TAMPERE UNIVERSITY OF TECHNOLOGY

DEEPAK REVANNA

Design and Implementation of Scalable FFT Processor for Wireless Applications

Master of Science Thesis

Examiners:
Prof. Jari Nurmi
M.Sc. Omer Anjum
Examiners and topic were approved in the Computing and Electrical Engineering Faculty Council meeting 15.Aug.2012.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
DEEPAK REVANNA: Design and Implementation of Scalable FFT Processor for Wireless Applications
Master of Science Thesis, 63 Pages, 5 Appendix Pages
March 2013
Major: Electronics and Communications Engineering
Examiners: Prof. Jari Nurmi, M.Sc. Omer Anjum
Keywords: FFT, FPGA, OFDM, SDR, Processor

In the recent past communication is predominantly becoming wireless which is a drastic shift from wired communication. Generally, the transmitted radio signal over a wireless channel is subject to more distortion, more interference and more noise than a signal over wired channel. In other words, the SNR of received signal over a wireless channel is comparatively lower compared to received signal over a wired channel. Hence, to recover original data from received signal, wireless communication systems have to be more robust and efficient in recovering original data. Wireless communication systems these days adopt efficient multi-carrier transmission technique such as OFDM in their transceivers. And majority of the commercial wireless standards are OFDM based.

OFDM based wireless standards demand highly efficient baseband hardware in communication systems. The baseband hardware needs to meet stringent design parameters such as high speed, low power, low area, low cost, highly flexible and highly scalable. Modern wireless systems support multiple standards to meet the demands of end user application requirements. A wireless system while supporting multiple standards, should also satisfy performance requirements of those supported standards. Wireless transceivers based on SDR platform support multiple wireless standards. Meeting performance requirements of multiple standards is a challenge while designing baseband hardware. To design an efficient OFDM baseband hardware, it is necessary to efficiently design its performance critical component. FFT computation is one of the most performance critical component in an OFDM system. Designing FFT hardware to support multiple wireless standards while meeting the above specified performance requirements is a challenging task.

In this thesis work a N-point scalable novel FFT processor architecture was proposed. A radix-2 fixed point 16-bit N-point scalable FFT processor was designed and prototyped using VHDL on an Altera Stratix V FPGA device 5SGSMD5K2F40C2. The processor was implemented targeting SDR platforms supporting multiple OFDM

based wireless standards. The processor operates at a maximum frequency of 200MHz and uses less than 1% of hardware resources on the FPGA. It meets the performance requirements of OFDM based wireless standards such as IEEE 802.11a/g, IEEE 802.16e, 3GPP-LTE, DAB and DVB-T. The FFT processor based on proposed novel architecture has a better performance in terms of speed, flexibility and scalability when compared to existing fixed as well as variable length FFT processors.

PREFACE

When I wanted to start work on my thesis I approached Prof. Jari Nurmi to give me an opportunity and to which he positively responded. He provided me with all the facilities and environment to work in the department. He was a mentor, a guide and consistently supported my work during bad times as well as good times. Hence, I would like to express my sincere gratitude towards Prof. Jari Nurmi for all his support. Mr. Omer Anjum who provided me with the research topic, shared his knowledge and guided me throughout my thesis work. I thank him for guiding, supporting and supervising me throughout my work. Mr. Roberto Airoldi was also my mentor who shared his technical knowledge and helped me during my work. He also taught me how to write research papers for publishing in international conferences. I would like to thank him for all his guidance, support, sharing knowledge and I cherish the moments of sharing office space with him. I express my thanks to Mr. Manuele Cucchi for his technical help and discussions during the course of my work. I would also like to thank Ms. Leyla Ghazanfari for being a very supportive and encouraging colleague. My family has loved, cared and supported me during tough times as well as good times. I am eternally grateful to my father Revanna, my mother Mangalamma and my brother Nagendra Prasad for showering me with their unconditional love and support.

Tampere, 22.Mar.2013

Deepak Revanna

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ALUT	Adaptive Look Up Table
CDM	Code Division Multiplexing
CentOS	Community ENTerprise Operating System
DAB	Digital Audio Broadcasting
DAC	Digital to Analog Converter
DFT	Discrete Fourier Transform
DIF	Decimation In Frequency
DIT	Decimation In Time
DSP	Digital Signal Processing
DVB-T	Digital Video Broadcasting-Terrestrial
EDA	Electronic Design Automation
FDM	Frequency Division Multiplexing
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
ICI	Inter Carrier Interference
IEEE	Institute of Electrical and Electronics Engineers
IFFT	Inverse Fast Fourier Transform
ISI	Inter Symbol Interference
LOS	Line Of Sight
LPF	Low Pass Filter
OFDM	Orthogonal Frequency Division Multiplexing
PSK	Phase Shift Keying
QAM	Quadrature Amplitude Multiplexing
RAM	Random Access Memory
RF	Radio Frequency
ROM	Read Only Memory
RTL	Register Transfer Level
SDC	Synopsys Design Constraint
SDR	Software Defined Radio
SNR	Signal-to-Noise Ratio
USB	Universal Serial Bus
VCD	Value Change Dump
VHDL	Very high speed integrated circuit Hardware Description Language
3GPP-LTE	3rd Generation Partnership Project-Long Term Evolution

1. INTRODUCTION

The wired telecommunication networks offer low-bit-rate services as well as high-bit-rate services. Voice services require low-bit-rates while broadband multimedia services require high-bit-rates. Wireless communication networks also provide the specified services. However, some of the high-bit-rate services are limited due to various performance constraints. During the course of time, there has been a growing demand for high-bit-rate services in wireless communication systems. Providing services over wireless channels is challenging because the mobile radio channels are more contaminated compared to wired channels.

The main characteristic of a mobile radio channel is multipath reception of transmitted signal. The received signal not only contains Line-Of-Sight (LOS) signal but also reflected signals. The reflected signals are delayed and distorted versions of transmitted signal. Transmitted signal undergoes reflections due to terrain features like trees, buildings, vehicles, hills, mountains and so on. The delayed signal causes Inter Symbol Interference (ISI) with LOS signal received at the receiver. ISI causes performance degradation of the transceiver and it is necessary to adopt suitable equalization technique in order to improve the performance. A broadband multimedia wireless communication system requires high-bit-rate transmission in terms of at least several megabits per second. Designing wireless transceivers to support such high data rates with compact and low-cost hardware is a challenging task. In order to overcome multipath-fading environment and in the meantime achieve high data rates, Orthogonal Frequency Division Multiplexing (OFDM) transmission scheme is used.

OFDM is a parallel data transmission technique which minimizes the influence of multipath fading through simpler equalization technique. OFDM is widely adopted in modern wireless communication systems. It has been adopted by major wireless standards such as Institute of Electrical and Electronics Engineers (IEEE) 802.11a/g, IEEE 802.16e, 3rd Generation Partnership Project-Long Term Evolution (3GPP-LTE), Digital Audio Broadcasting (DAB) and Digital Video Broadcasting-Terrestrial (DVB-T). A transceiver supporting multiple standards and which is based on Software Defined Radio (SDR) platform allows switching between multiple wire-

less standards at run time. Modern wireless transceivers based on SDR platform may support one or multiple standards. In any case, the transceiver should comply with performance requirements of all the standards it supports. The standards specify strict performance requirements in terms of high speed, low power, low cost, flexibility and scalability.

Meeting stringent performance requirements while supporting multiple wireless standards is the need of the hour. Since, OFDM based communication system is commercially adopted in major wireless standards, there is a huge amount of research interest in OFDM baseband digital signal processing. In order to design an efficient OFDM baseband hardware, its components require to be efficient. In an OFDM baseband hardware, FFT computation is one of the most computationally intensive operation which influences performance of the system. The baseband hardware has to be efficient and capable enough to compute FFT within the time constraints necessary to support multiple wireless standards. Baseband hardware should be scalable so that it supports multiple wireless standards as well as it should meet the performance constraints such as high speed, low area and low power consumption. Hence, the baseband hardware requires a scalable FFT module which meets the performance constraints required by multiple wireless standards.

Scope of the thesis work was to propose a N-point scalable novel FFT processor architecture, implement a radix-2 fixed point 16-bit N-point scalable FFT processor based on the proposed architecture using Very high speed integrated circuit Hardware Description Language (VHDL) and synthesize the processor on a Field Programmable Gate Array (FPGA). The processor implementation was simulated using ModelSim simulation tool from Mentor Graphics Corporation to measure its performance in terms of speed and scalability. Also, the processor was synthesized on an FPGA to measure the performance parameters such as maximum operating frequency, area and power consumption. The synthesis tool used was Quartus II version 12.1 from Altera Corporation and the FPGA was Altera stratix V 5SGSMD5K2F40C2.

The structure of this thesis is as follows: the Chapter 2 explains the background related to OFDM and research work on FFT processors, the Chapter 3 explains FFT processor architecture and its components in detail, the Chapter 4 is about implementation details such as simulation and synthesis, the Chapter 5 discusses results and its evaluation and finally the Chapter 6 draws conclusions based on the results achieved.

2. BACKGROUND

OFDM is an efficient multi-carrier transmission technique which is predominantly used in wireless transceivers. OFDM technique offers better spectral utilization and better performance compared to other transmission techniques in recovering original signal from received signal. Since, major wireless standards are based on OFDM transmission, there is a lot of research interest in this domain. Research in OFDM baseband hardware of transceiver is a challenging task. One of the major performance critical module of OFDM transceiver is FFT computation. One of the steps in creating an efficient OFDM transceiver is to create an efficient FFT module. In this regard, basics of OFDM based communication systems and FFT algorithm are described in detail below.

2.1 Wireless OFDM Systems

A simplex communication system based on OFDM is shown in Figure 2.1.

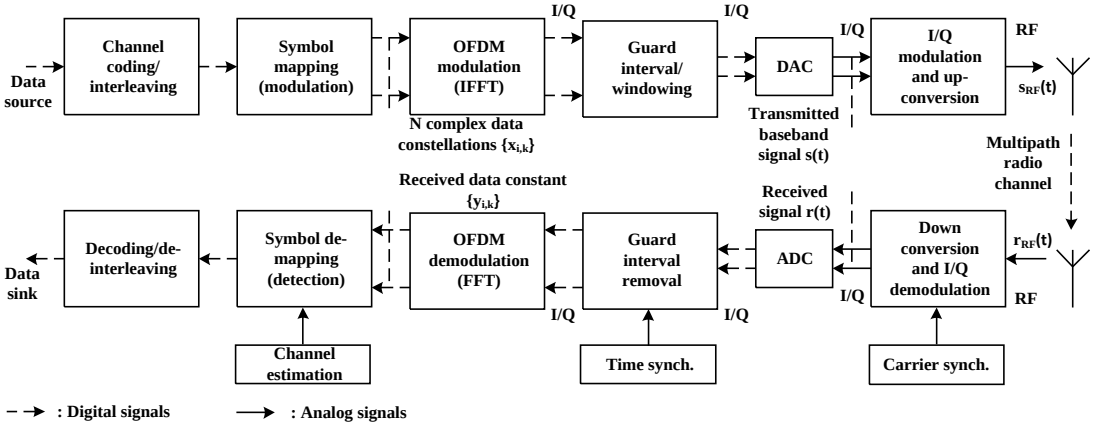

Figure 2.1: *An OFDM based simplex communication system [8].*

A communication system in general has a transmitter and a receiver which can be put together to form a transceiver. The transmitter modulates baseband digital signal, converts it into Radio Frequency (RF) signal using Digital to Analog Converter (DAC). The transmitted RF signal undergoes multipath fading, it is contaminated with thermal noise, distortions in radio channel and also undergoes Inter Symbol Interference (ISI). To recover original transmitted signal, the received signal has to

undergo equalization. Several equalization techniques are available, but a suitable technique is chosen based on Signal-to-Noise Ratio (SNR) requirements and other design constraints.

Inverse Fast Fourier Transform (IFFT) and FFT are used for modulating and demodulating the data constellations on orthogonal sub-carriers. These two signal processing algorithms are used instead of I/Q-modulators and demodulators. The input of IFFT is $x_{i,k}$ which is an N-point data constellation, where N is the number of IFFT/FFT points, i is sub-carrier index and k is an OFDM symbol index. N is chosen as a power of two, this helps in efficient implementation of IFFT and FFT algorithms for modulation and demodulation respectively. The described communication system consists of OFDM module which is our area of interest in the baseband domain. Hence, OFDM, OFDM modulation and demodulation are discussed in detail below.

2.2 OFDM

OFDM is a multi-carrier transmission technique which is widely popular in most of the available commercial wireless communication standards. As described in [7] an OFDM signal is made up of a number of sub-carriers or sub-channels which are orthogonal to each other. The bandwidth of an OFDM signal includes all the sub-carriers or in other words each sub-carrier shares the available bandwidth. The sub-carriers carry data individually and are modulated in amplitude as well as phase. The sub-carriers can be multiplexed using Frequency Division Multiplexing (FDM) or Code Division Multiplexing (CDM) technique. Orthogonality property of sub-carriers increases spectral utilization of the transmitted signal while reducing Inter Carrier Interference (ICI). The OFDM technique is an improvement over an FDM technique. Advantage of OFDM is that it requires a single filter for all the sub-carriers while FDM requires filter for each of the sub-carriers. But, disadvantage of OFDM is that it requires highly accurate frequency synchronization technique in order to avoid ICI. OFDM technique can be used to modulate a number of sub-carriers to carry data individually as detailed below.

Figure 2.2 describes modulation in baseband domain of an OFDM based communication system. Input data bits are split among different sub-carriers with the help of serial-to-paraller converter. The sub-carriers are assigned with a range of frequencies and each of them share the available bandwidth of an OFDM signal.

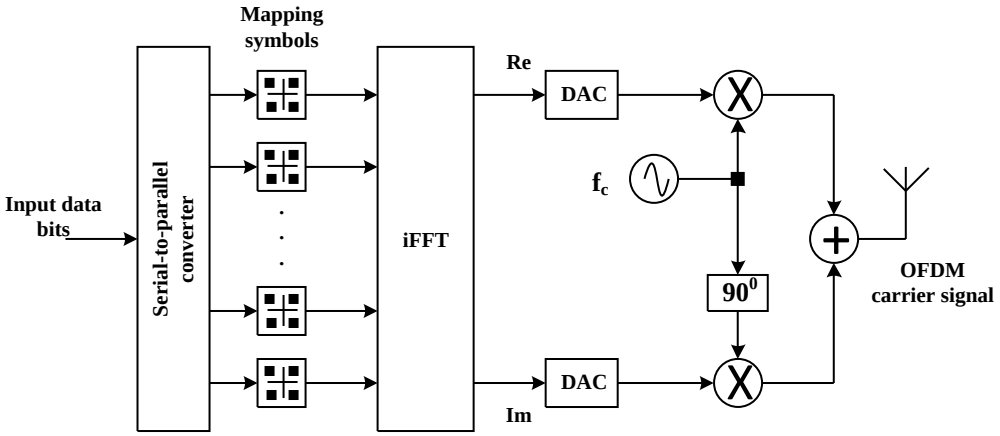

Figure 2.2: *OFDM modulation at transmitter [7].*

Each sub-carrier is modulated by data using Phase Shift Keying (PSK) or Quadrature Amplitude Modulation (QAM) technique. IFFT on sub-carriers transforms them to time domain and combines them together to form an OFDM signal. Resulting digital OFDM signal is converted to analog signal by Digital to Analog Converter (DAC). To transmit data bearing OFDM signal over radio channel, an RF carrier is modulated by the OFDM signal. After using OFDM modulation at transmitter, OFDM demodulation is used at receiver to recover data from radio signal.

OFDM demodulation at the receiver side is illustrated in Figure 2.3.

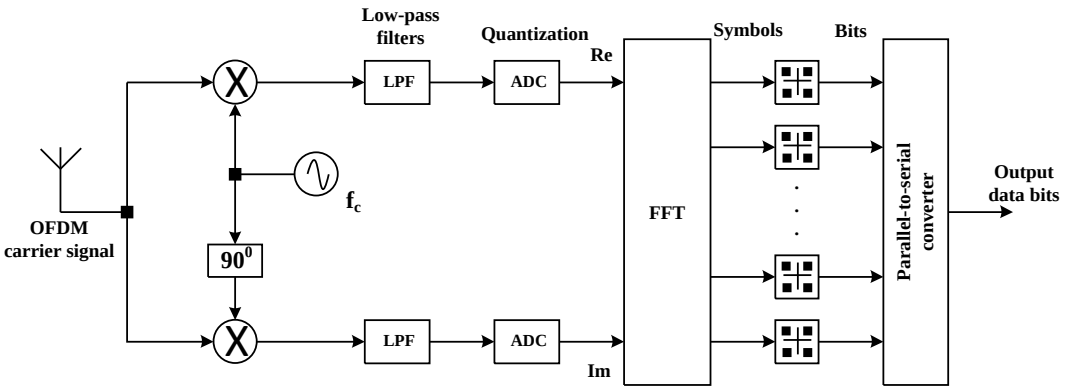

Figure 2.3: *OFDM demodulation at receiver [7].*

RF signal received from radio channel is down converted to separate data bearing OFDM signal from it. OFDM signal is complex valued consisting of real and imaginary parts. The resulting baseband OFDM signal is low pass filtered to eliminate unwanted harmonics present around baseband signal frequency.

OFDM analog signal is converted into digital signal through Analog to Digital Converters (ADC). Time domain digital OFDM signal is converted into frequency domain through FFT operation. Also, FFT operation disintegrates OFDM signal into its sub-carriers. Individual sub-carriers are demodulated separately to extract data from them. Symbols from sub-carriers are converted to bit streams using symbol detectors. The demodulator or symbol detector is in synchronization with the modulator which maps bit streams to symbols.

OFDM modulators and demodulators are used in wireless transceivers. And wireless transceivers support multiple wireless standards. OFDM transmission technique is widely popular and it is adopted commercially by a number of wireless standards.

2.3 OFDM Based Wireless Standards

Most of the wireless standards available are OFDM based. Some of the OFDM based wireless standards are IEEE 802.11a/g, IEEE 802.16e, 3GPP-LTE, DAB and DVB-T. Different standards specify time constraints for FFT computation as shown in Table 2.1.

	FFT Size	**FFT Period [μs]**
DAB	2048	1000
	1024	500
	512	250
	256	125
DVB-T	8192	896
	2048	224
IEEE 802.11a/g	64	3.2
IEEE 802.16e	256	8
3GPP-LTE	128	66.7
	256	
	512	
	1024	
	2048	

Table 2.1: *FFT computation time for OFDM based wireless standards.*

To support a specific standard, a communication system should meet the performance constraints set by that standard. FFT computation time is specified in micro seconds for different sizes of FFT. IEEE 802.11a/g and IEEE 802.16e support only specific size FFT computation while DAB, DVB-T and 3GPP-LTE support different FFT size computations. For DAB and DVB-T, FFT computation time varies in accordance with FFT size. However, in case of 3GPP-LTE, FFT computation time is the same irrespective of FFT size.

FFT operation is computationally intensive and is required to be performed within the time constraints specified by various wireless standards. Hence, FFT is studied in more detail before its implementation in hardware.

2.4 Fast Fourier Transform

FFT is a faster version of Discrete Fourier Transform (DFT). Computation of DFT/FFT of a time domain digital signal $x(n)$ results in converting it into a frequency domain signal. Analysis and processing of a discrete signal in frequency domain is more efficient than its analysis in time domain. The FFT algorithm was first developed and presented by Cooley and Tukey in [5]. It was developed in order to reduce number of complex multiplications and additions in DFT. An N-point DFT is given by,

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-(i\frac{2\pi nk}{N})} \tag{2.1}$$

where $k = 0, 1, 2 \ldots N-1$.
According to equation 2.1, DFT computation requires $N^2 - N$ complex additions and N^2 complex multiplications. An N-point FFT equation is given by,

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n) e^{-(i\frac{2\pi nk}{\frac{N}{2}})} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) e^{-(i\frac{2\pi nk}{\frac{N}{2}})} \tag{2.2}$$

where $W_N^k = e^{-i\frac{2\pi k}{N}}$, $k = 0, 1, 2 \ldots N-1$.
According to equation 2.2, the number of multiplications and additions are reduced to $\frac{N}{2} * log_2(N)$ and $N * log_2(N)$ respectively. Preferring FFT over DFT for hardware implementation means increased speed, reduced power consumption, reduced area and reduced cost.

FFT is computed in two different ways, Decimation In Time (DIT) and Decimation In Frequecny (DIF). In DIT algorithm, inputs are in bit reversed order and the outputs are in natural order. In DIF algorithm, the inputs are in natural order

and the outputs are in bit reversed order. According to Tran-Thong et al. [12], the DIT algorithm provides better signal-to-noise ratio when compared to DIF algorithm for a finite word length. Based on number of FFT inputs, the algorithm can be radix-2, radix-4, radix-8 or split-radix type. In radix-2 algorithm FFT size is a power of two, radix-4 FFT size is a power four while radix-8 FFT size is power of eight. And split-radix type involves mix of any of the specified radix combinations.

A radix-2 DIT FFT algorithm can be depicted as a butterfly diagram as shown in Figure 2.4. The figure describes 16-point FFT butterfly diagram where *x(n)*, *X(k)* are 16-point complex inputs and outputs respectively.

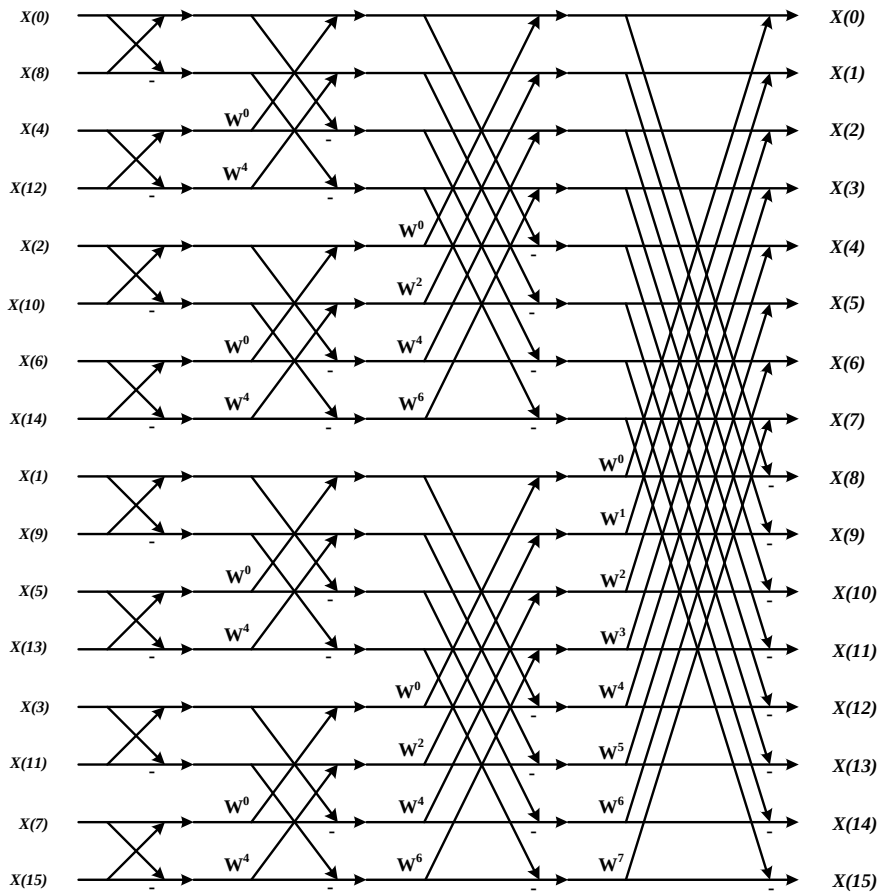

Figure 2.4: *Radix-2 DIT FFT butterfly diagram [1].*

Since, it is a DIT algorithm, the inputs to butterfly diagram are in bit reversed order and the outputs are in natural order. The upper half of butterfly diagram is symmetric with its lower half until the last stage. During the last stage, the upper half of input samples mingle with lower half before computation. This particular property of DIT algorithm is the basis of our address generation algorithm, dataflow algorithm and input data storage described later in this document. Considering an

N-point FFT, there are $log_2 N$ number of stages and each stage requires $\frac{N}{2}$ butterfly operations.

Butterfly operation is the basic entity of a butterfly diagram. The butterfly operation is pictorially described as shown in Figure 2.5.

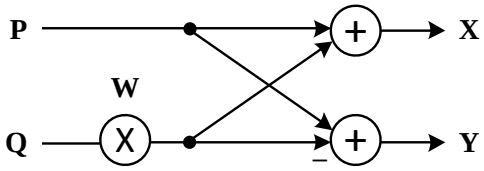

Figure 2.5: *Single radix-2 DIT butterfly operation.*

Butterfly operation can be illustrated in equation as,

$$X = P + WQ \quad \text{and} \quad Y = P - WQ \tag{2.3}$$

where P, Q are complex input values, W is an input twiddle factor and X, Y are complex output values. The output X is the result of addition while Y is the result of subtraction.

Studying existing research on FFT processors provides us with a information on their merits and demerits. Knowing demerits of existing research enables us to work on them and improve them according to our requirements. In the next section, existing research work on FFT processors is evaluated and summarized.

2.5 Research Work On FFT Processors

In recent times, research on OFDM wireless communication systems is focused extensively. In particular, research on FFT algorithm and its hardware implementation is a hot research topic. Focus of research is to optimize FFT algorithm and to find efficient hardware solutions. Since, SDR platforms based on OFDM technique support multiple wireless standards, OFDM communication systems need to support multiple standards. Since, FFT is an integral component of OFDM transceiver, FFT hardware should be capable of supporting multiple wireless standards. Hence, we have investigated existing research literature on variable length as well as fixed length FFT processors.

According to [14], in OFDM based wireless transceivers, FFT is one of the most power consuming and computationally intensive operation. Since, FFT is power

hungry and computational module, it is the main motivation behind research on FFT processor architectures. Fixed length FFT processors are proposed by Derafshi et al. in [2], H.Jiang et al. in [4], S.-N. Tang et al. in [11], K. George et al. in [3] which focus on specific FFT size and specific standard. Processor proposed by Y.-T.Lin et al. in [6] focuses on lower power consumption, the processor presented by B.Wang et al. in [13] focuses on higher speed and supports only 64-point FFT computation. The paper presented by Q.Zhang et al. in [15] the main focus is to reduce area consumption. Hence, some of the specified existing research works are based on specific FFT size targeting specific standard and are optimized for specific design parameter. Fixed length FFT processors can support only specific wireless standard. And they won't be scalable across multiple standards but they are optimized in terms of power, area, high speed and low cost. On the other hand, variable length FFT processors supporting multiple standards have to compromise in terms of high speed, low power and low area. Finding a reasonable balance between scalability and meanwhile achieving performance constraints is a design challenge. Hence, we have attempted to find a reasonable balance between low power, low area, low cost, high speed, flexibility and scalability of FFT processor.

Scalable FFT processor designed and implemented as part of the thesis work is based on a holistic approach adopted to achieve reasonable balance between scalability while meeting strict performance constraints. The processor is design time configurable to support a maximum FFT size N_{max}. Since, processor is based on radix-2 FFT algorithm, N_{max} can only be power of two. During runtime, the processor can support FFT computation of size varying from 16-point upto N_{max}-point. Hence, proposed processor architecture is configurable at design time and scalable at runtime. The proposed architecture can be extended to radix-4/8 FFT computations to achieve higher performance. In addition, the FFT processor can be used in non-OFDM systems where scalability is required.

3. SCALABLE FFT PROCESSOR

The scalable FFT processor supports N-point complex value radix-2 fixed point FFT computation. The processor is configurable at design time to required N_{max}-point (radix-2 values only) and after which at runtime it can perform FFT computation from 16-point to N_{max}-point. During design time data memory and twiddle factor memory are chosen so as to support N_{max}-point computation. Following are the major components of FFT processor and its block diagram representation is shown in Figure 3.1.

- Butterfly unit
- Data memory (RAM)
- Twiddle factor memory (ROM)
- Interconnect
- Address generation unit
- Control unit

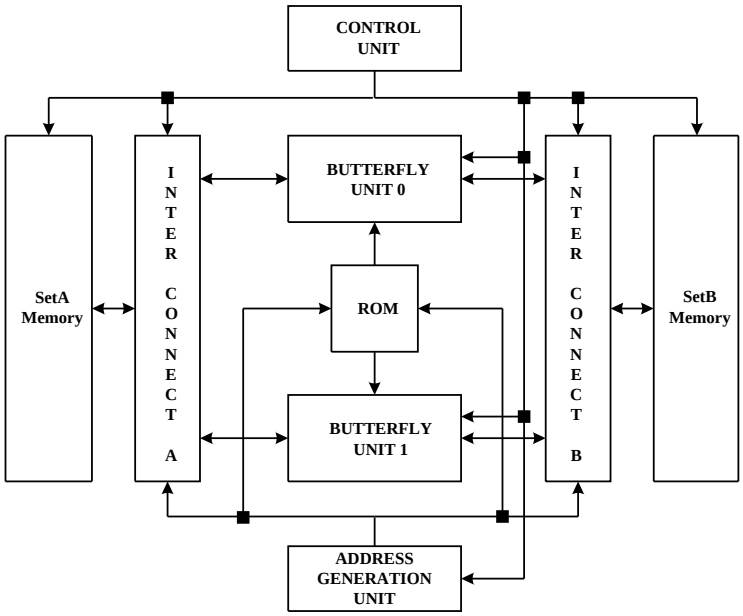

Figure 3.1: *Scalable FFT processor block diagram.*

The FFT processor uses two butterfly units which operate in parallel and compute two outputs per clock cycle. Two sets of data memory were chosen which were named SetA and SetB, each set contained four memory banks required for simultaneous access of four samples. One twiddle factor memory was used for storing $\frac{N_{max}}{2}$ twiddle factors to support N_{max}-point FFT computation. Two interconnects called interconnectA and interconnectB were used to form link between butterfly units and memory set SetA and SetB respectively. Address generation unit was used to generate addresses required to read input samples and twiddle factors for butterfly units. Control unit was required to generate control signals at required timing to co-ordinate and synchronize activities between rest of the components.

Figure 3.2 shows input/output ports of FFT core.

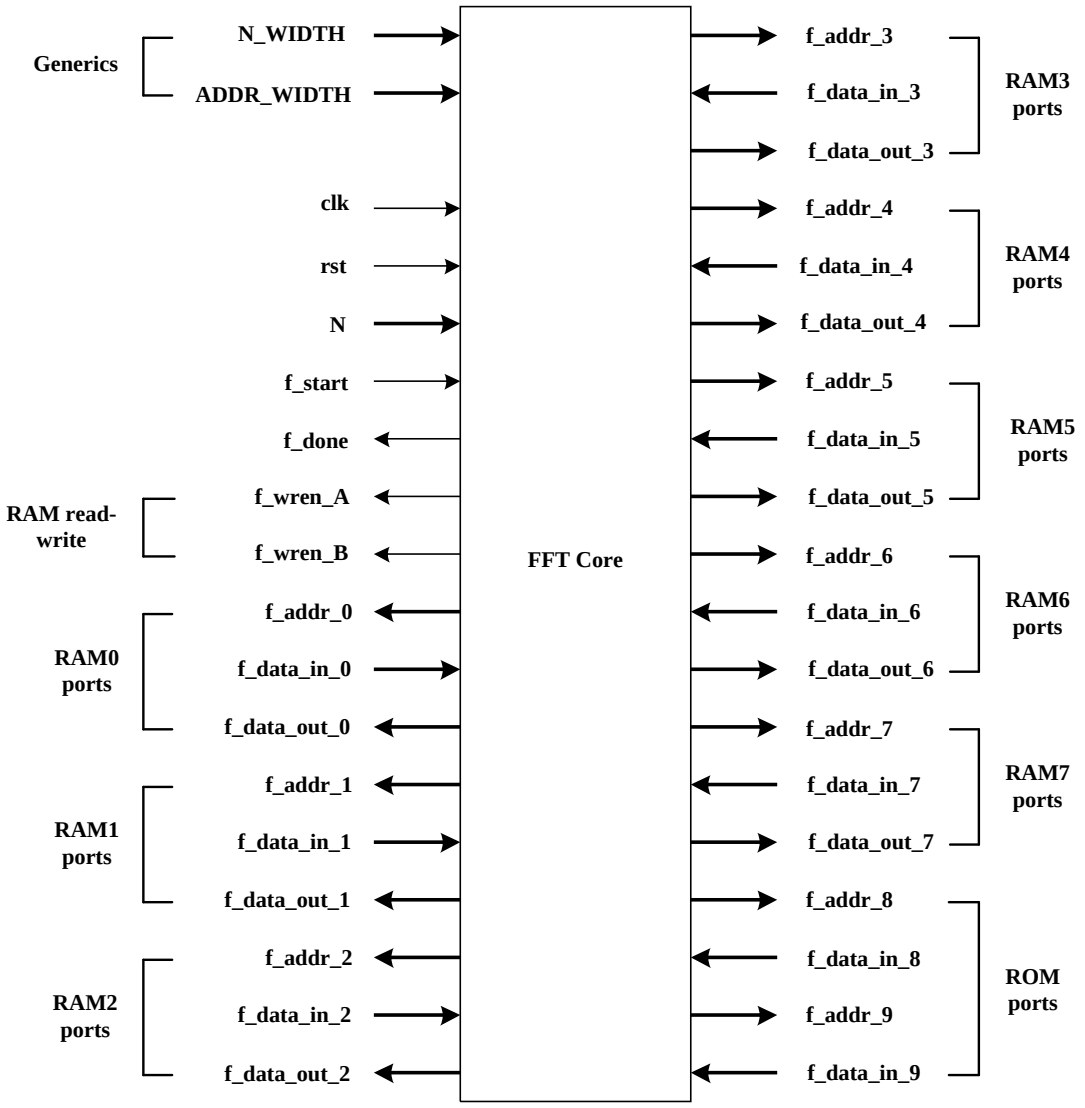

Figure 3.2: *FFT processor core pin details.*

Ports with thin line represent single bit pins while ports with thick line represent multi-bit pins. Ports named in capital letters are generics which allow design time configuration of modules. The address width ($ADDR\_WIDTH$) and N-width (N_WIDTH) are configurable at design time. The core has clk (clk), active low reset (rst), start port (f_start) to trigger beginning of computation, done port (f_done) to signal end of computation and a port to specify size of FFT (N). In addition, the core has address ($f\_addr\_0,\ldots,f\_addr\_7$), data in ($f\_data\_in\_0,\ldots,f\_data\_in\_7$) and data out ($f\_data\_out\_0,\ldots,f\_data\_out\_7$) ports for each of RAM memory banks. Read-write operation on SetA (f_wren_A) and SetB (f_wren_B) memory are controlled via write enable signals. And twiddle factor ROM access is through address (f_addr_8 *and* f_addr_9) and data in ($f\_data\_in\_8$ *and* $f\_data\_in\_9$) ports.

3.1 Internal Architecture

When FFT processor is in operation, the overall dataflow through the processor follows a ping-pong logic. In even numbered stages, input data is read from SetA and output data is stored in SetB. In odd numbered stages, input data is read from SetB and output data is stored in SetB. The pipelined internal architecture of scalable FFT processor is shown in Figure 3.3.

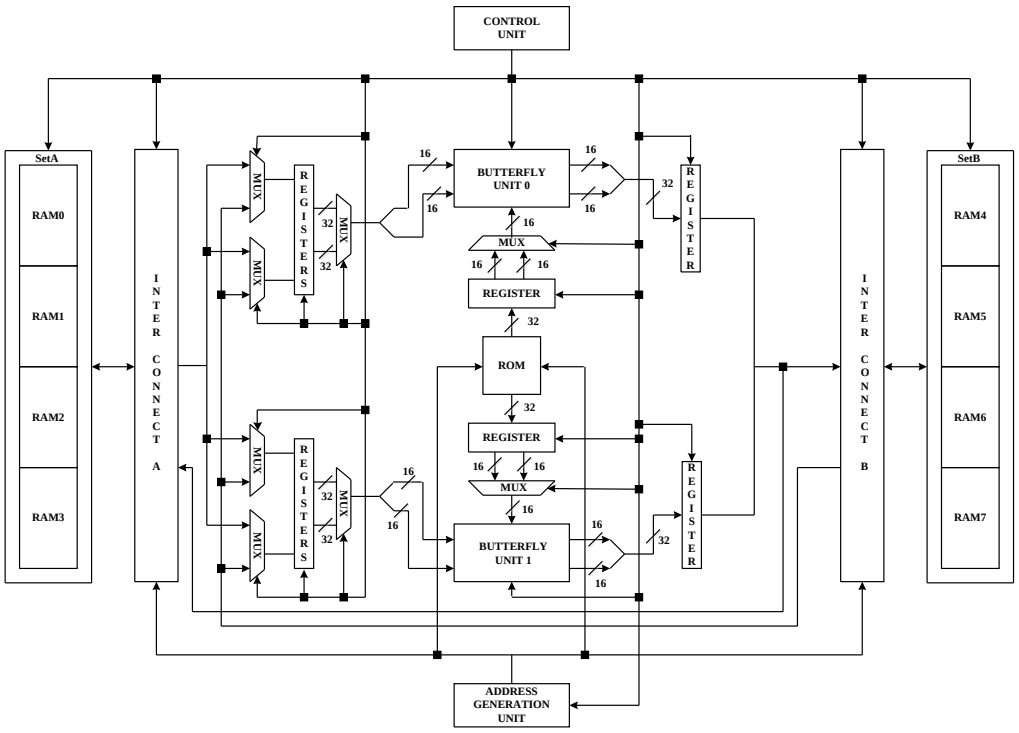

Figure 3.3: *FFT processor pipelined internal architecture.*

There are nine pipeline stages in the FFT core. The pipeline includes address generate, memory read, datapath from interconnectA to interconnectB and memory write. The nine pipeline stages include address generation stage, memory read stage, two stages before butterfly units, two stages inside butterfly units, two stages after butterfly units and memory write stage. Two pipeline stages of butterfly unit are shown in Figure 3.4. In the beginning of each stage, initial ten clock cycles are used up to fill up the pipeline because memory read/write latency is one clock cycle, address generation latency is one clock cycle and one extra clock cycle is required at the beginning of each stage. Hence, ten clock cycles are required to compute initial output samples in each stage and after which two outputs are computed every clock cycle. The number of clock cycles required by each stage in a pipelined architecture for an N-point FFT is given by,

$$cycles\_per\_stage = 10 + \frac{N}{2} \tag{3.1}$$

The total number of clock cycles required to compute FFT is given by,

$$cycles\_FFT = (cycles\_per\_stage * log_2(N)) + 2 \tag{3.2}$$

The FFT processor architecture is described in detail in terms of its components in the following sections.

3.2 Butterfly Unit

The butterfly unit shown in Figure 3.4 was adopted from [10] which was implemented by J.Takala et al.

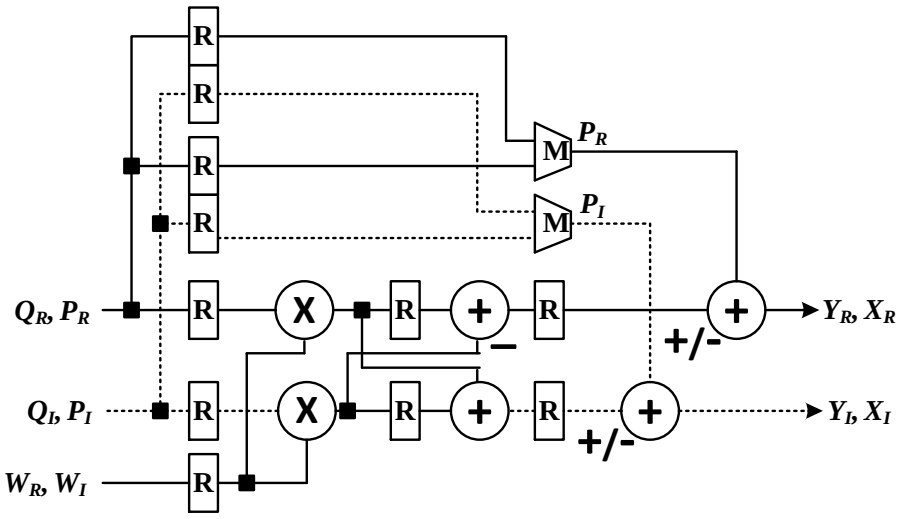

Figure 3.4: *Pipelined butterfly unit [10].*

The butterfly unit was designed to support radix-2 DIT butterfly operation. Since, FFT processor was required to be a fixed point processor, the butterfly unit implementation was modified to support Q-14 fixed point computation. There are three input ports, two output ports and they are 16-bit fixed point values. P, Q are complex input samples, W is twiddle factor and X, Y are complex output samples. Two out of three input ports are shared between Q_R (real part), Q_I (imaginary part) and P_R, P_I respectively. The third input port is for twiddle factor and it is shared between W_R and W_I. In Figure 3.4 the dotted lines indicate imaginary data and the thick line indicate real data. Inputs and outputs of butterfly unit are registered.

The butterfly unit consists of a complex multiplier and two adders as computational units. Complex multiplier contains two bit parallel real multipliers and two adders. Complex multiplication operation is computed in two staged pipeline to reduce number of real multipliers from four to two. The reduction in number of multipliers through pipeline operation is a reasonable compromise between high throughput, area and power efficiency.

When butterfly unit is in operation Q and P are read every alternate clock cycle and same is the case with W_R and W_I. Two clock cycles are required to fill up the pipeline in the beginning and thereafter every clock cycle one output (X or Y) of butterfly operation is produced. Thus, every two clock cycles one butterfly operation is completed per butterfly unit. The FFT processor employs two such butterfly units operating in parallel at any given time as shown in Figure 3.3.

The complex multiplier inside butterfly unit forms critical path of the FFT processor. The critical path includes a multiplier and an adder. The butterfly unit operation is controlled by control unit which issues register loads, multiplexer signals at appropriate time intervals. Two butterfly units operate in parallel in the FFT processor to compute two output samples per clock cycles which increases the throughput of the processor.

Figure 3.5 shows input/output pins of a butterfly unit. Butterfly unit has a clock port *clk* and an active low reset port *rst*. Imaginary parts of inputs *P, Q* are multiplexd into an input port *PQI* while real parts of *P, Q* are multiplexed into another port *PQ_R*. Real and imaginary parts of twiddle factor *W* are multiplexed into *WRI*. *load_P* and and *load_P2* are load signals for *P* input registers, *load_Q* is load signal for *Q* input register and *load_W* is load signal for *W* input register.

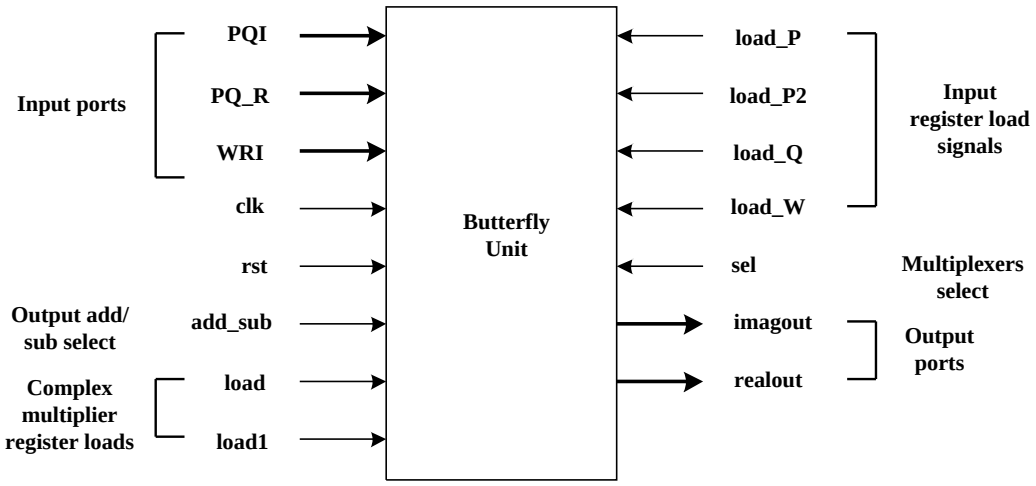

Figure 3.5: *Butterfly unit pin details.*

The load signals *load* and *load1* are for multiplier output registers and complex multiplier adder output registers respectively. Multiplexers are controlled via *sel* control signal. Addition or subtraction operations which are part of butterfly unit are controlled using *add_sub* port. The output ports *realout* and *imgout* are real and imaginary parts of output data respectively.

Waveforms of input/output ports of butterfly unit are shown in Figure 3.6 for 16-point FFT. The timing of control signals to butterfly unit at different stages of FFT are shown in the figure.

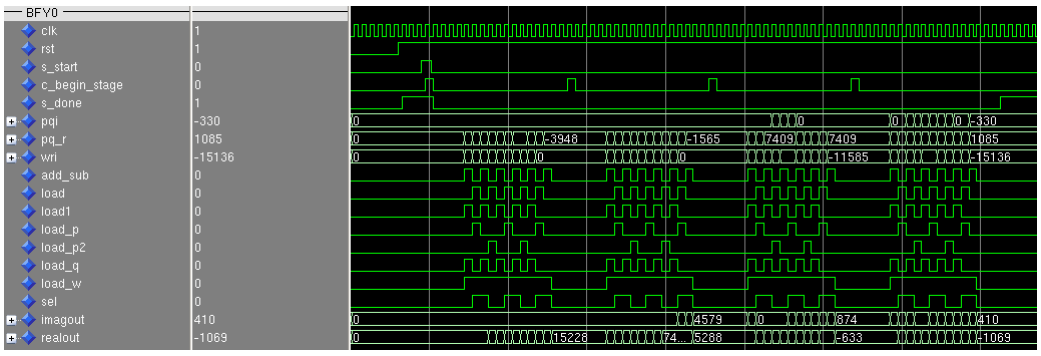

Figure 3.6: *Butterfly unit waveform for 16-point FFT.*

There are four stages in a 16-point FFT and timing of pipelined butterfly unit control signals follow similar pattern in every stage. The signals *c_ start*, *begin_ stage* and *s_ done* are not part of butterfly unit. But those signals are shown to clearly describe waveforms at the beginning of FFT computation, during computation and at the end of computation. However, *c_ start* indicates start of computation, *begin_ stage* indicates beginning of a stage and *s_ done* indicates end of FFT computation. Input data latency is four clock cycles and output data latency is seven clock cycles with respect to beginning of stage.

In a butterfly unit, multiplication is the most power, time and area consuming portion. Hence, multiplier operation is explained in more detail below.

3.2.1 Bit Parallel Multiplier

The multiplication of two complex numbers W and Q is given by,

$$WQ = (W_R + jW_I)(Q_R + jQ_I) = (W_RQ_R - W_IQ_I) + j(W_RQ_I + W_IQ_R) \quad (3.3)$$

where suffix W_R, Q_R are real parts and W_I, Q_I are imaginary parts. If we try to implement equation 3.3 straight away and without optimization it requires four multipliers and two adders. The critical path is decided by multiplier and adder combination. We can optimize complex multiplication to use three multipliers instead of four by following the equation 3.4 below.

$$WQ = W_I(Q_R - Q_I) + Q_R(W_R - W_I) + j[W_I(Q_R - Q_I) + Q_I(W_R + W_I)] \quad (3.4)$$

The area and power consumption can be reduced if two clock cycles are used for complex multiplication. As shown in Figure 3.4 it is implemented by sharing a common bus for both input samples P and Q, it means that first Q is read followed by P. The complex multiplication is pipelined by using two clock cycles for computation. The pipelined multiplication is given by,

$$A1 = Q_RW_R; \quad B1 = Q_IW_R \quad (cycle\#1)$$

$$A2 = Q_RW_I; \quad B2 = Q_IW_I \quad (cycle\#2)$$

$$WQ = (A1 - B2) + j(B1 + A2) \quad (3.5)$$

This approach uses only two multipliers and two adders. As demonstrated above it is not necessary to have real and imaginary parts of twiddle factor in the same clock cycle. Hence, a single bus can be shared to read real and imaginary parts of twiddle factor in consecutive clock cycles. The outputs of real valued multipliers are

registered thereby reducing critical path of complex multiplier and in turn critical path of butterfly unit. The outputs of adders of complex multiplier are registered as well. The butterfly unit is implemented by pairing up complex multiplier along with adders required for butterfly operation. Inputs of the butterfly unit are registered and it needs two sets of input registers for operand P since, next P is read while current W^*Q is computed.

3.3 Data Memory (RAM)

The input samples, intermediate samples and the output samples of FFT computation are stored in data memory. Data memory is RAM based and there are two sets called SetA and SetB. Butterfly operation is performed by reading input samples from one set of memory and writing output samples to a different set of memory. Hence, two sets of memory are used for storing samples at different stages during computation. SetA memory includes four memory banks RAM0, RAM1, RAM2, RAM3 while SetB memory includes four memory banks RAM4, RAM5, RAM6 and RAM7. Since, two butterfly units require four input samples per clock cycle we chose to use four memory banks to store data samples. The maximum size of each memory bank is decided by the maximum size of FFT computation N_{max} selected during design time. Maximum size of a memory bank is given by,

$$ram\_bank\_size = \frac{N_{max}}{4} \tag{3.6}$$

where the size is measured in terms of 32-bit words. The size of a memory bank decides its address bus width which is expressed as $log_2(ram\_bank\_size)$.

In memory a complex data sample is stored as a 32-bit word, higher 16-bits constitute real part while the lower 16-bits constitute imaginary part. Data format stored in memory is pictorially described as shown in Figure 3.7.

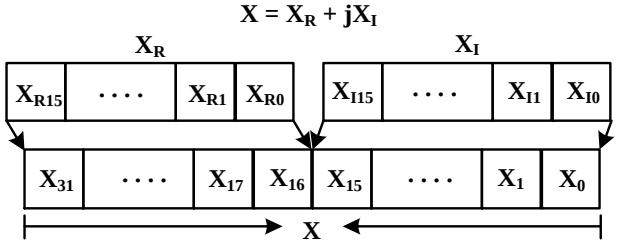

Figure 3.7: *Data format stored in memory.*

In the beginning of FFT computation input samples are always stored in SetA. The

input samples are bit reversed according to DIT FFT and split equally among four memory banks. Figure 3.8 describes order of input samples stored in memory at the beginning of FFT computation.

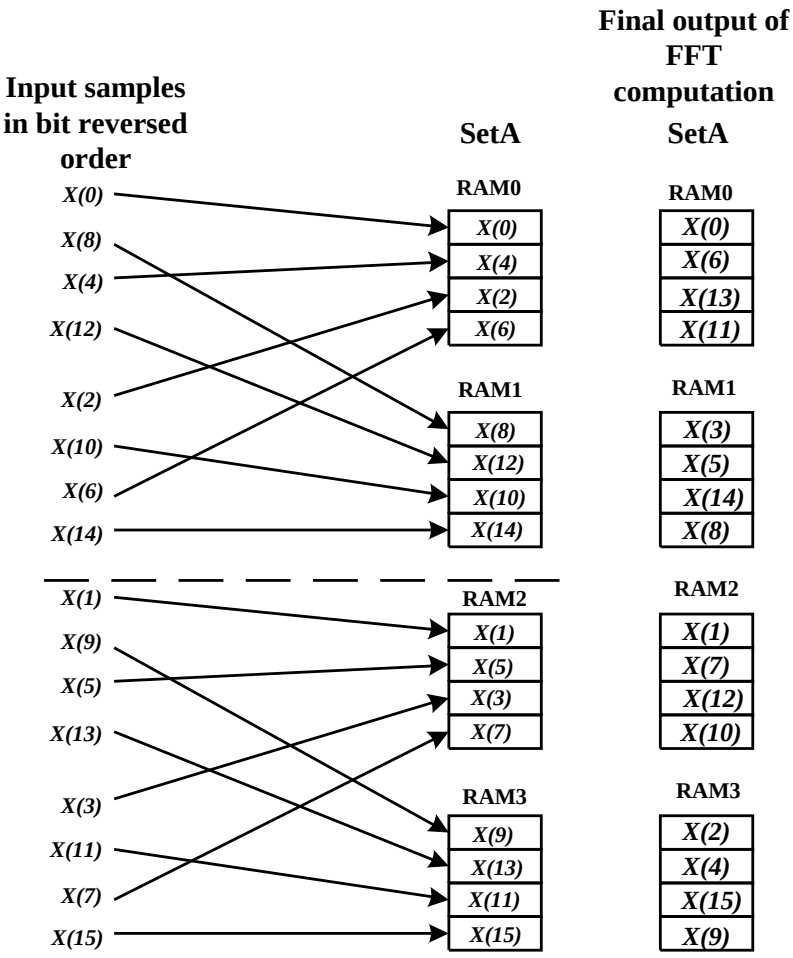

Figure 3.8: *Order of input sample at the beginning and the order of final output of 16-point FFT computation.*

The bit reversed input samples are equally split into two halves, the upper half inputs are stored in RAM0, RAM1 and the lower half in RAM2, RAM3. Order of input samples are such that it provides conflict free access for butterfly operation.

Final output of an N-point FFT computation might be available in SetA or SetB depending on number of stages. If number of stages are even, final outputs are stored in SetA and if number of stages are odd, final outputs are stored in SetB. Since, for a 16-point FFT number of stages are even, final outputs would be available in SetA and the order of outputs would be as shown in Figure 3.8. The order of outputs is decided by dataflow algorithm described in detail in later part of this document.

The memory banks are clocked dual port memories which enable access from within FFT core as well as from external world. Memory access (read-write) latency is one clock cycle.

Figure 3.9 shows input/output ports of a RAM memory bank. Generic port *FILE_NAME* is to specify memory initialization file and *ADDR_WIDTH* is to specify address port width during design time. RAM bank is a dual port memory and hence it has two sets of data and address ports.

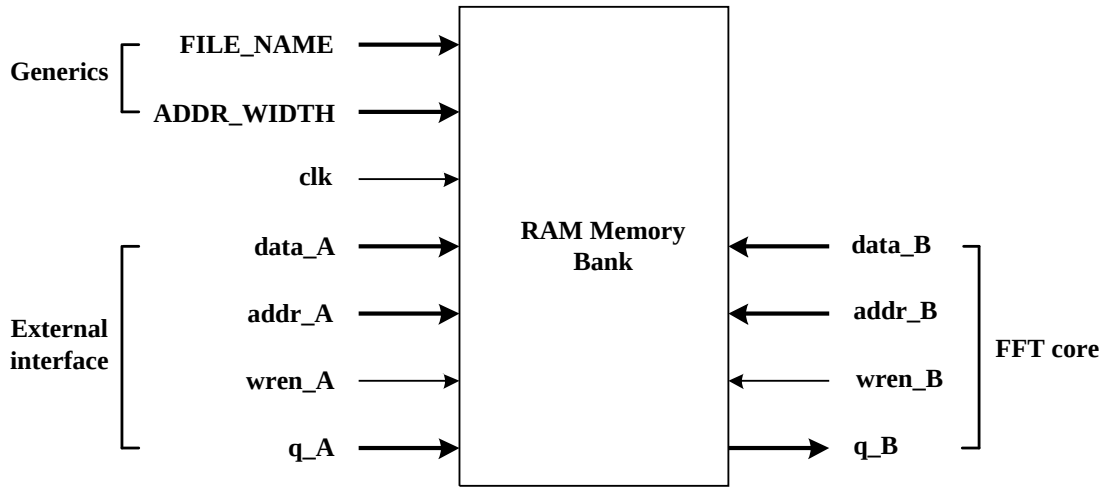

Figure 3.9: *RAM memory bank pin details.*

A RAM memory bank consists of input data ports (*data_A, data_B*), address ports (*addr_A, addr_B*), read-write enable signal (*wren_A, wren_B*) and output data ports (*q_A, q_B*). Ports with suffix 'A' are external interface ports and ports with suffix 'B' are FFT core interface ports.

Waveforms of SetA RAM bank (RAM0) ports are shown in Figure 3.10 for 16-point FFT. Write operation happens when *wren_B* is '1' and read operation happens when it is '0'. Memory read-write access latency is single clock cycle.

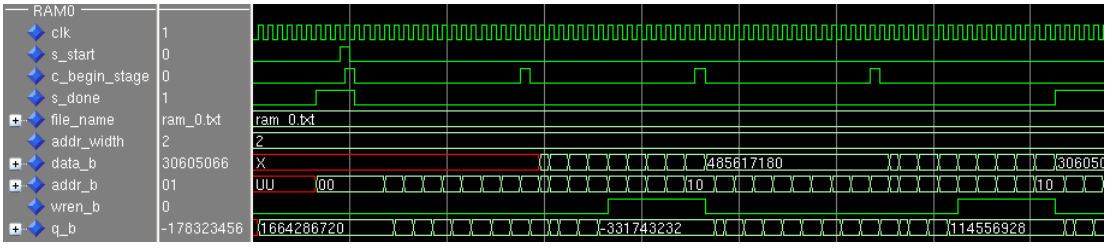

Figure 3.10: *SetA RAM memory bank (RAM0) waveforms.*

In the first stage data is read from SetA memory and in the next stage data is written to SetA memory. Likewise, read-write operations are switched between SetA and SetB memory in alternate stages.

Waveforms of SetB RAM bank (RAM4) ports are shown in Figure 3.11 for 16-point FFT.

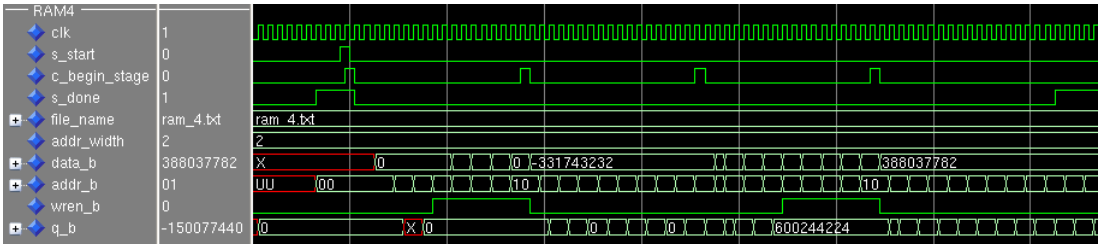

Figure 3.11: *SetB RAM memory bank (RAM4) waveforms.*

In the first stage data is written to SetB memory and in the next stage data is read from it. Read-write operations are switched between SetB and SetA memory in alternate stages.

Data samples are stored in data memory, to store twiddle factors we need twiddle factor memory which is described in detail in the next section.

3.4 Twiddle Factor Memory (ROM)

The twiddle factor memory is to store twiddle factors required by butterfly unit0 and butterfly unit1 during FFT computation. Twiddle factor memory is a clocked dual port ROM which allows reading two twiddle factors per clock cycle simultaneously as shown in Figure 3.3. For an N-point FFT, a maximum of $\frac{N}{2}$ twiddle factors are required for computation. Hence, a maximum of $\frac{N_{max}}{2}$ twiddle factors are stored in ROM at design time in order to support FFT computation from 16-point upto N_{max}-point. Hence, size of ROM is given by,

$$rom\_size = \frac{N_{max}}{2} \tag{3.7}$$

where size is measured in terms of 32-bit words. Since, ROM address bus width depends on maximum size of ROM, address bus width is expressed as $log_2(rom\_size)$.

Twiddle factors stored in ROM are in natural order and it is illustrated for a 16-point FFT in Figure 3.12. Twiddle factor is a complex value in which 16-bit real and imaginary parts are packed into a 32-bit word as shown in Figure 3.7.

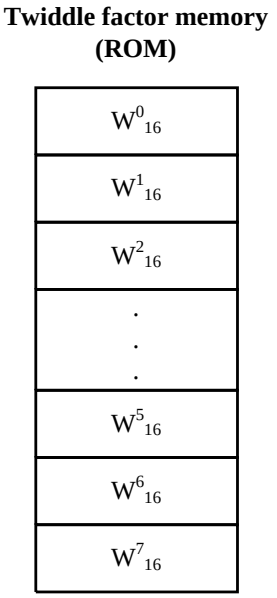

Figure 3.12: *Order of twiddle factors stored in ROM.*

The 32-bit twiddle factor is read from memory and it is unpacked into real and imaginary parts before feeding them to butterfly units as shown in Figure 3.3. The real and imaginary parts are fed through a common input port to butterfly unit in alternative clock cycles as shown in Figure 3.4.

Figure 3.13 shows input/output ports of a ROM memory.

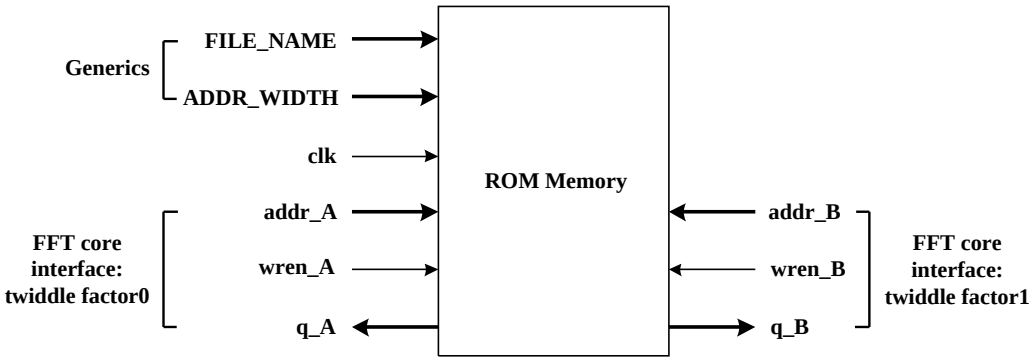

Figure 3.13: *ROM memory pin details.*

Generic port *FILE_ NAME* is to specify memory initialization file and *ADDR_ WIDTH*

is to specify address port width during design time. ROM bank is a dual port memory and hence it has two sets of data and address ports. A ROM memory consists of input data ports ($data\_A$, $data\_B$), address ports ($addr\_A$, $addr\_B$), read enable signal ($wren\_A$, $wren\_B$) and output data ports (q_A, q_B). The address, data and read enable ports interface with FFT core. Ports with suffix 'A' are meant for buttefly unit0 (twiddle factor0) while ports with suffix 'B' are meant for butterfly unit1 (twiddle factor1) inside FFT core.

Waveforms of ROM memory ports are shown in Figure 3.14 for 16-point FFT.

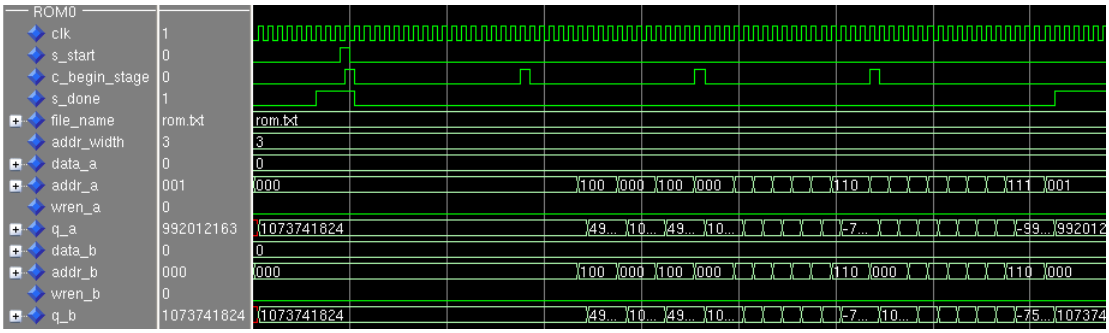

Figure 3.14: *ROM memory waveforms.*

Since, two twiddle factors are required for computation, dual port ROM is used to store twiddle factors. Read operation happens when $wren\_A$, $wren\_B$ are set to '0'.

We understood how twiddle factors are stored in ROM and how data samples are stored in data memory. Now, it is necessary to understand how data samples are guided to butterfly units after they are read from data memory and it is explained in following section.

3.5 Interconnect

The function of interconnect is to route data between memory and butterfly units as well as route address between memory and address generation unit as shown in Figure 3.3. An interconnect consists a number of multiplexers whose outputs are registered. The multiplexers act a switches linking inputs to appropriate outputs based on select signals from control unit.

An interconnect is described as in Figure 3.15.

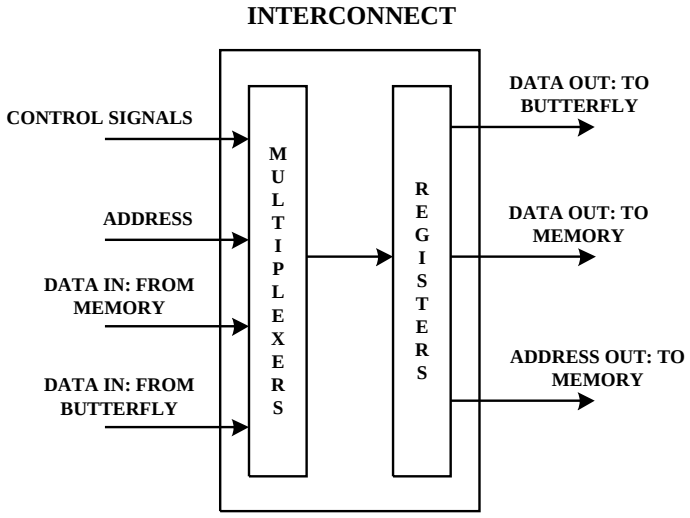

Figure 3.15: *Interconnect internal architecture and external interface.*

The address generation unit sends read/write addresses to interconnect which are routed to appropriate memory banks based on control signals from control unit. Interconnect also receives data samples from memory banks which are routed to appropriate butterfly units depending on control signals. After butterfly operation, data received from butterfly units are routed to memory for storage. Two such interconnects are used in FFT processor architecture and they are interconnectA and interconnectB shown in Figure 3.3. The interconnectA forms a connection between butterfly units and memory SetA while interconnectB forms a connection between butterfly units and memory SetB. Both the interconnects form link between address generation unit and memory sets SetA and SetB.

Figure 3.16 shows pin details of an interconnect module. Generic port *N_WIDTH* allows scalable value for N while *ADDR_WIDTH* is to scale address corresponding to N. Interconnect has clock (*clk*) and active low reset (*rst*) ports. The port *i_RW* indicates read/write operation for SetA or SetB memory. Last stage of computation is indicated by *i_last_stage* port. Whether data read from memory has to be flipped or not is indicated by *i_input_flip* port. Memory read address input ports are *i_read_data* and *i_read_data1* which are received from address generation unit. The *read_data1* is nine clock cycles delayed version of *read_data* because write operation begins after nine cycles from beginning of each stage. Memory store addresses are *i_store_add* and *i_store_sub* received from address generation unit. The output memory addresses are *i_addr_RAM0,...,i_addr_RAM3*, these are RAM bank addresses. Ports *i_bfy0_in_first, i_bfy0_in_second, i_bfy1_in_first, i_bfy1_in_second* are outputs of butterfly units and inputs to interconnect.

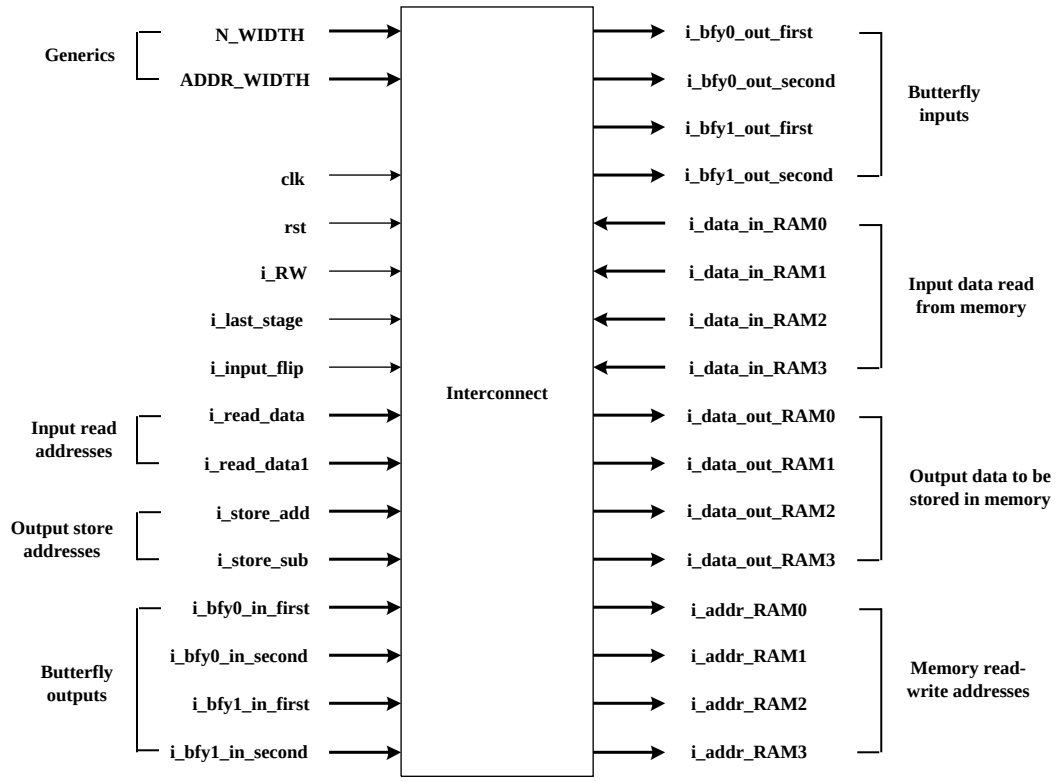

Figure 3.16: *Interconnect pin details.*

Ports *i_bfy0_out_first, i_bfy0_out_second, i_bfy1_out_first, i_bfy1_out_second* are outputs of interconnect and inputs to butterfly units. Data input from RAM banks are received through ports *i_data_in_RAM0,...,i_data_in_RAM3*. And data outputs to RAM banks are sent through *i_data_out_RAM0, ...,i_data_out_RAM3*.

Figure 3.17 shows waveforms for interconnectA for 16-point FFT computation.

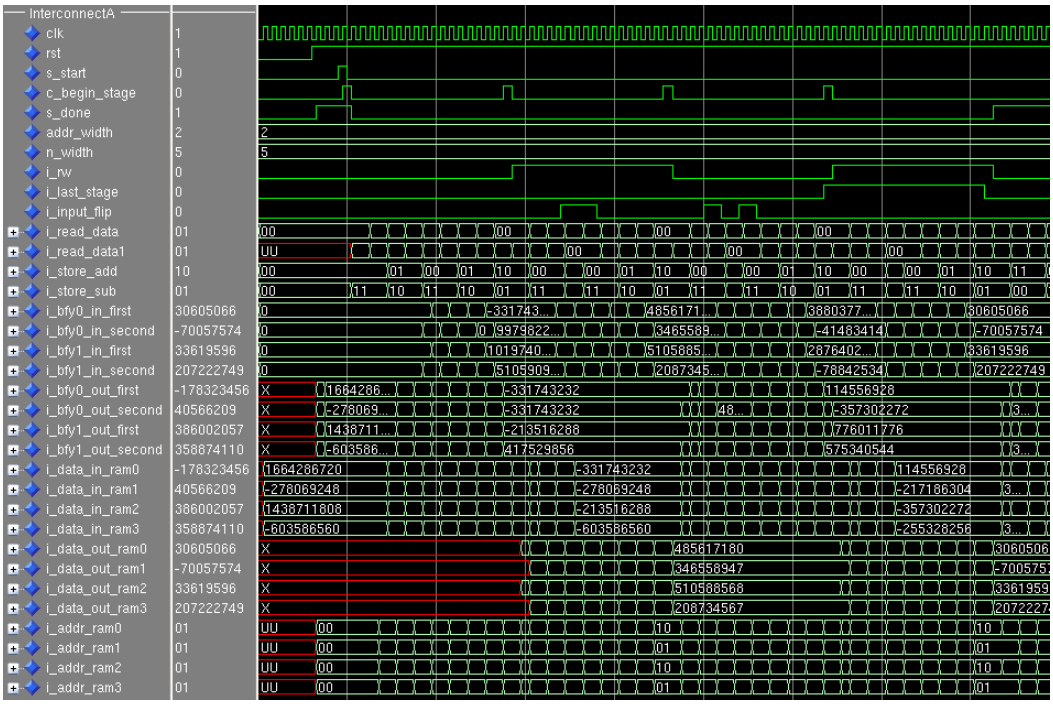

Figure 3.17: *InterconnectA waveforms.*

Since, it is linked to SetA memory, memory read occurs every even numbered stage. And memory write occurs every odd numbered stage as shown in the waveforms. Figure 3.18 shows waveforms for interconnectB for 16-point FFT computation.

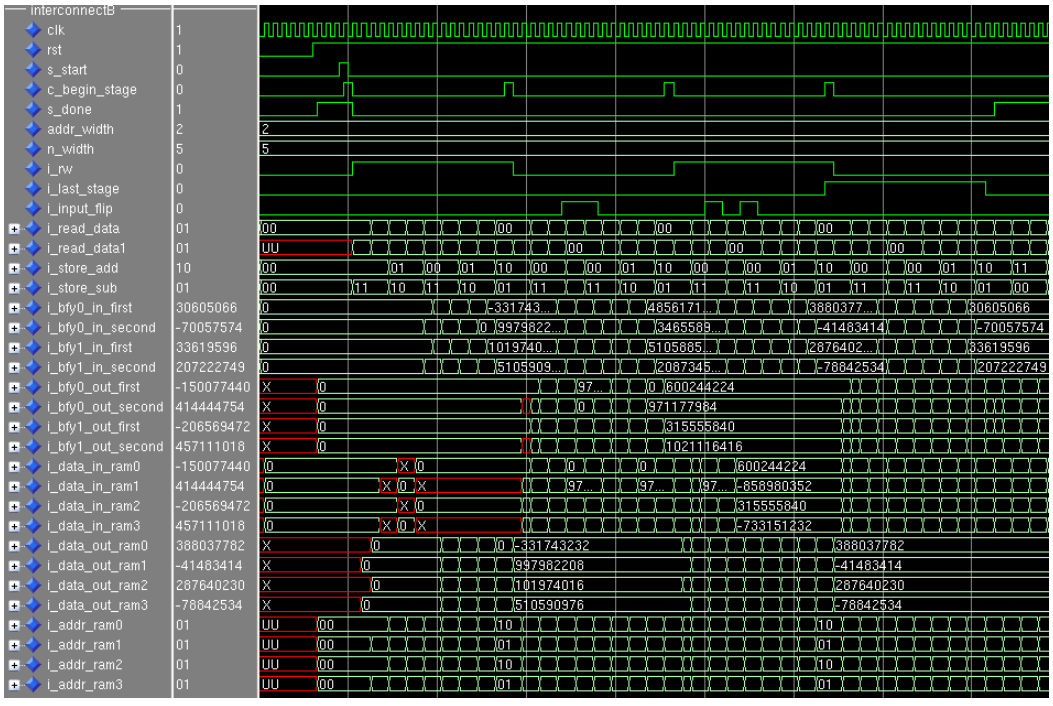

Figure 3.18: *InterconnectB waveforms.*

Since, it is linked to SetB memory, memory read occurs every odd numbered stage. And memory write occurs every even numbered stage as shown in the waveforms.

Interconnects form link between data memory and address generation unit, the data memory was dealt in detail in previous sections. Hence, following section deals with address generation unit in detail.

3.6 Address Generation Unit

The address generation unit is required to generate addresses for butterfly inputs, twiddle factors and butterfly outputs. And it should be capable of supporting N-point FFT computation in order to support scalable architecture. A novel address generation algorithm was developed to generate addresses for inputs, twiddle factors and outputs. The algorithm is also capable of supporting N-point FFT computation and provides conflict free access of data samples during computation. The address generation algorithm requires two m-bit counters, where $m = log_2(\frac{N}{4})$. The algorithm is described in detail below.

1. *Address to read inputs from memory:*
   A simple m-bit counter is used to generate read address.

$$read\_data = [a_{m-1}a_{m-2}.....a_1a_0] \tag{3.8}$$

2. *Address to store outputs to memory:*
   Another simple m-bit counter is used to generate store address.

$$store\_add = [b_{m-1}b_{m-2}.....b_1b_0] \tag{3.9}$$

$$store\_sub = [b'_{m-1}b'_{m-2}.....b'_1b'_0] \tag{3.10}$$

3. *Address to read twiddle factors from ROM:*
   For an N-point FFT there are log_2N number of stages, we assume stage index as s which is incremented at the end of each stage. Hence, s can take values $s = 0, 1, 2, ...log_2(N)$-1.

   * *Twiddle factor addresses for stages except last stage ($s = 0, 1, ...log_2(N)$-2):*
   Following steps are executed in sequence to generate twiddle factor addresses in the current stage.

   (i) $[d_{m-1}..d_0] = [a_{m-1}...a_0]\ XOR\ [0a_{m-1}...a_1]$

   (ii) $count\_gray = [0d_{m-1}d_{m-2}.....d_1d_0]$

$$= [e_m e_{m-1}..e_1 e_0]$$

(iii) $coefx = [e_m....e_{m-s}0_{m-s}0_{m-s-1}...0_1 0_0]$

$$= [f_{m+1} f_m ... f_1 f_0]$$

(iv) $Coef0Addr = [f_m f_{m-1} ... f_1 f_0]$

(v) $Coef1Addr = Coef0Addr$

* *Twiddle factor addresses for last stage ($s = log_2(N)$-1):*
Following steps are executed in sequence to generate twiddle factor addresses for last stage. Note: $coefy = [0_{m+1}0_m...0_1 1]$ at the beginning of the stage.

(i) $coefy = [f_{m+1} f_m ... f_1 f_0]$

(ii) $sum = [a_{m-1} a_{m-2}..a_1 a_0] + [f_m f_{m-1}..f_2 f_1]$

$$= [g_{m-1} g_{m-2} ... g_1 g_0]$$

(iii) $coefy = [0 g_{m-1} g_{m-2} ... g_1 g_0 f_0']$

$$= [f_{m+1} f_m ... f_0]$$

$f_m =$ '0' for first $\frac{N}{4}$ butterfly computations.

$f_m =$ '1' for next $\frac{N}{4}$ butterfly computations.

(iv) $Coef0Addr = [f_m f_{m-1} ... f_1 f_0]$

(v) $Coef1Addr = [f_m f_{m-1} ... f_1 f_0']$

To read inputs from SetA or SetB memory, *read_ data* address from step 1 is required. The *read_ data* is generated using an m-bit counter. To store outputs of butterfly units store addresses from step 2 are required. Two store addresses *store_ add*, *store_ sub* are required for storing output data. There are two butterfly outputs wherein one is result of addition and the other is result of subtraction. Outputs of butterfly units have to be stored such that there are no access conflicts while storing as well as when they are read in next stage. Hence, *store_ add* is required to store result of addition and *store_ sub* is required to store result of subtraction. The *store_ add* is generated using an m-bit counter while *store_ sub* is one's complement of *store_ add*. Twiddle factor addresses are generated in step 3 wherein *Coef0Addr* corresponds to butterfly unit0 and *Coef1Addr* corresponds to butterfly unit1. The logic is different for last stage and rest of the stages. To generate twiddle factor addresses, the steps have to be followed in the specified order. For stages except last stage, using *read_ data* a gray code value is generated which is appended with

suitable number of zeros. The number of zeros appended to form address depends on stage index s. Both the twiddle factor addresses are same for stages except the last stage. The last stage twiddle factor address is generated by adding *read_ data* with previous value of *Coef0Addr* and the last bit of result address is set to '0' for first half of stage and set to '1' for second half of the stage. The *Coef1Addr* is formed by inverting the least significant bit of *Coef0Addr*.

Address generation unit internal logic is pictorially presented in Figure 3.19. It consists of two m-bit counters one each for generating *read_ data* and *store_ add*.

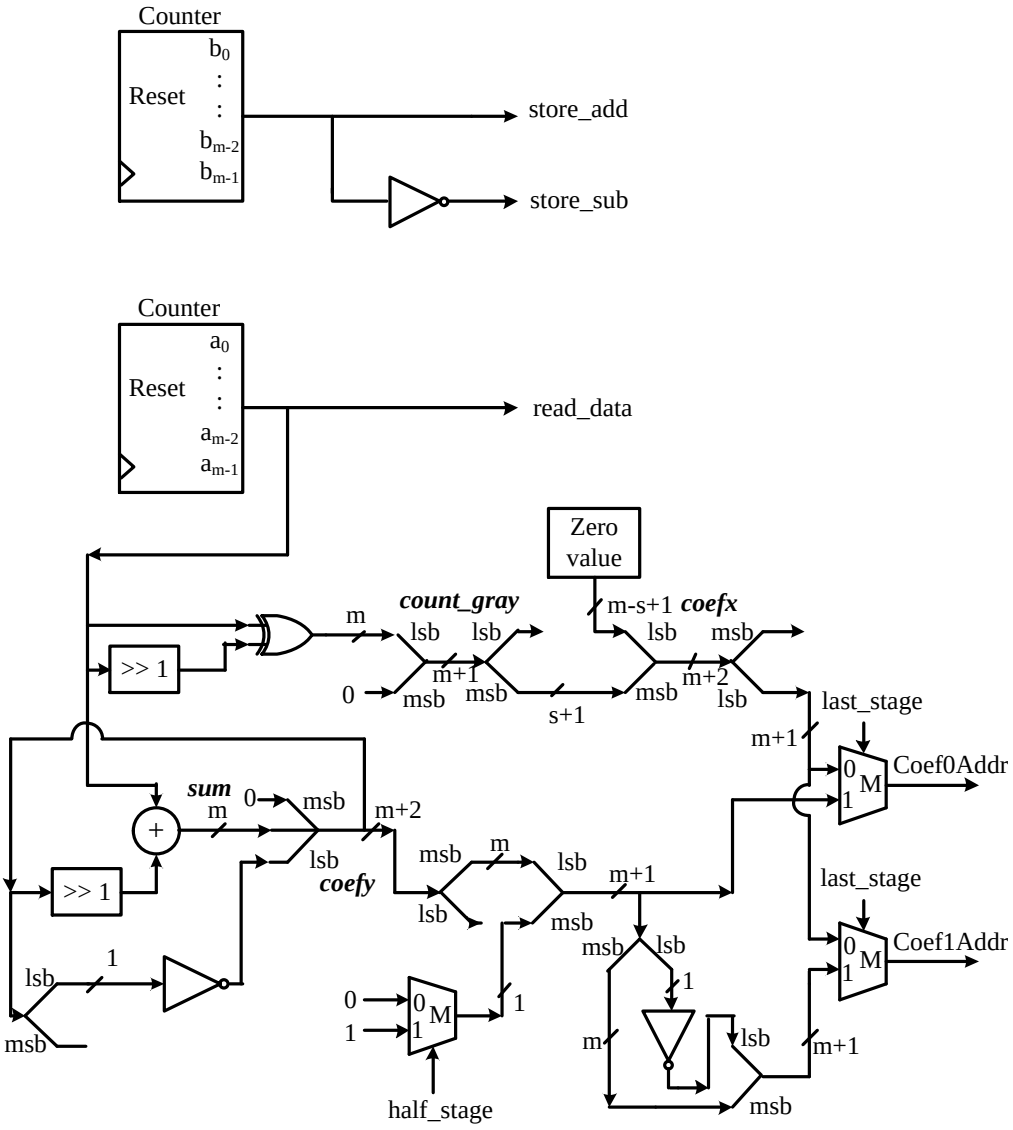

Figure 3.19: *Address generation unit internal logic.*

The twiddle factor address generation involves logic for last stage and for rest of the stages. For rest of the stages gray code is generated from *read_ data* and appended with zeros supplied by *zero value* module. And *Coef0Addr* and *Coef1Addr* are the same for rest of the stages as described before. For the last stage, previous *Coef0Addr* value is added with *read_ data*. The last bit of resulting address is set to '0' for first half of the stage and it is set '1' for next half of the last stage. The *Coef1Addr* is formed by inverting the leas significant bit of *Coef0Addr*. The select signal *last_ stage* differentiates last stage from rest of the stages. And *half_ stage* select signal differentiates first half of last stage from its second half.

Figure 3.20 shows pin details of address generation unit.

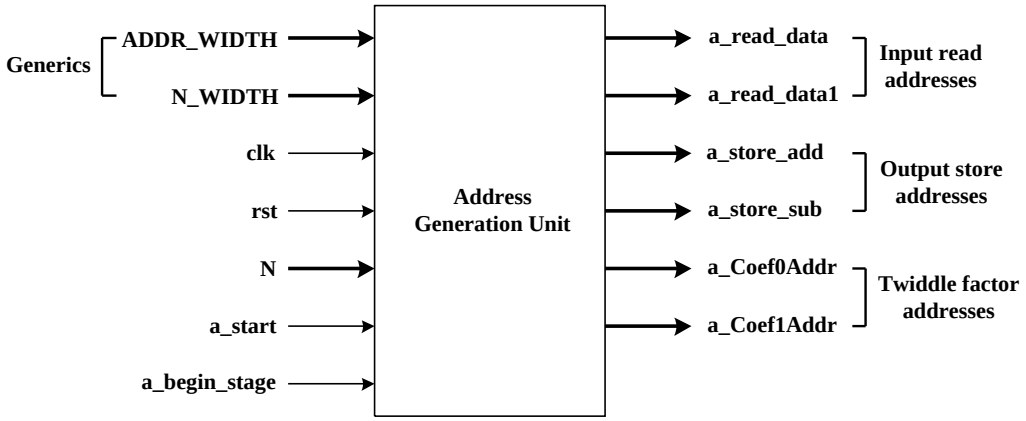

Figure 3.20: *Address generation unit pin details.*

Address generation unit has generic ports for specifying address width (*ADDR_ WIDTH*) and for specifying N width (*N_ WIDTH*). The clock port is *clk*, active low reset port is *rst* and FFT size port is *N*. The port *a_ start* is a start signal to start computation while port *a_ begin_ stage* is to indicate the beginning of each FFT stage. Read address ports are *a_ read_ data* and *a_ read_ data1*. *a_ read_ data1* is nine clock cycle delayed with respect to beginning of each stage. And it is required by interconnects in routing data to memory. Store address ports are *a_ store_ add* and *a_ store_ sub*. The twiddle factor address ports are *a_ Coef0Addr* and *a_ Coef1Addr*.

Figure 3.21 shows waveforms address generation unit for 16-point FFT.

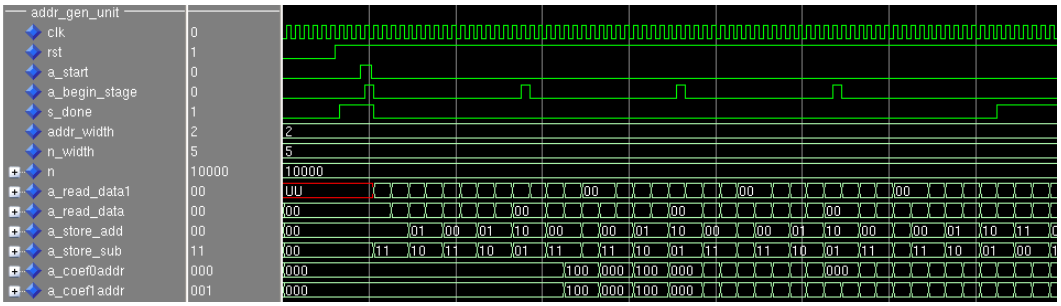

Figure 3.21: *Address generation unit waveforms.*

Address generation starts immediately after begin stage signal is received from control unit. And for every two clock cycles a new address is generated.

As we saw earlier, address unit generates addresses but there needs to be a module which co-ordinates activities of all the components in the processor. And such a module is called control unit which is discussed in detail in the following section.

3.7 Control Unit

The control unit controls, co-ordinates and synchronizes activities of rest of the components as shown in Figure 3.3. The timing of control signals issued have to be accurate in order to synchronize activities of different components. Moore state machine was applied to implement control unit and it is shown in Figure 3.22.

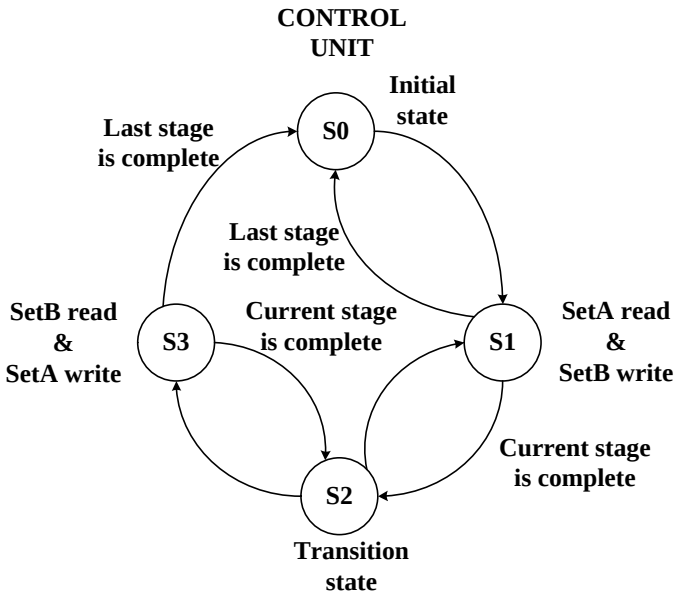

Figure 3.22: *Control unit state diagram.*

Control unit begins with initial state *S0* also called as reset state. In initial state, all the signals and variables are initialized to appropriate values. After initializing required signals and variables it changes state to *S1*. In *S1*, it generates timing for control signals related to following activities: address generation, SetA memory read, route inputs via interconnectA to butterfly units, butterfly operation, route outputs to SetB memory via interconnectB and store outputs to SetB memory. It moves to transition state *S2* if *S1* does not correspond to last stage of FFT. Otherwise, it moves to initial state *S0* indicating that it is end of FFT computation. In transition state *S2*, necessary control signals are re-initialized so as to prepare for generating control signals for next stage of FFT.

After re-initializing required signals and variables in transition state *S2*, it moves to *S3*. In *S3*, it generates timing for control signals related to following activities: address generation, SetB memory read, route inputs via interconnectB to butterfly units, butterfly operation, route outputs to SetA memory via interconnectA and store outputs to SetA memory. It moves to transition state *S2* if *S1* does not correspond to last stage of FFT. Otherwise, it moves to initial state *S0* indicating that it is end of FFT computation. In transition state *S2*, necessary control signals are re-initialized so as to prepare for generating control signals for next stage of FFT.

Likewise, the state transitions $S1 \rightarrow S2 \rightarrow S3$ and $S3 \rightarrow S2 \rightarrow S1$ are iterated in alternative FFT stages until completion of FFT computation. It begins with state *S0* and after completion of all stages it returns to *S0*. It is important to note that state *S1* generates control signals for even numbered stages, while state *S3* generates control signals for odd numbered stages.

Figure 3.23 shows pin details of control unit.

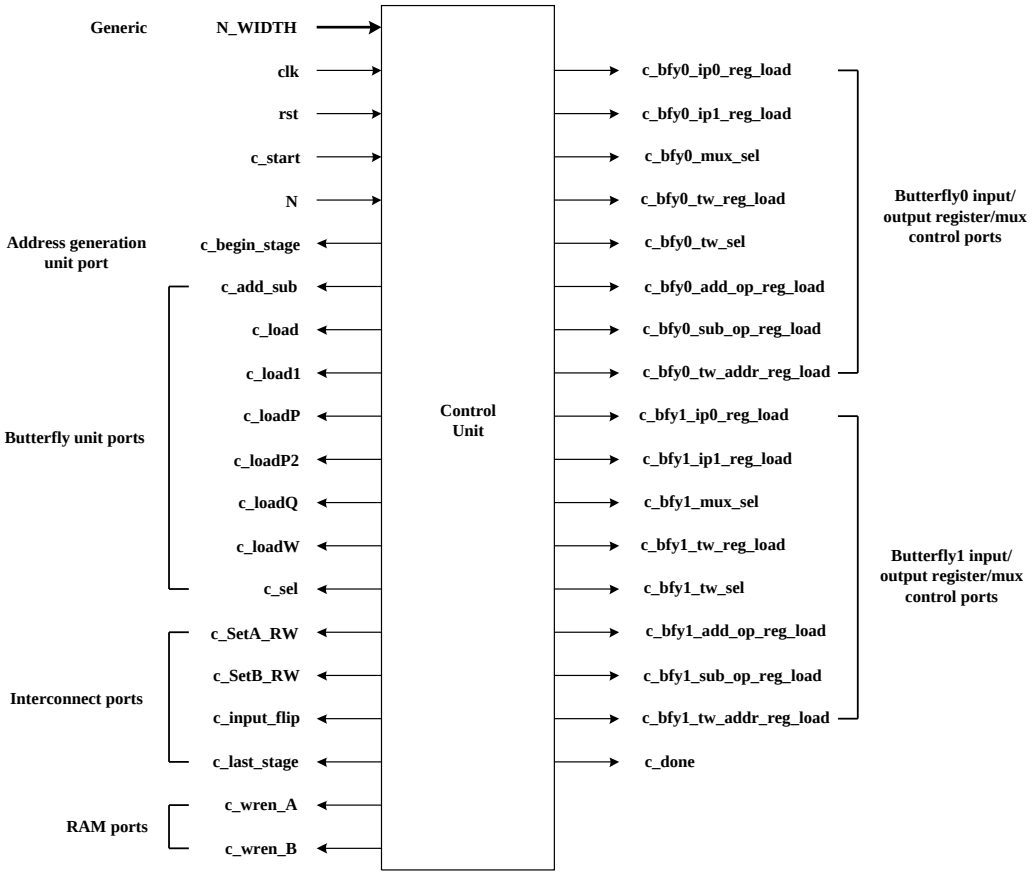

Figure 3.23: *Control unit pin details.*

Control unit signals, co-ordinate and synchronize activities of rest of the components of the processor. Generic port to configure size of FFT is *N*, there is clk port *clk*, reset port *rst* and *c_start* port to trigger beginning of computation. Beginning of each stage is synchronized with address generation unit through port *c_begin_stage*.

Ports corresponding to butterfly unit are: *c_add_sub* to choose addition/subtraction at the butterfly output, ports *c_load* and *c_load1* for complex multiplier registers load signals, ports *c_loadP, c_loadP2, c_loadQ, c_loadW* for input register loads, port *c_sel* for multiplexer select signal.

Ports corresponding to interconnects are: ports *c_SetA_RW, c_SetB_RW* for signaling read-write operation for different sets in each stage,port *c_input_flip* to indicate whether flip is required for current inputs read from memory, port *c_last_stage* to signal last stage of FFT.

Ports specific to RAM memory banks are: port *c_wren_A* is SetA read-write signal,

port *c_wren_B* is SetB read-write signal.

Butterfly unit0 register/multiplexer control ports are: input register load ports are *c_bfy0_ip0_reg_load* and *c_bfy0_ip1_reg_load*, input mux select port is *c_bfy0_mux_sel*, twiddle factor input register load port is *c_bfy0_tw_reg_load*, input twiddle factor select port is *c_bfy0_tw_sel*, twiddle factor address register load port is *c_bfy0_tw_addr_reg_load*, butterfly output register load ports are *c_bfy0_add_op_reg_load* and *c_bfy0_sub_op_reg_load*.

Butterfly unit1 register/multiplexer control ports are: input register load ports are *c_bfy1_ip0_reg_load* and *c_bfy1_ip1_reg_load*, input mux select port is *c_bfy1_mux_sel*, twiddle factor input register load port is *c_bfy1_tw_reg_load*, input twiddle factor select port is *c_bfy1_tw_sel*, twiddle factor address register load port is *c_bfy1_tw_addr_reg_load*, butterfly output register load ports are *c_bfy1_add_op_reg_load* and *c_bfy1_sub_op_reg_load*.

Figure 3.24 shows waveforms of control unit for a 16-point FFT computation.

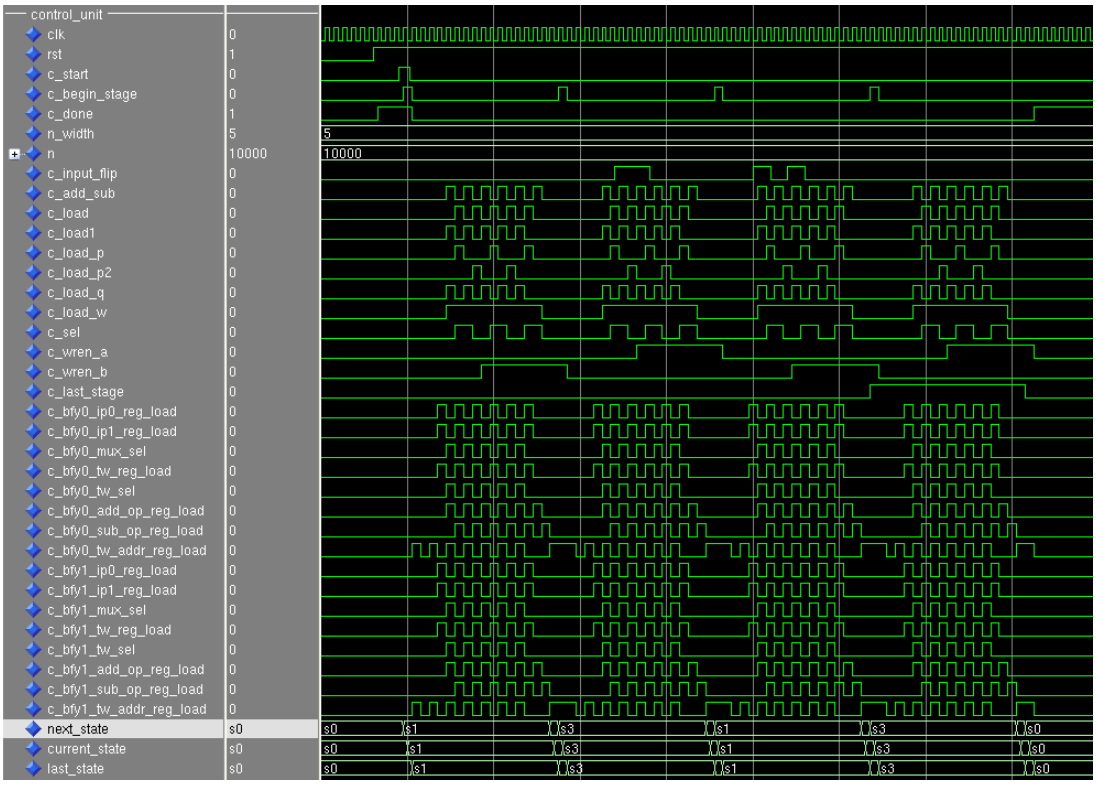

Figure 3.24: *Control unit waveforms.*

State transition throughout different FFT stages is evident in the waveforms and is in accordance with earlier description. And butterfly unit control signals are symmetric across FFT stages.

After understanding each component of FFT processor in detail, its functionality is better understood through understanding the dataflow through various components during computation. In this regard, a novel dataflow algorithm is described in the following section.

3.8 Dataflow Algorithm

The flow of data across various components of processor is the basis of FFT processor architecture. The dataflow algorithm describes dataflow through memory, interconnects and butterfly units. It also describes order of reading inputs from memory, storing outputs to memory, routing inputs to butterfly units, order of twiddle factor access and routing butterfly outputs to memory. A novel dataflow algorithm is pictorially described in Figure 3.25, the thick line describes butterfly unit0 and dotted line describes butterfly unit1.

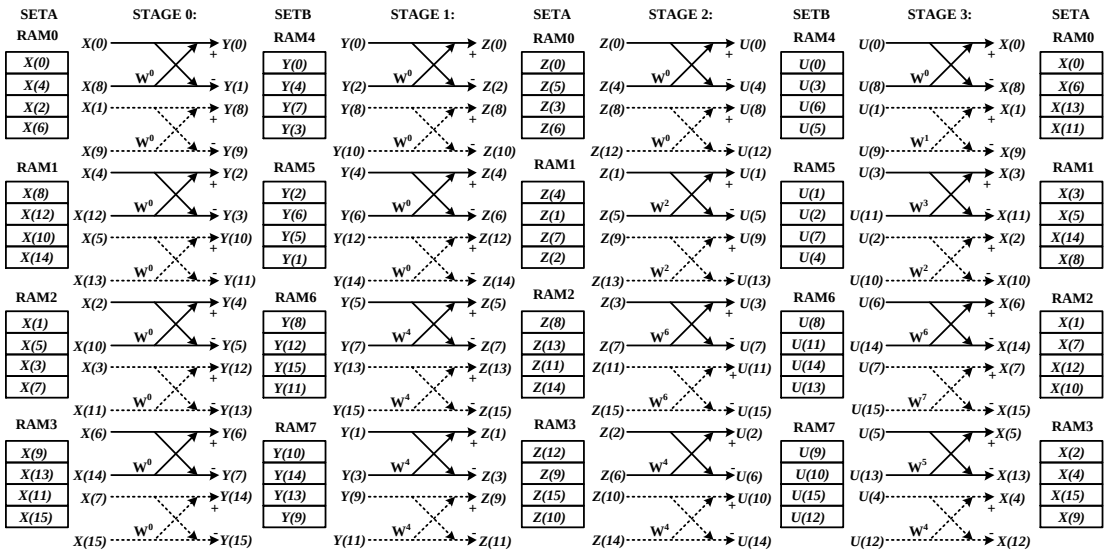

Figure 3.25: *16-point dataflow butterfly diagram for FFT processor architecture.*

The dataflow algorithm presented below is in accordance with Figure 3.25.

In even numbered stages, the inputs are read from SetA memory and they are routed to butterfly units via interconnectA. The twiddle factors are read from ROM and sent to butterfly units. After butterfly operation, the outputs are routed via interconnectB and stored in SetB memory.
In odd numbered stages, the inputs are read from SetB memory, they are routed to butterfly units via interconnectB. The twiddle factors are read from ROM and sent

to butterfly units. After butterfly operation, the outputs are routed via interconnectA and stored in SetA memory.

Dataflow with respect to butterfly unit0:
Even numbered stages:

- During stages except last stage:
  Read inputs from RAM0, RAM1. After butterfly computation, the result of addition and subtraction are stored in RAM4,RAM5.

- During last stage:
  Read inputs from RAM0, RAM2. After butterfly computation, the result of addition and subtraction are stored in RAM4,RAM5.

Odd numbered stages:

- During stages except last stage:
  Read inputs from RAM4, RAM5. After butterfly computation, the result of addition and subtraction are stored in RAM0,RAM1.

- During last stage:
  Read inputs from RAM4, RAM6. After butterfly computation, the result of addition and subtraction are stored in RAM0,RAM1.

Dataflow involving butterfly unit1:
Even numbered stages:

- During stages except last stage:
  Read inputs from RAM2, RAM3. After butterfly computation, the result of addition and subtraction are stored in RAM6,RAM7.

- During last stage:
  Read inputs from RAM1, RAM3. After butterfly computation, the result of addition and subtraction are stored in RAM6,RAM7.

Odd numbered stages:

- During stages except last stage:
  Read inputs from RAM6, RAM7. After butterfly computation, the result of addition and subtraction are stored in RAM2,RAM3.

- During last stage:
  Read inputs from RAM5, RAM7. After butterfly computation, the result of addition and subtraction are stored in RAM2,RAM3.

The FFT processor implementation details in terms of simulation and synthesis are described in detail in the following chapter.

4. IMPLEMENTATION

FFT processor was implemented using VHDL written in a Linux (CentOS) environment using emacs editor. Once FFT processor architecture was designed, its components were implemented separately. The functionality of individual components were verified by running test benches in ModelSim (simulation tool from Mentor Graphics Inc.). After individual modules were functionally verified, they were integrated to form the complete system. The FFT processor as a complete system was functionally verified using a system level test bench simulated using ModelSim tool. Initially, 16-point FFT computation was verified. Later, the processor was configured for different radix-2 sizes to verify their functionality. The simulation results were analyzed to determine the time taken by computation of different FFT sizes.

Following the simulation, synthesizable version of FFT processor was created from its simulation version. Later, the processor was synthesized on an Altera Stratix V FPGA device 5SGSMD5K2F40C2 using Quartus II version 12.1 tool. The synthesis results were analyzed to determine area, maximum frequency of operation and power consumption. The implementation details are discussed in detail in the following sections.

4.1 Simulation

The FFT processor was simulated before synthesizing it on an FPGA device. Simulation was carried out in order to verify the functionality of the processor and to validate the results. The RTL simulation was also performed to capture signal activities of different ports and signals in a Value Change Dump (VCD) file. The VCD file was used to carry out power analysis using post fit netlist data obtained after synthesis. The simulation phase is described in detail below.

4.1.1 Pre-simulation

Before simulating FFT processor necessary environment had to be set up. Simulation of FFT processor operation required RAM and ROM inputs. RAM and ROM data were provided in text files as inputs to simulation. RAM stores data samples whereas ROM stores twiddle factors required for FFT computation. Data samples

and twiddle factor values were represented in fixed point format. Q-14 fixed point format was used to represent floating point values. Since, datapath of the processor is 16-bit, lower fourteen bits were used to represent fraction part while upper two bits were used for integer part and sign. A test application was written in C to generate RAM and ROM text files containing input samples and twiddle factors respectively. For an N-point FFT, the test application generated Q-14 complex data, it packed real and imaginary parts of complex data into 32-bit value and stored them among four SetA memory text files. In addition, the test application generated $\frac{N}{2}$ twiddle factors, it packed real and imaginary parts of complex data into 32-bit value and stored them in ROM text file. The RAM and ROM text files were used in simulation to initialize the FFT processor RAM and ROM memories respectively.

All the VHDL files along with RAM and ROM text files were grouped inside a user defined project directory. For simulation, the tool used was ModelSim version 6.5c from Mentor Graphics Inc.. The simulation tool was used in a Linux (CentOS) environment. To launch ModelSim from command line, following command was executed from the project directory.

- vsim &

Before the first simulation run, a work folder was created inside project directory by executing the following command inside it.

- vlib work

4.1.2 Running Simulation

To run a simulation, the VHDL code files needed to be compiled. Since, there were many VHDL files to compile, a script file was created to compile all the files at once and save time while compiling frequently. Following command was used to compile VHDL files for simulation.

- vcom file_name1.vhd file_name2.vhd ... file_nameN.vhd

The following steps were followed in sequence to run simulation.

1. Compile VHDL files.
   The script file is executed using following command to compile VHDL files.
   - sh script_file_name.sh
2. Load the design.
   After compilation, design file was created with the name of top level entity inside the work directory. Before running simulation, the design was loaded using the following command.

- vsim work.top_level_entity_name

3. Add ports and signals to wave window.
   To observe the state and values of various ports and signals, they were loaded onto the wave window by executing the following command.
   - add wave signal1 signal2 .. signalN port1 port2 ... port N
4. Run simulation.
   The simulation was run for a finite time using the following command.
   - run time_in_nano_seconds

4.1.3 Post Simulation

A fixed point FFT algorithm was implemented in C and in matlab to calculate N-point FFT. The results from matlab are considered to be accurate when compared to C implementation results. Hence, C implementation results were verified with matlab results for an N-point FFT. FFT computation by the processor is comparable to C implementation. Hence, the results of simulation were validated against the C implementation results. The simulation results were analyzed further to determine FFT computation time taken by different FFT sizes.

4.1.4 VCD File Generation

To perform power analysis using PowerPlay power analyzer, it was required generate VCD file containing signal activities of the processor. Hence, VCD files for different FFT sizes were generated during RTL functional simulation. The following steps were executed in sequence to generate VCD files.

1. Compile VHDL files.
   The script file was executed using following command to compile VHDL files.
   - sh script_file.sh
2. Load the design.
   After compilation, design file was created with the name of top level entity inside work directory. The design was loaded for simulation using the following command.
   - vsim work.top_level_entity_name
3. Add ports and signals to wave window.
   To observe state and values of various ports and signals, they were loaded onto the wave window by executing the following command.

- add wave signal1 signal2 ... signalN port1 port2 ... portN

4. Create VCD file.
   During simulation, signal activities were captured in a VCD file by using following command.

   - vcd add -file file_name.vcd -internal -ports -r comp_instance_name/*

   The command recursively captures signal activities of ports and internal signals of all the hierarchical components. The "internal" switch enables capturing internal signal activities while "ports" switch enables capturing input/output port activities of components.

5. Run simulation.
   The simulation was run for a finite time using the following command.

   - run time_in_nano_seconds

6. Dump signal activities.
   Following command dumps all the signal activities and their value changes into VCD file.

   - vcd checkpoint

7. Exit simulation.
   Writing to VCD file finishes only when simulation is exited. Following command is used to exit simulation.

   - quit -f

4.2 Synthesis and Power Analysis

Synthesis required Quartus II version 12.1 from Altera Corp. which was installed on a Windows 8 machine. Altera Stratix V FPGA device 5SGSMD5K2F40C2 was used to synthesize FFT processor core. The following procedure was followed step by step to carry out synthesis on FPGA.

1. Create project.
   Created a new Quartus project for compiling VHDL files of FFT processor. And while creating the project added all the VHDL files to the project.

2. Set top level entity.
   A top level entity for the project which was an entry point to begin the compilation was specified.

3. Compile.
   Started compiling the project and fixed the compilation errors and warnings encountered. Compilation process goes through analysis and synthesis, followed by placement and routing, followed by timing analysis, followed by EDA netlist write operation.

4. Program FPGA.
   Programmed the FPGA device via USB port after the compilation was successful.

5. Analyze results.
   Analyzed the synthesis results, fitter summary and timing analyzer summary to obtain design parameters such as area, timing and maximum operating frequency.

After the synthesis was complete, power consumption was analyzed with the help of PowerPlay Power Analyzer tool available in Quartus II software. The PowerPlay Power Analyzer tool needed VCD file containing signal activities as an input to estimate the power consumption. The tool also required Synopsys Design Constraint (SDC) file to set up frequency at which power analysis was carried out. The following were the sequence of steps followed to carry out power analysis.

1. Launch PowerPlay Power Analyzer.
   Launched PowerPlay Power Analyzer tool in Quartus II software after compilation was successful.

2. Input (SDC) File.
   Specified the frequency of operation for power analysis using SDC file.

3. Input VCD file.
   The VCD file generated during RTL functional simulation was provided as an input to the PowerPlay Power Analyzer tool.

4. Set default toggle rate.
   Default toggle rates were set suitably depending on frequency at which power analysis was done. Generally, the toggle rates for a design module varies between 8-12% and hence, default toggle rate was chosen as 12% of frequency.

5. Start power analysis.
   Started the power analysis after setting up the environment and fixed the errors or warnings encountered during the same.

6. Analyze results.
   The dynamic power dissipation, static power dissipation and routing dynamic power dissipation were noted down and analyzed for different FFT points.

5. RESULTS AND EVALUATION

The FFT processor was functionally verified through RTL simulation using ModelSim version 6.5c. During simulation, FFT computation time for different FFT sizes are analyzed which are tabulated below. Before carrying out synthesis, the FFT core was configured to support maximum FFT size of $N_{max} = 2048$. The resulting FFT core (excluding memory components) was synthesized on an Altera Stratix V FPGA device 5SGSMD5K2F40C2 using Quartus II version 12.1 synthesis tool. The FFT core area consumption details are tabulated and discussed as below.

Resource usage for each of the FFT components and for FFT core as a whole are given in TABLE 5.1.

Table 5.1: *FFT Core Resource Utilization.*

Components	**Combinational ALUTs**	**Registers**	**DSP Blocks**
Butterfly Units	136	352	4
Interconnect	170	752	0
Address Generation Unit	482	199	0
Control Unit	291	111	0
FFT Core	1143	1754	4

The FFT core utilizes 1143 ALUTs out of 345200 available (<1%) and 4 DSP blocks out of 1590 available (<1%) on the FPGA device.

FFT processor was simulated in ModelSim for functional verification and to measure computation time. FFT computation time for different FFT sizes are given in TABLE 5.2. The simulation was carried out at a frequency $f_{max} = 200MHz$.

Table 5.2: *FFT Computation Time.*

N	**Clock Cycles**	**Total Time[μs]**
16	74	0.37
32	132	0.66
64	254	1.27
128	520	2.6
256	1106	5.53
512	2396	11.98
1024	5222	26.11
2048	11376	56.88

The plot of computation time against FFT size is given in Figure 5.1.

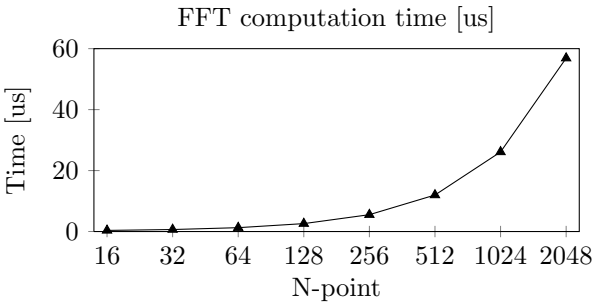

Figure 5.1: *FFT computation time as a function of N.*

Increase in FFT size results in almost linear increase in computation time till 256 points. But after 256 points, there is exponential increase in computation time.

According to TABLE 5.2, computation time (in micro seconds) for specified FFT size meets the timing constraints of various wireless standards such as IEEE 802.11a/g, IEEE 802.16e, 3GPP-LTE, DAB and DVB.

Power dissipation of FFT core was determined using PowerPlay Power Analyzer tool available in Quartus II synthesis tool. PowerPlay tool performs power analysis on post-fit netlist data obtained after synthesis. The tool requires VCD file containing signal activities of FFT core. As explained earlier, signal activities at RTL were captured during simulation in ModelSim. In addition, frequency at which power analysis to be done was specified through SDC file. Default toggle rate was set

at 12% of frequency for those signals which didn't have signal activity in VCD file. Operating voltage was set at 0.9V and the ambient temperature was set at 25 degree celsius. The power analysis results are presented in TABLE 5.3. Total dynamic power dissipation (in milliWatts) consists of dynamic power, static power and routing dynamic power dissipation of FFT core. Energy consumption expressed in micro Joules was calculated as $Energy = power * time$.

Table 5.3: *Power Analysis Summary.*

N	**Total Power Dissipation[mW]**	**Energy[μJ]**
16	399.05	0.147
32	412.46	0.271
64	423.46	0.537
128	407.83	1.06
256	434.20	2.4
512	428.57	5.1
1024	432.01	11.279
2048	434.44	24.71

According to TABLE 5.3, power dissipation varies slightly about an average value of approximately 420mW for different FFT size. Variation of power dissipation against FFT size is illustrated in Figure 5.2.

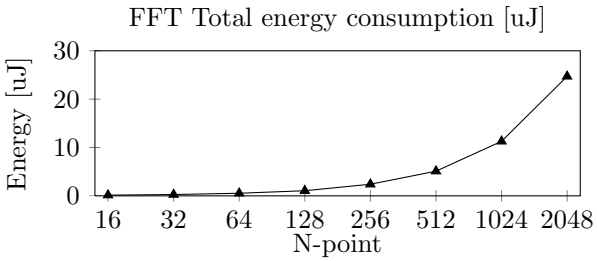

Figure 5.2: *FFT total energy consumption as a function of N.*

The dynamic power dissipation increases with increase in FFT size because of increased signal activities and increase in area. The energy consumption increases drastically with N whereas power consumption varies slightly around an average value.

The proposed FFT processor was compared in terms of computation time and scalability with the existing research work. Comparison information is provided in TABLE 5.4.

Table 5.4: *Comparison With Existing FFT Processors.*

	64 [μs]	**128 [μs]**	**256 [μs]**	**512 [μs]**	**1024 [μs]**	**2048 [μs]**	**f [MHz]**	**Scalable**
[13]	2.1	-	-	-	-	-	31.69	No
[2]	-	-	-	-	26	-	100	No
[3]	-	40.34	47.30	52.30	61.14	-	470	Yes
[6]	-	-	-	-		57	17.86	Yes
Proposed	1.27	2.6	5.53	11.98	26.11	56.88	200	Yes

The FFT processor based on our novel architecture outshines fixed length as well as variable length FFT processors.

6. CONCLUSIONS

A scalable FFT processor architecture was proposed. In order to realize it, a radix-2 fixed point 16-bit N-point FFT processor was implemented using VHDL. The FFT processor was simulated in ModelSim verify its functionality and to determine computation time. The FFT core was synthesized on an Altera Stratix V FPGA device 5SGSMD5K2F40C2 to determine area, maximum operating frequency and power consumption. Based on simulation and synthesis results proposed architecture meets the timing constraints of wide range of wireless standards such as IEEE 802.11a/g, IEEE 802.16e, 3GPP-LTE, DAB and DVB. Hence, the proposed architecture can be adopted in SDR platforms supporting above specified wireless standards. According to [9], higher radix FFT computation is faster than lower radix FFT for large number of FFT points. Hence, the proposed architecture can be extended to implement radix-4, radix-8 or split radix FFT processors with additional modifications. In order to extend the architecture to higher radix: suitable radix butterfly should be used, address generation algorithm should be extended to support required data access during computation, additional control signals should be added to control unit and the interconnect should be extended to accommodate additional data/address signals. In addition, the processor architecture can be adopted by any other non-OFDM based applications where a reasonable balance between specified design parameter is required.

REFERENCES

[1] C. Sydney Burrus. Signal Flow Graphs of Cooley-Tukey FFTs. *http://cnx.org/content/m16352/latest/*. Accessed: 21-Mar-2013.

[2] Z.H. Derafshi, J. Frounchi, and H. Taghipour. A high speed FPGA implementation of a 1024-point complex FFT processor. In *Second International Conference on Computer and Network Technology (ICCNT), 2010*, pages 312 –315, april 2010.

[3] K. George and C.-I.H. Chen. Configurable and expandable FFT processor for wideband communication. In *IEEE Instrumentation and Measurement Technology Conference Proceedings, 2007. IMTC 2007.*, pages 1 –6, may 2007.

[4] Haining Jiang, Hanwen Luo, Jifeng Tian, and Wentao Song. Design of an efficient FFT processor for OFDM systems. *IEEE Transactions on Consumer Electronics*, 51(4):1099–1103, nov. 2005.

[5] J.W.Cooley and J.W.Tukey. An algorithm for the machine calculation of the complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[6] Y.-T. Lin, P.-Y. Tsai, and T.-D. Chiueh. Low-power variable-length fast fourier transform processor. *IEEE Proceedings - Computers and Digital Techniques*, 152(4):499 – 506, july 2005.

[7] Marcus Majo. Design and implementation of an OFDM-based communication system for the GNU radio platform. Master's thesis, IKR, University of Stuttgart, 2009.

[8] Ramjee Prasad. *OFDM for Wireless Communications Systems*. Artech House, 31 Mar 2004.

[9] K.L.S. Swee and Lo Hai Hiung. Performance comparison review of radix-based multiplier designs. In *4th International Conference on Intelligent and Advanced Systems (ICIAS), 2012*, volume 2, pages 854–859, June.

[10] J. Takala and K. Punkka. Butterfly unit supporting radix-4 and radix-2 FFT. In *Proc. Int. Workshop Spectral Methods and Multirate Signal Process., Riga, Latvia*, pages 47–53, June 20-22 2005.

[11] Song-Nien Tang, Chi-Hsiang Liao, and Tsin-Yuan Chang. An area- and energy-efficient multimode FFT processor for WPAN/WLAN/WMAN systems. *IEEE Journal of Solid-State Circuits*, 47(6):1419–1435, june 2012.

[12] Tran-Thong and Bede Liu. Fixed-point fast fourier transform error analysis. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 24(6):563–573, dec 1976.

[13] Bingrui Wang, Qihui Zhang, Tianyong Ao, and Mingju Huang. Design of pipelined FFT processor based on FPGA. In *Second International Conference on Computer Modeling and Simulation, 2010. ICCMS '10.*, volume 4, pages 432–435, jan. 2010.

[14] Ning Zhang and R.W. Brodersen. Architectural evaluation of flexible digital signal processing for wireless receivers. In *Conference Record of the Thirty-Fourth Asilomar Conference on Signals, Systems and Computers, 2000.*, volume 1, pages 78–83 vol.1, 29 2000-nov. 1 2000.

[15] Qihui Zhang and Nan Meng. A low area pipelined FFT processor for OFDM-based systems. In *WiCom '09. 5th International Conference on Wireless Communications, Networking and Mobile Computing, 2009.*, pages 1–4, sept. 2009.

A. APPENDIX

FFT core was synthesized on an Altera Stratix V FPGA 5SGSMD5K2F40C2 using Quartus II software. Post synthesis, synthesized FFT core and its components can be viewed using Quartus II RTL viewer tool shown in the screen shots below.

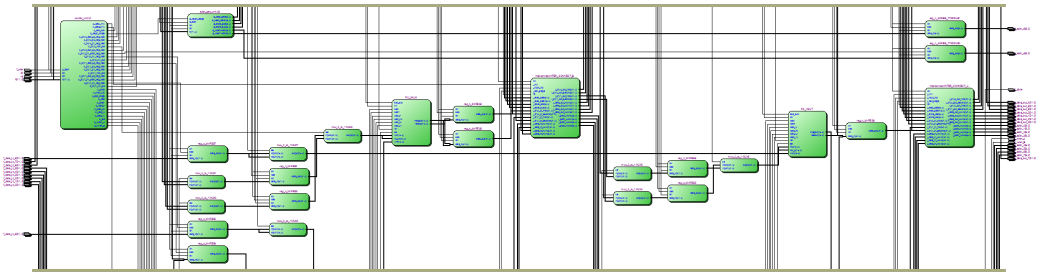

Figure A.1: *Synthesized FFT core in RTL viewer.*

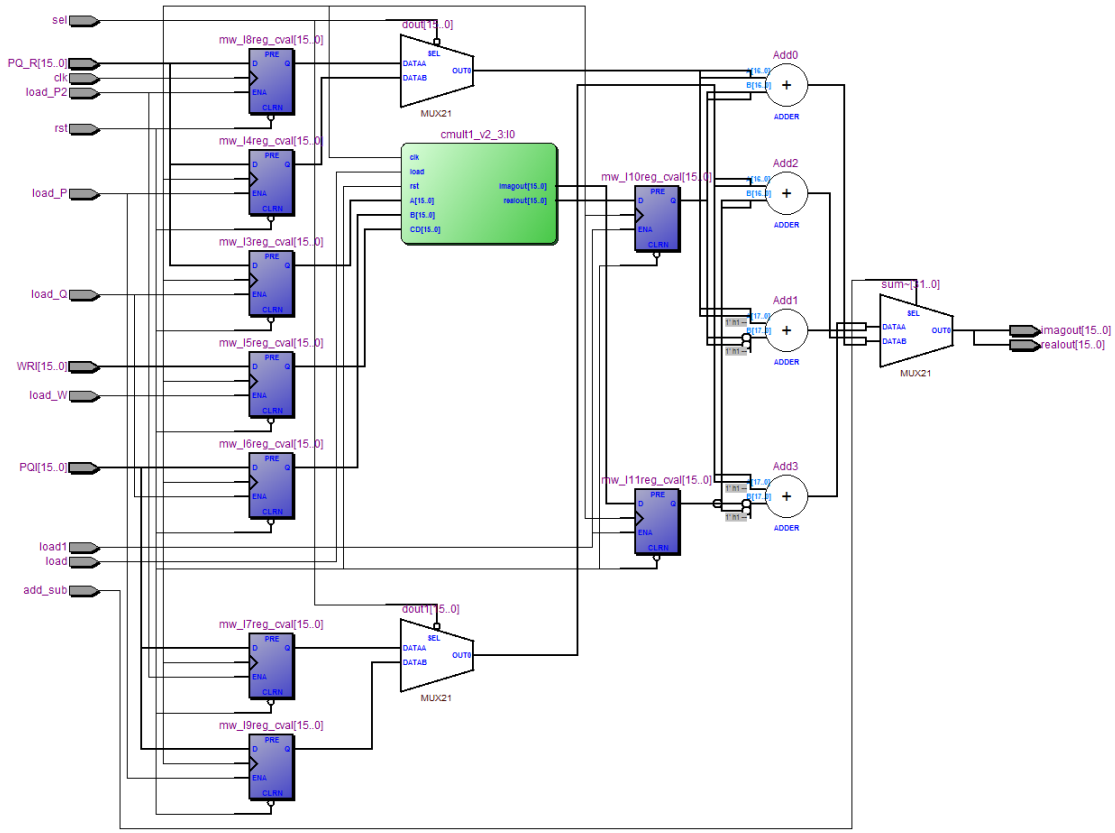

Figure A.2: *Synthesized butterfly unit in RTL viewer.*

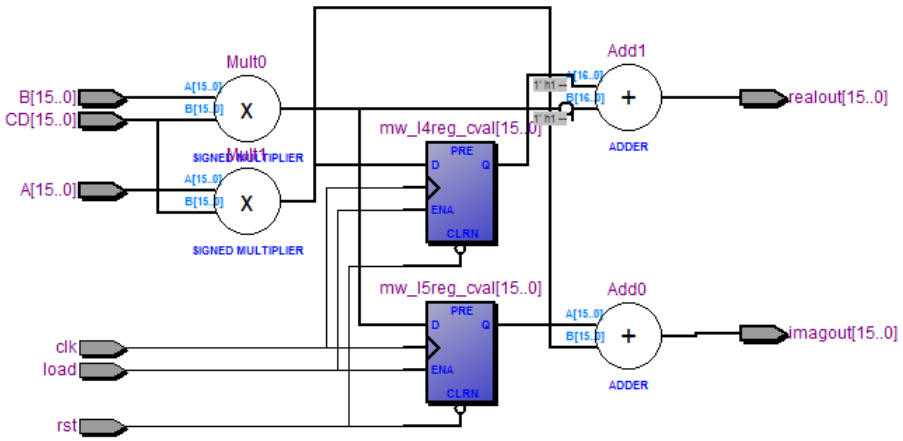

Figure A.3: *Synthesized complex multiplier in RTL viewer.*

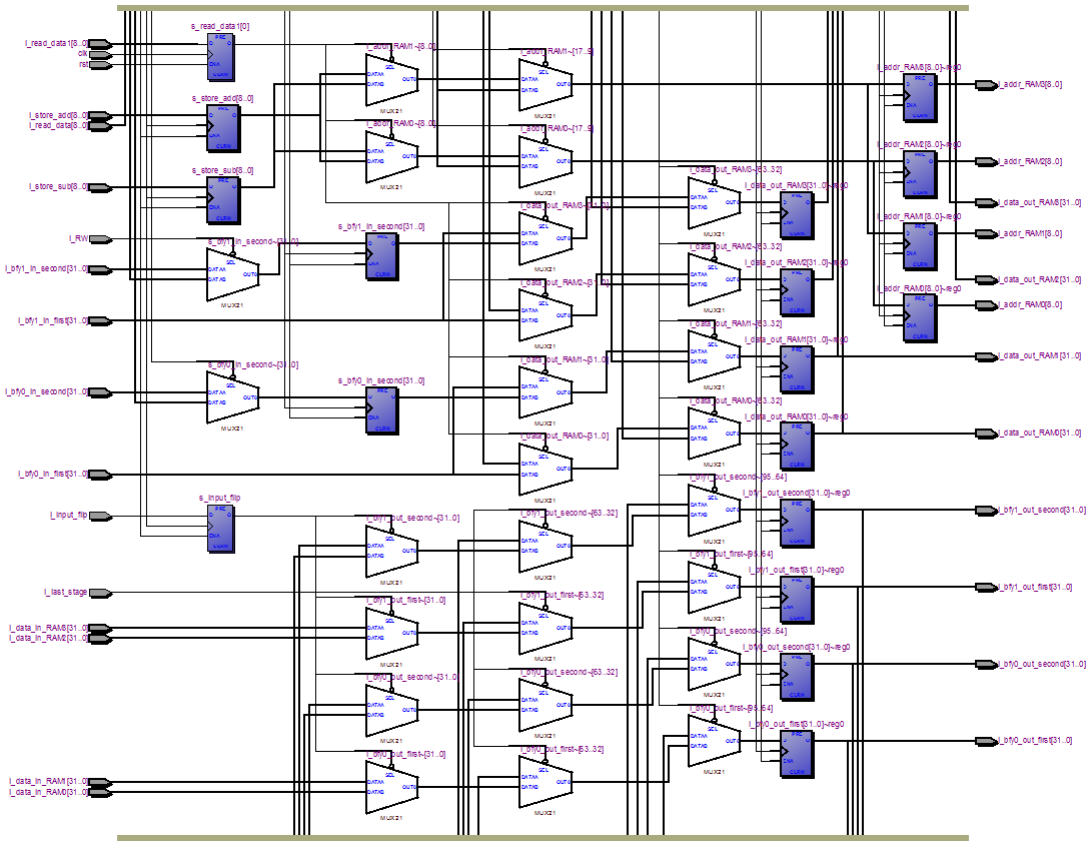

Figure A.4: *Synthesized interconnect in RTL viewer.*

Figure A.5: *Synthesized address generation unit in RTL viewer.*

Figure A.6: *Synthesized control unit in RTL viewer.*

The chip planner tool in Quartus II software provides a pictorial view of synthesized FFT core location on the FPGA chip. Dark blue region in Figure A.7 is the location of FFT core on FPGA chip. The core was placed in a localized region which indicates efficient placement and routing of FFT core.

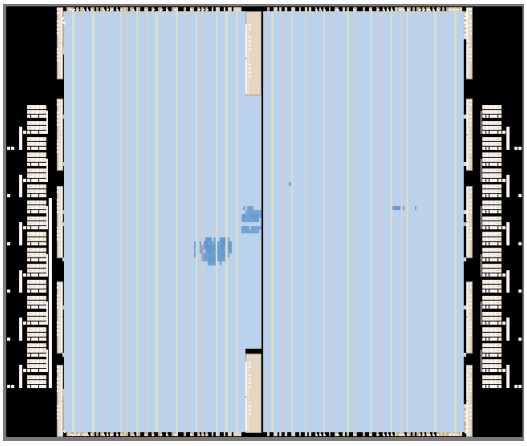

Figure A.7: *Location of FFT core on the FPGA chip, courtesy: Chip planner tool.*

Placement of synthesized design over a localized region is efficient in a sense that it reduces static and dynamic power consumption, reduces clock skew issues and allows economic usage of available FPGA chip area.

Design partition planner tool in Quartus II enables us to view the design partition of FFT core components as shown in Figure A.8.

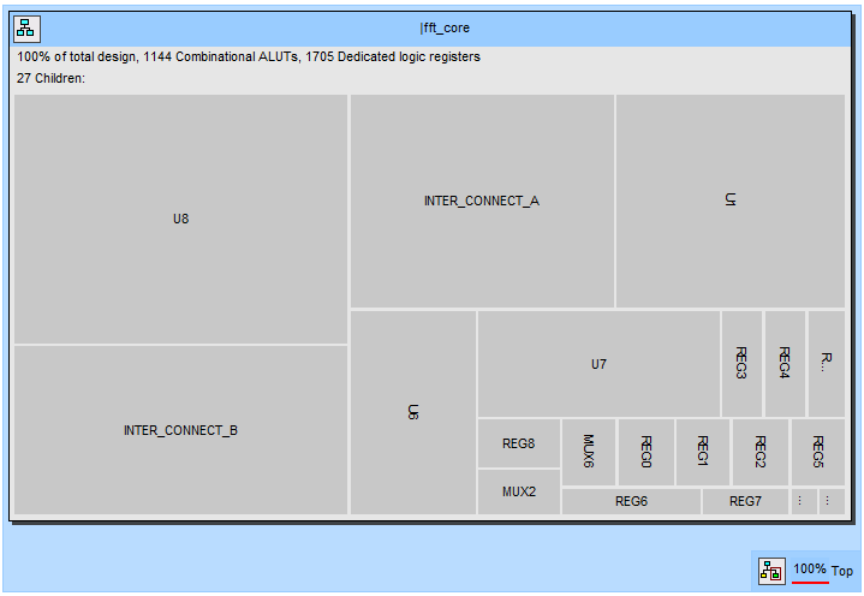

Figure A.8: *Partition of FFT core components on the FPGA chip, courtesy: Design partition planner tool.*

Major design partitions are mapped as mentioned below.
U8: Address Generation Unit
U1: Control Unit
U6: Butterfly Unit0
U7: Butterfly Unit1

INTER_CONNECT_A: InterconnectA
INTER_CONNECT_B: InterconnectB