

Ultra-low power FFT Processor Design for IoT Devices

Reza Homainejad

Bachelor of Engineering
Computer Engineering Major



Department of Electronic Engineering
Macquarie University

October 22, 2017

Supervisor: Dr Ediz Cetin

ACKNOWLEDGMENTS


I would like to thank to my supervisor, Dr. Ediz Cetin, who offered great help in the fully understanding of the 64 -point FFT algorithms, and providing me support in the world of signal processing.

Reza Homainejad,

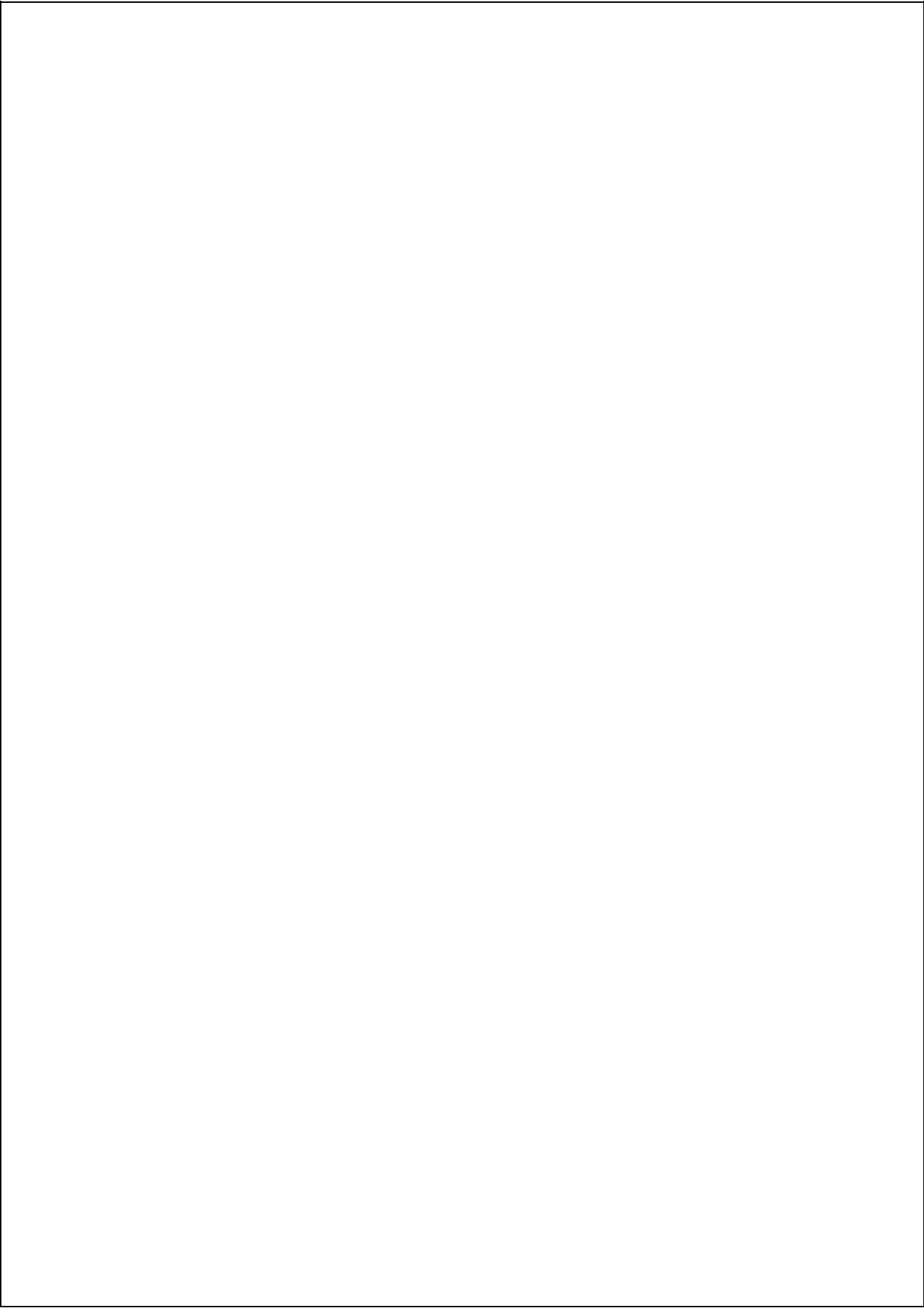
STATEMENT OF CANDIDATE

I, Reza Homainejad, declare that this report, submitted as part of the requirement for the award of Bachelor of Engineering in the Department of Electronic Engineering, Macquarie University, is entirely my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualification or assessment at any academic institution.

Student's Name: Reza Homainejad

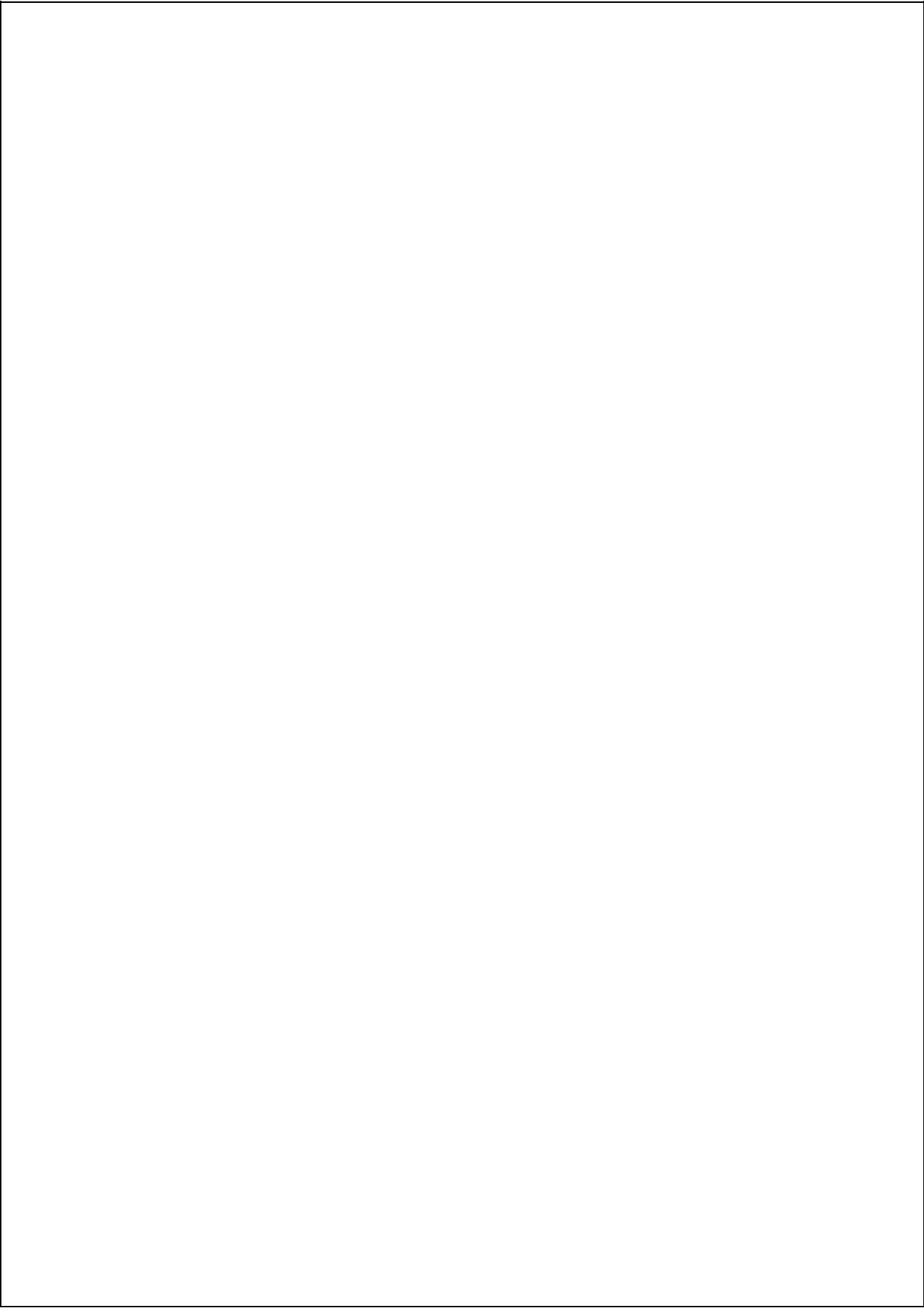
Student's Signature: 

Date: 19/08/17



ABSTRACT

In a world where sensor data can be collected through smart grid and sensor backup applications, reducing power consumption has become one of the most important factors for data transmission. FFT Processing is capable of low power, high throughput sensor data collecting. The IEEE 802.11ah standard is designed with the Physical Layer(PHY) and the Medium Access Control (MAC). This paper discusses developing a FFT processor for this new standard. A simulation of the FFT processor is created in MATLAB, along with a VHDL component that handles the FFT architecture. These 2 are then tested for competency and validity, before then tested each other for evaluation and results.



Contents

| | |
|-----------------------------------------------------------------------------------------------------------------|-----|
| ACKNOWLEDGMENTS..... | iii |
| STATEMENT OF CANDIDATE..... | v |
| Contents | ix |
| Chapter 1 Introduction..... | 1 |
| Chapter 2 The characteristics of the IEEE 802.11ah WiFi Standard and Its requirements for A FFT Processor | 2 |
| 2.1 The IEEE 802.11 | 2 |
| Chapter 3 Fast Fourier Transform: The Algorithms and Architecture That Define It..... | 5 |
| 3.1 Introduction..... | 5 |
| 3.2 The Discrete Fourier Transform..... | 5 |
| 3.3 The Fast Fourier Transform | 6 |
| 3.2.1 Decimation-in-time and Decimation-in-frequency FFT Algorithms | 6 |
| 3.2.2 Higher Radix Algorithms..... | 12 |
| 3.3 FFT architecture types | 13 |
| 3.3.1 Memory Architecture | 13 |
| 3.3.2 Cache Memory Architecture | 15 |
| 3.3.3 Array Architecture..... | 16 |
| 3.3.5 Pipeline Architecture..... | 16 |
| 3.4 Conclusion..... | 19 |
| MATLAB and VHDL Implementation..... | 22 |
| 4.1 MATLAB Implementation | 22 |
| 4.2 VHDL Implementation..... | 23 |
| 4.2.1 The radix-2 SDF Data flow..... | 23 |
| Conclusion | 27 |
| Chapter 5 | 28 |
| Verification and Evaluation..... | 28 |
| 5.1 Verification | 28 |

| | |
|-----------------------------------------------|----|
| Testbench..... | 28 |
| MATLAB Results..... | 28 |
| 5.2 FPGA Results | 30 |
| 4.3 Evaluation | 32 |
| 4.3.1 Circular Buffer..... | 32 |
| 4.3.2 Different Adders | 33 |
| Conclusion | 35 |
| Conclusions and Future Work..... | 36 |
| Abbreviations..... | 38 |
| Appendix A MATLAB Code of FFT Processer | 39 |
| A.2 Generalised-point radix-2 FFT code..... | 39 |
| Appendix B: Consulation Forms | 42 |
| Bibliography..... | 44 |

List of Figures

| | |
|-----------------------------------------------------------------------------------|----|
| Figure 1: 8-point DIT FFT, based on [14]..... | 9 |
| Figure 2: 8 point DIF FFT, based on [14] | 11 |
| Figure 3: Single Memory Architecture of a FFT, Based on [5]..... | 13 |
| Figure 4: Dual Memory Architecture of the FFT. Based on [5]..... | 15 |
| Figure 5: FFT Cache Architecture, based on [5] | 16 |
| Figure 6 Array FFT architecture, based on [5] | 16 |
| Figure 7: Pipeline FFT Architecture, based on [5]..... | 17 |
| Figure 8 The R2SDF | 18 |
| Figure 9: The Butterfly processing block of the FFT Processor..... | 24 |
| Figure 10 A full Adder Block..... | 24 |
| Figure 11: A representation of the complex multiplier..... | 26 |
| Figure 12: FFT algorithm vs Builtin for real values. | 29 |
| Figure 13: FFT algorithm vs Builtin for complex values..... | 29 |
| Figure 14: Value Difference between the FFT algorithm and the MATLAB builtin..... | 30 |
| Figure 15 Modelsim Simulation results of the FFT Processor..... | 31 |
| Figure 16 Modelsim Simulation results of the twiddle factor multiplication..... | 31 |
| Figure 17 A representation of a circular buffer..... | 33 |
| Figure 18 Area Comparison of different Adders, from [9] | 34 |
| Figure 19 Latency comparison of different Adders, from [9] | 34 |

List of Tables

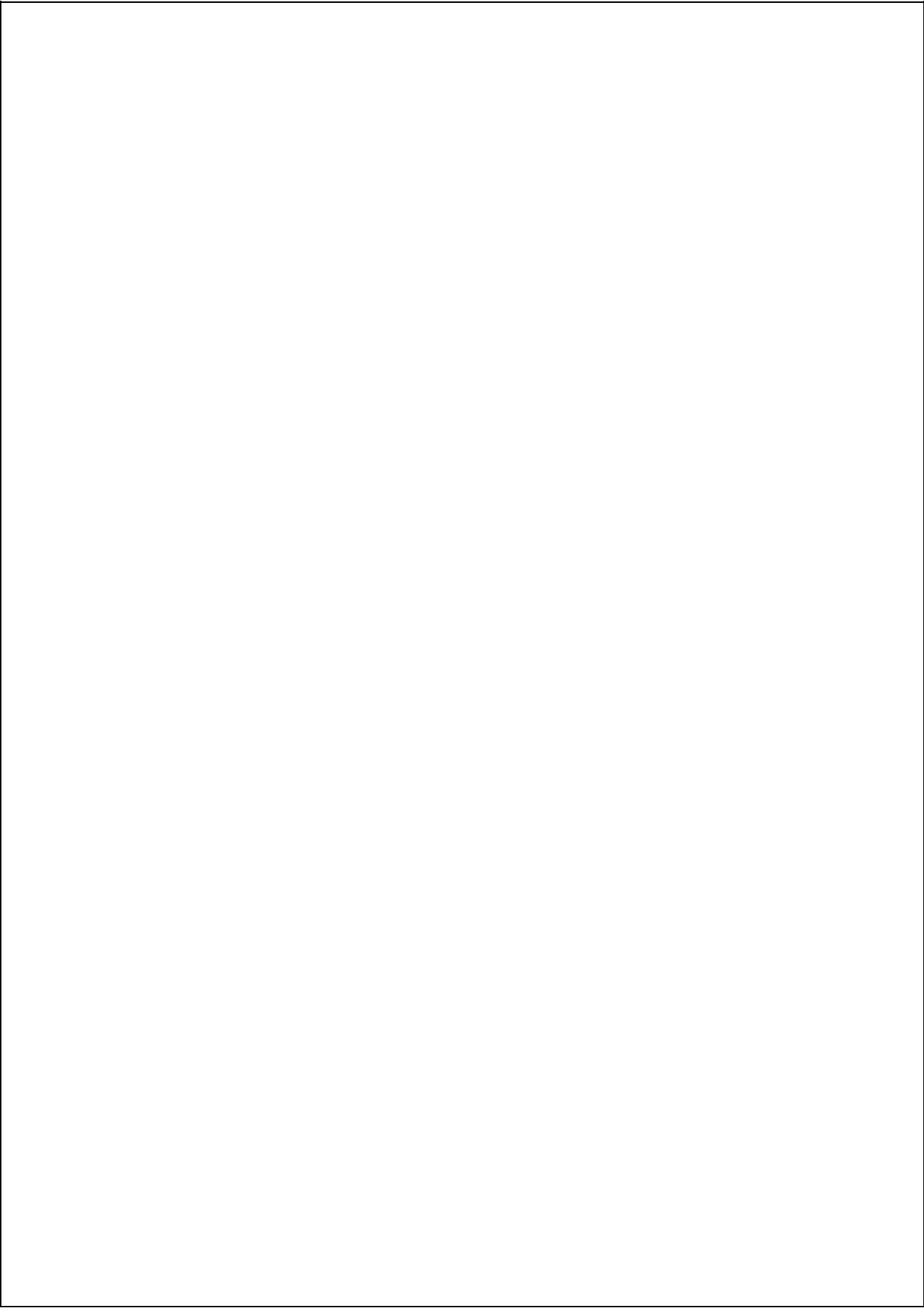
Table 1: IEEE 802.11ah parameters.....3

Table 2: Comparison of Overall Noise Power for DIF FFT, based on [8]12

Table 3: Comparison of Overall Noise Power for DIT FFT, based on [8]12

Table 4: FFT Pipeline Architecture Design analysis..... 17

Table 5: Resource usage of Virtex4.....32



Chapter 1 Introduction

Current technology is advancing to the point where Sensor data and long-range communications between two machines are becoming more plausible to achieve. The technology for the Internet of Things(IoT) is becoming more sophisticated with the advent of newer technologies and better Wireless Network Standards. The biggest problem, though, is that the technology itself must become more and more energy efficient. Several solutions have been made to provide ultra-low power, power conserving solutions.

The Project will be discussing on implementing a FFT processor on the IEEE 802.11ah standard for IoT communications. Reducing power consumption while dealing with data transmission is essential for the FFT processor to operate competently.

The report is separated into 6 chapters. Each chapter will talk in detail into important portions of the project. Chapters 2 and 3 will describe in detail the background of the IEEE 802.11ah standard, as well as FFT architecture and history.

Chapter 4 will talk in detail regarding the implementation of a FFT processor simulator in matlab, as well as a FFT processor with a specific architecture in VHDL.

Chapter 5 will describe the verification process used to validify the FFT processor in MATLAB and in VHDL, and provides improvements to the current design to make it more power efficient.

Finally, the project will conclude with a brief paragraph on the future work of FFT.

Chapter 2 The characteristics of the IEEE 802.11ah WiFi Standard and Its requirements for A FFT Processor

This section of the report will briefly talk about the IEEE 802.11 standard. It will specifically talk more about the properties of the IEEE 802.11ah standard, its properties and uses, and what it is required to build a FFT processor for that standard.

2.1 The IEEE 802.11

The Wifi Standard, or IEEE 802.11, is a set of specifications for MAC and PHY material to be implemented in Wireless Local Area Network(WLAN)[10]. The Institute of Electrical and Electronics Engineers(IEEE) Standards Association maintains every current version of the IEEE standard, and archives them when they're superseded. The IEEE carry standards for the 900MHz, 2.4, 3.6, 5, and 60 GHz frequency bands [16].

One common example of the IEEE 802.11 standard that is used commercially is the IEEE 802.11ac. Designed for the 5 GHz band, the 802.11ac carries considerable throughput. This project will delve further into another specification standard, the IEEE 802.11ah.

2.2 The IEEE 802.11ah Standard

The IEEE 802.11ah, or Ha-Low, is a standard for the wireless network (WLAN). Designed by the 802.11ah task group, the scope of IEEE 802.11ah is to enhance Medium Access Control(MAC), and Physical layer design (PHY) [10]. The system has been standardized as of 2015, and global Industrial, Scientific, and Medical(ISM) bands have been established.

The Ha Low carries substantial advantages, as explained by Aust. Et al [11]:

“

- Longer range and less power consumed due to optimal propagation characteristics below 1 GHz.
- No licensing and regulatory issues (ISM band).
- License-exempt in various different countries.
- Almost clear co-existence issues.
- Easy to understand, follow and to implement for network device manufacturers.
- Enrichment of current wireless communication devices, e.g., IEEE 802.11a/b/g/n.”

Aust Et al., and Sum et al mention the uses of the HaLow Standard, mainly smart grid and smart utility applications, sensor networks, backhaul networks for servers, rural communication, and machine-to-machine(M2M) communication [10] [11].

The currently allocated bands that use the HaLow standard in Australia is the 918-926 band, determined by the Australian Commissions and Media Authority [16].

Table 1: IEEE 802.11ah parameters

| System Parameters | IEEE802.11ah |
|-------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| Number of required FFT Points | 32 |
| Radio Frequency | 918-926(Australia), 917.5-923.5 (South Korea), 916.5-927.5 (Japan), 863-868 (Europe), 755-787 (China), 866-869 and 920-925 (Singapore),902-928 (US) |
| Channel Bandwidth(MHz) | 1-16 |
| Modulation Type | BPSK, QPSK, 16QAM, 64QAM, 256QAM |
| Number of Subcarriers | 26 |
| Data Rate | 17.7 Mbps (256QAM, $N_{ss} = 4$) |

2.3 The Requirements of the FFT Processor for the IEEE 802.11ah Wifi Standard

Based on the table in appendix A given by Aust et al [11], the FFT processor needs to follow a set of requirements so it can be a hub for sensor transmission. A good example of FFT requirements is shown through N. Li's Thesis, which in its second chapter talked about the specifications of the FFT for the Multi-Band Orthogonal Division Multiplexing (MB-OFDM) Ultra-Wide Band(UWB) standard [9]. Much of the requirements presented here is based on the draft of the IEEE 802.11ah standard. The radio frequency of the ISM bands is between 918 and 926 [13], so a sampling frequency of 922 MHz is required. There are 24 data subcarriers, and 2 pilot subcarriers, totalling up to 26 subcarriers. This means the FFT size in this case must be around 32 points. The period for this FFT must be 34.7 07 nano seconds. This is calculated by inversing the sample frequency given f_s .

Chapter 3 Fast Fourier Transform: The Algorithms and Architecture That Define It

3.1 Introduction

The following section will provide an overview of the FFT, the algorithms that make up an FFT Processor and the terminology used internationally, the different types of FFT that are more often used, and the architecture of the FFT Processors.

3.2 The Discrete Fourier Transform

The Discrete Fourier Transform Formula is defined as:

In a sequence of N -points, the DFT of such a sequence is defined as a sum of all points multiplied by their twiddle factor components, or:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-\frac{j2\pi nk}{N}}, \quad k = 0, 1, 2, \dots, N-1 \quad (1)$$

$X(k)$ and $x(n)$, for general purposes, are complex equations containing a series of values. n and k are integers.

$e^{\frac{j2\pi nk}{N}}$ is most often represented as the Twiddle Factor, also represented as the symbol W_N^{nk} . It is described as:

$$W_N^{nk} = e^{\frac{j2\pi nk}{N}} = \cos\left(\frac{2\pi nk}{N}\right) - j\sin\left(\frac{2\pi nk}{N}\right) \quad (2)$$

W_N can be replaced with the first equation, leading to:

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{nk} , \quad k = 0, 1, 2, \dots, N-1$$

DFT can take any number of points and create a result based on the twiddle factor and the value of $x(n)$. The Twiddle factor, also known as the primitive n th root of unity, can also take any N points. However, because of this, the same amount of adders and multipliers is required.

3.3 The Fast Fourier Transform

A DFT and FFT, functionality wise, provide the same result. However, a DFT requires N^2 operations, while an FFT requires $n \log N$ operations. The Fast Fourier Transform, devised in 1965 by James Cooley and John Tukey, is an efficient method for calculating DFT [1]. It was based on I. J. Good's algorithm for a Fourier series, which multiplies a N -vector by an $N * N$ matrix into m matrices, where m is proportional to $\log N$. The algorithm Cooley and Tukey formulated provided an optimization of redundancy, and the pattern of redundancy.

Since the article's release, there have been designs based on Cooley and Tukey's work [1] that implement the FFT formula and making them more efficient. Different algorithms were implemented for computing the DFT, but they provide the same result. Some examples of this are shown below.

3.2.1 Decimation-in-time and Decimation-in-frequency FFT Algorithms

FFT Algorithms, in general, tend to reduce the amount of computation needed to perform DFT algorithms. This is done by allocating stages of the previous *record* of the DFT into a new set of allocated DFTs. There are two different algorithms that handle the allocation: Decimation-in-time FFT, and Decimation-in-frequency FFT. For the purposes of the following examples, the difference will be shown through radix-2 8-point FFT figures.

Decimation-in-Time FFT Algorithms decompose the pre-allocated sequence $x(n)$ into smaller sub-sequences of even and odd parts. Based on the process shown in [14], Since:

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{nk}, \quad k = 0, 1, 2, \dots, N-1$$

Then we can separate this sequence into two parts, the even part and the odd part.

$$\begin{aligned} X(2m) &= \sum_{n=0}^{N/2-1} x(2m)W_N^{2mk} + \sum_{n=0}^{N/2-1} x(2m+1)W_N^{(2m+1)k}, \quad k = 0, 1, 2, \dots, N/2-1 \\ &= \sum_{n=0}^{N/2-1} x(2m)W_N^{2mk} + W_N^k \sum_{n=0}^{N/2-1} x(2m+1)W_N^{2mk} \end{aligned}$$

Where $x(n)$ is divided into the even sequence $x(2m)$, and the odd sequence $x(2m+1)$.

The twiddle factor is affected by this, as:

$$W_N^2 = e^{\frac{4\pi i}{N}} = e^{\frac{2\pi i}{\frac{N}{2}}} = W_{\frac{N}{2}}^2$$

Because of this, the final equation for a DIT FFT algorithm is:

$$X(k) = \sum_{n=0}^{N/2-1} x(2n)W_{\frac{N}{2}}^{nk} + W_N^k \sum_{n=0}^{N/2-1} x(2n+1)W_{\frac{N}{2}}^{nk}, \quad k = 0, 1, 2, \dots, N/2 - 1$$

Since each figure is divided into 2 parts, the amount of complex additions and multiplications for calculation that is needed to operate the algorithm is $O\left(\left(\frac{N}{2}\right)^2\right)$.

The DIT FFT algorithm requires $O(N)$ complex multiplications and additions for each once divided N-point, leaving with a complex multiplication and addition requirement of $N + 2 * \left(\frac{N}{2}\right)^2$ operations. Compared to the amount of operations for a standard DFT calculation, the DIT FFT is 50% more efficient.

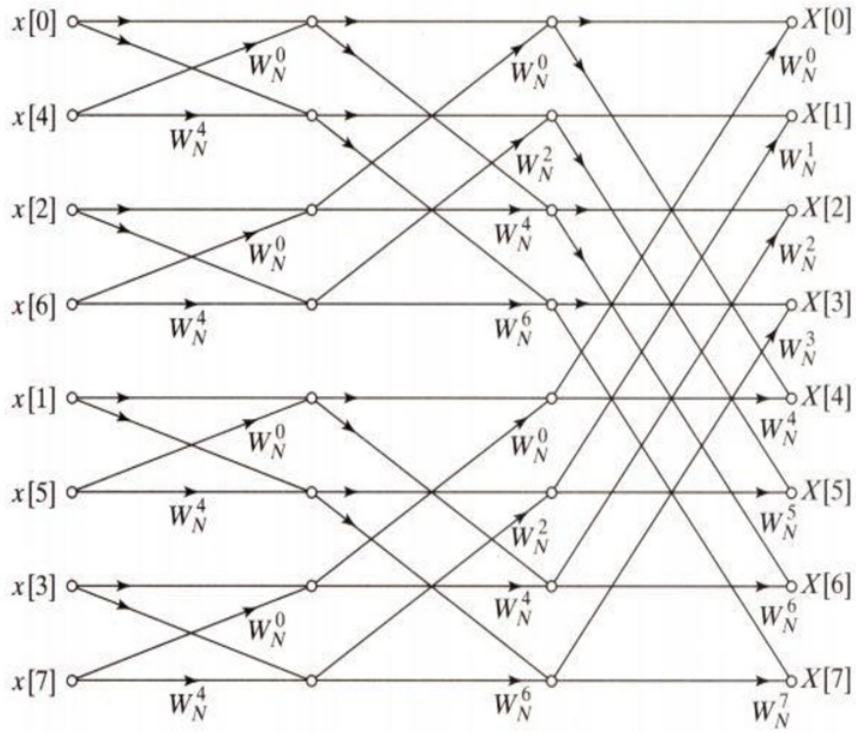


Figure 1: 8-point DIT FFT, based on [14]

From the example shown above, the amount of operations of a radix-2 N-Point FFT is $O(N \log N)$, which is less than DFT's $O(N^2)$ operations. Overall, the multiplication is done before the additions. The inputs in this case is bit-reversed in the input sequence, while the output sequence is in normal order.

DIF FFTs perform their algorithm by dividing the outputs of each DFT into a smaller sequence.

To better show this equation in action, and based on [14]:

$$X(2m) = \sum_{n=0}^{N-1} x(n) W_N^{2mn}, \quad k = 0, 1, 2, \dots, N-1$$

$$= \sum_{n=0}^{N-1} x(n)W_N^{2mn} + \sum_{n=\frac{N}{2}}^{N-1} x(n)W_N^{2mn}$$

Like the DIT formula, the formula can be altered. As shown below:

$$X(2m) = \sum_{n=0}^{N-1} x(n)W_N^{2mn} + \sum_{n=0}^{N-1} x\left(n + \frac{N}{2}\right)W_N^{2m\left(n+\frac{N}{2}\right)}, \quad k = 0,1,2 \dots, N-1$$

$$\begin{aligned} &= \sum_{n=0}^{N-1} x(n)W_N^{2mn} + \sum_{n=0}^{N-1} x\left(n + \frac{N}{2}\right)W_N^{2mn} \\ &= \sum_{n=0}^{N-1} \left(x(n) + x\left(n + \frac{N}{2}\right)\right)W_N^{2mn} \end{aligned}$$

DIF's amount of operations is the same as DIT's amount, however it performs the additions before the multiplications. Not only that, but the inputs in DIF are in normal order, while the outputs are to be allocated in bit-reverse order.

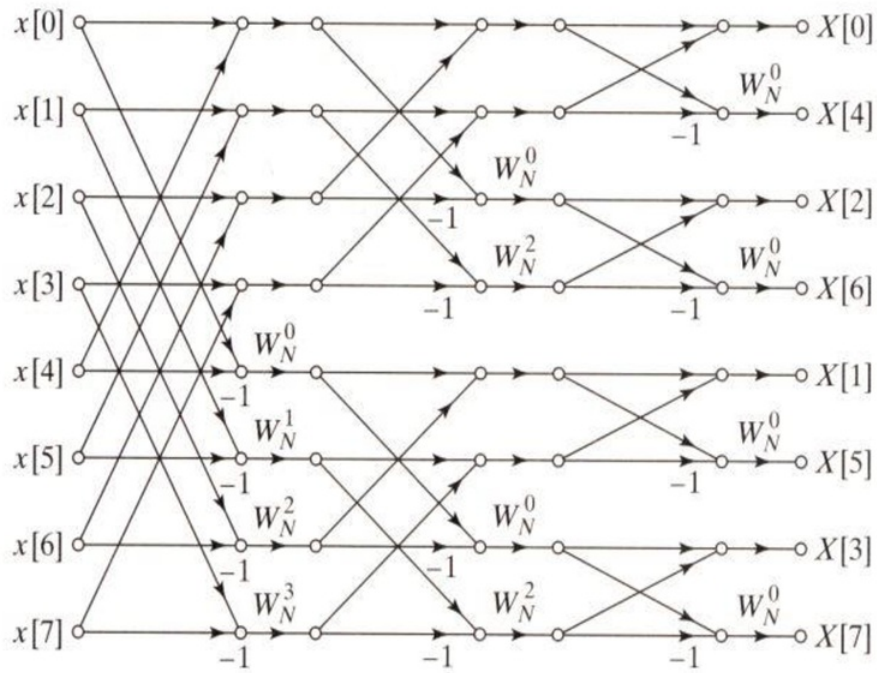


Figure 2: 8 point DIF FFT, based on [14]

Both DITs and DIFs have common requirements for the algorithms to function: they both require bit-reversal sequences, and require a set amount of adders and multipliers in each stage. However, one of the biggest differences between DIT and DIF algorithms is with quantization. When the input involves a large amount of values, quantization refers to the process of mapping those inputs into a smaller, but more countable set. However, Signal-to-Quantization-Errors(SQNR) occur while the algorithm goes through this.

Wei-Hsin Chang and Truong Q. Nguyen tested quantization errors and FFT algorithms in [8], where they compared DIT and DIF FFT algorithms and tested them with algorithmic SQNR to see how many errors are given based on the algorithm and N-point. The results show that in most scenarios, a DIT FFT algorithms carries less quantization errors than DIF FFT.

Table 2: Comparison of Overall Noise Power for DIF FFT, based on [8]

| FFT Size | Radix-2 | Radix-4 | Split-Radix |
|----------|---------|---------|-------------|
| 8 | 8 | 4 | 4 |
| 16 | 64 | 32 | 28 |
| 32 | 352 | 176 | 148 |
| 64 | 1664 | 832 | 684 |

Table 3: Comparison of Overall Noise Power for DIT FFT, based on [8]

| FFT Size | Radix-2 | Radix-4 | Split-Radix |
|----------|---------|---------|-------------|
| 8 | 4 | 8 | 8 |
| 16 | 28 | 32 | 64 |
| 32 | 140 | 208 | 200 |
| 64 | 620 | 688 | 840 |

3.2.2 Higher Radix Algorithms

A data sequence in FFT algorithms by default go divides a sequence and generates samples of N to the power of 2. A radix-4 FFT, then, would divide the sequence and generate samples of N to the power of 4. The amount of twiddle factors is also increased, as in the case of a DIT radix-4 FFT algorithm:

$$X(k) = \sum_{n=0}^{\frac{N}{4}-1} x(4n)W_N^{4nk} + \sum_{n=0}^{\frac{N}{4}-1} x(4n+1)W_N^{(4n+1)k} + \sum_{n=0}^{\frac{N}{4}-1} x(4n+2)W_N^{(4n+2)k} \\ + \sum_{n=0}^{\frac{N}{4}-1} x(4n+3)W_N^{(4n+3)k}, \quad k = 0, 1, 2, \dots, N-1$$

Overall, a radix-4 FFT requires less multipliers, but in return requires a lot more adders. For example, a 16-point FFT would require 3 multipliers and 10 adders if the sample rate was radix-4, whereas for a radix-2 it would take 4 multipliers and 8 adders.

3.3 FFT architecture types

Each FFT architecture can be described as either memory-based, or pipeline based, as told from [7].

3.3.1 Memory Architecture

Memory Architecture, or memory based architecture, uses memory in each stage where it must read and write data. Data exchanges between the memory and the operation are done by a dual serial bus. There are two different types of memory Architecture: **single memory architecture** and **dual memory architecture**.

In the singular memory architecture, data exchanges are taken between the processor and the memory at every stage. One butterfly processing period is

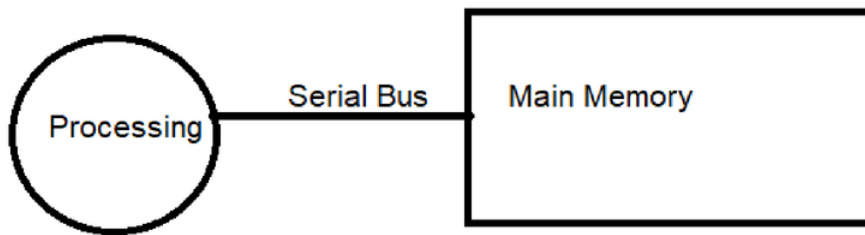
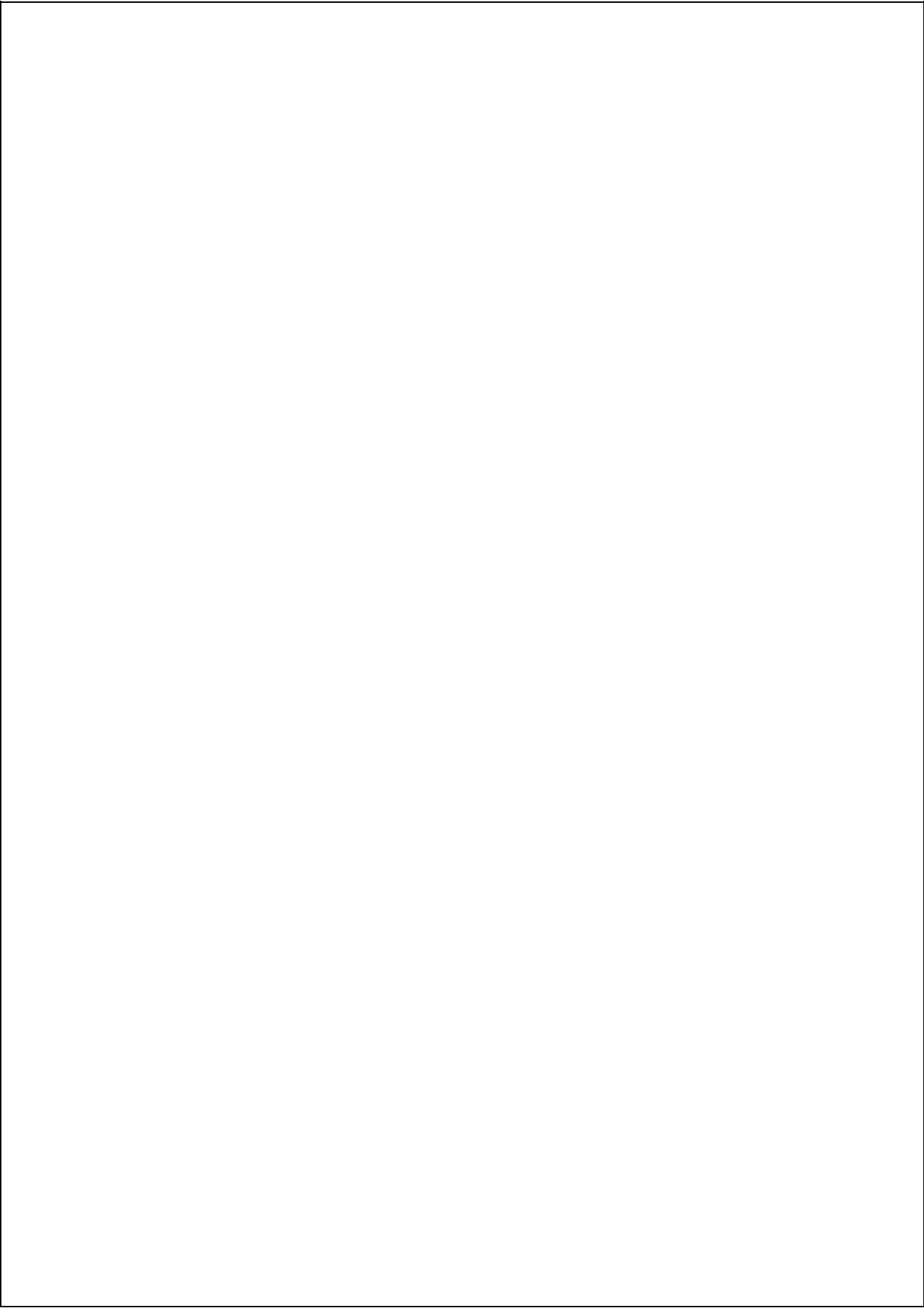


Figure 3: Single Memory Architecture of a FFT, Based on [5]

done for each stage, best represented by figure 3.



In dual-memory architecture, both memory blocks are connected to the main processing block. The data from one memory is processed, and moved to the other memory block. This repeats for each processing stage, and behaves like tennis. A good example of Dual memory architecture is the Honeywell DASP Processor [4].

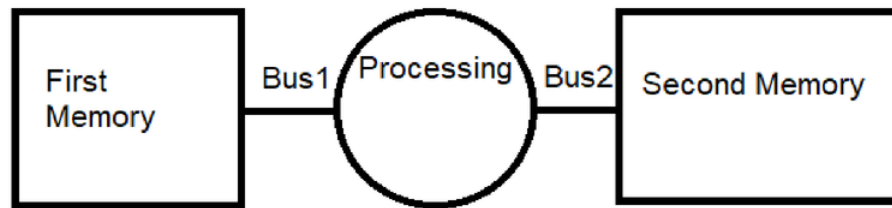


Figure 4: Dual Memory Architecture of the FFT. Based on [5]

While this architecture is the most direct out of the rest, it is also the most power-consuming due to the reliance on memory than anything else.

3.3.2 Cache Memory Architecture

Cache Memory Architecture follows a similar approach to a single memory architecture, in that it includes a step in between the data exchanges. This extra step is the data cache, and it communicates with the processing and with the memory. It is not as popular because FFT processors have very poor locality, and data caches rely on a sufficient locality to carry its efficiency prior to the inception of this design.

Cache Memory Architecture is faster, and is more energy efficient. This makes it better to use than single memory architecture. However, Cache memory architecture carries the flaw in that it creates increased controller complexity. A good example of Cache Memory Architecture B.M. Baas's 1024-point cache memory processor [5], of which he provides the benefits and downfalls of the architecture in detail.

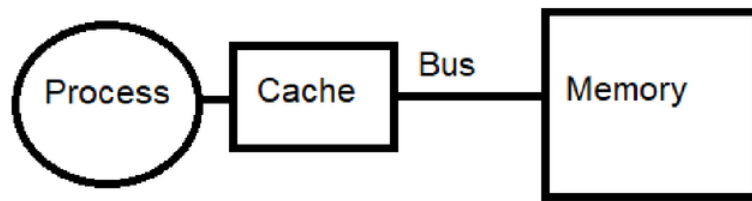


Figure 5: FFT Cache Architecture, based on [5]

3.3.3 Array Architecture

An Array Architecture is one of the most complicated FFT architectures, because it requires an array of processes that carry a local buffer to perform the computations, as shown in Figure 6. This means area space is a factor for the architecture.

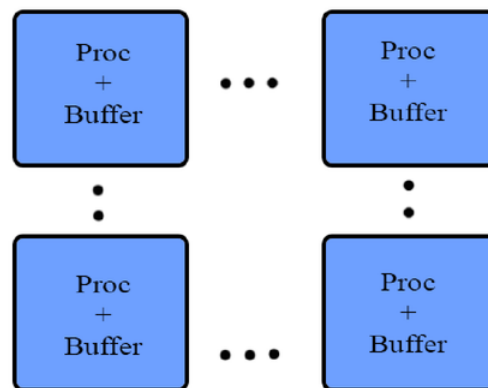


Figure 6 Array FFT architecture, based on [5]

The FFT Processor that uses array architecture is O'Brian Et al's FFT Processor Design, which uses 4 data paths and 4 memory banks on one chip [5].

3.3.5 Pipeline Architecture

Pipeline FFT architecture, shown in Figure 7, revolves around the process and the buffer interleaving several times in a cascading order. Between each stage

there is a communicator, and at the last stage is an un-scrambler. This is also a complicated memory architecture, and isn't as flexible as the other architectures. Pipeline provides high throughput and data efficiency.

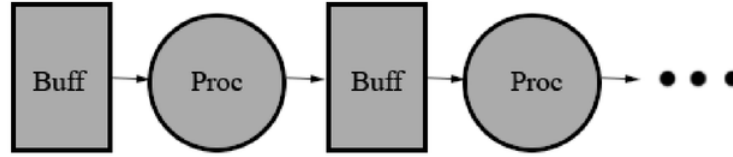


Figure 7: Pipeline FFT Architecture, based on [5]

There are some common FFT pipeline architectures, such as Multipath Delay Communicator(MDC), Singlepath Delay Communicator (SDC), and Singlepath Delay Feedback (SDF). Pipeline Architectures also rely on multipliers as well as butterfly structures, but one thing they have in common is that they take data through a 'pipeline' that allows FFT generation to be quickly produced. A multiplier-butterfly ratio shows the number of multipliers for the FFT to the butterflies, shown below.

Table 4: FFT Pipeline Architecture Design analysis

| FFT Pipeline Architecture | Complex Multipliers | Complex Adders | Memory Size | Control Logic | Efficiency of add/sub and mult blocks. | |
|---------------------------|---------------------|----------------|--------------------|---------------|----------------------------------------|-----|
| R2SDF | $\log_2 N - 2$ | $2 \log_2 N$ | $N - 1$ | Simple | 50% | 50% |
| R2MDC | $\log_2 N - 2$ | $2 \log_2 N$ | $\frac{3N}{2} - 2$ | Simple | 50% | 50% |
| R4SDF | $\log_4 N - 1$ | $8 \log_4 N$ | $N - 1$ | Medium | 25% | 75% |
| R4MDC | $3(\log_4 N - 1)$ | $8 \log_4 N$ | $\frac{5N}{2} - 4$ | Medium | 25% | 25% |
| R4SDC | $\log_4 N - 1$ | $3 \log_4 N$ | $2N - 2$ | Complex | 100% | 75% |

| | | | | | | |
|---------------------|----------------|--------------|---------|--------|-----|-----|
| R2 ² SDF | $\log_4 N - 1$ | $3 \log_4 N$ | $N - 1$ | Simple | 75% | 75% |
|---------------------|----------------|--------------|---------|--------|-----|-----|

R2SDF (Radix-2 Single Delay Feedback)

The radix-2 Single-Delay feedback is a very common FFT Architecture design. In the pipeline process, 1 signal is passed through to one stage of the processing component, where it is stored inside a data register temporarily. This “delays” the processing until enough data has been registered. Once enough data is stored, the processors begin to perform the butterfly processing. The output data is then multiplied with the twiddle factors and then sent to the next stage, or if the data goes through enough stages, is delivered as the output data of the circuit.

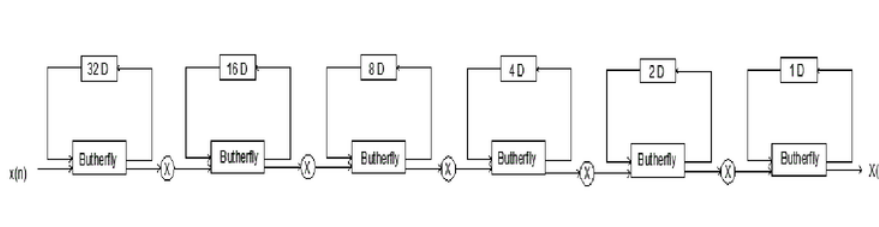


Figure 8The R2SDF

R2MDC (Radix-2 Multi-Delay Communicator)

The radix-2 Multi-Delay Communicator is another type of FFT Architecture. In the pipeline process, the one signal goes through a data register and through a communicator. After enough data is stored, it performs the computation. The R2MDC takes the remaining results of the data to another data register, where it goes through twiddle factor multiplication before going through the next communicator, or until it is finished.

R4SDF (Radix-4 Single Delay Feedback)

The radix-4 single delay feedback behaves similarly in function to the R2SDF, but it provides different results due to using the radix-4 algorithm instead of radix-

2. The Radix-4 algorithm creates more adders while making less multipliers, which is good for handling memory space. This is reflected in the R4SDF design.

R4MDC (Radix-4 Multi Delay Communicator)

The radix-4 Multi Delay Communicator also behaves similarly to the radix-2 counterpart. With more than 1 twiddle factor needed for multiplication, the communicator uses more control counters to toggle different twiddle multiplication when it needs to.

R4SDC (Radix-4 Single Delay Communicator)

The radix-4 Single Delay Communicator differs from the previous architectures. Using a modified radix-4 algorithm, it utilizes most of the multipliers and adders to reduce the memory requirement. While this provides a highly efficient processor, its structure is complicated, making it difficult to recreate.

R2²SDF (Radix-2 Squared Single Delay Feedback)

The radix-2 Squared Single Delay Feedback also differs from the previous architectures. He and Torkelson implemented a radix-2 algorithm, but altered the design so that the data go through different butterfly additions to mimic the radix-4 multiplication algorithm [6]. While this is less efficient than the R4SDC, it is overall a lot easier to reproduce.

For the purposes of this project, R2SDF will be used for the construction of the FFT processor. This is mainly due to the simplicity of its design, and due to the differences between FFT algorithms, this FFT Processor will follow the DIT algorithm for less errors.

3.4 Conclusion

This chapter has briefly reviewed both the algorithms that define the Fourier Transform, as well as the architecture that is used along with it. Radix-2 DIF and DIT algorithms have a simple structure, but carry different complications when quantization comes into play.

FFT architecture is a relevant part of the FFT design process, and each architecture provides its own benefits and drawbacks. Finding the right architecture is relevant for the overall power-saving. However, the application of the FFT algorithm and the architecture will go on in the next chapter.

Chapter 4

MATLAB and VHDL Implementation

With the design of the FFT Processor secured, the next step is to implement the design into MATLAB and into VHDL. In this chapter, a FFT Processor is designed in MATLAB, where it is compared with a built-in MATLAB FFT algorithm. Later, a VHDL implementation of the R2SDF is designed and tested upon, with ideas to improve on its overall design.

4.1 MATLAB Implementation

For the MATLAB implementation, the project follows a set of stages. The validity of this FFT processor is tested with MATLAB's inbuilt FFT operation, $fft(x)$ under a set of N-points that would be normal under the processor's algorithm. This comparison is then graphed to show any differences between the two. The MATLAB code of this FFT processor can be seen in appendix A.

Two arrays exist in the MATLAB implementation: Stage and Twiddlefactor. Twiddlefactor handles all the twiddle factors for each stage, while Stage handles all the data inputs that go through each stage.

4.2 VHDL Implementation

4.2.1 The radix-2 SDF Data flow

The pipeline data flow for the Xilinx VHDL design is described as 5 big blocks along with one final block. These big blocks are the stages of the processor, and they contain a control counter, butterflies, memory stages, and a twiddle multiplier.

The R2SDF structure was taken instead of the other designs is because it is the simplest. Most designs mentioned earlier in chapter 3 require too much power due to the reliance of memory blocks.

The Butterfly processing block.

Figure 11 in the next page shows an example of how one of the stages operates. Essentially, in each stage, the input data $x(n)$ goes through a multiplexer, where it stores the input in one of the shift registers. When the control is 1, the multiplexer function is altered, and takes in the difference between $x(n)$ and the data in the shift register. Likewise, the output is officially initiated when the control is 1.

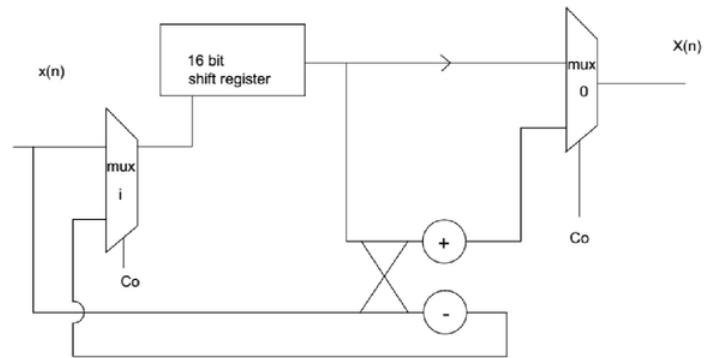


Figure 9: The Butterfly processing block of the FFT Processor

The Full Adder

A full adder has 3 inputs and 2 outputs. In this scenario we call these inputs A_1, A_2 , and Carry in (C_{in}), and we call the outputs Sum and Carry out (C_{out}).

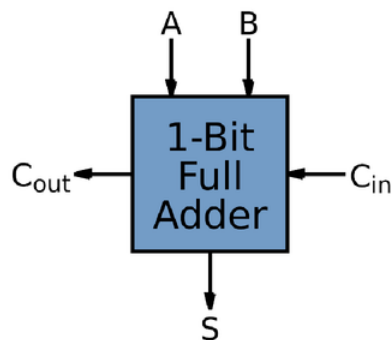


Figure 10 A full Adder Block

The Control Signal

The control signal in the VHDL component is essentially a binary counter. Each part of the binary counter is designated into each stage, acting as its control signal for the purposes of butterfly addition and subtraction, and complex multiplication.

The Shift Register

The complex data is received and is then sent to the shift register, shown in Figure 11. The Shift register is a collection of First In, First Out D-flip flops. It is named as such because at each clock rate, it shifts from one register to the next, up until the end of the register, where it is used for the butterfly process.

The Complex Multiplier

The complex multiplier, shown in figure 12, relies on 3 multipliers, and 5 adders. Normally in a common complex multiplication, we'd get the idea that

$$\begin{aligned}X(n) &= x(n) * W_n^{nk} \\X(n) &= (x_0 + jy_0) * (x_1 + jy_1) \\&= x_0x_1 + jx_0y_1 + jy_0x_1 - y_0y_1 \\R &= x_0x_1 - y_0y_1, I = x_0y_1 + y_0x_1\end{aligned}$$

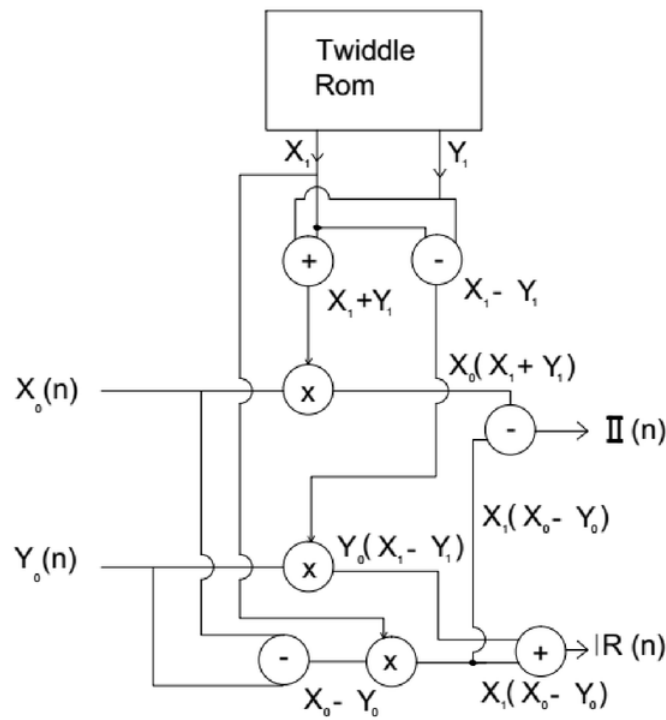


Figure 11: A representation of the complex multiplier

The complex multiplier can be simplified however, into 3 sets of equations, each requiring 1 set of multipliers.

$$\begin{aligned}
 I &= x_0(x_1 + jy_1) - Z \\
 R &= y_0(x_1 - jy_1) + Z \\
 Z &= x_1(x_0 - y_0)
 \end{aligned}$$

This representation is implemented and used in the VHDL component of the complex multiplier.

Due to the nature of the DIT algorithm, a bit-reversal isn't necessary to do at the end, since it assumes all the data at the beginning are already bit reversed.

Conclusion

Chapter 5

Verification and Evaluation

5.1 Verification

In Xilinx, if a given code was written down correctly in a Hardware Description Language, such as VHDL, it needs to be synthesised. Synthesis compiles the data in the VHDL code into a series of gates that connect. This chapter will discuss the power consumption and area space the current FFT design holds.

Testbench

To test whether the VHDL and the matlab FFT are working correctly, both versions are given the same input data. This data is, for the purposes of this project, a sine wave. After both simulations could process the data, the data is compared to the MATLAB built in.

MATLAB Results

Figure 8, 9 and 10 give the results of the FFT simulation, compared with the FFT built in algorithm in Matlab. Both the real values and the complex values show little to no difference in the overall results compared to the matlab builtin, showing that the code correctly simulates the FFT algorithm. Figure 10 shows the differences in values between the MATLAB builtin and the FFT Algorithm recreation. The difference in value is too small that any inaccuracies in the FFT algorithm are so small that they are hardly noticeable.

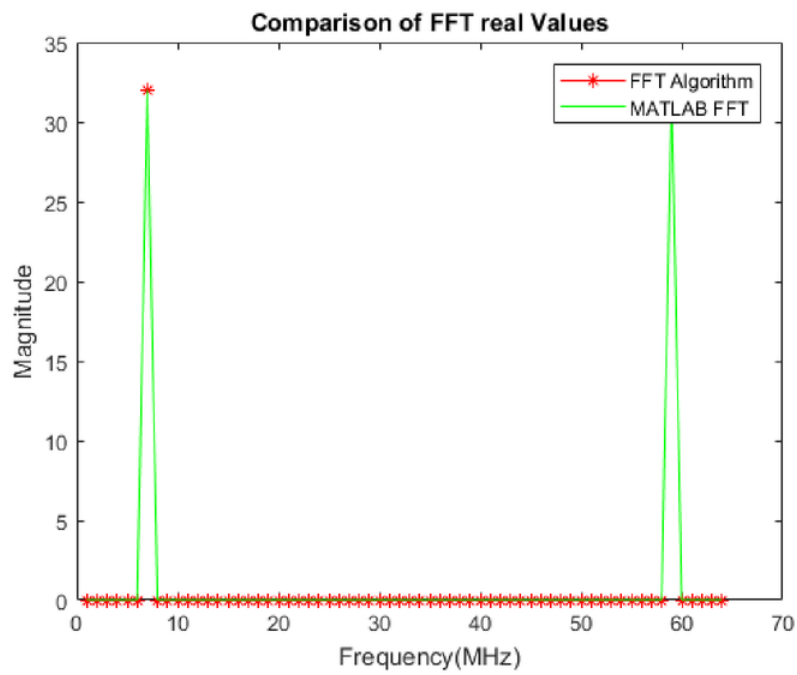


Figure 12: FFT algorithm vs Built-in for real values.

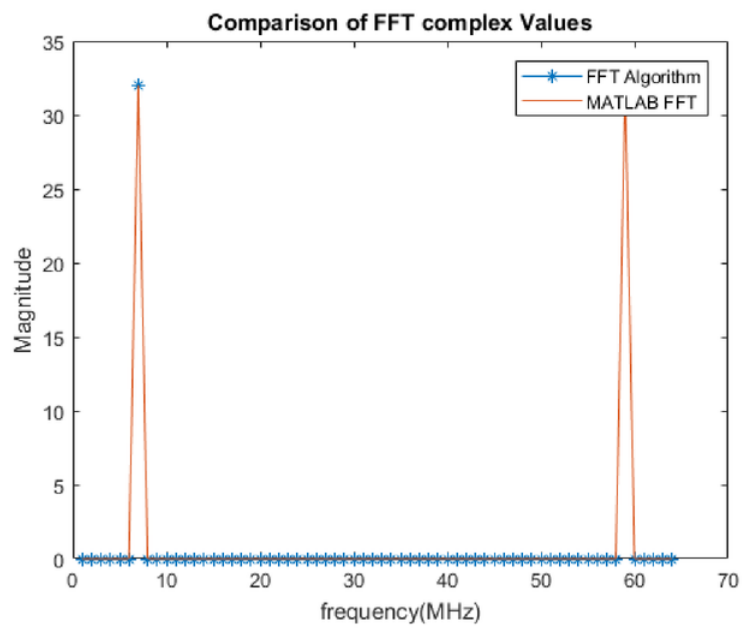


Figure 13: FFT algorithm vs Built-in for complex values

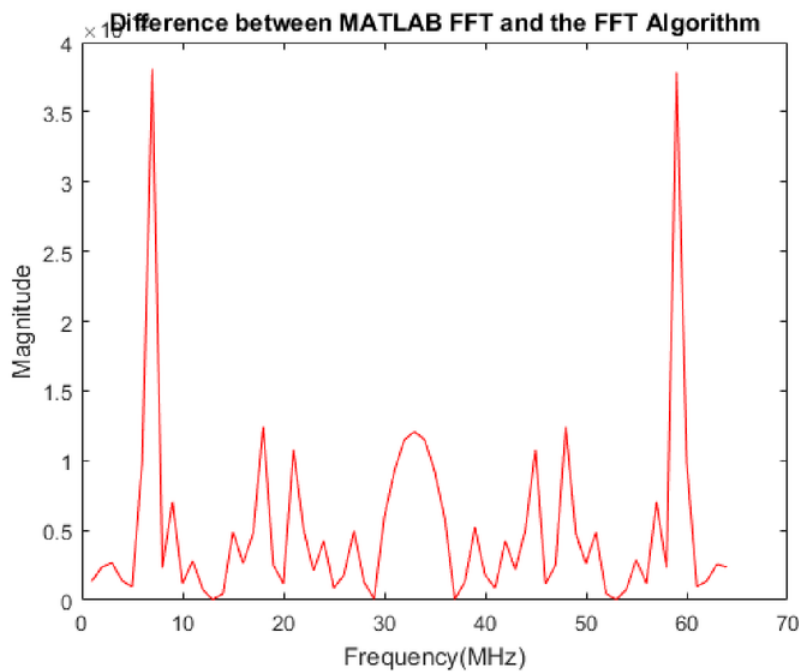


Figure 14: Value Difference between the FFT algorithm and the MATLAB builtin

5.2 FPGA Results

The target FPGA for this FFT design was Virtex 4. Virtex 4 is a reliable and common FPGA family to test on. To ensure the VHDL code was working correctly, a testbench was made to see how it operates with different inputs, while checking the accuracy of the outputs. To further evaluate the efficiency of the architecture, a second copy of the FFT design was made. This second copy has been modified to designa. The VHDL must compensate by adding in an extra memory slot for the 4th multiplication.

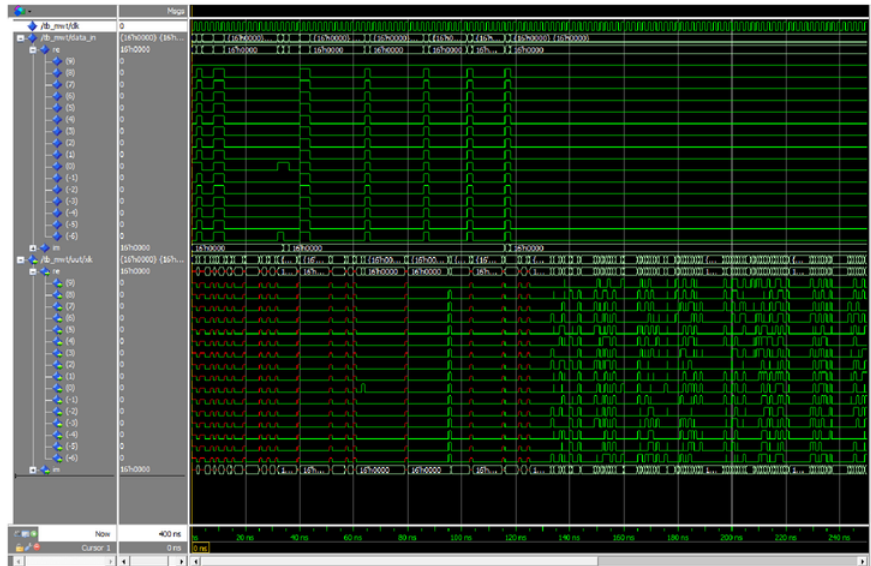


Figure 15 Modelsim Simulation results of the FFT Processor.

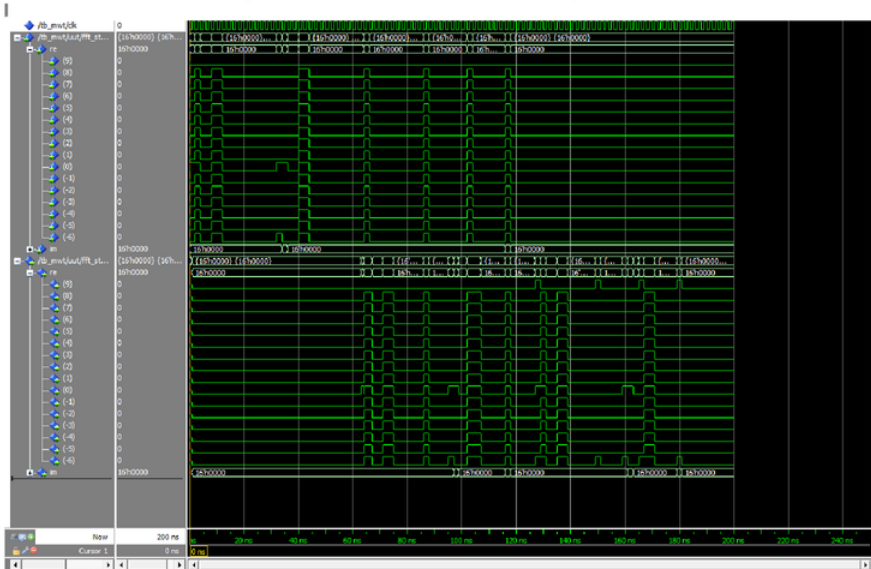


Figure 16 Modelsim Simulation results of the twiddle factor multiplication

Table 5: Resource usage of Virtex4

| Logic Utilization | Used |
|--------------------------------|-------|
| Number of Slice Flip Flops | 372 |
| Total Number of 4 input LUTs | 4,170 |
| Number used as Shift registers | 190 |

4.3 Evaluation

While the current design is already implemented, due to time constraints there weren't any ways to improve the design to make it more efficient. Much of the implementation in this project relied on the R2SDF architecture. This section will detail improvements that could improve the efficiency of the overall processor.

4.3.1 Circular Buffer

The current FFT Processor design utilizes shift registers. Shift registers store the data in the first register, and when new data arrives, it stores the data from the first register while it shifts the old data to the next register. While it is simple to utilize, it isn't power efficient to make all registers react at the same time.

To compensate this, a circular buffer could be implemented into the FFT processor. A Circular buffer behaves similarly to a shift register. The difference between the two is with its function. Circular buffers write data in one register, but moves to another register to write the next set of data. Similarly, it reads data from one register, and moves to another to read the next set of data. This is done through pointers, which only provides the address to the register, and nothing else.

Both shift registers and circular buffers follow the First-In First-Out(FIFO) process, which makes little to no difference to the way the overall processor should operate.

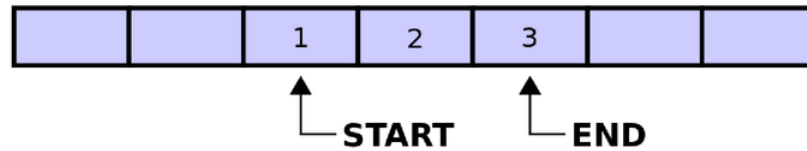


Figure 17 A representation of a circular buffer.

4.3.2 Different Adders

Much of the butterfly process in the FFT processor relies on the use of full adders. A simple full adder doesn't take much area space, but in high numbers, the large number of adders may cause a high-power requirement. If a low-power FFT processor was to be designed, one of the most important ideas to consider is using different full adder designs.

There are different full adder designs to be considered, shown in graphs made by Nuo Li[9]. A good example is a Kogg-Stone full adder design, which provides good latency at the cost of high area usage in the FFT processor. Another good example of a good adder design is the carry skip, which carries low area costs but its latency is not as decent.

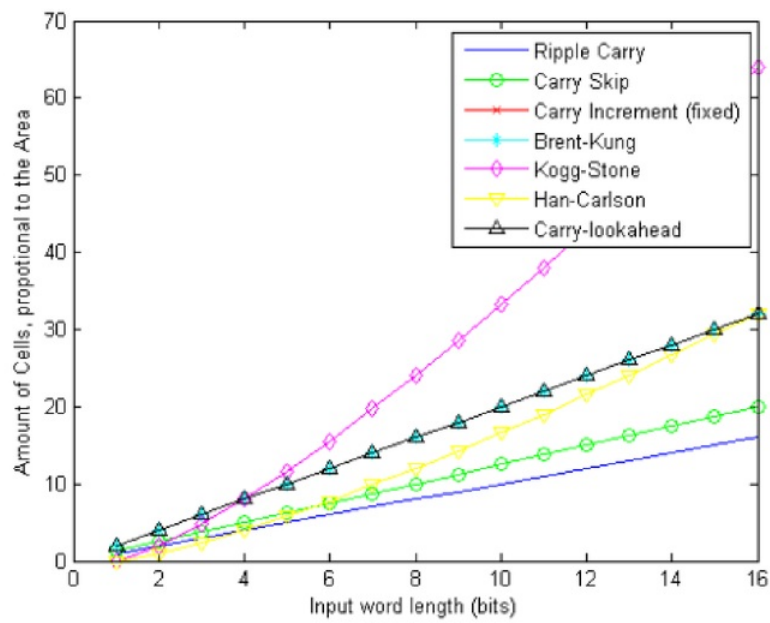


Figure 18 Area Comparison of different Adders, from [9]

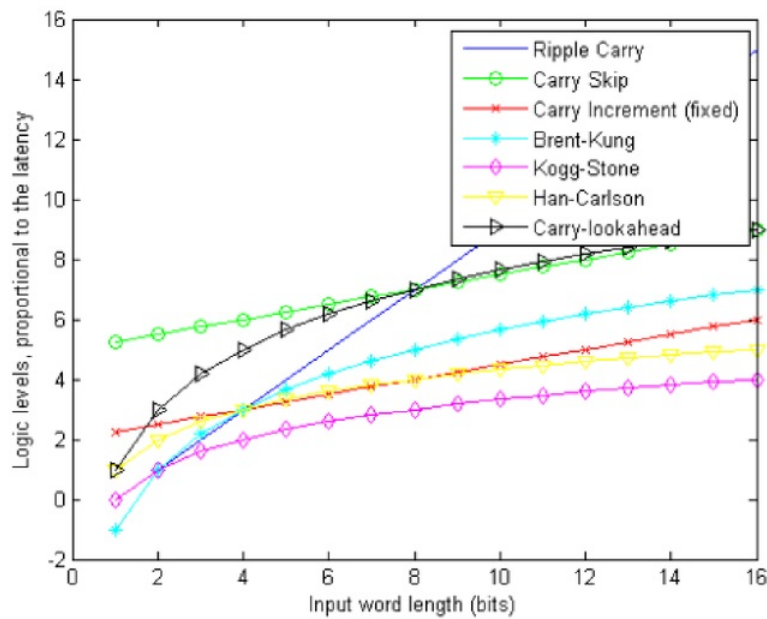


Figure 19 Latency comparison of different Adders, from [9]

Conclusion

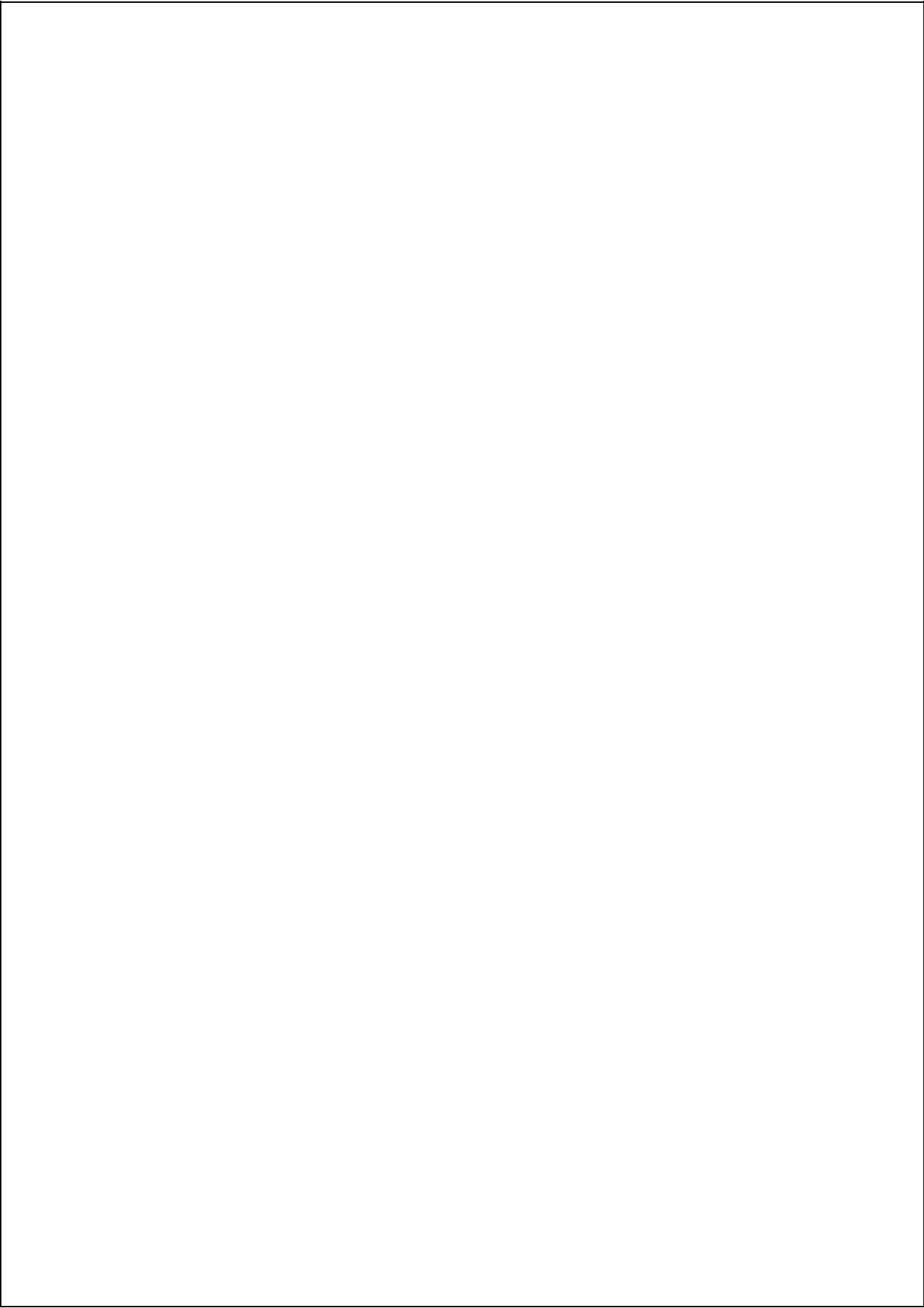
This chapter of the document described the verification of the FFT Processor that was designed, how it worked in both the matlab and the VHDL variations, and how despite the success of the work, how it can still be improved in the future.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

For the first half of this project, a Literature Review and a background of the IEEE 802.11 standard have been complete. A FFT algorithm code has already been written in matlab, and the result is validated by the inherent MATLAB function FFT. The requirements of a FFT processor to work under the 802.11ah standard has already been set up and recorded, and FFT hardware layout is currently underway. Overall, progress has been running smoothly and working on a steady pace.



Chapter 7

Abbreviations

| | |
|---------------------|---------------------------------------------------|
| DFT | Discrete Fourier Transform |
| DIF | Decimation in Frequency |
| DIT | Decimation in Time |
| FFT | Fast Fourier Transform |
| IEEE | Institute of Electrical and Electronics Engineers |
| ISM | Industrial, Science, and Medical |
| M2M | Machine to Machine |
| MAC | Medium Access Control |
| MB-OFDM | Multi-Band Orthogonal Division Multiplexing |
| PHY | Physical Layer |
| R2SDF | Radix-2 Single Delay Feedback |
| R2MDC | Radix-2 Multi-Delay Communicator |
| R4SDF | Radix-4 Single Delay Feedback |
| R4MDC | Radix-4 Multi-Delay Communicator |
| R4SDC | Radix-4 Single Delay Communicator |
| R2 ² SDF | Radix-2 Squared Single Delay Feedback |
| WLAN | Wireless Local Area Network |
| UWB | UltraWideBand |

Appendix A MATLAB Code of FFT Processor

A.1 Overview

The portion of this report will provide the MATLAB code of a generalised-point radix-2 FFT processor. This Processor can calculate any point that is the exponent of 2(4,8,16, etc). The Inputs for the code is the frequency Coefficient, and the amount of points the input provides is based on the value of N.

A.2 Generalised-point radix-2 FFT code

```
clear all
tic
N=64; %points of an array
fc=21;
fs=N;
Ts=1/fs;
tvals=0:Ts:1-Ts;
x=cos(2*pi*fc*tvals)+ sin(2*pi*fc*tvals);
M=1;
j=log2(N); %create a integer, j, for the while loop
Butterfly = zeros(j-1,N); % a twiddle factor group Butterfly
Stage = zeros(j+1,N);
n = 0;
p=(0:N-1)';
t=bitrevorder(p);
for S=1:j
```

```

for B=1:N
    b = reverse(dec2bin(B-1,j+1));
    if b(S) == '1'
        Butterfly(S,B) = exp(-1j*2*pi/(2^(S))).^(n);
        n = n+1;
    else
        Butterfly(S,B) = 1;
    end
    if 2^(S-1) == n
        n = 0;
    end
end
end

for S=1:j+1
    for B=1:M:N
        if M == 1
            Stage(S,B) = x(t(B)+1);
        else
            for n=0:M-1
                if M == N
                    if n+1 <= N/2
                        Stage(S,n+1) = Stage(S-1,n+1) + Stage(S-1,n+1+N/2);
                    else
                        Stage(S,n+1) = Stage(S-1,n+1-N/2) - Stage(S-1,n+1);
                    end
                elseif n <= M/2-1
                    Stage(S,B+n) = Stage(S-1,B+n) + Stage(S-1,B+n+M/2);
                    Stage(S,B+n) = Stage(S,B+n) * Butterfly(S,B+n);
                else
                    Stage(S,B+n) = Stage(S-1,B+n-M/2) - Stage(S-1,B+n);
                    Stage(S,B+n) = Stage(S,B+n) * Butterfly(S,B+n);
                end
            end
        end
    end
end

```






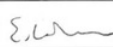



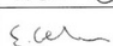
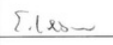
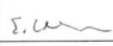
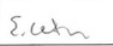
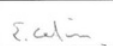
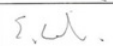
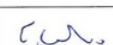
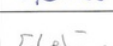


```
        end
    end
end
M=M*2;
end
toc
Stage_re = abs(real(Stage));
Stage_im = abs(imag(Stage));
```

Appendix B: Consulation Forms

This appendix contains the consultation meetings attendance form as required by the department. Both the supervisor and the student had to sign on the consultation meetings form for the record of the meetings.

Consultation Meetings Attendance Form

| Week | Date | Comments (if applicable) | Student's Signature | Supervisor's Signature |
|------|----------|-----------------------------|-----------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
| 1 | 1/8/17 | — |  |  |
| 2 | 10/8/17 | — |  |  |
| 3 | 17/8/17 | — |  |  |
| 4 | 24/8/17 | — |  |  |
| 5 | 1/9/17 | — |  |  |
| 6 | 7/9/17 | — | |  |
| 7 | 15/9/17 | — | |  |
| 8 | 5/10/17 | — | |  |
| 9 | 13/10/17 | — | |  |
| 10 | 20/10/17 | — | |  |
| 11 | 27/10/17 | — | |  |
| 11 | 27/10/17 | — | |  |

Bibliography

[1-16]

- [1] J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297-301, 1965.
- [2] T. H. Joo and A. V. Oppenheim, "Effects of FFT coefficient quantization on sinusoidal signal detection," in *ICASSP-88, International Conference on Acoustics, Speech, and Signal Processing*, 1988, pp. 1818-1821 vol.3.
- [3] S. Magar, S. Shen, G. Luikuo, M. Fleming, and R. Aguilar, "An application specific DSP chip set for 100 MHz data rates," in *ICASSP-88, International Conference on Acoustics, Speech, and Signal Processing*, 1988, pp. 1989-1992 vol.4.
- [4] S. Magar, S. Shen, G. Luikuo, M. Fleming, and R. Aguilar, "An application specific DSP chip set for 100 MHz data rates," in *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88, 1988 International Conference on*, 1988, pp. 1989-1992: IEEE.
- [5] J. O'Brien, J. Mather, and B. Holland, "A 200 MIPS single-chip 1 k FFT processor," in *Solid-State Circuits Conference, 1989. Digest of Technical Papers. 36th ISSCC., 1989 IEEE International*, 1989, pp. 166-167: IEEE.
- [6] H. Shousheng and M. Torkelson, "A new approach to pipeline FFT processor," in *Proceedings of International Conference on Parallel Processing*, 1996, pp. 766-770.
- [7] B. M. Baas, "A low-power, high-performance, 1024-point FFT processor," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 3, pp. 380-387, 1999.
- [8] W. H. Chang and T. Q. Nguyen, "On the Fixed-Point Accuracy Analysis of FFT Algorithms," *IEEE Transactions on Signal Processing*, vol. 56, no. 10, pp. 4673-4682, 2008.
- [9] N. Li and N. P. v. d. Meijs, "A Radix 2² based parallel pipeline FFT processor for MB-OFDM UWB system," in *2009 IEEE International SOC Conference (SOCC)*, 2009, pp. 383-386.
- [10] C. S. Sum, H. Harada, F. Kojima, Z. Lan, and R. Funada, "Smart utility networks in tv white space," *IEEE Communications Magazine*, vol. 49, no. 7, pp. 132-139, 2011.
- [11] S. Aust, R. V. Prasad, and I. G. M. M. Niemegeers, "IEEE 802.11ah: Advantages in standards and further challenges for sub 1 GHz Wi-Fi," in *2012 IEEE International Conference on Communications (ICC)*, 2012, pp. 6885-6889.
- [12] S. H. Aust, "Advanced Wireless Local Area Networks in the Unlicensed Sub-1GHz ISM-bands," 2014.

- [13] Australian Communications and Media Authority, Spectrum Planning and Engineering Branch. (2015). *Frequency plan for services in the 900 MHz band 890 - 960 MHz*.
- [14] K. i. Kido, *Digital fourier analysis: fundamentals*. Springer, 2015.
- [15] T. H. Tran, S. Kanagawa, D. P. Nguyen, and Y. Nakashima, "ASIC design of MUL-RED Radix-2 Pipeline FFT circuit for 802.11ah system," in *2016 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS XIX)*, 2016, pp. 1-3.
- [16] "IEEE Draft Standard for Information Technology--Telecommunications and information exchange between systems Local and metropolitan area networks--Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Very High Throughput to Support Chinese Millimeter Wave Frequency Bands(60 GHz and 45 GHz)," *IEEE P802.11aj/D5.0, February 2017 (Amendment to IEEE Std 802.11REVMc 8.0, as amended by IEEE Std 802.11ah 10.0, IEEE Std 802.11ai 11.0 and IEEE Std 802.11ak 3.0 and IEEE Std 802.11aq 7.2)*, pp. 1-321, 2017.