Abstract

The efficiency of algorithms are predominantly judged based on the computational complexity i.e the number of arithmetic operations. However, limited attention has been given to actual implementation efficiency and the cost of inherent control structures necessary for the algorithm. We consider in this contribution the implementation efficiency of two computational efficient FFT algorithms with a focus on the power consumption when applied to an OFDMA scenario. We seek to address the question if computational complexity can be used as a metric for power efficient implementations.

1 Introduction

Our objective in this paper is to investigate whether the application of computational efficient FFT algorithms translates into power efficient algorithms in an application that requires only a subset of FFT points to be calculated. Power efficiency is an important metric for battery driven devices and as more advanced and power demanding features are being added to already existing features in mobile devices, there is a need for considering power efficient architectures which is the focus in this paper. Our motivation is the scenario shown in Figure 1, which we term Orthogonal Frequency Domain multiple access, OFDMA, based cooperative relaying. In this scenario, cooperation is established as the mobile devices form’s an ad-hoc short-range cluster that cooperates by exchanging broadcast information from the base station using relaying.

In Orthogonal Frequency-Division Multiplexing (OFDM) the frequency spectrum is divided into a set of mutually orthogonal subcarriers on which data are modulated onto before transmission. In OFDMA, the subcarriers may be assigned to different communication paths as exemplified in Figure 2, where the spectrum are divided into blocks of subcarriers. The primary motivation for considering the scenario in Figure 1 and the use of OFDMA is to obtain power efficiency while maintaining requirements in terms of data rate.

A number of references has addressed the use of computational efficient DFT algorithms that can be used in OFDMA [9] [10]. They are based on [12] by H.V. Sorensen et.al which introduced an algorithm named FFT Transform Decomposition and showed that this algorithm was superior in terms of computation complexity compared to alternative algorithms such as the FFT pruning first derived in [8]. However little has been reported on actual implementation efficiency and the cost of the inherent control structures of the proposed algorithms. The premise is that while
classical evaluation methods for efficient DFT algorithms intended for OFDM systems are based on computational count of additions and multiplications, these measures only gives a theoretical indication of which algorithm is optimal in a given situation, but does not take into account the required control structures needed to manage and maintain timing of input and output data for the computational units in the implementation of the algorithm.

When considering the actual implementation of an algorithm the cost function changes from computational count to algorithmic execution time, hardware utilization and power consumption symbolized in the joint cost function in (1)

$$Cost = Power \times Area \times Time$$  \hspace{1cm} (1)

We study the implementation of two FFT algorithms, namely the split radix algorithm used for calculating a full FFT and the Transform Decomposition algorithm used for calculating a subset of FFT points. We investigate how well the measure of computational complexity compare to power consumption on an FPGA and compare the performance of the split-radix algorithm to the transform decomposition algorithm given in [12]. Our objective is therefore to address the question if computational complexity can be used as metric for power efficient implementations. The remaining of the paper is organized as follows: Section 2 briefly describes our system model. Section 3 reviews the derivation the split-radix algorithm and the transform decomposition algorithm. Section 4 presents the methodology of implementing the algorithms onto a Cyclone III FPGA and in section 5 we discuss the results obtained. Section 6 presents our conclusions.

2 OFDMA and system model

For the application of FFT algorithms, we consider the system model of a downlink OFDMA system illustrated in figure 3. As the focus in this paper is on the mobile station, we assume that the receiver receives a sequence of data-symbols. The data symbols for a given user are obtained by performing the FFT operation on the received signal. As the focus of this paper is the implementation of the FFT block, we will assume perfect synchronization and channel estimation as well no nonlinear effects from the RF section which will compromise the orthogonality of subcarriers and limit the performance of the FFT. This results in a simplified system model shown in figure 4 where ram blocks are the interface blocks to the implemented FFT core and “Start” and “Done” indicate control signals for the FFT calculation.

Figure 3. System model of OFDMA downlink.

Figure 4. System model with principal blocks.

3 Computational efficient FFT algorithms

Many references has dealt with the development of efficient algorithms for the calculation of the DFT given by

$$X[k] = \sum_{n=0}^{N-1} x[n]W_N^{nk}, k = 0, 1, \ldots, N - 1$$  \hspace{1cm} (2)

3.1 Computing the full length FFT

The FFT algorithm known to have the lowest computational complexity is the split-radix algorithm. To the best of the authors knowledge [7] presented the split-radix FFT algorithm with the lowest computational complexity. This algorithm was based on a recursive implementation of the split-radix algorithm proposed by [4]. Unfortunately from the perspective of hardware implementation, recursive algorithms are less suited than iterative solutions. For our purpose we therefore use the Split-radix algorithm given in [4] and [11] for the computation of a full-length FFT. Table 1 gives an overview of FFT algorithm complexity for a 1024 point FFT.

3.1.1 Split-radix FFT algorithm

The split-radix algorithm has been derived in detail in [4] and [11]. For our purpose we consider the split radix FFT based on a radix 2 Decimation-in-Frequency flow graph where it is observed that even and odd numbered DFT points are calculated independently. The Split radix algorithm exploits that by dividing calculations into radix 2 and radix 4 decompositions, see [4] and [11].
The split radix algorithm is derived by noting that computation of even points of a $N$ length sequence can be written as

$$X[2k] = \sum_{n=0}^{N/2-1} (x(n) + x(n + N/2))T_{N/2}^{nk},$$

$$k = 0, 1, \ldots, \frac{N}{2} - 1$$

and the odd points as

$$X[4k+1] = \sum_{n=0}^{N/4-1} (x(n) - x(n + N/2) - jx(n + N/4) + jx(n + 3N/4)).$$

$$X[4k+3] = \sum_{n=0}^{N/4-1} (x(n) - x(n + N/2) + jx(n + N/4) - jx(n + 3N/4)).$$

Repeated decompositions as given by the above equations then constitute the N-point DFT calculation.

The L-shaped butterfly structure for the split radix is shown Figure 5 together with the basic form a FFT butterfly used to perform arithmetic operations in an FFT algorithm.

### 3.2 Computing individual FFT points

For the case where only a subset of FFT points need to be computed, direct computation of (2) might be considered if the number of points is less than $\log_2(N)$. But in general direct computation of (2) is not efficient [11]. Other efficient algorithms for a subset of output points was first reported by [6] and later [8] deriving the FFT pruning for the Radix 2 FFT. A more efficient algorithm was reported in[12] named “Transform Decomposition” based on a Divide-and-Conquer approach. Besides offering higher efficiency in terms of computational complexity, the algorithm is also more flexible with respect to the location of DFT points to be calculated compared to the pruning algorithms. The efficiency together with flexibility makes the Transform Decomposition algorithm attractive and is therefore chosen for computing a subset of individual FFT points.

#### 3.2.1 The Transform Decomposition algorithm

For a detailed derivation of the algorithm we refer to [12]. The derivation assumes that $N$, the length of the DFT is a power of 2 and that $P$, the number of output points that needs to be computed is divisible by $N$.

Then if

$$Q = \frac{N}{P}$$

$$n = Q \cdot n_1 + n_2$$

$$n_1 = 0, \ldots, P - 1$$

and $n_2 = 0, \ldots, Q - 1$ \hspace{1cm} (6)

Then (2) can be re-written as a double sum

$$X(k) = \sum_{n_2=0}^{P-1} \sum_{n_1=0}^{Q-1} x(n_1 Q + n_2) W_N^{n_1 Q + n_2 k}$$

(7)

After some manipulation (7) becomes

$$X(k) = \sum_{n_2=0}^{P-1} \sum_{n_1=0}^{Q-1} x_{n_2} (n_1) W_P^{n_2 k}$$

(8)

With $\{\cdot\}_P$ is modulus $P$.

(8) is recognized as two DFT’s with the square bracket expressing a $P$ length DFT. The efficiency of the transform decomposition algorithm stems from the fact the DFT’s can be computed using e.g the split radix algorithm or any FFT algorithm that works for a $P$ length algorithm. This makes the efficiency of transform decomposition algorithm independent of the location of the FFT points to be calculated as opposed to the FFT pruning algorithms.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Operations</td>
<td>33968</td>
<td>34824</td>
<td>38584</td>
</tr>
</tbody>
</table>

Table 1. Number of nontrivial real-additions and multiplications for split radix and radix 2 FFT algorithms for a 1024 point FFT.

![Figure 5. Example of an L-shaped datapath performing one step in the split radix FFT.](image-url)
3.3 Example of Transform decomposition algorithm

An example of the transform decomposition algorithm is given in Figure 5 for an $N = 16$, $P = 8$ and DFT indexes $k = [3, 10]$.

![Figure 6. Transform decomposition example with DFT indexes 3 and 10 for an $N$ length DFT.](image)

3.4 Computational complexity of efficient DFT algorithms

We now turn to the examination of computational complexity for the two algorithms to provide the basis for later comparison with the power performance. For the split radix algorithm it is seen by observing the L-shaped butterfly in Figure 5 that it consists of two complex multiplications and six complex additions as well as two real additions for each two point butterfly operation. The computational complexity of the transform decomposition algorithm, depends on the subdivision factor $P$ as this defines the $Q$, number of $P$ length DFT’s that has to be computed. The split radix algorithm is used to calculate each $P$ length DFT. The $Q$ times recombination of the these sub-DFT’s add the following complexity

\[ n_{\text{complex add/sub}} = Q \left( n_{\text{length}} (P) SRFFT \right) + (Q - 1) L \]

\[ n_{\text{complex mult}} = Q \left( n_{\text{length}} (P) SRFFT \right) + QL \]  

(9)

Figure 7 provides a comparison of the computational complexity for the computation of a 1024 point DFT with a subset of output points. It is seen that direct computation of (2) quickly becomes inefficient as expected while the transform decomposition provides a complexity reduction compared to a full length FFT. The complexity of the transform decomposition algorithm is shown with $P$ chosen to minimize complexity for a given number of output points.

4 Implementation of algorithms onto a Cyclone III FPGA

As implementation platform, a Cyclone III from Altera was chosen, utilizing the well established Altera Quartus tool chain. The development kit further provide access to physical power measurements and the use of Alteras power estimation tool PowerPlay© [1] [3]. For an initial estimate of system requirement in terms of resource usage on the FPGA we utilize a previous result from [2, table 4] which reported the resource usage for a 1024 point FFT. Comparing that result to the available device ensured appropriate resources for the proposed algorithms.

4.1 Implementation methodology

The applied implementation methodology is based on a Finite State Machine with Datapath approach described in [5] and giving an overall architecture as shown in Figure 8. Implementation of each algorithm will entail the design of a data path where the algorithm calculation units are contained. The control structure consists of a Finite State machine (FSM), which is used to setup the data path to perform the relevant operations on the data and an address generator used to retrieve and store data in memory.

In the following, we will briefly describe the implementation of the datapaths for split radix and the transform decomposition algorithms.
4.2 Implementation of Split radix algorithm

Repeated decompositions of an N length DFT into even and odd numbered sequences following equations 3, 4 and 5 leads to a sequence of L-butterfly’s and the final decomposition using two-point DFT’s, constitutes the datapath to be implemented for split radix FFT algorithm. The structure as a result of the decompositions for a 32 point DFT is shown in Figure 9. Starting at level 0, a block of total length 32 consisting of 8 L-butterflies is calculated. Next at level 1, 4 L-butterflies are calculated giving a total block length of 16. At level 2, three blocks of length 8 starting at addresses 0, 16 and 24 are calculated. Level 3 consists of 5 4-point L-butterflies, and level 4 of 11 2-point butterflies. As may be derived from the figure, the SRFFT consists of log2(N) levels at which a specific number of L-butterfly blocks need to be calculated. To optimize use of calculation units the L-butterfly is done using pipelining as shown in figure 10. The figure shows the scheduling of computations in the L-butterfly, resulting in a total latency of 7 clock cycles. It is noted that all the involved computations are complex and that data not operated on during a cycle is buffered for alignment in time e.g. $y[0]$ and $y[1]$ in figure 10. The generation of twiddle factors has been implemented as look-up tables. The final two-point DFT is implemented as complex addition/subtraction whereas each resulting real valued operation is performed with an execution time of one clock cycle.

4.3 Implementation of the Transform Decomposition algorithm

Referring back to (8), the datapath of the transform decomposition algorithm is the calculation of number, $Q$, DFT’s of length $P$. Each of these $P$-length DFT’s are calculated using the split radix algorithm, and therefore the implementation of the transform decomposition reuses the implementation described above for the split radix algorithm. The transform decomposition requires an additional implementation of the recombination given by (8). For the recombination, we utilize the proposed optimization in [12]. The optimization uses the principles of the Goertzel algorithm [6] by rewriting (8) to

$$y_k[j] = \sum_{m=0}^{j-1} x_{Q-m-1} \left( \{k\}_P \right) T_N^{k(j-m-1)}$$

(10)

The expression in (10) can be viewed as a convolution of $x_{Q-m-1} \left( \{k\}_P \right)$ and

$$H_k(z) = \frac{z^{-1}}{1 - z^{-1}T_N^k}$$

(11)

(11) is implemented as a Direct form II realization shown in figure 11. This implementation offers a reduction in com-
plexity in terms of a reduction of multiplication per iteration. Although the introduction of a feedback in the transfer characteristic creates a potential for instability, the gain in complexity is attractive and the optimization is used in the implementation of the transform decomposition algorithm. The following section presents the results of the evaluation of the algorithms discussed above.

5 Results

The results below has been obtained using the system constraints listed in table 2. The results obtained in section are based on simulations as well as measurements of power consumption. The following section briefly outlines the methodology and the performance measure used to obtain the results. For the implementation and validation of the relation between complexity and power consumption, values of \( Q = 32 \) and \( P = 32 \) is chosen for the transform decomposition algorithm.

### 5.1 Power estimation and measurements

For the simulation of power consumption we utilize the Altera provided Powerplay tool [1] [3, chapter 10]. This tool utilizes device models for estimating power consumption based information from device utilization as well as simulations giving information about the signal activities and environmental settings conditions such as supply voltages and ambient temperature. The FPGA development board used for the implementation also offers the possibility of carrying out measurements on specific device supply voltages. This enables the verification of simulations against actual measurements.

The device power consumption is composed of two contributions, dynamic and static power consumption.

\[
P_{\text{device}} = P_{\text{dyn}} + P_{\text{static}}
\]

Whereas dynamic power consumption is given as [1]

\[
P_{\text{dyn}} = \left[ 0.5CV^2 + I_{sc}V \right] f \cdot a
\]

\( C \) being load capacitance, \( I_{sc} \) being short-circuit current, \( f \) and \( a \) being clock frequency and probability of signal change in a clock-cycle. Finally \( V \) is supply voltage. The static power consumption is written as

\[
P_{\text{static}} = A e^{BT} + C
\]

With constants \( A, B \) and \( C \) being device constants.

To compare simulation results and measurements a measure of mean power defined as dynamic and idle related Energy per OFDM symbol time was defined as

\[
P_{\text{mean}} = \frac{1}{T_s} (E_{\text{active}} + E_{\text{idle}})
\]

### 5.2 Split radix power consumption

The power consumption of a full 1024 point FFT was simulated and measured, the results are shown in table 3. Results are using full clock speed at 50MHz and at 16.67MHz. The justification for reducing the clock speed is the specification of maximum processing time of 0.5ms for the full 1024 point FFT. The results illustrate the effect on power consumption when lowering the clock frequency. It is noted that that simulations seem to underestimate the power consumption.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>System Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame duration [ms]</td>
<td>2</td>
</tr>
<tr>
<td>Full FFT length</td>
<td>1024</td>
</tr>
<tr>
<td>FFT Time [ms]</td>
<td>0.5</td>
</tr>
<tr>
<td>Number of subcarriers</td>
<td>1 - 250</td>
</tr>
</tbody>
</table>

**Table 2. System constraints for FFT size and timing performance.**

<table>
<thead>
<tr>
<th></th>
<th>Simulated [mW]</th>
<th>Measured [mW]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Clock</td>
<td>33.0 mW</td>
<td>38.8 mW</td>
</tr>
<tr>
<td>Reduced Clock</td>
<td>27.3 mW</td>
<td>33.4 mW</td>
</tr>
</tbody>
</table>

**Table 3. Simulated and measured mean power consumption of 1024 Split radix FFT.**
of output points calculated, the simulated results are shown in table 4. Measured power consumption is given in table 5.

\[
\begin{array}{c|c}
\text{n} & P_{\text{mean}} \text{ [mW]} \\
\hline
4 & 39.29 \\
20 & 39.60 \\
60 & 39.79 \\
80 & 39.96 \\
100 & 40.04 \\
120 & 40.27 \\
160 & 40.51 \\
200 & 40.82 \\
248 & 41.29 \\
\end{array}
\]

Table 4. Simulation results for Transform decomposition algorithm. Mean power consumption, \( P_{\text{mean}} \), simulated for each number demodulated subcarriers, \( n \).

The mean value is composed of weighted contributions of idle and active power consumptions given by

\[
P_{\text{mean}} = \frac{t_{\text{active}}}{T_s} P_{\text{active}} + \left( 1 - \frac{t_{\text{active}}}{T_s} \right) P_{\text{idle}} \quad (15)
\]

With \( T_s = 2 \text{msec} \). Again, it is noted that simulation results underestimate power consumption. Furthermore it is noted that idle power consumptions constitutes a considerable portion of the total mean power consumption. Figure 12 compares the simulated as well as measured result of the mean power consumption shown in Table 4. The immediate results that can be concluded from figure 12 is that even though the transform decomposition algorithm offer reduced computational complexity, this advantage does not translate into a decrease in power consumption. Looking at the results in table 5 indicate that the reason for this result should be found in the high idle power consumption. Table 6 presents simulated results for the case of maximum number of calculated FFT points for the transform decomposition algorithm, which as indicated in figure 12 gives the biggest difference in performance for the two algorithms.

\[
\begin{array}{c|c|c|c}
\text{Group} & \text{Power Usage} & \text{Power Usage} \\
& \text{TD FFT [mW]} & \text{SR FFT [mW]} \\
\hline
\text{Dynamic} & 28.45 & 16.24 \\
\text{Static} & 66.39 & 66.33 \\
\text{I/O} & 9.57 & 7.57 \\
\text{Total} & 104.4 & 90.14 \\
\end{array}
\]

Table 5. Measured power consumption for transform decomposition FFT with \( n \) calculated subcarriers

\[
\begin{array}{c|c|c|c|c}
\text{n} & t_{\text{active}} \text{ [\mu sec]} & P_{\text{mean}} \text{ [mW]} & P_{\text{active}} \text{ [mW]} & P_{\text{idle}} \text{ [mW]} \\
\hline
4 & 123 & 45.7 & 45.0 & 56.7 \\
20 & 143 & 45.9 & 45.1 & 57.2 \\
60 & 210 & 46.1 & 44.7 & 57.7 \\
80 & 240 & 46.5 & 45.0 & 57.9 \\
100 & 267 & 46.6 & 44.8 & 58.4 \\
120 & 300 & 46.5 & 44.5 & 58.2 \\
160 & 363 & 47.2 & 44.7 & 58.6 \\
200 & 423 & 47.7 & 44.7 & 58.8 \\
248 & 493 & 48.4 & 45.0 & 58.8 \\
\end{array}
\]

Table 6. Comparison of simulated power consumption between split radix and transform decomposition algorithm for \( N = 248 \) FFT points

Figure 12. Comparison of power consumption between split radix and transform decomposition algorithm

Table 6 gives a summary of power consumption based on a power analysis using the Powerplay tool. The tool reports the consumption in terms of dynamic, static and I/O. While idle and static power consumption is not directly related, it is deemed valid assuming that any clock circuitry is disabled during idle operation to relate the static power consumption to the idle power. Based on the results obtained it seems relevant to consider a power-off state in the design to eliminate idle power.

Eliminating the contribution from idle power in (15), the mean power consumption for the transform decomposition
algorithm given by (15) is modified as shown in figure 13. The result in figure 13 is based on measurements on power consumption assuming zero idle power contribution. The result shows the significance of reducing idle power and in that case the transform decomposition algorithm becomes feasible up to approximate 100 calculated FFT points.

6 Conclusions

We have investigated power consumption of two FFT algorithms implemented on an FPGA. Measurements were compared with power consumption analysis using the Altera PowerPlay® tool. Based on the results it is noted that computational complexity and power consumption is not readily comparable. In order for the computational complexity and power consumption to be comparable it is necessary to consider the effect of idle power. If idle power are assumed eliminated then the two metrics can be considered comparable. Secondly it was shown that minimizing clock frequency contributes to a reduction in power consumption. Then clock frequency and idle power becomes a tradeoff as lowering clock frequency increase the requirement of parallel execution with an increased area as a result, increasing static power consumption. Further design space explorations revealed additional potential for reducing power consumption. The following points summarizes the findings:

- Minimize idle power by disabling clocks and power off idle circuitry
- Minimize system clock to fit constraints
- Minimize area utilization
- Optimize time vs. area trade-off

It is clear that other factors is involved, e.g device cooling, and the results should be verified using other platforms such as DSP and microcontroller to investigate whether the findings are generally applicable.

References