University of Thessaly
Department of Electrical and Computer Engineering

Diploma thesis by Kyriakos Manti

"Design and Implementation of Scalable FFT Processor Architecture for OFDM Based Communication Systems"

Supervisors:
Stamoulis Georgios
Moondanos Ioannis

Volos 2015

ACKNOWLEDGMENT

Upon completion of my thesis, I would like to thank my supervisor Dr. Stamoulis Georgios and my co-supervisor Dr. Moondanos Ioannis for their trust and excellent cooperation we have had during this project and my studies.

I would also like to thank Ph.D. candidate Antoniadis Charalampos for his assistance and guidance on this work.

Finally, i would like to thank my family and friends for their unconditional support and encouragement.

ABSTRACT

We live in an era of mobile data revolution. Users demand services and applications from mobile communication systems that go far beyond mere voice and telephony, leading to a high growth in the wireless electronic device market (i.e. smart-phones and laptops). The growth in data use by mobile services and applications, such as audio and video streaming has become a driving force for the development of the next generation of wireless standards. As a result, new standards are being developed to provide the data rates and network capacity necessary to support worldwide delivery of these types of rich multimedia application.

The modern wireless standards are predominantly based on OFDM communication systems, a method of encoding digital data on multiple carrier frequency. Various wireless devices in recent times support multiple wireless standards and demand efficient transceivers. In an OFDM based transceiver, the baseband hardware needs to meet stringent design parameters, such as high speed, low power, low area, low cost, flexibility and scalability, to be efficient across multiple standards. To design an efficient OFDM baseband hardware, it is necessary to efficiently design its performance critical component. FFT computation is one of the most computationally intensive operations which influence the performance of the system in an OFDM system.

The baseband hardware requires a scalable FFT module which meets the performance constraints required by multiple wireless standards. The scope of this thesis work was the implementation of scalable FFT processor in which an existing scalable radix-2 N-point FFT processor architecture was adopted. The FFT processor was designed and implemented in Verilog, (an HDL) and functionality verified through RTL simulation according, to the specifications of the proposed architecture.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The growing demand in the multimedia services has increased the need for faster services in the wireless communication systems but some of the high-bit rate services are limited due to various performances constrains. Therefore, providing services over wireless channels is a challenging task due to the fact that the mobile radio channels are more demanding compared to wired channels.

A broadband multimedia wireless communication system requires fast transmission and the designing of wireless transceivers to support high data rates with compact and inexpensive hardware; this forms challenging task. The standards specify strict performance requirements in terms of high speed, low power, low cost, flexibility and scalability. In order to avoid the multipath-fading environment and to achieve high data rates at the same time, the Orthogonal Frequency Division Multiplexing (OFDM) transmission scheme is being used.

OFDM is a parallel data transmission technique which minimizes the influence of multipath fading through simpler equalization technique. This technique is being widely used in the wireless communication systems since it predominates in terms of spectral utilization and performance, compared with other techniques like recovering original signal from received signal. One of the major performance critical modules of OFDM transceiver is Fast Fourier Transform (FFT) computation, an efficient OFDM transceiver which is being characterized by an efficient FFT module. In OFDM baseband hardware, FFT computation is one of the most computationally intensive operations which influence the performance of the system. The baseband hardware has to be capable enough to compute FFT within the time constraints necessary to support multiple wireless standards and also be scalable. In addition, it has to meet the criteria of high speed, low area and low power consumption.

In order to support digital communication standards, integrated circuits are commonly based on FFT of some length. Flexible length makes the design more usable for configurable circuits. As the transform length increase, the amount of arithmetic involved becomes excessive. This makes FFT one of today's most important tools in digital signal processing, as it enables the efficient transformation between time and frequency domain. Since, FFT is an integral component of OFDM transceiver, research on FFT algorithm and its hardware implementation is focused extensively.

The focus of related researches is to optimize the FFT algorithm and to find efficient hardware solutions. Some of the researches work is based on specific FFT size, targeting specific standard and optimized for specific design parameter. Fixed length FFT processors can support only specific wireless standard and they will not 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. In search for a reasonable balance between scalability and achieving performance constraints, a scalable FFT processor architecture was presented by D. Revana et-al. in [1].

Scalable FFT processor architecture is based on radix-2 FFT algorithm, while the size of FFT can only be a number to the power of two. The architecture is configurable at design time to support a maximum FFT size and scalable at runtime while it can support any radix-2 FFT size

from 16 to maximum size. The proposed architecture was implemented as part of the thesis work.

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

2. FAST FOURIER TRANSFORM (FFT)

The FFT algorithm was presented by Cooley and Tukey in [2] with an aim to compute Discrete Fourier Transform (DFT), with significant reduction in number of computations. DFT computation of a time domain digital signal x(n) results in its conversion into a frequency domain signal. Analysis and processing of a discrete signal in frequency domain is more efficient than an analysis in time domain. In fact, reduced computations due to FFT algorithm helped to decrease power consumption, area and increase require system throughput. Direct computation of N-point DFT would require $N^2 - N$ complex additions and N^2 complex multiplication operations according to equation given by:

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

where $k = 0, 1, 2 \ldots N - 1$.

However, using FFT algorithm total number of additions and multiplications reduces to $N * \log_2(N)$ and $\frac{N}{2} * \log_2(N)$ respectively. An N-point FFT equation is given by:

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

where $W_N^k = e^{-\left(j\frac{2\pi k}{N}\right)}, k = 0, 1, 2 \ldots N - 1$.

FFT is computed in two different ways. Decimation In Time (DIT) and Decimation In Frequency (DIF). DIT FFT algorithm is found to provide better signal-to-noise ratio in comparison with DIF FFT for a finite word length according to Tran-Thon et al. in [3].

Based on the 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 of four while radix-8 FFT size is a power of eight. Split-radix type involves mixing of any of the specified radix combinations. A radix-2 DIT FFT algorithm can be depicted as a butterfly diagram as shown in figure 2.1.

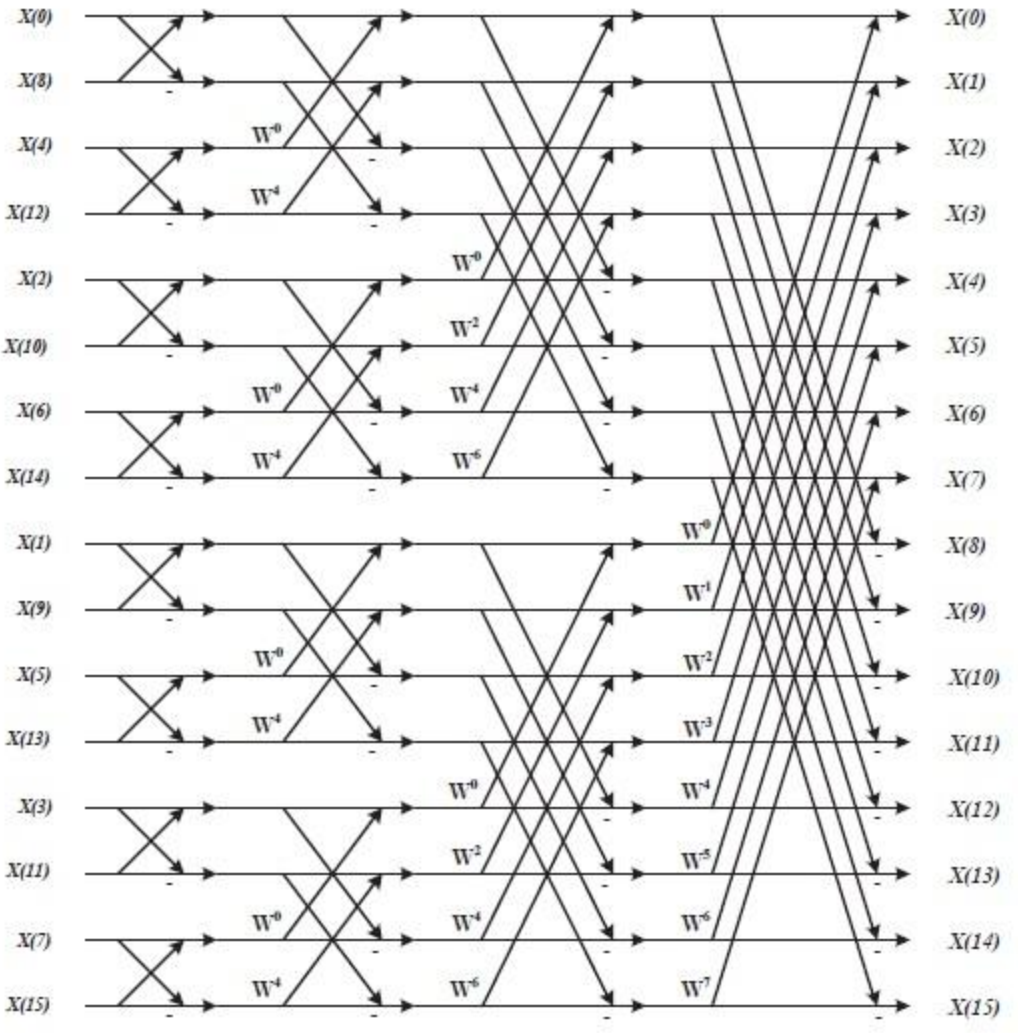


Figure 2.1: Radix-2 DIT FFT butterfly diagram

The figure 2.1 describes 16-point FFT butterfly diagram where $x(n), X(k)$ are 16-point complex inputs and outputs respectively. Since, it is a DIT algorithm, the inputs $x(n)$ are in bit reversed order and outputs $X(k)$ are in natural order. The butterfly computation with upper half data samples is symmetric with the lower half till the last stage. In the last stage, butterfly computation merges data samples from lower half and upper half. Such a property of DIT FFT is the basis for address generation scheme and input data storage in data memory of FFT processor. Considering an N-point FFT, there are $\log_2(N)$ number of stages and each stage contains $\frac{N}{2}$ butterfly operations.

3. SCALABLE FFT PROCESSOR

Scalable FFT processor architecture was adopted from [1] which was presented by D. Revana et-al. in. It was designed to support N-point complex value radix-2 fixed point FFT computation. The architecture of processor is configurable at design time for required maximum FFT size ***Nmax.*** Once the processor is configured for ***Nmax***, at runtime it supports any radix-2 FFT size from 16 to ***Nmax***.

An external interface communicates with the FFT processor in order to specify the size of FFT computation and then sends the data samples for computation. After FFT computation the external interface receives the results. This is achieved by defining a configuration channel, a write channel and a read channel respectively. Each channel produces suitable availability and validly signals, depending on the state of processor. Figure 3.1 shows input/output ports of the FFT processor.

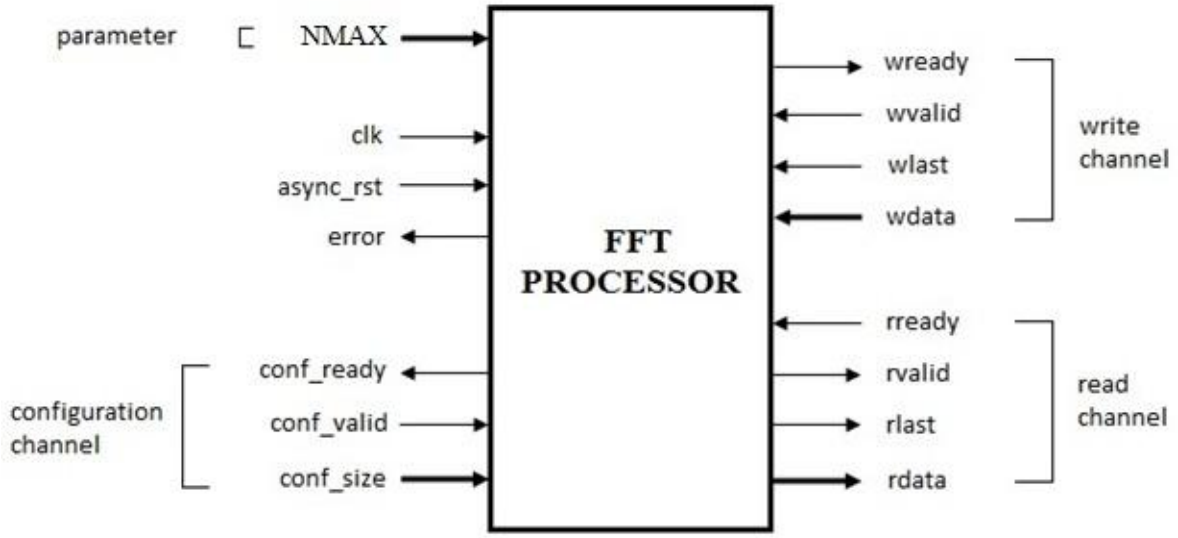


Figure 3.1: FFT processor core port details

Ports marked with thin lines represent single bit pins while ports marked with thick lines represent multi-bit bins. Ports named in capital letters are parameters which allow design time configuration of modules. The port ***NMAX*** is the maximum FFT size ***Nmax***, which is configurable at design time. The processor has a clock port ***clk*** and an active high reset signal ***async_rst*** which is synchronized and supplied to components of the processor via ***rst*** signal. Furthermore it has an error signal ***error*** which becomes activated when the given runtime FFT size and the number of given data sample do not match a prerequisite for the FFT processor to start. Each channel contains two ports. ***Ready*** signal indicates when it can accept the data and

valid signal indicates when the data are valid. Write channel and read channel also include ***last*** signal to indicate the transfer of the final item in transaction.

The FFT processor was required to be a fixed point processor. Fixed point is a simple, yet very powerful way to represent fractional numbers in computer. By reusing all integer arithmetic circuits of a computer, fixed point arithmetic is orders of magnitude faster than floating point arithmetic which was used to represent floating point values. Since, data path of the processor is 16-bit, it used Q-14 fixed point format in which the lower fourteen bits were used to represent fraction part and the upper two bits were used for integer part and sing. This representation supports any fractional number between -2 to 2.

The major components of FFT processor and its block diagram representation are shown in the following figure 3.2.

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

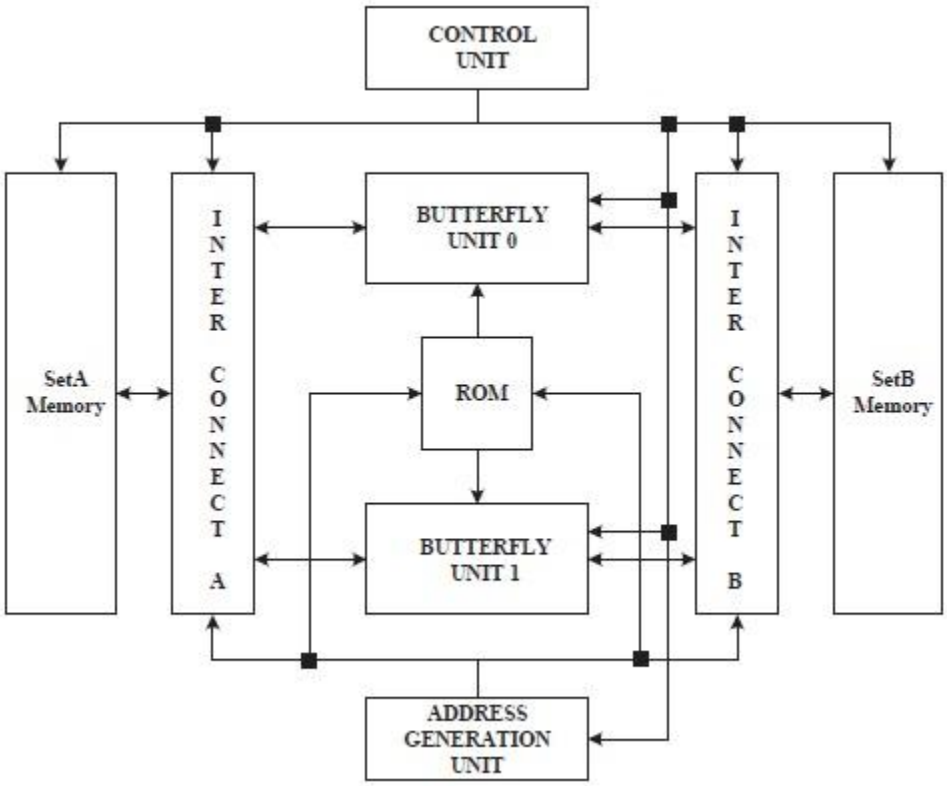


Figure 3.2: Scalable FFT processor block diagram

Two butterfly units are used, which operate in parallel and compute two outputs per clock cycle. Data memory storing data samples includes two sets called ***SetA*** and ***SetB***, with each set containing four RAM banks for simultaneous access of four samples. Twiddle factors are stored in a ROM. During design time, data memory and twiddle factor memory are suitable chosen to store a 32-bit word and to support ***Nmax***-point FFT computation. Two interconnects called ***interconnectA*** and ***interconnectB*** are used to connect butterfly units and address generation unit with data memory set ***SetA*** and ***setB*** respectively. The Address generation unit is used to generate addresses required to read input samples and twiddle factor for butterfly units. The Control unit is required to co-ordinate and synchronizes activities of the rest of the components.

When FFT processor is in operation, the overall dataflow through the processor is pipelined and follows ping-pong logic. The internal pipelined architecture of FFT processor is shown in Fig. 3.3.

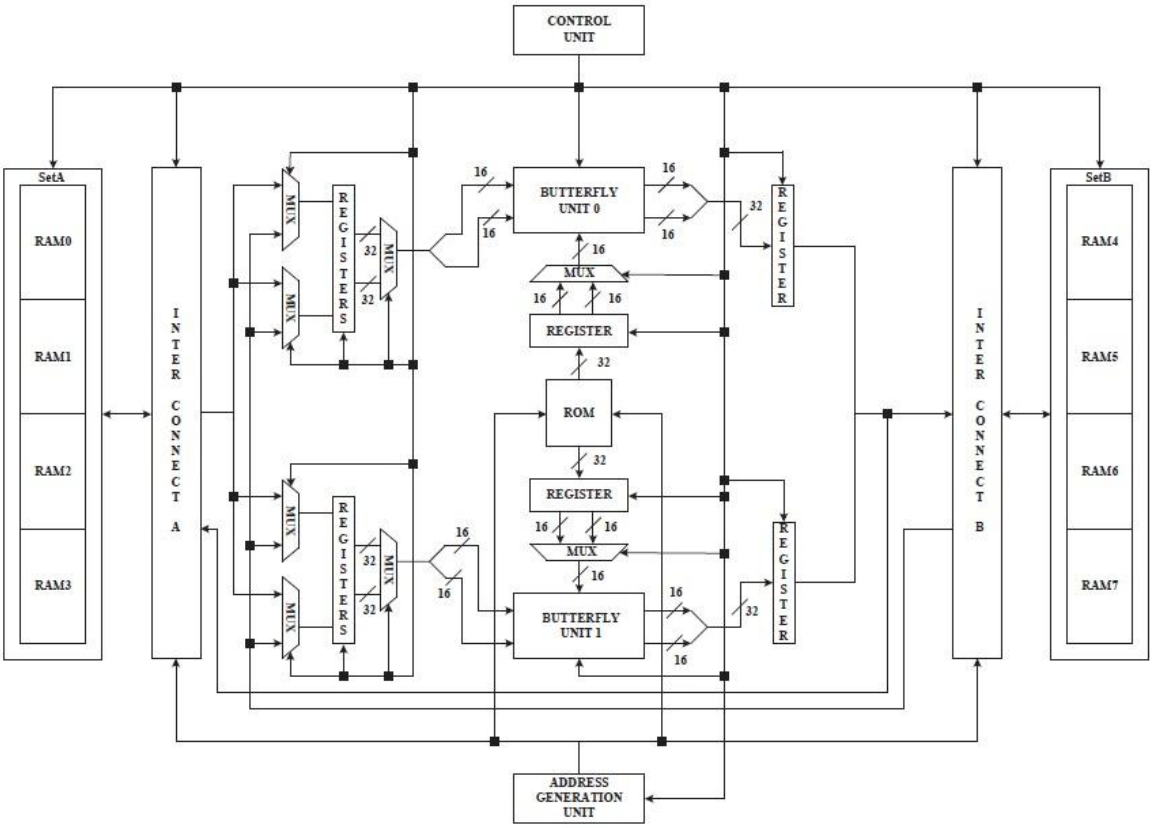


Figure 3.3: FFT processor pipelined internal architecture

According to pipelined architecture, there are nine pipeline stages. These stages are address generation stage, memory read stage, two stages before butterfly units (pre-computation), two stages inside butterfly units (computation), two stages after butterfly units (post-computation) and memory write stage. Each pipeline stage is a registered activity which consumes one clock cycle. At the beginning of each stage, one additional clock cycle is required and thereafte ten clock cycles are needed to fill up pipeline. Therefore, ten clock cycles are

required to compute initial output samples in each stage, 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:

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

The total number of clock cycles required to compute FFT for an N-point is given by:

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

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

3.1 BUTTERFLY UNIT

The butterfly unit was adopted from [4] which was implemented by J.Takala et al. It was designed to support radix-2 DIT butterfly operation. Butterfly operation is the basic entity of a butterfly diagram and it is pictorially described as shown in Figure 3.4.

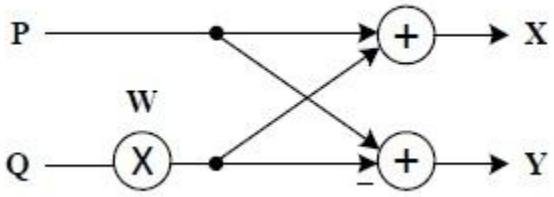


Figure 3.4: Single radix-2 DIT butterfly operation [4]

Butterfly operation can be illustrated in equation as,

$$X = P + WQ \quad \text{and} \quad Y = P - WQ \tag{3.3}$$

where P and Q are complex input values, W is an input twiddle factor and X and Y are complex output values. The output X is the result of addition while Y is the result of subtraction. The multiplication (W*Q) is based on bit-parallel multipliers method which is explained in more detail below. The multiplication of two complex numbers W and Q is given by:

$$WQ = (W_R + jW_I)(Q_R + jQ_I) = (W_RQ_R - W_IQ_I) + j(W_RQ_I + W_IQ_R) \tag{3.4}$$

where indexes **"R"** and **"I"** represent the real and imaginary part of complex numbers respectively. Implementing the equation (3.3), four multipliers and two adders are required. After the optimization of the complex multiplication, it requires three multipliers instead of four according by following equation (3.4).

$$WQ = W_I(Q_R - Q_I) + Q_R(W_R - W_I) + j[W_I(Q_R - Q_I) + Q_I(W_R + W_I)] \quad (3.5)$$

The area and power consumption can be reduced if two clock cycles are used for complex multiplication. The reduction in number of multipliers through pipeline operation is a reasonable compromise between high throughput, area and power efficiency. Hence, the internal architecture of butterfly unit where was implemented is shown in Figure 3.5.

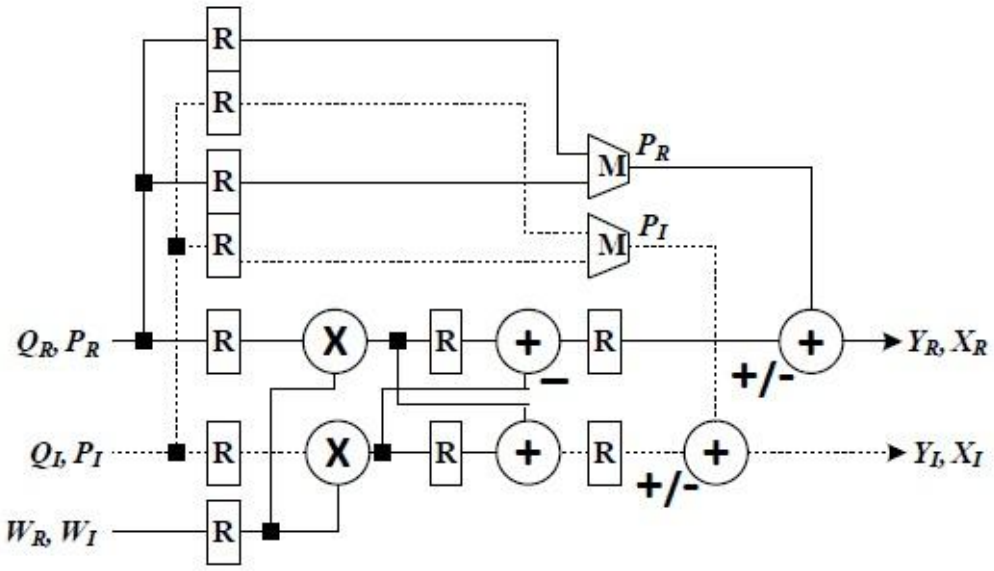


Figure 3.5: Pipelined butterfly unit

This approach uses only two multipliers and two adders. The complex multiplication is pipelined by using two clock cycles for computation. The pipelined multiplication is given by:

$$A_1 = Q_R W_R; \quad B1 = Q_I W_R \quad \text{(cycle\#1)}$$

$$A_2 = Q_R W_I; \quad B2 = Q_I W_I \quad \text{(cycle\#2)}$$

$$WQ = (A1 - B2) + j(B1 + A2) \quad (3.6)$$

Hence, the butterfly unit was implemented by sharing two out of three input ports for both input samples Q_R, Q_I and P_R, P_I respectively while the third input port is shared to read W_R and W_I of twiddle factor. Since FFT processor is required to be a fixed point computation, inputs ports are 16-bit values. When butterfly unit is in operation, complex values Q and P are read every alternate clock cycle and same is the case with W_R and W_I. Furthermore, a complex multiplier (two bit parallel real multipliers) and two adders were implemented as computation units and were integrated in the butterfly unit. The outputs of real valued multipliers are registered, thereby reducing the critical path of complex multiplier and thereafter the 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. The butterfly unit was implemented to support Q-14 fixed point computation. Inputs and outputs are 16-bit fixed point values. Two output ports are shared between X_R, X_I and Y_R, Y_I respectively, being packed into 32-bit complex value X and Y.

Butterfly unit operation is controlled by control unit which issues register loads and multiplexers signals at appropriate time intervals. Two butterfly units operate in parallel in the FFT processor to compute two output samples per clock cycles, increasing the throughput of the processor. Figure 3.6 shows input/output ports of butterfly unit.

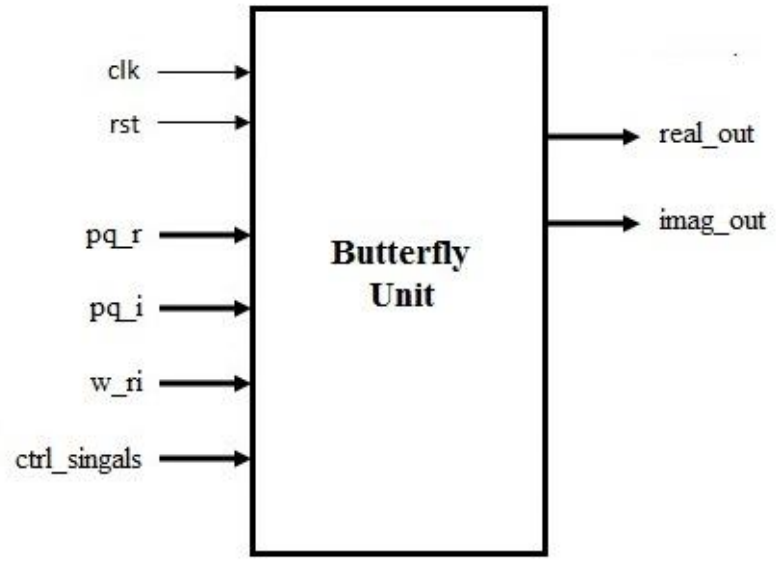


Figure 3.6: Butterfly unit port details

Butterfly unit has a clock port ***clk*** and an active high reset signal ***rst***. Real parts of inputs P and Q are multiplexed into an input port ***pq_r*** while imaginary parts of P and Q are multiplexed into port ***pq_i***. Real and imaginary parts of twiddle factor W are multiplexed into port ***w_ri***. Control signals are driven into an input port ***ctrl_signals*** which is an 8-bit value and

contains the signals ***load_q***, ***load_p1***, ***load_p2***, ***load_w***, ***load_mul***, ***load_add_sub***, ***sub_enable*** and ***sel***. The order at which signals are mentioned above is from lower bit to upper bit of ***ctrl_signals*** value***.*** ***Load_q*** and ***load_w*** are load signals for Q and W input registers. ***load_p1*** and ***load_p2*** are load signals for P inputs registers. ***load_mul*** and ***load_add_sub*** are load signals for multiplier output registers and complex multiplier adder output register respectively. Multiplexers are controlled via ***sel*** control signal. Addition or subtraction operations which are part of butterfly unit are controlled using ***sub_enable*** signal, which if being active high, the output has a result of subtraction. The output ports ***real_out*** and ***imag_out*** are the real and imaginary parts of output data respectively.

Figure 3.7 shows waveforms of butterfly unit ports for 16-point FFT.

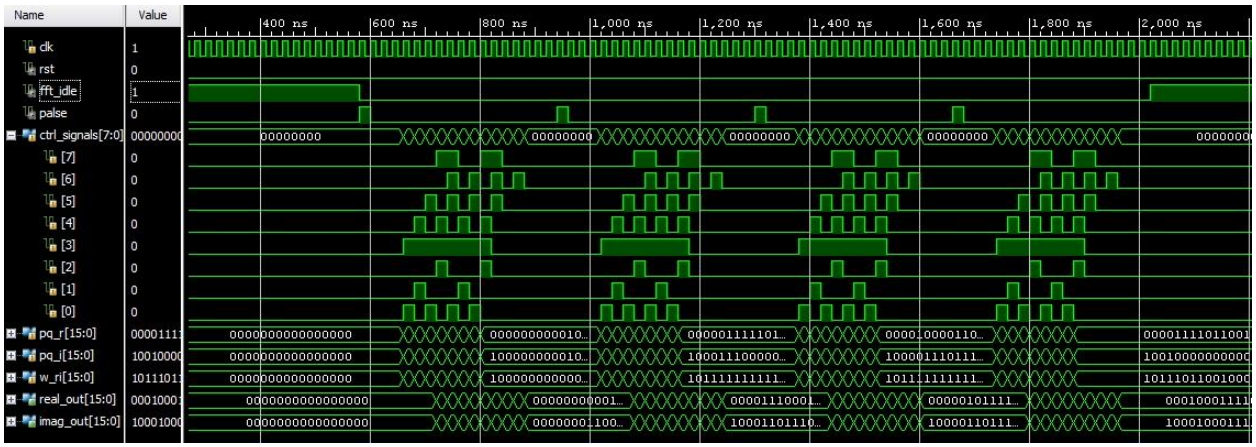


Figure 3.7: Butterfly unit waveform for 16-point FFT

As shown in figure 3.7, there are four stages in a 16-point FFT and timing of pipelined butterfly unit control signals follow a similar pattern in every stage. ***fft_idle*** and ***palse*** signals are part of the understanding of the waveforms. ***fft_idle*** signal indicates the beginning of FFT computation when it becomes low and the end of computation when it becomes high. ***palse*** signal indicates the 1st clock cycle in each stage. The input data latency is four clock cycles and the output data latency is seven clock cycles with respect to the beginning of stage. Initially, the butterfly units compute the addition result and at the next clock cycle they compute the subtraction result for each stage of computation. The results of addition and subtraction are routed every alternate clock cycle. Two butterfly units have the same timing of control signals.

3.2 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 it consists of two sets called ***SetA*** and ***SetB***. Butterfly operation is performed by reading input samples from one set of memory and by writing output samples to a different set of memory. Since, two butterfly units require four input samples per clock cycle, each set has to include four memory banks to concurrent access to four samples at any given time. Hence, ***SetA*** memory includes four memory banks ***RAM0***, ***RAM1***, ***RAM2*** and ***RAM3*** while ***SetB*** memory includes four memory banks ***RAM4***, ***RAM5***, ***RAM6*** and ***RAM7***. The maximum size of each memory bank is decided by the maximum size of FFT computation ***Nmax*** selected during design time. The maximum size of a memory bank is given by:

$$\text{ram\_bank\_size} = \frac{Nmax}{4} \tag{3.7}$$

where the size is measured in terms of 32-bit words. The size of a memory bank defines its address bus width which is given by:

$$\text{address\_ram\_width} = \log_2(\text{ram\_bank\_size}) \tag{3.8}$$

In memory, complex data samples are stored as a 32-bit word, higher 16 bits constitute the real part while 16 lower-bits constitute the imaginary part. Data format stored in memory is pictorially described as shown in Figure 3.8.

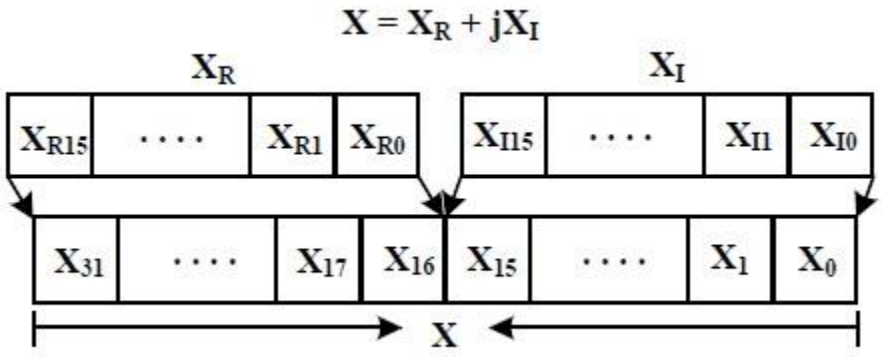


Figure 3.8: Data format stored in memory

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 3.9 describes the order of input samples stored in memory at the beginning of FFT computation.

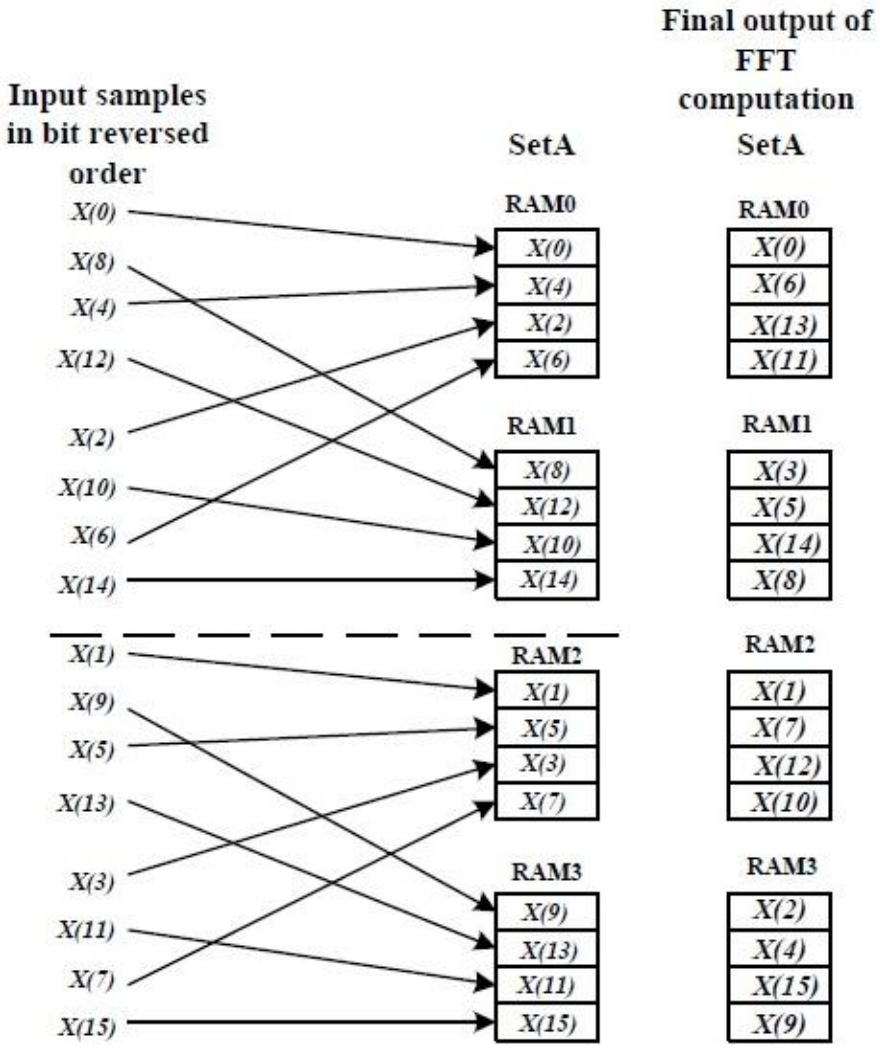


Figure 3.9: Order of input sample at the beginning and the order of final output of 16-point FFT computation

The bit reversed input samples are equally split into two halves, the upper half inputs stored in ***RAMO***, ***RAM1*** and the lower half inputs in ***RAM2***, ***RAM3***. The order of input samples is 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 the number of stages. If the number of stages is even, final outputs are stored in SetA otherwise if number of stages is odd, final outputs are stored in ***SetB***. Since for a 16-point FFT, the number of stages is even, final outputs would be available in ***SetA*** and the order of outputs would be as shown in Figure 3.9.

Eight suitable RAM memory banks were chosen to support ***Nmax***-point FFT computation and were integrated into the FFT processor. Each RAM memory bank is clocked dual port memory, enabling access from within the FFT processor for computations as well from an axi unit which is used to form a link between external interface and memories. Memory access (read/write) latency is one clock cycle. Figure 3.10 shows input/output ports of a memory bank.

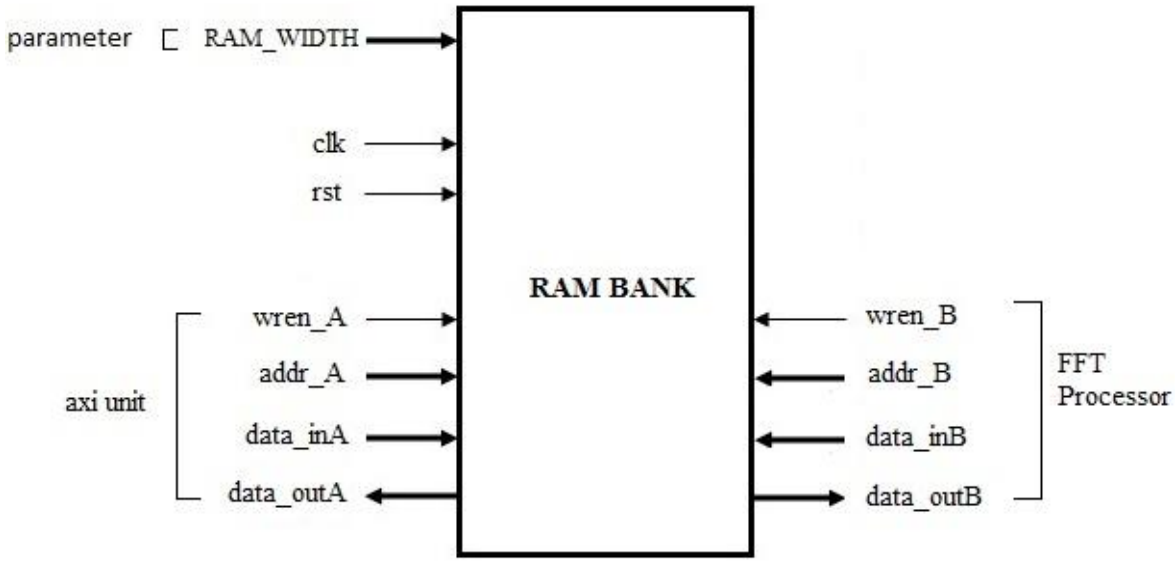
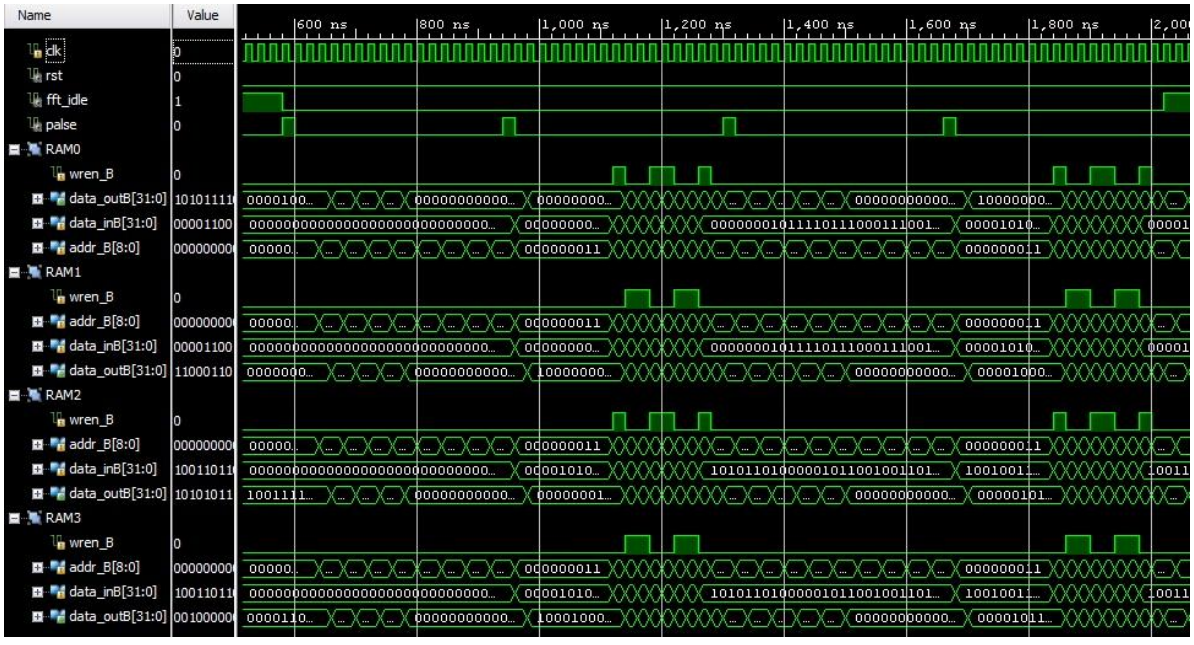


Figure 3.10: RAM memory bank port details

RAM_WIDTH port is a local parameter into FFT processor, which is computed by equation (3.8) based on given ***NMAX*** and it is used to specify the address width during design time. RAM bank has a clock port ***clk*** and an active high reset signal ***rst***. It consists of input data ports ***data_inA*** and ***data_inB***, address ports ***addr_A*** and ***addr_B***, write enable signals ***wren_A*** and ***wren_B*** and output data ports ***data_outA*** and ***data_outB***. Ports with suffix 'A' are axi unit ports and ports with suffix 'B' are FFT processor ports.

Waveforms of memory sets ***SetA*** and ***SetB*** for FFT processor ports are shown in Figure 3.11 (a) and (b) respectively for 16-point FFT.

(a)



(b)

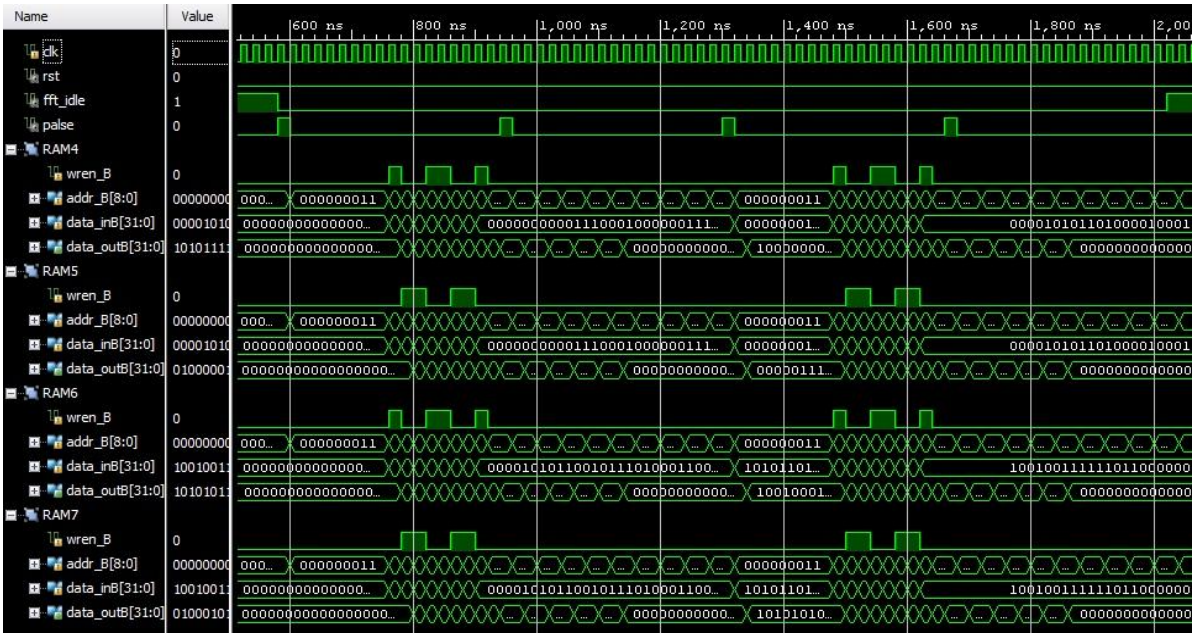


Figure 3.11: (a) Memory set SetA waveforms (b) Memory set SetA waveforms

Figure 3.11(a) and (b) illustrates the FFT processor operation. As shown in the first stage, data are read from ***SetA*** memory and the results are written to ***SetB***. In the next stage, data are read from ***SetB*** and the results are written to ***SetA***. Read/write operations are switched between ***SetA*** and ***SetB*** memory in alternate stages. Write operation takes place when ***wren*** signal is high and read operation takes place when signa is low. Memory read/write access latency is single clock cycle.

3.3 TWIDDLE FACTOR MEMORY (ROM)

The twiddle factor memory is used to store twiddle factors required by butterfly unit0 and butterfly unit1 during FFT computation, as shown in Figure 3.3. The maximum number of twiddle factors required for an N-point FFT computation is $\frac{N}{2}$. Hence, the maximum size of ROM unit to support up to ***Nmax***-point FFT computation is given by:

$$\text{rom\_size} = \frac{Nmax}{2} \tag{3.9}$$

where size is measured in terms of 32-bit words. The size of ROM defines its address bus width, is given by:

$$\text{address\_rom\_width} = \log_2(\text{rom\_size}) \quad (3.10)$$

Twiddle factors stored in ROM are in natural order, before a FFT processor initiates the computations. 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 3.8. It is illustrated for a 16-point FFT in Figure 3.12.

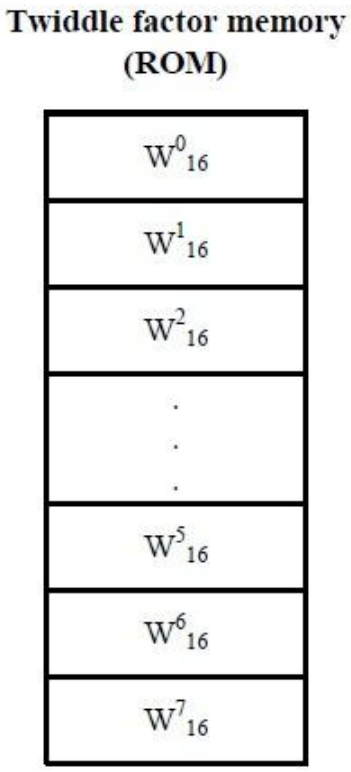


Figure 3.12: Order of twiddle factors stored in ROM

Twiddle factor memory was chosen to support ***Nmax***-point FFT computation and was integrated into the FFT processor. Twiddle factor memory is a clocked dual port ROM which allows reading two twiddle factors per clock cycle for the butterfly units. It grants access within the FFT processor for computations as well as to an axi unit for initialization of the twiddle factor memory, depending on the given FFT size. Memory access (read/write) latency is one clock cycle. Figure 3.13 shows input/output ports of a ROM memory.

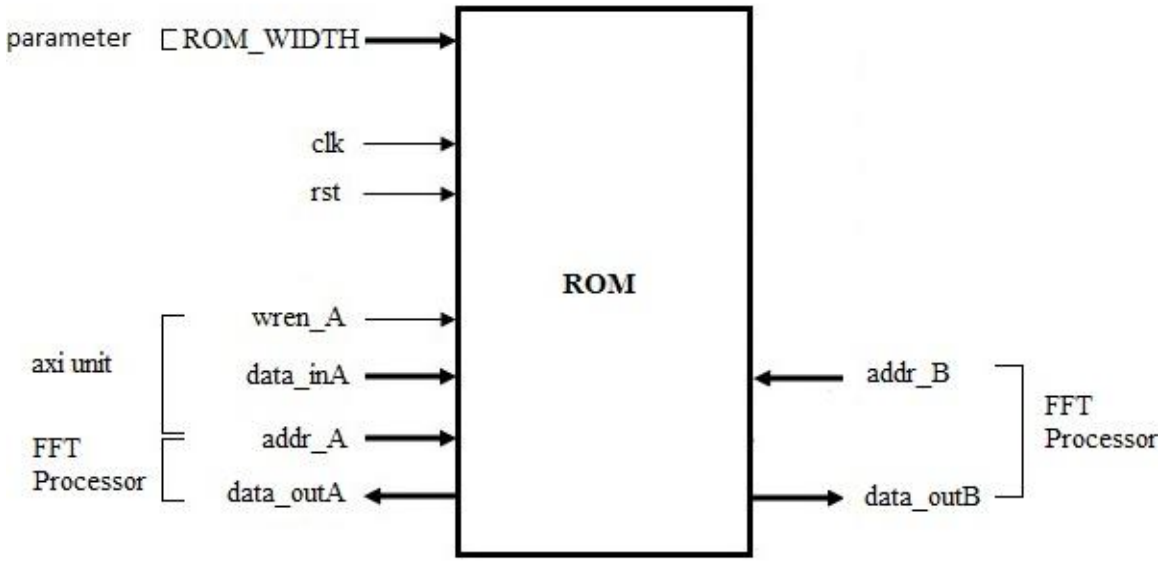


Figure 3.13: ROM memory port details

ROM_WIDTH port is a local parameter which is computed by equation (3.10) based on the given ***NMAX*** and it is used to specify rom address width during design time. ROM is a dual port memory which supplies twiddle factors for butterfly units. It has a clock port ***clk*** and an active high reset signal ***rst***. It consists of address ports ***addr_A*** and ***addr_B*** and output data ports ***data_outA*** and ***data_outB*** which are used by the FFT processor. Ports with suffix 'A' are meant for butterfly unit0 (twiddle factor0) while ports with suffix 'B' are meant for butterfly unit1 (twiddle factor1). Axi unit uses a write enable signal ***wren_A***, input data port ***data_inA*** and address port ***addr_A*** to initialize the twiddle factor memory. Address values from axi unit and FFT processor are multiplexed into port ***addr_A***. ***addr_A*** port is set to drive address value from axi unit while a FFT processor is idle. Twiddle factor memory consists of input data ports ***data_inA*** and ***data_inB***, address ports ***addr_A*** and ***addr_B***, write enable signals ***wren_A*** and ***wren_B*** and output data ports ***data_outA*** and ***data_outB***. Ports with suffix 'A' are axi unit ports and ports with suffix 'B' are FFT processor ports.

Waveforms of twiddle factor memory for FFT processor ports are shown in Figure 3.14 for 16-point FFT.

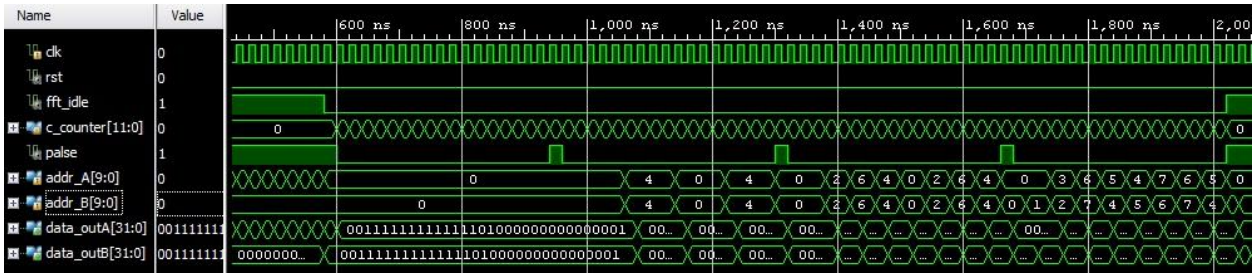


Figure 3.14: ROM memory waveforms

Figure 3.14 illustrates the FFT processor operation which is reading two twiddle factors per clock cycle for the butterfly units at different stages of FFT. Memory read/write access latency is single clock cycle.

3.4 INTERCONECT

Interconnect is the link between butterfly units and address generation with data memory. Two interconnects are used in FFT processor architecture and they are ***interconnectA*** and ***interconnectB*** as illustrated in Figure 3.4. The ***interconnectA*** forms a connection between butterfly units and memory ***SetA*** while ***interconnectB*** forms a connection between butterfly units and memory ***SetB***. The internal architecture of interconnect is described in Figure 3.15.

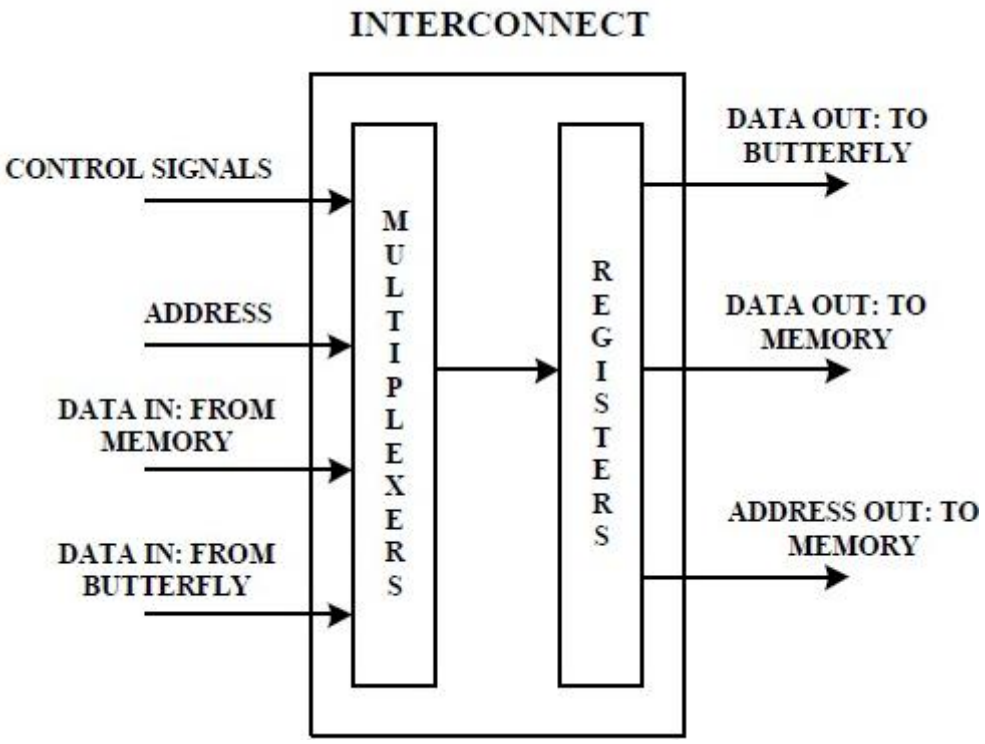


Figure 3.15: Interconnect internal architecture and the external interface

Interconnect consists of a number of multiplexers whose outputs are registered. The multiplexers act like switches, linking inputs to appropriate outputs based on selected signals from the control unit.

The interconnect unit was implemented in order to perform a reading/writing operation to the memory set at which a connection is established. The interconnect unit receives read and store addresses from the address generation unit. When it performs a reading operation, the read address is selected by the multiplexer and becomes registered. Thereafter, the read address is

routed to data memory set for reading. The four read data from memory set are registered in the interconnect unit which then routes read data to butterfly units for computation. If interconnect performs a writing operation, data is received as results from butterfly units and the write address is selected via multiplexer. Data and address are registered and then routed to memory for storage.

When FFT processor is in operation, in even numbered stages, the inputs are read from memory ***SetA*** and they are routed to butterfly units via ***interconnectA***. After butterfly operation, the outputs are routed via ***interconnectB*** and stored in memory set ***SetB***. In odd numbered stages, the inputs are read from memory ***SetB*** and they are routed to butterfly units via ***interconnectB***. After butterfly operation, the outputs are routed via ***interconnectA*** and stored in ***SetA*** memory. The four read data from memory set are separated and in some occasions flip before being supplied into butterfly units. The interconnect unit specifies the flow of data based on dataflow algorithm, which is described in detail in the relevant section (section 4).

Figure 3.16 shows input/output ports of an interconnect unit.

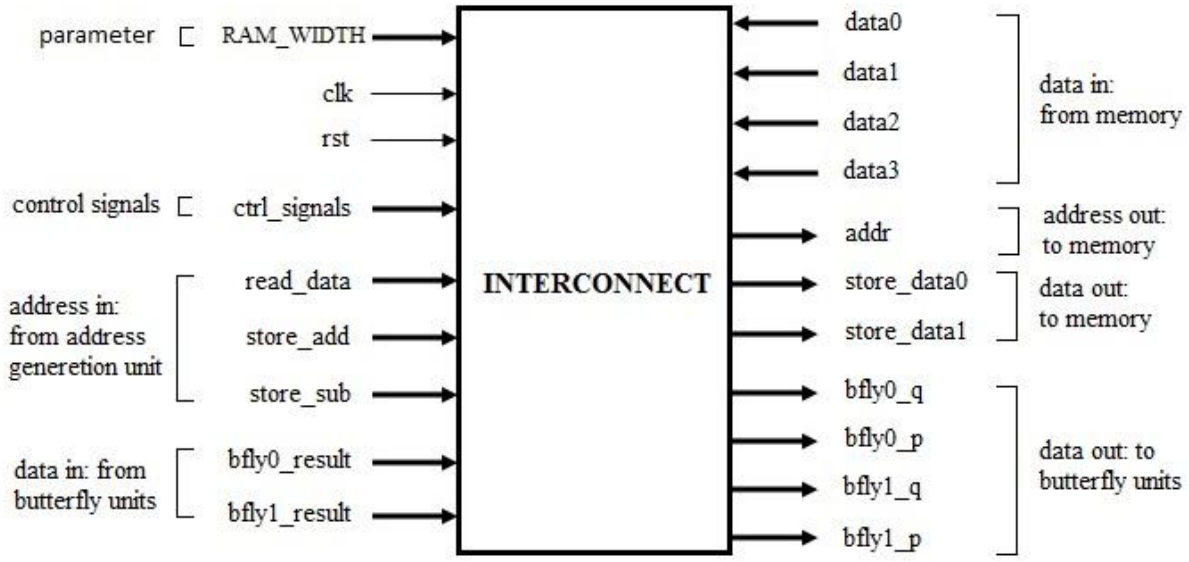


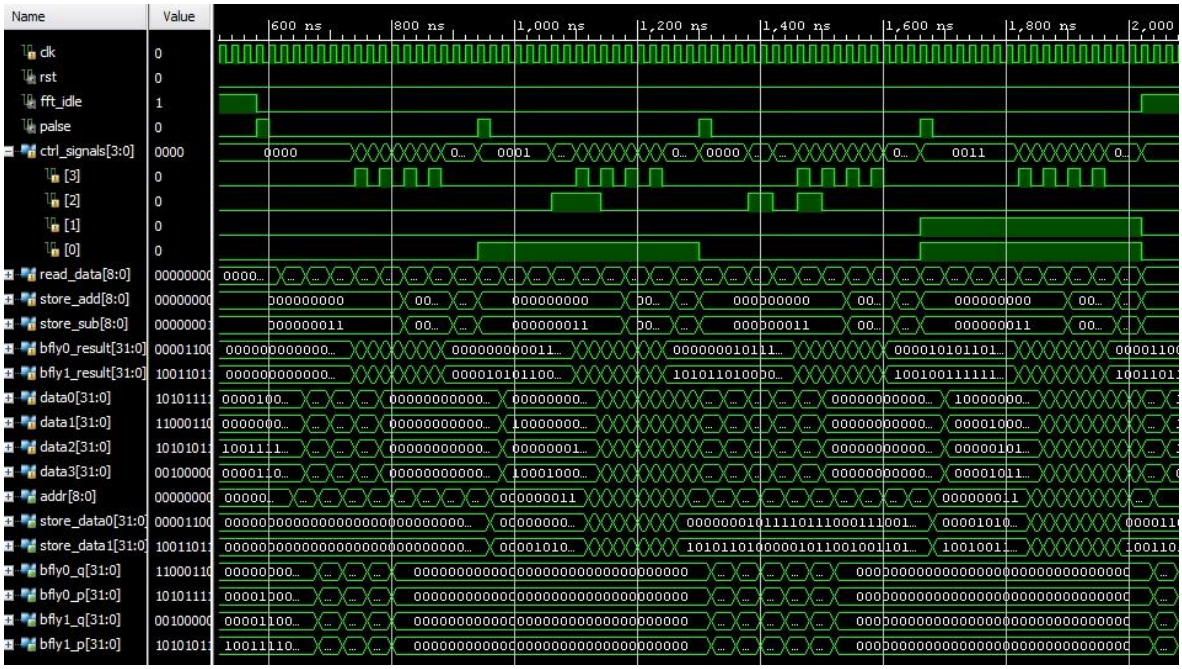
Figure 3.16: Interconnect port details

The interconnect unit uses the local parameter port ***RAM_WIDTH*** to specify ram address width. It has a clock port ***clk*** and an active high reset signal ***rst***. Control signals are driven into an input port ***ctrl_signals*** which is a 4-bit value and contains ***RW*** signal, ***last_stage*** signal, ***flip*** signal and ***add_sub*** signal. The order at which signals are mentioned above is from lower bit to upper bit of ***ctrl_signals*** value**.** ***RW*** signal specifies if the interconnect performs a reading or writing operation. ***last_stage*** signal indicates the last stage of FFT computation and ***flip*** signal indicates whether the read data from memory is to be flipped or not. The last control signal is ***add_sub***, which indicates whether the outputs from butterfly units are results of addition or subtraction. Memory read address is received through input port ***read_data*** and memory store

addresses are received through input ports ***store_add*** and ***store_sub***. ***store_add*** and ***store sub*** ports indicate the store addresses of addition and subtraction results respectively. The address is selected to be registered depending on the current operation of interconnect and it is routed via output port ***addr***. ***addr*** port is an input port to the memory set in which a connection is established. Four data inputs from memory set are received through ports ***data0***, ***data1***, ***data2*** and ***data3***. When interconnect performs a reading operation, ***last_stage*** and ***flip*** signals specify, via multiplexers, the flow of the read data before being registered into interconnect. The four read data which were registered are routed to butterfly units via output ports ***bfly0_q***, ***bfly0_p***, ***blfy1_q*** and ***bfly1_p***. ***bfly0_result*** and ***bfly1_result*** are input ports of interconnect, which contain the results of butterfly units. The results are registered and routed to memory set via output ports ***store_data0*** and ***store_data1*** respectively. When interconnect performs a writing operation, ***add_sub*** signal specifies, via a multiplexer the write address which will be registered and routed to memory set.

Figure 3.17 (a) and (b) shows waveforms of ***interconnectA*** ports and ***interconnectB*** ports respectively for 16-point FFT.

(a)



(b)

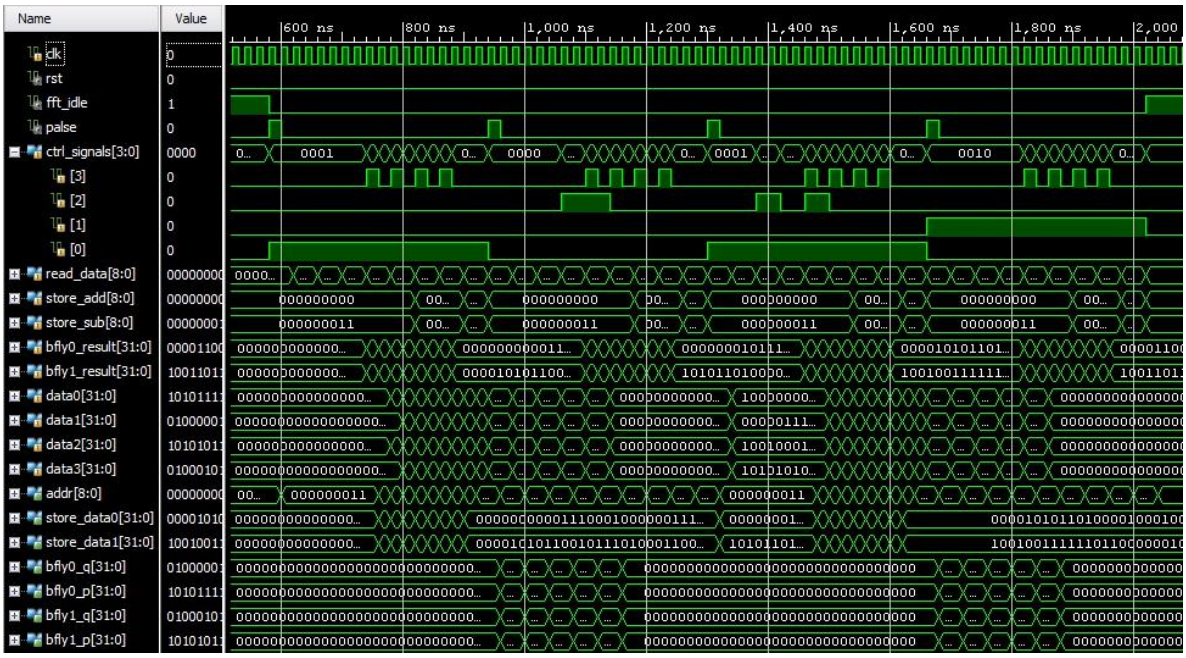


Figure 3.17: (a) InterconnectA waveforms (b) InterconnectB waveforms

As shown in a figure 3.17 (a) and (b), ***interconnectA*** performs a reading operation in every even numbered stage and a writing operation in every odd numbered stage. ***InterconnectB*** performs a writing operation in every even numbered stage and a reading operation in every odd numbered stage. ***last_stage*** and ***flip*** signals specify the flow of the read data before being registered and supplied to butterfly units. When ***last_stage*** signal is high, it indicates that it is the last stage. When ***flip*** signal is high, it indicates that the current read data from memory have to flip. The results of addition and subtraction are driven every alternate clock cycle by butterfly units. When ***add_sub*** signal is high, it indicates that the current data received by butterfly units are subtraction results. Otherwise, when ***add_sub*** signal is low, it indicates that the current data received by butterfly units are addition results.

3.5 ADDRESS GENERATION UNIT

Address generation unit generates addresses in each stage of FFT computation for reading input data samples, twiddle factors and storing outputs data samples. The address generation unit that was implemented is based on an address generation scheme which was developed to support N-point FFT computation.

The address generation algorithm provides conflict free access of data samples during computation. It requires two m-bit counters, where $m = \log_2(N/4)$, according to equation 3.8. 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.

$$read\_data = [a_{m-1}a_{m-2}.....a_1a_0] \quad (3.11)$$

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

$$store\_add = [b_{m-1}b_{m-2}\ldots\ldots b_1b_0] \quad (3.12)$$

$$store\_sub = [b'_{m-1}b'_{m-2}\ldots\ldots b'_1b'_0] \quad (3.13)$$

3) Address to read twiddle factors from ROM:
For an N-point FFT there are $\log_2 N$ 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, ... $\log_2(N) - 1$.

- Twiddle factor addresses for stages except last stage (s = 0, 1, … $\log_2(N)-2$):

(i) $[d_{m-1}..d_0] = [a_{m-1}\ldots a_0]$ XOR $[0a_{m-1}\ldots a_1]$
(ii) $count\_gray = [0d_{m-1}d_{m-2}\ldots..d_1d_0]$
$= [e_me_{m-1}..e_1e_0]$
(iii) $coef\ x = [e_m\ldots.e_{m-s}0_{m-s-s}\ldots0_10_0]$
$= [f_{m+1}f_m\ldots f_1f_0]$
(iv) $Coef0Addr = [f_mf_{m-1}\ldots f_1f_0]$
(v) $Coef1Addr = Coef0Addr$

- Twiddle factor addresses for last stage (s = $\log_2(N)-1$):
Note: coefy = $[0_{m+1}0_m\ldots0_11]$ at the beginning of the stage.

(i) $coefy = [f_{m+1}f_m\ldots f_1f_0]$
(ii) $sum = [a_{m-1}a_{m-2}..a_1a_0] + [f_mf_{m-1}..f_2f_1]$
$= [g_{m-1}g_{m-2}\ldots g_1g_o]$
(iii) $coefy = [0g_{m-1}g_{m-2}\ldots g_1g_of'_0]$
$= [f_{m+1}f_m\ldots f_0]$

f_m = *'0'* for first $\frac{N}{4}$ butterfly computations.
f_m = *'1'* for next $\frac{N}{4}$ butterfly computations.

(iv) $Coef0Addr = [f_mf_{m-1}\ldots f_1f_0]$

(v) $Coef1Addr = [f_m f m_{m-1} \ldots f_1 f'_0]$

The address ***read_data*** (step 1) is used to read inputs from either SetA or SetB memory, depending on FFT stage. There are two outputs to butterfly operation; one is a result of addition and the other is a result of subtraction. Hence, in step 2 there are two store addresses generated. Two outputs from butterfly unit have to be stored in different memory banks and in different address locations to avoid access conflicts. Therefore, the address ***store_add*** stores the result of addition and ***store_sub*** stores the result of subtraction. Step 3 presents twiddle factor address generation logic where ***Coef0Addr*** corresponds to butterfly unit0 and ***Coef1Addr*** corresponds to butterfly unit1. The steps described are executed in the specified order in order generate addresses ***Coef0Addr*** and ***Coef1Addr***. The internal logic of address generation unit is shown in figure 3.18.

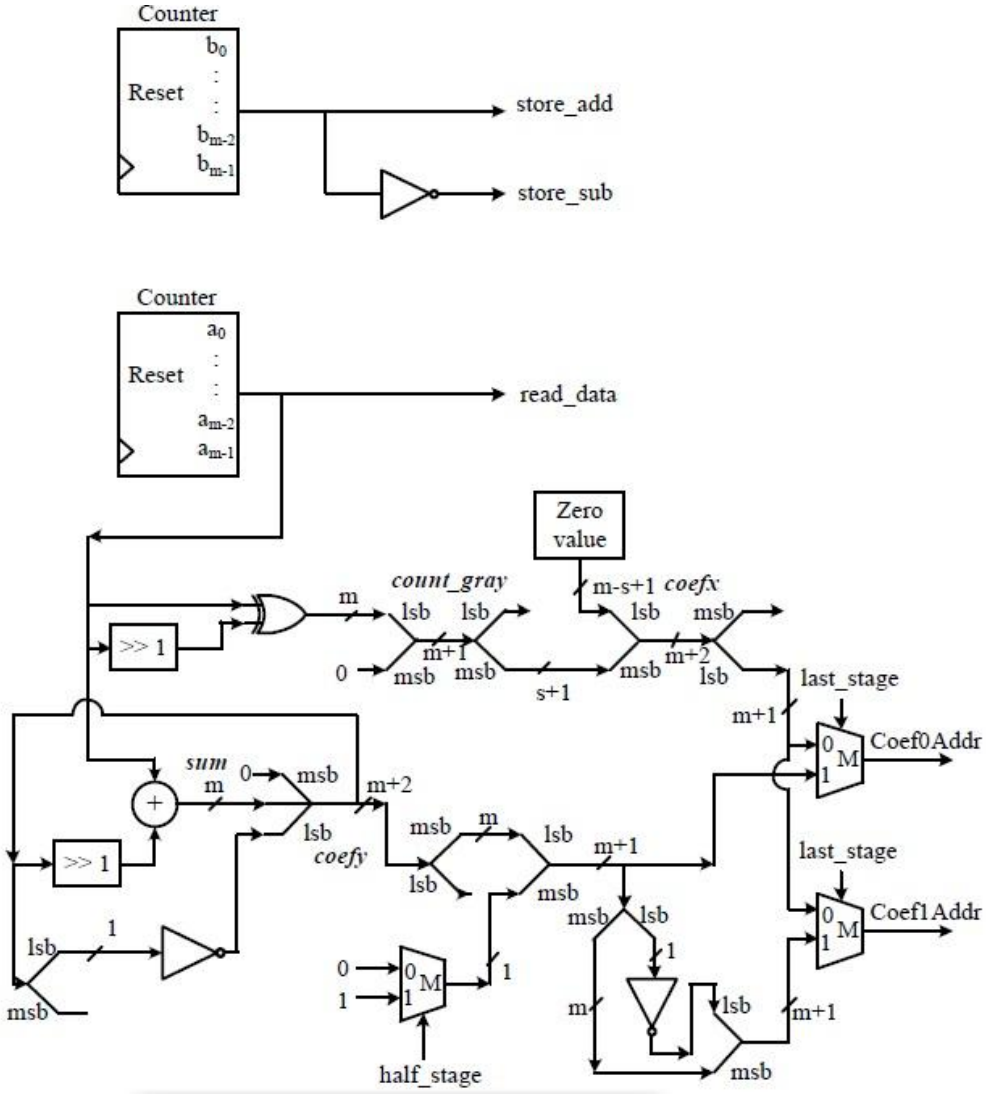


Figure 3.18: Address generation unit internal logic

Address generation unit was implemented based on the description of algorithm, as mention above. Figure 3.19 shows input/output ports of address generation unit.

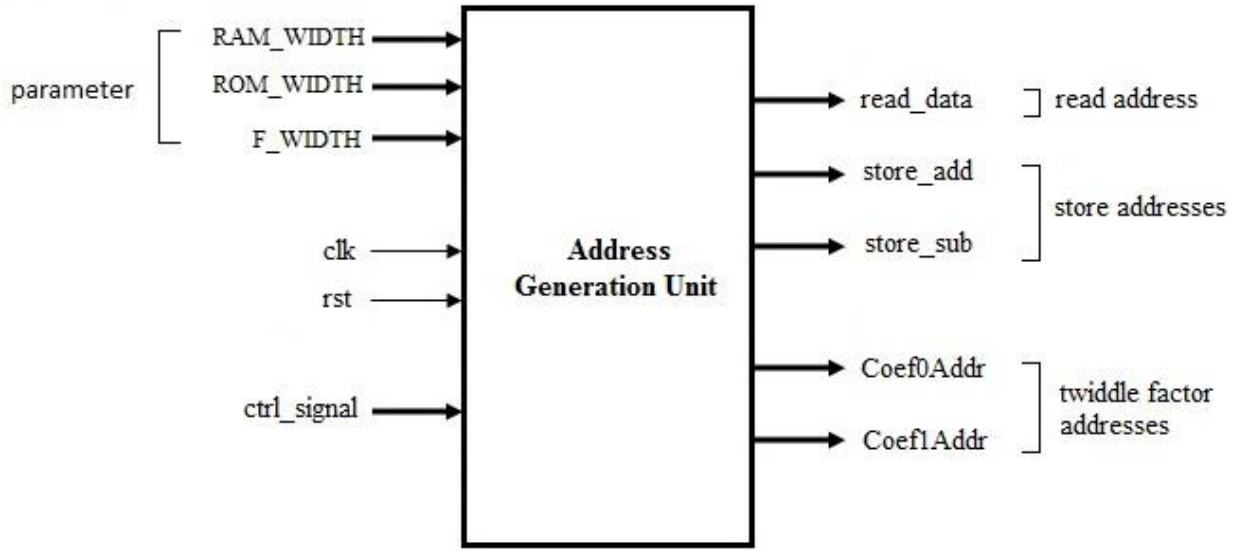


Figure 3.19: Address generation unit port details

The address generation unit uses the local parameter ports ***RAM_WIDTH*** and ***ROM_WIDTH*** to specify ram and rom address width respectively. Furthermore, ***F_WIDTH*** port is used as a local parameter to support the variables width for the maximum FFT. ***F_WIDTH*** value is computed by equation $\log_2(\boldsymbol{NMAX})+1$. The unit has a clock port ***clk*** and an active high reset signal ***rst***. Control signals are driven into an input port ***ctrl_signals*** which is (2*F_WIDTH+2)-bit value. ***ctrl_signals*** port contains ***fft_idle*** signal, ***last_stage*** signal, ***curr_size*** value and ***c_counter*** value. The order at which signals are mentioned above is from lower bit to upper bit of ***ctrl_signals*** value***. fft_idle*** signal indicates whether the FFT processor is idle. It is used by address generation unit to start generating addresses for given FFT size when FFT processor is set into operation. ***last_stage*** signal indicates the last stage of FFT computation. ***curr_size*** consists the value of the given FFT size and ***c_counter*** consists the value of current clock cycle in the stage. ***c_counter*** is used to define the values of addresses at proper time. Read address port is ***read_data*** and store address ports are ***store_add*** and ***store_sub***. The addresses ***read_data*** and ***store_add*** are m-bit counters whereas ***store_sub*** is one's complement version of ***store_add***. The twiddle factor address ports are ***Coef0Addr*** and ***Coef1Addr***.

Figure 3.20 shows waveforms of address generation unit for 16-point FFT

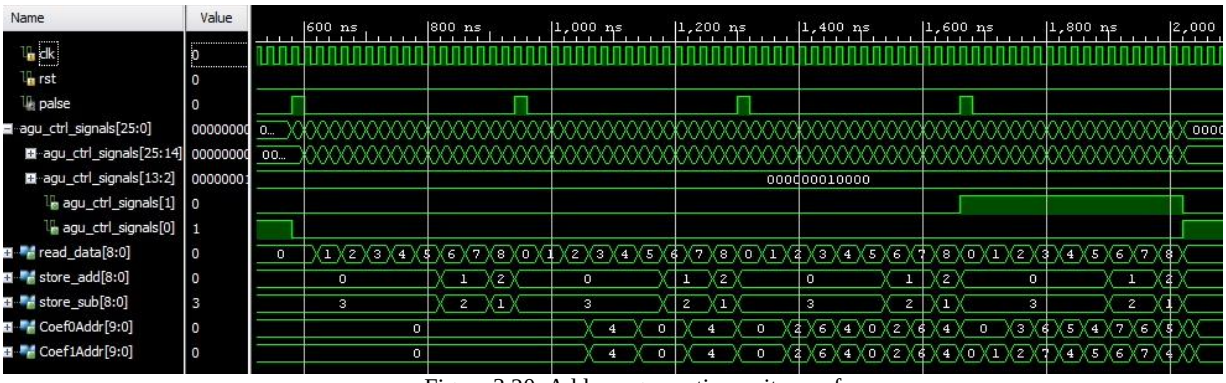


Figure 3.20: Address generation unit waveforms

As shown in a figure 3.20, address generation unit starts immediately after ***fft_idle*** signal is set to '0' by the control unit. For every two clock cycles, a new read address is generated and at the 8th clock cycle and for every four clock cycles, new store addresses are generated. The two multiplexer signals ***half_stage*** and ***last_stage*** enable coefficient address selection depending on the state of butterfly computations in a given stage. The ***last_stage*** signal is set to '1' for the last stage and to '0' for the rest of the stages, differentiating the last stage of FFT from the rest of the stages. The ***half_stage*** signal is computed into address generation unit based on ***c_counter***. In the last stage of FFT computation, the ***half_stage*** signal is set to '0' for the first $\frac{N}{4}$ butterfly computations and to "1" for the next $\frac{N}{4}$ butterfly computations. The twiddle factor addresses ***Coef0Addr*** and ***Coef1Addr*** are delayed by two clock cycles at the time that they were calculated. The reason is that the twiddle factor addresses are routed directly to ROM and they require two clock cycles of delay in order to synchronize with the read data from RAM before supplying them to butterfly units.

It is important to show the delay of twiddle factor addresses and store addresses generation, as is necessary for the proper flow of data. Figure 3.21 is a pictorial description of timeline for the nine pipeline stages.

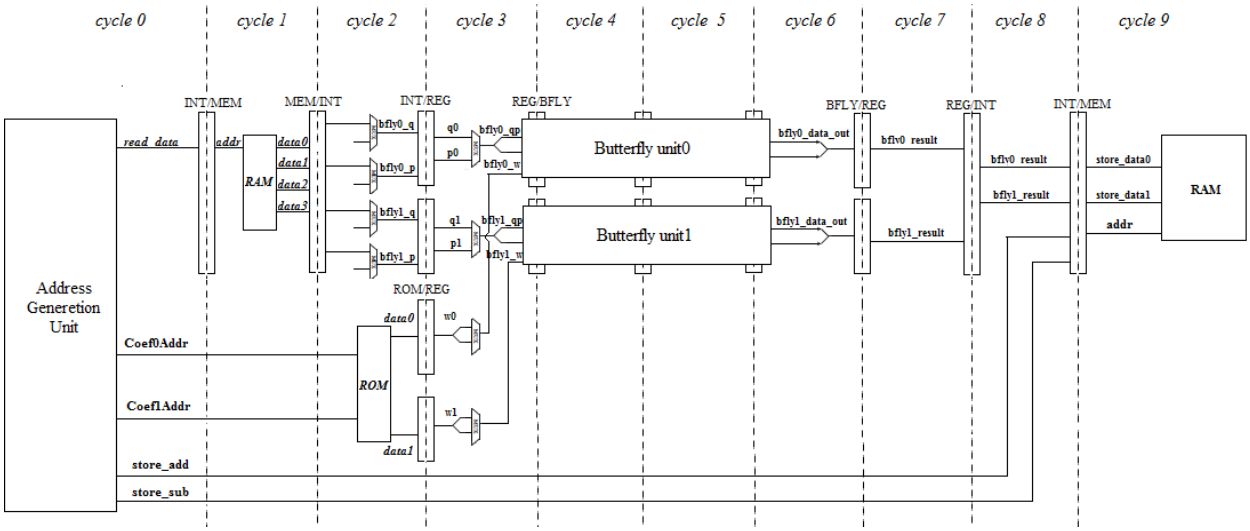


Figure 3.21: Timeline of nine pipeline stages.

3.6 CONTROL UNIT

The control unit is the most important unit of FFT processor. It generates control signals at proper timing to control and coordinate the activities of all other units in the processor as shown in Figure 3.3. Control unit is implemented using Moore state machine, as shown in figure 3.22.

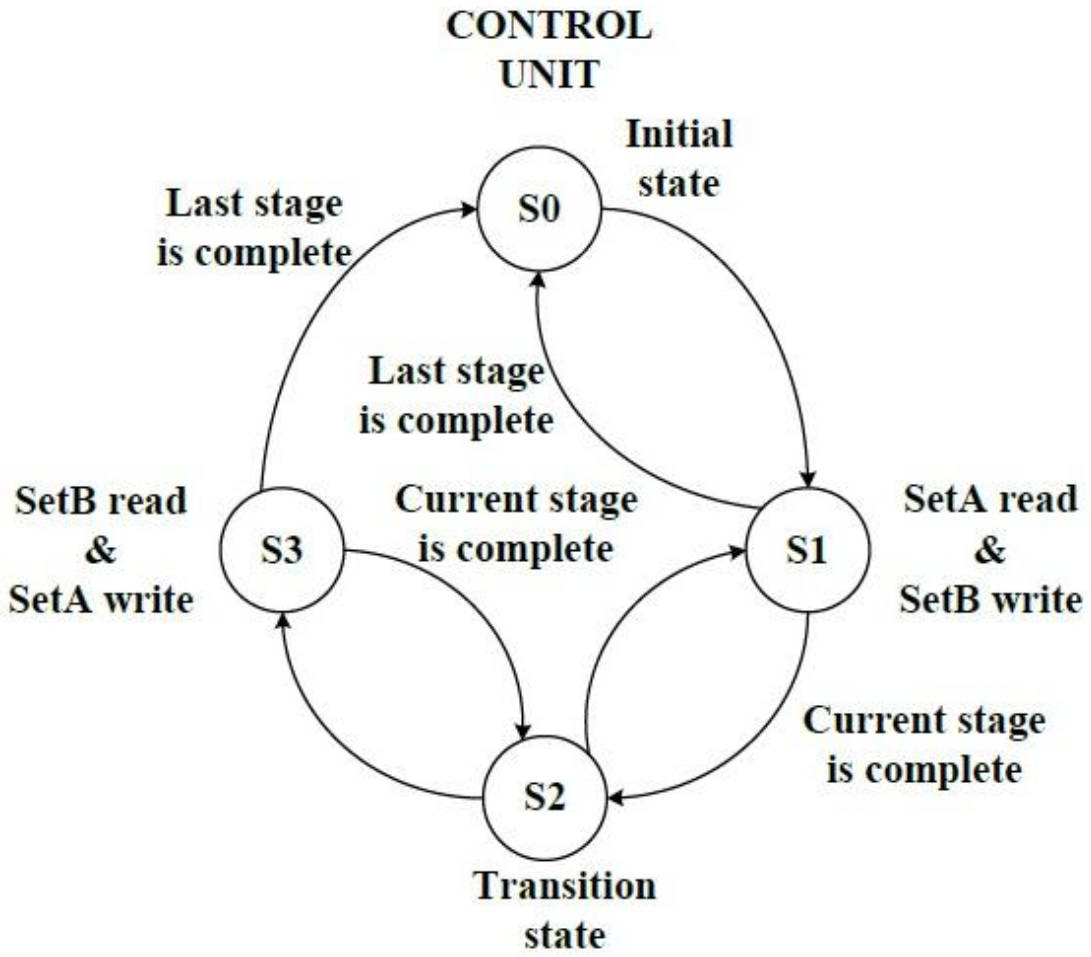


Figure 3.22: Control unit state diagram

At the beginning, control unit starts with state S0 in which the FFT processor is idle. In state S0, all the signals and variables are initialized depending on required N-point computation. After being initialized, when it receives the suitable signal to start computing, it moves to state S1. In state S1, it generates control signals for the following activities: address generation from address generation unit, read input samples from ***SetA*** memory, route inputs to butterfly units via ***interconnectA***, butterfly units operation, route outputs to ***SetB*** memory via ***interconnectB*** and write butterfly outputs to ***SetB*** memory. If the last stage of FFT computation is completed, it

moves from state S1 to state S0. Otherwise, after completing an FFT computation stage in S1 it moves to transition state S2 where it prepares itself before generating control signals for next stage.

From transition state S2 it moves to state S3. In state S3 it generates control signals for the following activities: address generation from address generation unit, read input samples from ***SetB*** memory, route inputs to butterfly via ***interconnectB***, butterfly units operation, route the outputs to ***SetA*** memory via ***interconnectA*** and write butterfly outputs to ***SetA*** memory. If the last stage of FFT computation is completing an FFT computation stage in S3 it moves to state S0. Otherwise, after completing a FFT computation stage in S3 it moves to transition state S2, where it prepares itself before generating control signals for next stage. The process of state transitions S1 → S2 → S3 and S3 → S2 → S1 are repeated in every alternate stage until it returns to state S0 at the end of FFT computation.

It is important to note that in order to generate suitable control signals, it was required to define a clock cycle counter. The counter is used to count the clock cycles that are needed in each stage. The number of clock cycles required by each stage is given by equation 3.2. This achieves the generation of control signals at proper time. Figure 3.23 shows input/output ports of control unit.

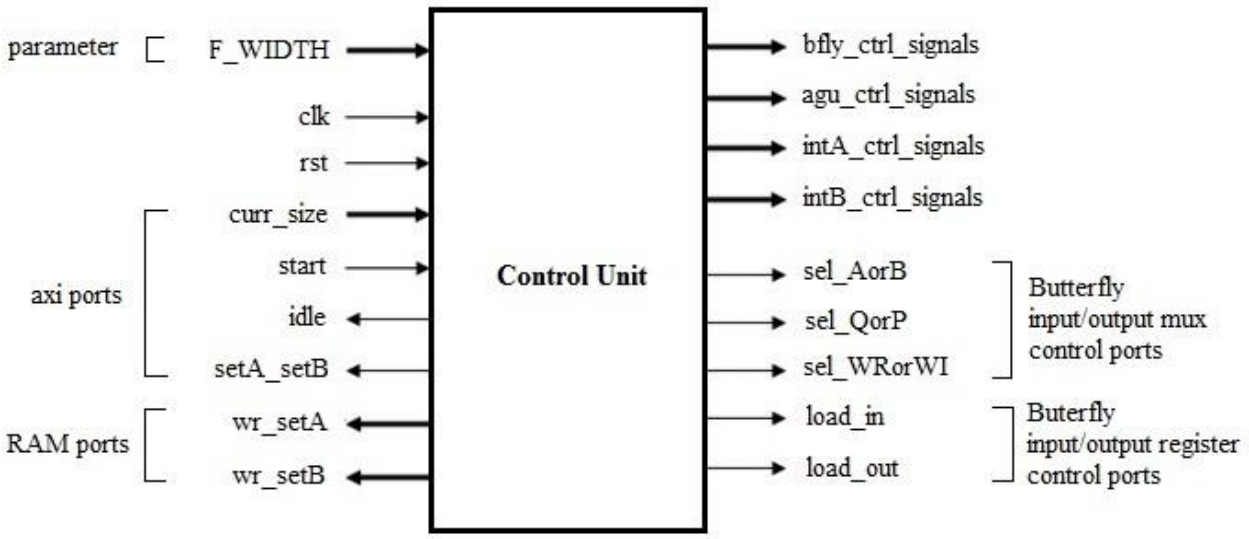


Figure 3.23: Control unit port details

The control unit uses the local parameter port ***F_WIDTH*** to specify the variables for the maximum FFT size. The unit has a clock port ***clk*** and an active high reset signal ***rst***. ***curr_size*** port has the value of the given FFT size and ***start*** port indicates the start of FFT computation. ***idle*** signal indicates whether the FFT processor is idle and ***setA_setB*** signal indicates which set of memory has stored the result at the end of FFT computation. ***wr_setA*** port and ***wr_setB*** port are 2-bit value for read-write signaling to memory sets ***SetA*** and ***SetB*** respectively. ***bfly_ctrl_signals*** port and ***agu_ctrl_signals*** port are corresponding to control signals of butterfly

units and address generation unit respectively. ***intA_ctrl_signals*** port and ***intB_ctrl_signals*** port correspond control signals of ***interconnectA*** and ***interconnectB*** respectively.

Registers and multiplexers were included in pipeline architecture of FFT between interconnects and butterfly units. Multiplexer signals specify the flow of data pre-computation of butterfly units. ***sel_AorB*** multiplexer signal specifies the interconnect unit from which, the read data will be registered before being supply them into butterfly units. ***sel_QorP*** multiplexer signal specifies the complex value (Q or P) which will be routed into butterfly units and ***sel_WRorWI*** multiplexer signal specifies the value (WR or WI) which will be routed into butterfly units. ***load_in*** and ***load_out*** are load signals for input registers in butterfly units and output registers of butterfly units respectively.

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

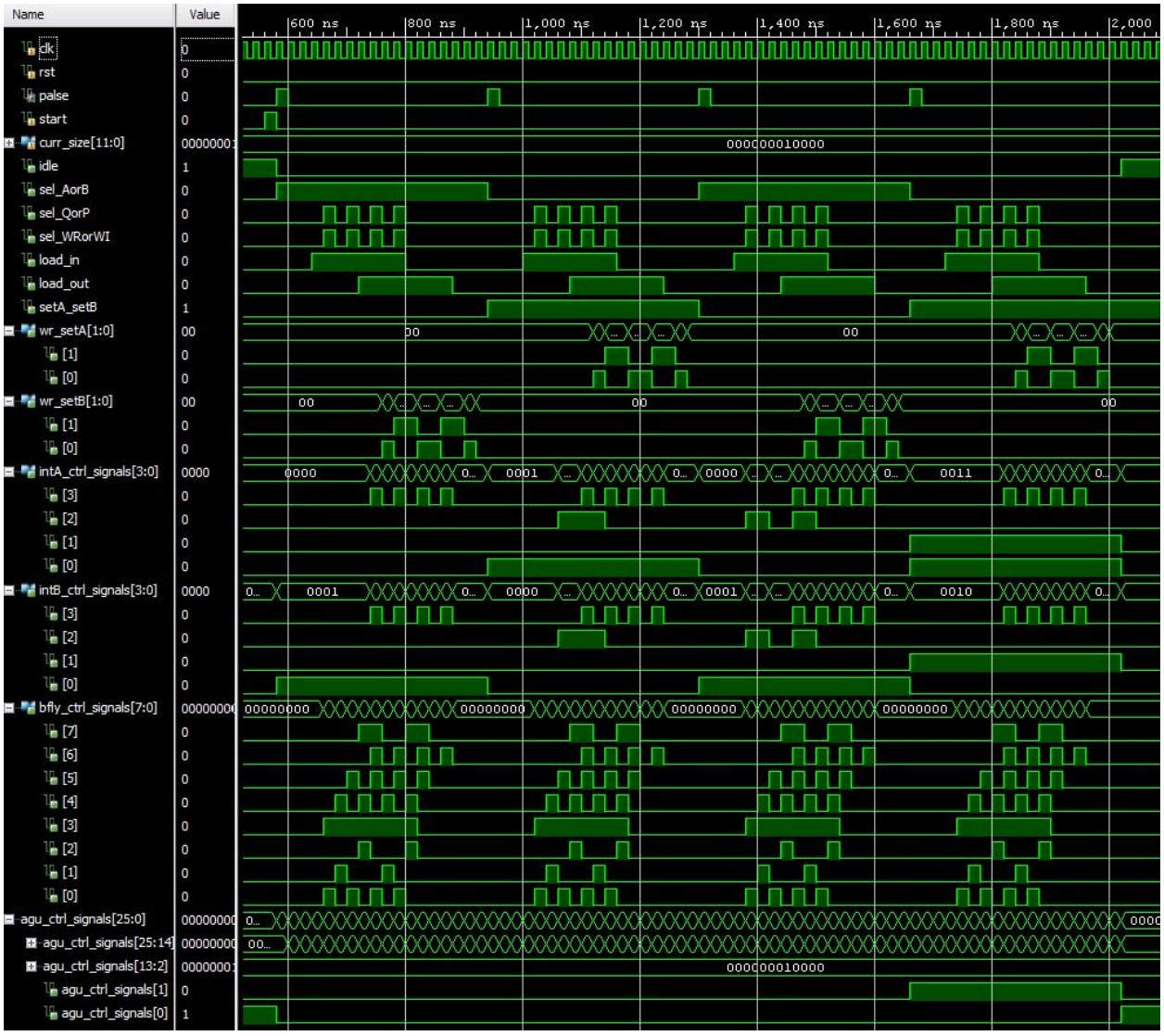


Figure 3.24: Control unit waveforms

3.7 AXI UNIT

Axi unit was implemented in order to manage the configuration, write and read channels of FFT processor. It is based on rules of communication protocol which were defined by the external interface. The properties of transmission that the protocol defines are the packet size, handshaking and error type.

The three transaction channels use the same ***valid***/***ready*** handshake process to transfer data. This two-way flow control mechanism means that both the master and slave can control the rate at which the information moves between them. The source generates the ***valid*** signal to indicate when data information is available and the destination generates the ***ready*** signal to indicate whether it can accept the information. Transfer occurs only when both the ***valid*** and ***ready*** signals are high. All channels transfer 64-bit value of data. Write channel and read channel also include ***last*** signal to indicate the transfer of the final item in transaction.

The external interface specifies the size of FFT computation via the configuration channel and axi unit enables the receipt of samples data via the write channel. For each sample data received, a writing operation is performed in order to store the input samples data in memory set ***SetA***. The 64-bit samples data are segregated into higher 32 bits which constitute the real part in floating point and into lower 32 bits which constitute the imaginary part in floating point. The 32-bit data are routed into toFixed units which convert them to 16-bit values in fixed point. They are then packed into 32-bit complex value and are stored in memory in bit reversed order, as described above in figure 3.9. The write latency for each data received is one clock cycle and the number of clock cycles required for writing input samples data for an N-point FFT is N clock cycles in continuous transmission flow.

When Axi unit receives the first sample data via the write channel, it starts to fill the twiddle factor memory if the given size differs from the previous FFT computation. Cordic generation unit, which supports fixed point computation, was integrated into axi unit in order to compute twiddle factors according to the equation,

$$W_N^k = e^{-(j\frac{2\pi k}{N})}, \quad k = 0, 1, 2 \ldots \frac{N}{2}, \tag{3.14}$$

where is expressed as,

$$W_N^k = \cos\left(\frac{2\pi k}{N}\right) + \text{jsin}\left(\frac{2\pi k}{N}\right) \tag{3.15}$$

The cos and sin are the real and imaginary part of twiddle factor respectively and are computed by the cordic generation unit. The results are merged into 32-bit complex value and are stored in rom memory. It takes a new input angle every clock cycle and gives results 15 clock cycles later. The total number of clock cycles required to initialize the twiddle factor memory for N-point FFT is $15 + \frac{N}{2}$ clock cycles.

The axi unit informs the control unit to start the FFT computations when the samples data inputs are stored in memory set ***SetA*** and twiddle factor memory is filled with data which is required to support the given FFT size. On the other hand, when the given FFT size and the number of given samples data do not match, axi unit informs the external interface for error. After completing a FFT computation, the control unit informs the axi unit memory set in which the final results are stored. Then, as shown in figure 3.9 the results have to be sorted in physical order before being supply to external interface via the read channel. The axi unit reads the final results from memory set and stores them in physical order to the other memory set. $\frac{N}{4}$ clock cycles are required to sort the results. Thereafter, when the read channel is enabled for transfer, the sorted data are segregated into higher 16-bits and lower 16-bits which are converted to 32-bits floating point by toFloat unit and are supplied in 64-bit value to external interface. The number of clock cycles required to transfer the final sorted data for an N-point FFT is N clock cycles in continuous transmission flow.

Figure 3.25 shows input/output ports of axi unit.

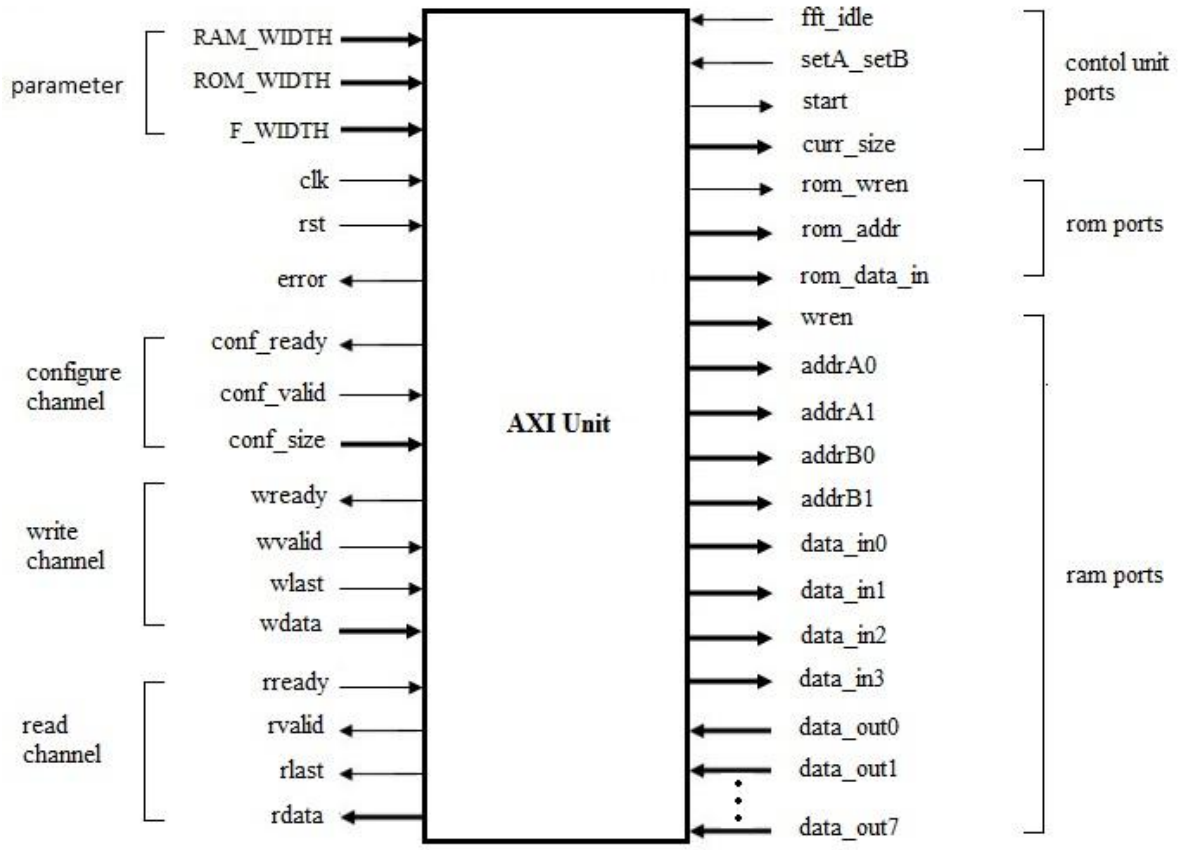


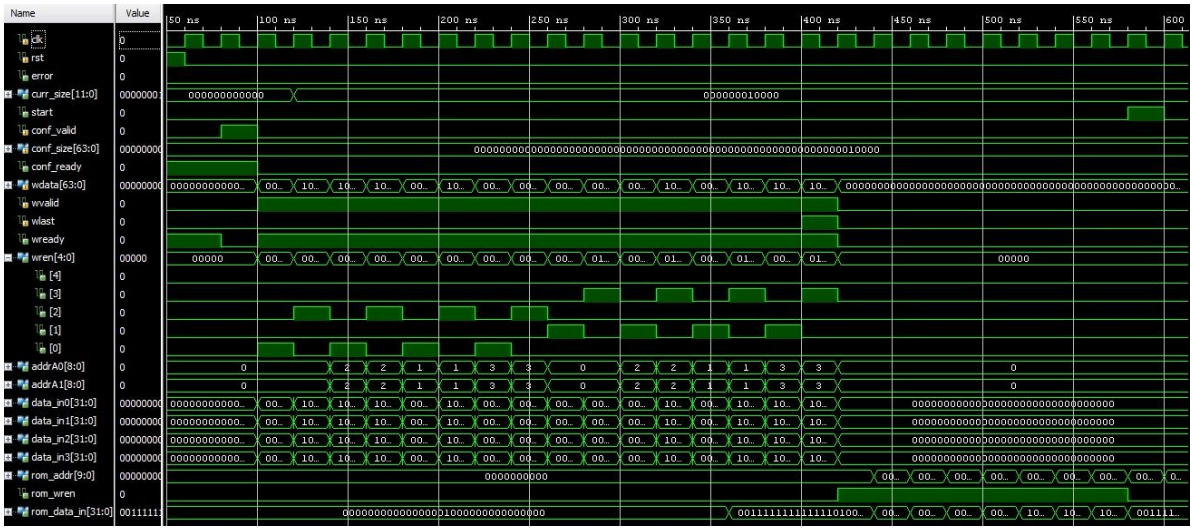
Figure 3.25: Axi unit port details

The axi unit uses the local parameter ports ***RAM_WIDTH***, ***ROM_WIDTH*** and ***F_WIDTH*** to specify ram address width, rom address width and the variables with the

maximum FFT size respectively. The unit has a clock port ***clk*** and an active high reset signal ***rst***. ***error*** signal indicates whether the given FFT size and the number of given data sample do not match a prerequisite for the FFT processor to start. Each channel contains three ports. ***ready*** signal indicates when it can accept the data, ***valid*** signal indicates when the data are valid and ***data*** port which has 64-bit value for transfer. Write channel and read channel also include ***last*** signal to indicate the transfer of the final item in transaction. All channel signals and ports are driven via the FFT processor. ***curr_size*** port has the value of the given FFT size and ***start*** signal indicates the start of FFT computation. ***idle*** signal indicates whether the FFT processor is idle and ***setA_setB*** signal indicates which set of memory was stored by the final results. Write enable signal ***rom_wren***, address port ***rom_addr*** and input data port ***rom_data_in*** are used to initialize the twiddle factor memory. Write enable port ***wren*** is 4-bit value for write signaling to RAM banks of the memory sets ***SetA*** and ***SetB***. **addrA0** and **addrA1** ports have the values of addresses of RAM banks of the memory sets ***SetA***. **addrB0** and **addrB1** ports have the values of addresses of RAM banks of the memory sets ***SetB***. Data in ports ***data_in0***, ***data_in1***, ***data_in3*** and ***data_in4*** are driven to four RAM banks in each memory set. Data out ports for memory set ***SetA*** are ***data_out0***,..***data_out3*** and for memory set ***SetB*** are ***data_out4***,..***data_out7***.

Figure 3.26 (a) and (b) shows waveforms of axi unit operations for 16-point FFT.

(a)



(b)

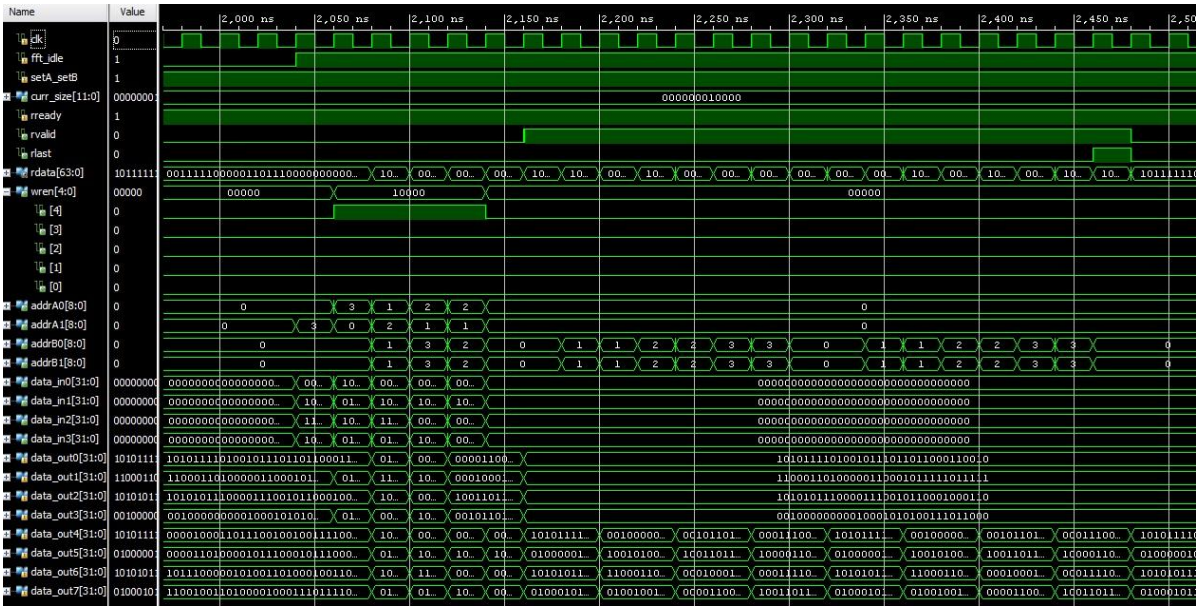


Figure 3.26: Axi unit waveforms (a) writing of the initial values to memory SetA and twiddle factor memory (b) result sorting and reading operation

4. DATAFLOW ALGORITHM

The flow of data across various components of processor is the basis of FFT processor architecture. The dataflow algorithm describes reading of inputs from memory, routing of inputs to butterfly units and routing outputs of butterfly units to memory. The units of FFT processor were implemented to succeed the proper flow of data based on the algorithm which described as follows.

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 supplied to butterfly units. After butterfly computations the outputs are routed via ***interconnectB*** and stored in ***SetB*** memory.

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

Dataflow involving butterfly unit0:

Even numbered stages:

- During all 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 all 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 all 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 all 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 flow of data for 16-point FFT is pictorially described in datagram of figure 4.1. Continues line butterfly belongs to butterfly unit0 and dotted line butterfly belongs to butterfly unit1.

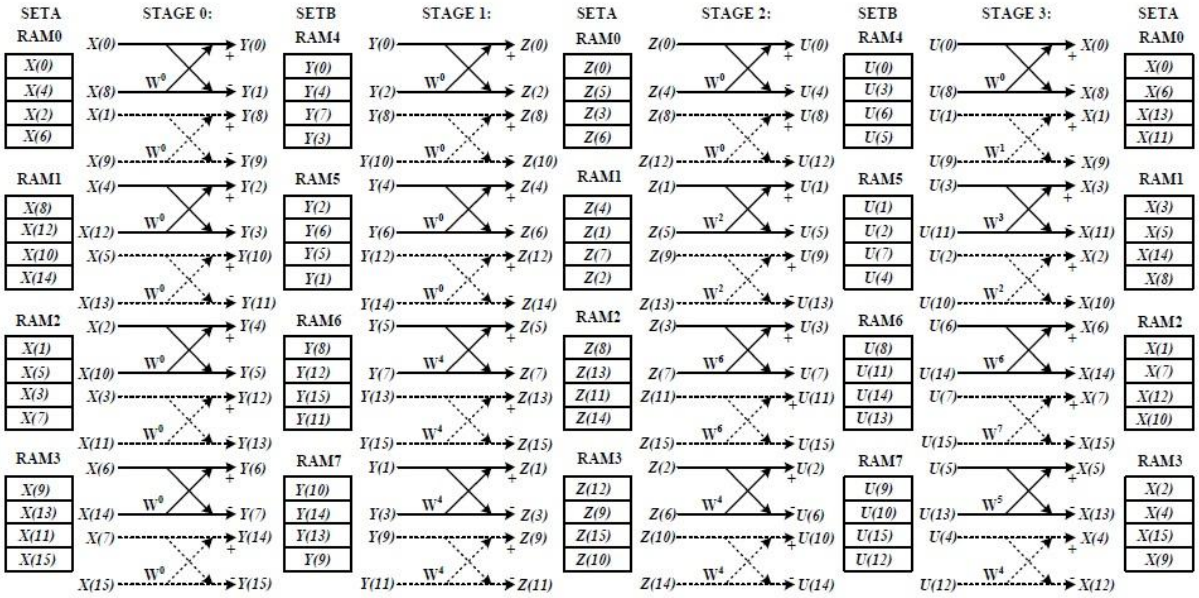


Figure 4.1: 16-point dataflow butterfly diagram for FFT processor architecture

Figure 4.1 shows the 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.

5. RESULTS

The FFT processor was implemented in Verilog, a Hardware Design Language (HDL) using the simulation tools ModelSim and Vivado. Initially, FFT processor components were implemented separately and were verified by running test benches. After individual modules were functionally verified, they were integrated to form the complete system. The FFT processor as a complete system was functionally verified by simulating operations of an external interface in a test bench. During simulation, the processor was configured for different radix-2 sizes from 16 to 2048 as maximum FFT size ***Nmax***. FFT computation time in clock cycles for different FFT sizes was verified by equation (3.2) and is given in Table 5.1.

N	Clock Cycles
16	74
32	132
64	254
128	520
256	1106
512	2396
1024	5222
2048	11376

Table 5.1: FFT Computation Time

As shown in table, 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.

6. CONCLUSION

The FFT processor was implemented and functionally verified through RTL simulation according to the scalable FFT processor architecture which was proposed. According to [1], the architecture outperforms the existing fixed and variable length FFT processor in terms of speed, power, area, flexibility and scalability, attributes required by a wide range of multiple wireless standards. Furthermore, the architecture can be extended to radix-4/8 computations in order to achieve higher performance. In addition, the FFT processor can be used in non-OFDM systems where scalability is required.

REFERENCE

[1] D. Revanna, O. Anjum, M. Cucchi, R. Airoldi and J. Nurmi, "A Scalable FFT Processor Architecture for OFDM Based Communication Systems"

[2] J.W.Cooley and J.W.Tukey, "An algorithm for the machine calculation of the complex Fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[3] Tran-Thong and B. Liu, "Fixed-point fast Fourier transform error analysis," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 24, no. 6, pp. 563–573, dec 1976.

[4] J. Takala and K. Punkka, "Butterfly unit supporting radix-4 and radix-2 FFT," in *Proc. Int. Workshop Spectral Methods and Multi rate Signal Process., Riga, Latvia*, June 20-22 2005, pp. 47–53.