez
University Paris VI, LIP6/SOC Laboratory, 4, place Jussieu, 75252 Paris Cedex 05, France
Email: sophie.belloeil@lip6.fr

Index Terms— automatic, optimization, redundant arithmetic, pattern matching, CAD tool
In order to make those two arithmetics compatible, we have at disposal all kind of arithmetic operators accepting both redundant or non redundant inputs/outputs. We also are able to convert one representation to the other. For example, the conversion from a CS representation to a NR representation is the addition between the two terms composing the redundant number.

C. Architectures

1) Adder: The architecture of a mixed adder (adding a CS number and a NR number) is shown in Figure 1.

A similar architecture exists for a redundant adder (adding two CS numbers). Both adders provide a CS output made of the effective sum and the carries (Output = S + C).

![Fig. 1. Implementation of a mixed adder](image)

The architecture presented shows the main advantage of redundant arithmetic: it allows to suppress the carry propagation in the computation of an addition.

2) Multiplier: The principle of multipliers’ architectures is shown in figure 2.

Those architectures are divided into four parts:
- Recoding (optional): Recoding of the inputs in R form
- Partial Products + Sum (mandatory): Effective computation of the multiplication
- Conversion (optional): Conversion of the output in NR form

![Fig. 2. Implementation of a multiplier](image)

As we can see, the output of the Wallace tree is in CS form, so authorizing the output of the multiplier to be in a CS form allows the deletion of a CS to NR converter. CS multipliers are therefore bound to have a better critical time than classical multipliers, but a bigger area because of the input recoders (especially with two R inputs).

D. Advantage of redundant/mixed arithmetics

The main drawback of classical arithmetic is the carry propagation which is, indeed, the main issue for critical paths. Therefore, many researches have been done in order to minimize the carry propagation time in addition computation, such as carry-skip, carry-lookahead or parallel prefix adders [13]. All those architectures manage to minimize the carry propagation time, but do not suppress carry propagation. Furthermore, the shorter the carry propagation time is, the bigger the area is.

Whereas removing carry propagation is not possible using classical arithmetic, using redundant arithmetic allows to suppress it as shown in the mixed adder architecture in Figure 1. Consequently, the time to perform an addition of two numbers is constant and independent of the number of digits. Besides, the area of redundant and mixed adders is smaller than most classical adders area. Addition being an essential operator, the potential benefit of using redundant and mixed adders is significant for timing and for area. This benefit represents the main advantage of redundant arithmetic.

III. AUTOMATION TOOL

A. Aim

The aim of our CAD tool is to take advantage of redundant arithmetic by introducing automatically redundant operators in the process of circuit design.

B. VLSI design flow

Our CAD tool is part of a classical VLSI design flow as shown in Figure 4 and takes place just before logical synthesis. It is part of the Coriolis platform [14] and its input format is therefore made in Stratus, the procedural netlist description language [15] of this platform. From this functional specification of a circuit and a knowledge in arithmetic, we obtain an optimized virtual description (i.e. before structural mapping) of the circuit. The tool modifies the specification of the different operators and interconnections, while ensuring the feasibility of the mapping toward a target technology. After this process, the VLSI design flow remains unchanged.

C. Pattern matching

1) Pattern: A pattern is a collection of arithmetical operators (typically two or three operators) and their interconnections.

1 formerly named Tsunami
2) **Principle:** The aim of the tool is to search for patterns which are bound to be replaced by more competitive ones. The new patterns are composed of redundant operators, which make them more optimized than the previous ones, in terms of timing and area.

3) **Rule:** The pattern couples are called **rules**.
   - The fundamental assumption is that a pattern and its substitute have the same behavior and the same inputs/outputs.
   - The second assumption is that the substitute pattern of a rule is more optimized than the one it is substituted to.

4) **Set:** We have chosen to deal with patterns that contain chains of two operators, because working on bigger chains would become complicated to handle without leading to better results. A set of twenty-four rules has therefore been established in order to be sufficient to handle most cases of connection between adders and multipliers. With a smaller set, results might not be optimal. In opposition, a bigger set would not lead to better results, and would make tough the choice of an optimal rule if several patterns matched.

5) **Evaluation:** Each pattern has been evaluated in terms of timing and area in order to verify that the substitute redundant pattern of each rule has better performances than the one it is substituted to. The architectures we used for adders and multipliers are [12], [13]:
   - NR adder and CS/NR converter: Sklansky algorithm
   - NR/R multiplier: Booth modified algorithm

The results obtained with the rules presented are shown in Table I which outlines the performances of the two patterns of each rule, in terms of timing and area, before and after optimization. Those results tend to prove that an optimization with redundant operators can improve timing and area. We can see that the improvements are up to 64.5% for the critical path, and 62.5% for the area. We can also notice that, in most cases, the bigger the number of digits is, the better the improvements are.

The evaluation of each rule has shown that, in every rule, the timing is improved with the substitution of the pattern. As for the area, only one kind of rules alters it: the ones that introduce redundant multipliers. As already discussed, those operators are indeed big. Whereas some studies suggest not to use redundant multipliers [6], we have decided to let the designer choose whether or not to use such kind of rules, depending on the goal to achieve: an optimization of the timing with a possible deterioration of the area allowed or forbidden.

6) **Description of the rules:** The rules are described in a configuration file. We have chosen the .xml file format which...
is a quite simple kind of description so that designers can easily add or delete one pattern if they want to.

Therefore:
- On one hand, our tool is suitable for designers who want to use automatically redundant arithmetic without having the needed arithmetic knowledge.
- On the other hand, the simpleness of the patterns’ description allows designers who have the arithmetic knowledge to modify the list of patterns.

One can consequently notice one of the advantages of using the proposed pattern matching approach: compared to other solutions, this approach is very modular and our tool can be dedicated to basic optimization, it can as well be a testing platform for new architectures for example.

### D. Algorithm

Based on the pattern matching approach, our algorithm is a recursive process which skims through a model with non redundant operators, from its outputs to its inputs, and introduces redundant operators by doing local transformations. This bottom-up approach is inspired by [9]. This algorithm is illustrated in the example of Figure 7.

```plaintext
Algorithm (net) is
  if (net is an input)
    stop
  while (pattern = SearchPattern())
    Replace (pattern)
    operator = input operator of the net
    foreach input of the operator
      Algorithm (input)
    foreach output of the cell
      Algorithm (output)
```

### E. Multiple operation trees

One important issue to handle is the management of multiple operation trees such as shown in Figure 8.

Let us consider the operation of Figure 8.a:
- A conventional transformation is to optimize each tree separately, which produces a circuit with the minimum area, but a non optimal optimization of the timing (because of an extra cost of CS to NR conversions): This kind of transformation is therefore called Priority to the area and the result is shown in Figure 8.b.
- In order to deal with that problem, another behavior of the tool has been implemented which can be called Priority to the timing: The rule is to treat every expression with a separate tree and without any resource sharing. This generates a circuit with a minimal timing, but an excessive overload as for the area. The result is shown in Figure 8.c.
- One last solution has been implemented in order to be able to make trade-offs between area and timing as discussed in [11]: it allows to optimize the timing while minimizing the area penalty. This behavior is therefore called Trade-off and the result, shown in Figure 8.d, is the most optimal.
In view of modularity, the three possible ways to handle the issue of multiple operation trees are implemented in the tool. Designers can therefore choose which one to use depending on their needs.

![Multiple operation trees](image)

**Fig. 8.** Multiple operation trees

**F. Conclusion**

The innovative parts of our work are:

- Applying a very well known technique in order to convert operators into a redundant form
- Including mixed and redundant adders as well as mixed and redundant multipliers

Furthermore, as already discussed, we mainly aim at modularity and flexibility. Following that idea, two factors are provided:

- Several parameters exist in order to specify which behavior to choose concerning (1) the use of redundant multipliers (2) the way to handle multiple operation trees
- The set of patterns provided relies on our arithmetic knowledge and is bound to provide the best results but designers can nevertheless choose to modify it thanks to the easiness of the description

As for the treatment of subtraction, we have chosen to replace it by adding the negation (using two’s complement) results.

More precisely: (1) concerning the subtraction, that is:

\[ a - b = a + \text{not}(b) + 2 \]

The use of only the CS representation is indeed quite common and leads to good results [6]–[11]. Note that we also consider the possibility of using the BS representation in order to deal with the substraction which could lead to even better results.

**IV. EXPERIMENTAL RESULTS**

First of all, we tested our algorithm on arithmetic computations which are typically used in applications involving signal, image and control processing and are therefore representative of arithmetical blocks which can be instanciated in real benchmarks. Second of all, we tested our algorithm on several benchmarks: four different implementations of a filter and two different implementations of a DCT macro generator.

In order to perform those tests, we used the place and route tools of the Cadence CAD System using the Alliance [16] Standard Cell Library in 0.35 μm. Several tables are going to be presented, showing the results in terms of timing and area, before and after optimization.

**A. Designs with additions, multiplications and subtractions**

1) Transformation of designs with additions: Firstly, we tested our algorithm on summation tree designs of 16 bits operands. The results are summarized in Table II. Those results are quite satisfactory, reducing significantly both timing and area. One can also notice that improvements of area are more effective as the number of operands in expression increases. This is due to the good performances of redundant adders: small area combined with a constant timing.

<table>
<thead>
<tr>
<th>Sum</th>
<th>Area (mm²)</th>
<th>Time (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Before</td>
<td>After</td>
</tr>
<tr>
<td>5 operands</td>
<td>0.08 ref</td>
<td>-37.5%</td>
</tr>
<tr>
<td>4 operands</td>
<td>0.12 ref</td>
<td>-50%</td>
</tr>
<tr>
<td>5 operands</td>
<td>0.16 ref</td>
<td>-56.2%</td>
</tr>
<tr>
<td>6 operands</td>
<td>0.2 ref</td>
<td>-55%</td>
</tr>
<tr>
<td>7 operands</td>
<td>0.24 ref</td>
<td>-58.3%</td>
</tr>
<tr>
<td>8 operands</td>
<td>0.28 ref</td>
<td>-60.7%</td>
</tr>
<tr>
<td>9 operands</td>
<td>0.32 ref</td>
<td>-62.5%</td>
</tr>
<tr>
<td>10 operands</td>
<td>0.36 ref</td>
<td>-63.9%</td>
</tr>
</tbody>
</table>

**TABLE II**

**RESULTS FOR ADDITION EXPRESSIONS**

2) Transformation of designs with a mixture of additions, subtractions and multiplications: Secondly, our algorithm was tested on designs with a mixture of additions, subtractions and multiplications. We used 8 bits operands for those tests. The results are summarized in Table III. Those results show that our algorithm can also be applied to such kind of designs. Both timing and area are again significantly reduced. We can nevertheless notice that the area is less improved for designs with multiplications, due to the big area of redundant multipliers, it is even deteriorated once. We can also notice one degradation of the timing in a design with a substraction: this degradation is due to the adjunction of an inverter in the middle of the arithmetic operators chain, such as shown in Figure 9.

3) Conclusion: Those designs are quite trivial, but they are a first step in order to demonstrate the usefulness of redundant operators. They can show on which kind of designs our algorithm can be applied to. Compared to results exposed in [8]–[10], our results seem to have the same order of magnitude.
the area is better improved with our tool, but the timing less improved, (2) concerning the designs with multiplications, the timing is better improved with our tool (the improvement of the area depends of the exemple) which strenghens our choice of using redundant/mixed multipliers.

B. Filters

In order to test the performances of our tool, we optimized a Finite Impulse Response filter (FIR). Figure 10 presents the architecture of a FIR.

![Architecture of a filter](image)

**Fig. 10. Architecture of a filter**

Given the architecture presented (with no multiple operation tree), we used our optimization tool with the option **Priority to the timing**. We optimised four types of this filter: (1) with four or eight coefficients, (2) with input data and constants encoded on eight or sixteen bits. Table IV shows the results obtained, resulting in highly increased performances in timing and area, compared to the classic architecture implementation. This is because: (1) all multipliers outputs are converted into CS form, which suppresses the CS to NR converters, and (2), all slansky adders are therefore transformed into redundant adders. As a consequence, all arithmetical operators used have a better timing and a better area than the previous ones. Note also that the bigger the number of coefficients is, the better the improvements are, for the timing as well as for the area. Concerning the number of bits, it seems that the timing is better improved as it grows as opposed to the area which is less improved.

<table>
<thead>
<tr>
<th>Type</th>
<th>Area (mm²)</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before</td>
<td>After</td>
<td>Before</td>
</tr>
<tr>
<td>4coef/8bits</td>
<td>0.65</td>
<td>0.46</td>
</tr>
<tr>
<td>4coef/16bits</td>
<td>1.95</td>
<td>1.3</td>
</tr>
<tr>
<td>8coef/8bits</td>
<td>1.32</td>
<td>0.9</td>
</tr>
<tr>
<td>8coef/16bits</td>
<td>3.97</td>
<td>2.94</td>
</tr>
</tbody>
</table>

**Table IV**

**Optimizations of a FIR**

C. Discrete Cosine Transform generator

We optimized a Discrete Cosine Transform generator [4] also. Several algorithms have been proposed in order to compute the 1-D DCT:

- The Loeffler Signal Flow Graph [17] has been widely used: this implementation permits to compute the 1-D DCT of 8 pixels in only one cycle. The corresponding architecture is shown in Figure 12.
- A graph partitionning is possible in order to obtain a rate of 1 pixel/cycle. This architecture is shown in Figure 11.

Both implementations contain multiple operation trees. Those designs are therefore good examples in order to test the performances of the three kinds of options of the tool: **Priority to timing**, **Priority to area** and **Trade-off**.

1) Partitionning graph: Figure 11 shows the architecture of the 1-D DCT partitionning graph.
• **Time:**
  - Improvement of the critical time whatever the option of the tool is
  - Better improvement with options **Priority to the timing** and **Trade-off** because more patterns can be substituted

• **Area:**
  - **Priority to the area:** Improvement of the area
  - **Priority to the timing:** Deterioration of the area because of the lack of resource sharing
  - **Trade-off:** Improvement of the area thanks to the resource sharing

Note that the architecture generated with the option **Trade-off** of our tool is the same one than the hand-coded one presented in [4].

<table>
<thead>
<tr>
<th>Mode</th>
<th>Area (mm²)</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Before</td>
<td>After</td>
</tr>
<tr>
<td>Priority to the area</td>
<td>1.88</td>
<td>1.65</td>
</tr>
<tr>
<td>Priority to the timing</td>
<td>1.88</td>
<td>1.95</td>
</tr>
<tr>
<td>Trade-off</td>
<td>1.88</td>
<td>1.61</td>
</tr>
</tbody>
</table>

**TABLE V**
OPTIMIZATIONS OF THE PARTITIONNING GRAPH

2) **Complete Signal flow graph:** Figure 12 shows the architecture of the 1-D DCT complete signal flow graph.

![Complete Signal flow graph](image)

**TABLE VI**
OPTIMIZATIONS OF THE COMPLETE SIGNAL FLOW GRAPH

<table>
<thead>
<tr>
<th>Mode</th>
<th>Area (mm²)</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Before</td>
<td>After</td>
</tr>
<tr>
<td>Priority to the area</td>
<td>3.96</td>
<td>3.9</td>
</tr>
<tr>
<td>Priority to the timing</td>
<td>3.96</td>
<td>7.4</td>
</tr>
<tr>
<td>Trade-off</td>
<td>3.96</td>
<td>4.1</td>
</tr>
</tbody>
</table>

3) **Conclusion:** The examples of the Discrete Cosine Transform generators corroborate the two following assumptions:
(1) the use of the Carry-Save representation leads to good results, even for benchmarks with several subtractions, and
(2) several multipliers in a benchmark can restrict a lot the reduction of the area. Those two examples contain several operators with multiple fan out also, which leads to the following conclusion as for the different kind of treatments:
• **Time:** Options **Priority to the timing** and **Trade-off** always give a better result than option **Priority to the area**
• **Area:**
  - Option **Priority to the area** always gives a better result than option **Priority to the timing**
  - The result obtained with option **Trade-off** is dependent from the context, it can be better or worse than with option **Priority to the area**

Furthermore, we can conclude that the choice of those options change significantly the results obtained with our tool. It is therefore important to have the control of the option to use in function of the performances to achieve.

V. FUTURE WORKS

A. **Borrow-Save**

The use of the two’s complement for the treatment of the subtraction is common and leads to good results. However, the introduction of an inverter cell is not always optimal: the design of Table III illustrated in Figure 9 is a good example. Therefore, the use of the BS representation can be an interesting idea which needs to be studied in order to improve the results obtained.

B. **Timing and area analysis**

Another idea is to be able to estimate automatically the performances of the patterns in terms of:
• **Area:** addition of all the areas of the operators
• **Timing:** critical path considering the critical paths of the operators
This would improve the modularity of the tool, allowing to automatically evaluate all the patterns. A designer could therefore easily test new patterns. Furthermore, it would improve the perrennity of the tool. The set of rules that have been tried indeed depends on the architectures of the arithmetic operators as well as on the technology used. Our tool could therefore be obsolete because of a new architecture of adder for example. Being able to estimate automatically all rules would avoid that issue. In addition, applying such an evaluation on the circuits to optimize would allow to create new behaviors of the tool such as searching for patterns only while skimming through the critical path. This process would minimize the changes done to the design while ensuring an improvement of the critical path.

C. Optimal solution

The chosen approach is an heuristic based on positive cases (the set of rules). It therefore is a greedy algorithm generating an optimized solution but not necessarily the best one. Considering the time to process the complete signal flow graph of the 1-D DCT in classical arithmetic: 27.5 seconds, the optimizations take from 46.6 seconds to 200 seconds given the behavior chosen as for multiple operation tree. The overcost not being important, the implementation of an algorithm generating all possible solutions and choosing the optimal one can appear as an interesting approach.

D. Treatment of the non arithmetic blocks

Making a specific treatment to non arithmetic blocks can be a good idea for heterogeneous designs containing both arithmetic and non arithmetic operators. A process such as the one shown in Figure 13: duplication of a non arithmetic block, e.g. a multiplexor (in order to use two mixed adders instead of two classical adders), would allow to improve performances.

VI. CONCLUSION

This paper describes an algorithm based on pattern matching techniques, that introduces redundant operators in high computational digital circuits. The idea of using pattern matching techniques in order to find operators that can be redundant is new and leads to good results.

More powerful solutions for this problem have existed for some time, such as including the use of redundant arithmetic in high level synthesis [6], [7]. Those approaches are nevertheless far less flexible than our approach. The advantages of using the proposed pattern matching approach are indeed based on flexibility and modularity as first of all several parameters exist in order to guide the behavior of the tool and second of all the set of rules can easily be modified in order to test new architectures.

Experimental results indicate that our work can be used effectively on several designs with mixture of additions, substraction and multiplication. This encourages us, first of all to investigate the different kinds of improvements which can be applied to this algorithm, and second of all, to create new benchmarks (such as a distance computation unit) on order to corroborate our results.

REFERENCES