MICROPIPELINE A DATAPATH OF FFT

A Thesis

Presented to the

Faculty of

San Diego State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Electrical Engineering

by

Truptiben Dhirajlal Patel

Summer 2010
SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the Thesis of Truptiben Dhirajlal Patel:

Micropipeline a Datapath of FFT

Santosh Nagaraj, Chair
Department of Electrical and Computer Engineering

Sridhar Seshagiri
Department of Electrical and Computer Engineering

Samuel Kassegne
Department of Mechanical Engineering

5/12/2010
Approval Date
Copyright ©2010
by
Truptiben Dhirajlal Patel
All Rights Reserved
DEDICATION

I would like to dedicate my thesis to my loving parents and family for all their caring support they have given throughout my life.
Fast Fourier Transform (FFT) is one of the most important algorithms in digital signal processing. FFT is widely used algorithm in various areas like image and speech processing, medical electronics and telecommunication. Design of FFT processor requires better approach, as growing the need of faster and small battery operated devices. Micropipelined design has many advantages over the synchronously pipelined design in terms of area, latency and power consumption.

FFT processor does lots of complex additions and multiplications. We remarks that first two stages of FFT computation have all the trivial multiplications where one or both of the multiplicand and multiplier are 0, 1 or -1. We have proposed a low latency multiplier, which is disable a power hungry multiplier in trivial case and directly compute multiplier’s output. Our proposed control circuit for handshaking do not require bounded delay element, so it helps to improve latency and area of the micropipelined FFT processor. Our designed 8-point micropipelined FFT processor computes all stages in 15 clock cycles and having 273.5 MHz maximum clock frequency.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>viii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>ix</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>x</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Related Works</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Key Contribution</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Organization of Thesis</td>
<td>3</td>
</tr>
<tr>
<td>2 FFT ALGORITHMS</td>
<td>4</td>
</tr>
<tr>
<td>2.1 The DFT Algorithm</td>
<td>4</td>
</tr>
<tr>
<td>2.2 FFT Algorithm</td>
<td>5</td>
</tr>
<tr>
<td>2.2.1 Divide and Conquer FFT Algorithm</td>
<td>5</td>
</tr>
<tr>
<td>2.2.2 Twiddle Factor</td>
<td>5</td>
</tr>
<tr>
<td>2.3 Radix-2 Butterflies</td>
<td>6</td>
</tr>
<tr>
<td>2.4 Radix-2 DIT FFT Algorithm</td>
<td>7</td>
</tr>
<tr>
<td>3 ARCHITECTURE OF FFT</td>
<td>9</td>
</tr>
<tr>
<td>3.1 General Architecture of FFT</td>
<td>9</td>
</tr>
<tr>
<td>3.1.1 Input Buffer Block</td>
<td>9</td>
</tr>
<tr>
<td>3.1.2 Output Buffer Block</td>
<td>10</td>
</tr>
<tr>
<td>3.1.3 FFT Computation Block</td>
<td>11</td>
</tr>
<tr>
<td>3.1.3.1 Architecture of Radix-2 Butterfly</td>
<td>11</td>
</tr>
<tr>
<td>3.1.3.2 Iterative Modified Booth Multiplier</td>
<td>12</td>
</tr>
<tr>
<td>3.2 Pipelined FFT Architecture</td>
<td>14</td>
</tr>
<tr>
<td>3.2.1 Synchronously Pipelined FFT Architecture</td>
<td>14</td>
</tr>
<tr>
<td>3.2.2 Asynchronously Pipelined FFT Architecture</td>
<td>15</td>
</tr>
<tr>
<td>3.2.3 Comparison of Synchronous and Asynchronous Design</td>
<td>15</td>
</tr>
</tbody>
</table>
3.3 GALS FFT Architecture ..................................................................................16

4  HANDSHAKING PROTOCOLS AND ELEMENTS FOR ASYNCHRONOUS DESIGN .................................................................19

4.1 Asynchronous Handshaking Protocol ..............................................................19
  4.1.1 Two-Phase Hand-Shaking Protocol .........................................................19
  4.1.2 Four-Phase Protocol Handshaking Protocol ..........................................20

4.2 Elements of Asynchronous Circuit Design ......................................................21
  4.2.1 Muller C-Element ...................................................................................21
  4.2.2 Sutherland’s Capture and Pass Latch .....................................................22
  4.2.3 Paul Day’s Transparent Latch ...............................................................23

4.3 Sutherland’s 2-Phase Micropipeline ...............................................................25

4.4 Four-Phase Asynchronous Design .................................................................27

5  PROPOSED FFT ARCHITECTURE ..................................................................28

5.1 Need for the New FFT Architecture ..............................................................28

5.2 Proposed FFT Architecture 1 ...........................................................................28

5.3 Proposed FFT Architecture 2 ...........................................................................30
  5.3.1 Proposed Control Circuit for Handshaking .............................................30
  5.3.2 Proposed Architecture of the Iterative Booth Multiplier .......................31

6  SIMULATION AND SYNTHESIS RESULTS ......................................................33

6.1 Simulation Results ...........................................................................................33

6.2 Synthesis Results ............................................................................................33

7  CONCLUSION AND FUTURE WORK .............................................................36

7.1 Conclusion .......................................................................................................36

7.2 Future Work ....................................................................................................37

REFERENCES .........................................................................................................38

APPENDIX

VHDL CODE AVAILABLE ON CD-ROM
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2.1. Twiddle Factor for 8-point FFT Computation</td>
<td>6</td>
</tr>
<tr>
<td>Table 3.1. Booth Recoding Strategy</td>
<td>14</td>
</tr>
<tr>
<td>Table 4.1. Truth Table for Muller C-Element</td>
<td>22</td>
</tr>
<tr>
<td>Table 6.1. Performance Parameters of Proposed-1 FFT Architecture</td>
<td>34</td>
</tr>
<tr>
<td>Table 6.2. Performance Parameter of Proposed-2 FFT Architecture</td>
<td>34</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

PAGE

Figure 2.1. Data-flow graph of DIT radix-2 butterfly..............................................................6
Figure 2.2. Data-flow graph of 8-point FFT .............................................................................8
Figure 3.1. FFT block diagram .................................................................................................9
Figure 3.2. Pin diagram of input buffer block ........................................................................10
Figure 3.3. Pin diagram of output buffer ................................................................................10
Figure 3.4. Pin diagram of FFT computation block .................................................................11
Figure 3.5. Block diagram of radix-2 butterfly .......................................................................12
Figure 3.6. Strategy for finding booth record digit .................................................................13
Figure 3.7. Iterative modified booth multiplier ....................................................................13
Figure 3.8. Synchronously pipelined FFT architecture ..........................................................14
Figure 3.9. Asynchronously pipeline FFT architecture ..........................................................15
Figure 3.10. GALS FFT architecture ....................................................................................17
Figure 4.1. Two-phase handshaking ......................................................................................19
Figure 4.2. Four-phase handshaking protocol .......................................................................20
Figure 4.3. Logical and dynamics representation of muller C-element .................................21
Figure 4.4. Sutherland’s capture-pass data storage element ...............................................22
Figure 4.5. CMOS capture-pass implementation .................................................................23
Figure 4.6. Data storage element for four-phase protocol ....................................................24
Figure 4.7. Paul Day’s pass transistor transparent latch .......................................................24
Figure 4.8. Two-phase micropipeline without data processing .............................................25
Figure 4.9. Micro-pipeline structure with data processing ..................................................26
Figure 4.10. Four-phase asynchronous FIFO structure .......................................................27
Figure 5.1. Asynchronous radix-2 butterfly architecture ......................................................29
Figure 5.2. Radix-2 butterfly with proposed control circuit for handshaking ......................30
Figure 5.3. Proposed architecture of booth multiplier .........................................................31
ACKNOWLEDGEMENTS

I would like to express the deepest appreciation to Dr. Premanand Chandramani and Dr. Santosh Nagaraj who have attitude and the substance of genius, they continually enlighten me not only in term of thesis subject but also in terms of pragmatic knowledge. Without their guidance and persistent help this thesis would not have been possible.

I would also like to thank my committee members Dr. Sridhar Seshagiri and Dr. Samuel Kassegne for their diligent effort.
CHAPTER 1

INTRODUCTION

FFT processor is the most important block of digital signal processing and it involved in wide range of applications like broad band’s systems, digital TV, in areas like medical electronics, radar and real time systems. FFT processor does lots of complex computations, so it needs better approach while designing.

Power, performance and area are the main factors in CMOS processor design. The clock signal is a major source of power consumption in synchronous processor design. Study shows that clock network consume up to 30% of the total power in synchronous processor design. Power consumption in CMOS design is directly proportional to the operating frequency. So, as the speed of clock increasing in synchronous designs, the power consumption is also increases. The higher power consumption due to clock network in synchronous design leads to design of asynchronous (micropipelined) FFT processor. In asynchronous design, blocks are activated while they are in use, so it reduces power consumption [1]. The concept of designing a fully asynchronous design is not practical because its increase a design complexity and requires extra hardware. These drawbacks of asynchronous design can be overcome by designing a globally asynchronous and locally synchronous design (GALS) FFT processor in which the block are locally working synchronously and globally asynchronously. The GALS designs have advantages of both the synchronous and asynchronous design.

Another major factor of CMOS design is the performance, which relies on the operating speed. The delay involved in the circuit should be as low as possible for the high performance integrated circuits. Asynchronous FFT processor works on higher speed as the synchronous FFT processor. The operating speed of the asynchronous design is based on critical delay of each block rather than critical delay of all blocks. Asynchronous design occupy little bit more area than the synchronous design, but it significantly improves the performance and power consumption of the FFT processor.
1.1 RELATED WORKS

The research group Yong Li, Zhiying Wang, JianRuan, and Kui Dai has implemented a low-power Globally Synchronous Locally Asynchronous (GSLA) FFT processor [2]. Their GSLA FFT was implemented in 0.18µm 1P6M CMOS standard cell ASIC technology. They have tested a GSLA FFT for power, performance and area requirement and achieve a low power FFT design at the cost of larger area requirement.

Another research group R. Seshasayanan, S. K. Srivatsa and V. Sugavaneswaran implemented a Multi-Clock Domain (MCD) Globally Asynchronous and Locally Synchronous (GALS) FFT processor [3]. Their designed 8-point FFT has been simulated using Active-HDL and synthesized using Altera APEX20KE technology. Our work is related to their work done but we have further divided the locally synchronous part of the GALS FFT processor in GSLA design. Their GALS FFT processor used synchronous FFT computational block which is a heart of the processor and does lots of complex computation. We have noticed that if we divide this part in GSLA design, we can improve FFT processor’s performance.

1.2 KEY CONTRIBUTION

Lots of research has been done to improve the speed of the FFT processor. GALS FFT design has a many advantages over the synchronous and fully asynchronous FFT designs in terms of area and speed. So we have used the GALS architecture to improve a speed of the FFT processor.

We have designed 8-point GALS FFT processor. While designing we have remark that we can improve design’s latency by further dividing locally synchronous parts of the GALS FFT processor in GSLA pipelined architecture. We have proposed a new FFT architecture which used GSLA pipelined structure for locally synchronous part of GALS design and we get significant results for the delay and latency.

FFT algorithm does lots of complex multiplications and additions. Multipliers are the major source of power consumption and introduce more delay. For the 8-point FFT, we remarks that first two stage of computation have all the trivial multiplication, where one or both of the multiplicand and multiplier are 0, 1 or -1. For trivial case, we have disabled
power hungry multipliers and directly counted output in one clock cycle. It requires little bit of more area but it improves latency and reduces the power consumption.

Available handshaking protocols for the asynchronous design requires buffer in request line to delay the request signal, which is equal to the critical path delay of the combinational logic between two pipelined stages. This delay element occupies more area and limits the latency of the design, when delay of the combinational block between two pipelined stages is less than the critical delay. We have proposed control circuit for handshaking in asynchronous design, which works on operating speed of the combinational logic rather than the critical path delay.

1.3 Organization of Thesis

In chapter 1, brief introduction and key contribution of the thesis are presented. Chapter 2, FFT algorithm discusses various methods for deriving FFT algorithm and its applications. It also presents radix-2 FFT algorithm which we have used to compute FFT.

Chapter 3 presents various available architecture to implement FFT processor’s hardware like general architecture, synchronously and asynchronously pipelined architecture, globally asynchronous and locally synchronous (GALS) pipelined architecture. Comparison of different architectures and why the GALS architecture is better than others are discussed. Architecture of sub-part of FFT like radix-2 butterfly, input and output buffer is also discussed.

Chapter 4 covers two-phase and four phase protocols, which are used by asynchronously pipeline design to communicate. It also discusses the elements involved in asynchronous control circuit like muller-C and memory element. Need for the new FFT architecture and proposed FFT architecture is discussed in chapter 5. Simulation and synthesis results of proposed FFT architectures are listed in chapter 6. Chapter 7 summarizes the thesis work and listed the future works that can be done to improve the proposed FFT architectures.
CHAPTER 2

FFT ALGORITHMS

Fast Fourier transfer (FFT) is a widely used basic block of DSP processor. The FFT is one of the most popular methods for calculating a Discrete Fourier transform (DFT). There are several other methods to calculate the DFT but FFT is incredibly more efficient and fast method. It reduces the computation time by hundreds. This chapter covers the DFT algorithm and method of computing DFT by FFT algorithm.

2.1 THE DFT ALGORITHM

The discrete fourier transform is most popular DSP algorithm. It can be used to implement the linear convolution of the two sequences in digital filtering operation. It is also used for spectral analysis of the signals.

The N-point DFT, X (k) for the sequence x (n) of length-N can be defined as

\[ X(k) = \sum_{n=0}^{N-1} x(n). W_N^{nk}, k \in [0, N - 1] \]  

(2.1)

Where

\[ W_N = e^{-j2\pi/N} \] is the Nth root of unity.

The inverse of discrete Fourier transform (IDFT) of the sequence x (k) can be defined as

\[ x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k). W_N^{-nk}, n \in [0, N - 1] \]  

(2.2)

As shown in the equation (2.1), to calculate each sample of DFT requires N-1 complex addition and N complex multiplication. So, computation of the N-point DFT sequence requires (N-1) N complex addition and N^2 complex multiplication. The total number of computation requires to calculate N-point DFT is rapidly increasing as the length of sequence (N) increases. N-point FFT algorithm only requires N log2 (N) computation and it is highly preferable in applications that required to compute all DFT samples [4].
2.2 FFT ALGORITHM

The FFT algorithm is used to calculate DFT with less number of computations. The communication systems, which are use an FFT algorithm that are also requires inverse FFT algorithm. The DFT and IDFT of the sequence can be compute using the same FFT hardware. To compute IFFT using same FFT hardware we just have to swap the real and imaginary part of the inputs and outputs. FFT algorithm basically decompose the N-point DFT computation into smaller size DFT computation and takes the advantage of the periodicity and symmetry of the complex number $W_N^k$. There are various algorithms available for the FFT computation but basically two algorithm are in use that are Decimation-in-time (DIT) FFT algorithm and Decimation-in-frequency (DFT) FFT algorithm. We have used Decimation-in-time algorithm for FFT computation.

2.2.1 Divide and Conquer FFT Algorithm

Decimation-in-time (DIT) algorithm spilt the time domain sequence in even and odd parts. The DFT of sequence $x(n)$ can be given as

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^kn$$

where $W_N^k = e^{-j2\pi/N}$ (2.3)

Now divide N-point DFT into two small N/2-point DFTs. For that we decimate sequence $x(n)$ into odd and even sequences as below

$$X(k) = \sum_{n=0}^{N/2-1} x(2n)W_N^kn + \sum_{n=0}^{N/2-1} x(2n+1)W_N^{(2n+1)k}$$

$$= \sum_{n=0}^{N/2-1} x(2n)W_N^{kn} + \sum_{n=0}^{N/2-1} x(2n+1)W_N^{nk}$$

$$= X_e(k) + W_N^kX_o(k), \quad k = 0, ..., N - 1$$ (2.6)

In equation (2.6), $X_e(k)$ and $X_o(k)$ are the N/2–point DFTs which are even and odd time sequences respectively. Use of the periodicity in $W_N$ is the most important step in deriving an FFT algorithm. The above process shows the general method for deriving divide and conquers FFT algorithms. This re-expression of the N-point DFT into N/2-point DFTs is also called Danielson-Lanczons lemma because it first noted by this two author.

2.2.2 Twiddle Factor

In FFT algorithm, a twiddle factor is a trigonometric constant coefficient that is multiplied by the data. It is also referred as root-of-unity and used to recursively combine
smaller DFTs in Cooley-Turkey FFT algorithm. The Equation of the twiddle factor is expressed in equation (2.7).

\[(2.7)\]

We have used MATLAB to calculate the twiddle factor of the N-point FFT. Table 2.1 listed a twiddle factor that required for calculating 8-point FFT.

<table>
<thead>
<tr>
<th>TWIDDLE FACTOR</th>
<th>DECIMAL VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 + 0 \text{i})</td>
<td></td>
</tr>
<tr>
<td>(0.70 - 0.70 \text{i})</td>
<td></td>
</tr>
<tr>
<td>(0 - 1 \text{i})</td>
<td></td>
</tr>
<tr>
<td>(-0.70 - 0.70 \text{i})</td>
<td></td>
</tr>
</tbody>
</table>

### 2.3 Radix-2 Butterflies

A butterfly is a part of FFT computation that combines the results of smaller DFTs into the larger size DFT, or vice versa. The radix-2 butterfly is a DFT of length \(N=2\) that takes two inputs \(x_0\) and \(x_I\) and computes two output \(y_0\) and \(y_I\). The shape of the data flow graph for the 2-point DFT is shown in Figure 2.1 is like butterfly, so it is named radix-2 butterfly. Cooley-Turkey’s FFT algorithm is use butterfly to break down a DFT of composite size \(n = r \times m\) into \(r\) smaller transform of \(m\) size, where \(r\) is “radix” of the transform. These smaller DFTs are then combined with \(r\)-size butterflies which are pre-multiplied by twiddle factor (root of unity).

DIT FFT algorithm is pre-multiply twiddle factor to radix-2 butterfly while DIF FFT algorithm is post-multiply to radix-2 butterfly. Now, by substituting the value of N=2 in equation (2.3), we can derive the equation for the 2-point DFT as below:

\[ X(k) = \sum_{n=0}^{1} x(n)W_2^{nk}, k = 0,1 \] (2.8)

\[ = x(0)W_2^0 + x(1)W_2^1 \] (2.9)

\[ = \begin{cases} x(0) & k = 0 \\ x(0) + X(1), & k = 1 \end{cases} \] (2.10)

### 2.4 Radix-2 DIT FFT Algorithm

For N-point DFTs, the radix-2 algorithm is a special case of common factor algorithm, where N is power of 2. The indices \(n\) and \(k\) in the equation (2.4) can be represented as below to derive a radix-2 algorithm

\[ n = 2^{a-1}n_{a-1} + 2^{a-2}n_{a-2} + ... + n_0 = \sum_{\beta=0}^{a-1}2^\beta \cdot n_\beta \] (2.11)

\[ k = 2^{a-1}k_{a-1} + 2^{a-2}k_{a-2} + ... + k_0 = \sum_{\beta=0}^{a-1}2^\beta \cdot k_\beta \] (2.12)

Where \(n, k = 0,1\) and \(i = 0,1,...,\alpha - 1\)

\[ N = 2^\alpha , \alpha \in N \]

\[ (k_{a-1}, k_{a-2}, ..., k_0) = \sum_{n_0=0}^{1} \sum_{n_1=0}^{1} \sum_{n_0=0}^{1} x(n_{a-1}, n_{a-2}, ..., n_0).W_N^{(\sum_{\beta=0}^{a-1}2^\beta \cdot n_\beta)(\sum_{\beta=0}^{a-1}2^\beta \cdot k_\beta)} \] (2.14)

The last term is defined as

\[ W_N^{\left(\sum_{\beta=0}^{a-1}2^\beta \cdot n_\beta\right)\left(\sum_{\beta=0}^{a-1}2^\beta \cdot k_\beta\right)} = W_N^{2^{a-1}k_{a-1}\sum_{\beta=0}^{a-1}2^\beta \cdot n_\beta} \cdot W_N^{2^{a-2}k_{a-2}\sum_{\beta=0}^{a-1}2^\beta \cdot n_\beta} \cdots W_N^{k_0\sum_{\beta=0}^{a-1}2^\beta \cdot n_\beta} \] (2.15)

Now we can write \(W_N^{-N} = (e^{-j2\pi/N})^N = 1\)

Now substituting this value in equation (2.15)

\[ G_0 = W_N^{2^{a-1}k_{a-1}\sum_{\beta=0}^{a-1}2^\beta \cdot n_\beta} = W_N^{2^{a-1}k_{a-1}\sum_{\beta=0}^{a-1}2^\beta \cdot n_\beta} = W_N^{2^{a-1}k_{a-1}n_0} \]

\[ G_1 = G_0 = W_N^{2^{a-2}k_{a-2}\sum_{\beta=0}^{a-1}2^\beta \cdot n_\beta} = W_N^{2^{a-2}k_{a-2}\sum_{\beta=0}^{a-1}2^\beta \cdot n_\beta} \]

\[ \ldots \ldots \ldots \]

\[ G_{\alpha-1} = W_N^{k_0\sum_{\beta=0}^{a-1}2^\beta \cdot n_\beta} \] (2.16)

Substituting the value of equation (2.16) in equation (2.14)
This summation can be divided into sequential summation as below

To derive the FFT in final stage, reorder the output data in natural order and it is done by bit-reversing

This algorithm is reduced the computational complexity by $O(N \log_2 N)$ butterfly operations and the computation has also been divided into $\log_2 N$ different steps, which is an advantage of considering pipelining in hardware. The dataflow graph of 8-point FFT is depicted in Figure 2.2.

CHAPTER 3

ARCHITECTURE OF FFT

This chapter discusses FFT architectures. There are various architectures available to implement FFT, but pipelined architecture is the most widely used. It also discusses the pros and cons of synchronously and asynchronously pipelined architecture and introduces Globally Asynchronous and Locally Synchronous (GALS) FFT architecture.

3.1 GENERAL ARCHITECTURE OF FFT

The simple architecture of FFT computation is shown in Figure 3.1. It does mainly consist of three blocks: input buffer block, FFT computation block and output buffer block.

![Figure 3.1. FFT block diagram.](image)

3.1.1 Input Buffer Block

Inputs to the FFT processor are given by this block. We have designed 8-point FFT processor. It has eight complex inputs. We have used 24-bit to represent each real and imaginary part of the inputs. If all the 8-points input are provided simultaneously, the processor must need 8 x 24 x 2 = 384 pins to load all inputs, which is not practically feasible. So, it used serial-input and parallel input buffer, which takes new 24-bit input data at every clock cycle and it takes 8 clock cycles to load all inputs. The input buffer block consists of following pins: 24-bit data lines, 3-bit selection lines, clock signal, write enable signal, buffer full signal and 2x8 x 24 output lines. Figure 3.2 shows an internal structure of input block. Buffer stores the new data when write_en signal is on. Select line is used to select the address in buffer where data is going to be store and initially it is “000”.

...
3.1.2 Output Buffer Block

Output block is basically parallel in and serial out buffer, which takes all the input at same time and generate the one output at every clock cycle. This block consist of following pins: (8, 24) input lines, write enable and read enable pin, (1,24) output line, 3 selection lines. When the output from the FFT computational block is ready, it will make write enable line of the output buffer block to goes high and store all the data in the output buffer. When read_en is high, output buffer will send out one output at every clock cycle. For the 8-point FFT, output buffer takes 8-clock cycles to send out the outputs. Figure 3.3 depicted the pin diagram of the output block.
3.1.3 FFT Computation Block

FFT computational block is the heart of FFT processor. This block receive its input from the input buffer block and send its outputs to the output buffer block as inputs. It has output_ready, 8-inputs and 8-outputs pin and each pin is a 24-bit bus. Output_ready signal is high, when FFT computational block produce new outputs and it enable output buffer for write operation. This block is consisting of four butterflies which are working in parallel. By reusing the hardware, all the 3 stages are computing only by four butterflies. Finite state machine (FSM) is used to keep track for the computing stage of FFT. In the final state, it will produce the outputs. The internal structure of the FFT computation block is illustrated in Figure 3.4.

![Figure 3.4. Pin diagram of FFT computation block.](image)

3.1.3.1 ARCHITECTURE OF RADIX-2 BUTTERFLY

The FFT computation block is uses radix-2 butterfly as a basic block to compute the FFT. It has two inputs and two outputs. The equation for radix-2 butterfly is

\[
A \rightarrow B \rightarrow A - B \\
\text{Where } A \text{ and } B \text{ are the complex numbers. For actual hardware representation, equation (3.1)}
\]

and equation (3.2) are break down into real and imaginary components [5].

\[
(3.1) \\
(3.2) \\
(3.3) \\
(3.4)
\]
Architecture of radix-2 butterfly is shown in Figure 3.5. It used one 39 bits modified booth multiplier to multiply input (of 24-bits) and twiddle factor (16-bits) and two carry look ahead adder.

![Block diagram of radix-2 butterfly.](image)

### 3.1.3.2 Iterative Modified Booth Multiplier

The booth multiplication is a technique that is faster and requires smaller circuit by recoding the numbers that are multiplied. It is a standard technique, which provides significant improvements over the long multiplication technique. We have used radix-4 booth recording technique. This technique will reduce the number of partial products required to calculate multiplier’s output. So it helps to improve power consumption and reduces the propagation delay.

The Booth recode is a multiplier term for which we consider the bits in blocks of three, such that each block overlaps the previous block by one bit as shown in Figure 3.6. Grouping of the bits for booth record digit is starting from the LSB. For the first block we only use two bits of the multiplier since there is no previous block to overlap.
The architecture of the booth multiplier is shown in Figure 3.7. To multiply 24-bits multiplicand with 16-bit multiplier, it will take 8 clock cycles. The booth digit selection unit records the booth digit and selects the partial product. Adder will add the partial product from the booth selection unit to the previous partial product and shifter will shift right the partial product by 2-bits. Possible booth record digit and corresponding partial product is listed in Table 3.1.
Table 3.1. Booth Recoding Strategy

<table>
<thead>
<tr>
<th>Booth Record</th>
<th>Partial Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1* Multiplicand</td>
</tr>
<tr>
<td>010</td>
<td>1* Multiplicand</td>
</tr>
<tr>
<td>011</td>
<td>2* Multiplicand</td>
</tr>
<tr>
<td>100</td>
<td>-2* Multiplicand</td>
</tr>
<tr>
<td>101</td>
<td>-1* Multiplicand</td>
</tr>
<tr>
<td>110</td>
<td>-1* Multiplicand</td>
</tr>
<tr>
<td>111</td>
<td>0</td>
</tr>
</tbody>
</table>

3.2 PIPELINED FFT ARCHITECTURE

Pipelining is an approach of inserting a register between two combinational logics, which are clocked synchronously or asynchronously. Pipelining improve a processor’s throughput. Synchronous and asynchronous pipelining are the two main types of pipelining.

3.2.1 Synchronously Pipelined FFT Architecture

In synchronous pipelining register is inserted between two stages which are clocked synchronously. Figure 3.8 shows synchronously pipelined FFT architecture. At the rising edge of the clock signal register becomes transparent and transfer the input to the output.

Figure 3.8. Synchronously pipelined FFT architecture.
The clock period should be equal or more than the critical delay of the all blocks. Suppose, for the architecture shown in Figure 3.8, if input buffer, FFT computational block and output buffer block have a critical delay of 10ns, 20 ns and 10ns respectively then clock period should be equal or more than 20ns. So, it will limit the maximum operating frequency of the designs.

3.2.2 Asynchronously Pipelined FFT Architecture

In asynchronous pipelining, register is inserted between two combinational blocks which is clocked asynchronously. Registers uses a handshaking protocol for synchronization. Figure 3.9 shows asynchronously pipelined FFT architecture. When the register have output ready, it will send request signal to second register. When second register gets the data from the first register it will send a acknowledge signal to first register. The operating speed of the design depends upon the critical path delay of each combination delay.

![Figure 3.9. Asynchronously pipeline FFT architecture.](image)

3.2.3 Comparison of Synchronous and Asynchronous Design

Asynchronous design offer following advantages over synchronous design [6].

1. Avoidance of clock-skew: In Synchronous circuit designs, the clock skew is a phenomenon in which the clock signal arrives at different time at different component on the same chip due to many causes like wire-interconnect length, capacitive coupling, material imperfections, temperature variations and the difference in input capacitance at the device using clock. Clock skew limits the maximum frequency of the operation. Clock skew problem in synchronous design can be avoided by using
balanced clock distribution network, but it is expensive and requires extensive delay modeling and simulation. Absence of the global clock in asynchronous design avoids clock skew problem.

2. Low power: Power consumption can be reduced by asynchronous circuit, which avoids two significant problem of the synchronous design. The one problem in synchronous design is that all parts are clocked, even if they are not performing a useful function. And second problem is the clock signal, which is a heavy load itself, required larger drivers and major amount of power to drive it. These problems can be avoided by using clock-gating but increase design complexity and requires special attention. Asynchronous design can avoid these problems without extra efforts and complexity.

3. Modularity: The performance of the synchronous design can be modified only by increasing the global clock frequency for that requires reimplementation of the design required. In contrast, the performance of the asynchronous design can be improved by changing an active part of the circuit, which is the part that affects the constraint of communication protocol.

4. Better than worst-case performance: The operating speed of the synchronous design is fixed. The clock period should minimum to accommodate all the worst case variations. Typically, all the worst-case conditions do not coincide at the same time and do not encounter very frequently. In contrast, operating speed of the asynchronous design is not fixed and it depends on the delay of the each combinational block, which gives better performance than the synchronous design.

However there are also some disadvantages of asynchronous design listed below [6]:

1. Complexity: Asynchronous design is use handshaking protocol for the synchronization between two blocks which require extra hardware and increase design complexity.

2. Deadlock: Greater attention is required to design an asynchronous design while avoiding deadlock and hazards in the circuit [7].

3. Lack of Resources : Most of the today’s CAD tools specially tools for testing and test vector generation are not suitable for the asynchronous design

### 3.3 GALS FFT Architecture

The advantages of pipelined architecture are small area, high data throughput and a simple control unit. If the design uses synchronous operation, delay is added between each radix-2 stage and between the two blocks inside the radix-2 butterfly. It will increase latency between input and output frames. The big problem is that the extra hardware will increase the die size and the power consumption. The above problem can be solved in three ways. One way is to change the control unit, second way is to design fully asynchronous structure and the third way is to create a globally asynchronous and locally synchronous (GALS) FFT
architecture. The first option of keeping the FFT architecture completely synchronous would increase the latency of the design. The second option have more complex control system and resulting in a system that is harder to understand. The third option would only have a slightly different control structure then synchronous structure but it improve the design latency and reduce the power consumption. So this pros and cons of the GALS structure inspire to design GALS FFT.

In the Figure 3.10 shown the architecture of GALS FFT which used in [3]. The input and output buffer blocks are working at clk1 frequency and FFT computation block is working at clk2 frequency. For communication between two different clock domain systems we need asynchronous handshaking protocol.

![Figure 3.10. GALS FFT architecture.](image)

The data in FFT design is always transfers in a single direction. The data always flows from the input buffer block to the FFT computation blocks and then from the FFT computation block to the output block. So, the input buffer block is designed with the single pair of request and acknowledgment signals. The FFT computation block has to communicate with input buffer block as well as the output buffer block. Hence the FFT computation block is designed with two sets of handshaking signals (request and acknowledge signals). The output buffer block communicates only with the FFT computation block and the data transfer is only one way, so output buffer block is designed with a single pair of the handshaking signals. The input buffer block, raised Req1 signal when output is
ready from input buffer block and the FFT computational block responds it by raising the Ack1 signal and samples the data from the data bus. The Input block then turns low the Req1 signal and the FFT computational block follows this by turning low the Ack1 signal. The same process applies to the communication between the FFT computation block and the output buffer block.
CHAPTER 4

HANDSHAKING PROTOCOLS AND ELEMENTS FOR ASYNCHRONOUS DESIGN

In asynchronous designs, registers use handshaking protocol to communicate with each other. This chapter discusses handshaking protocols and their control circuit. It also present brief introduction of elements that are used in design of control circuits.

4.1 ASYNCHRONOUS HANDSHAKING PROTOCOL

Different types of protocol available for the handshaking in asynchronous design. Basically we can categorize them in two types: 2-phase protocol and 4-phase protocol. Control circuits of hand-shaking protocols are basically containing two wires: request line and acknowledge line for handshaking. All the hand-shaking protocols have some pros and cons, depending upon design specification they are used in asynchronous circuit design.

4.1.1 Two-Phase Hand-Shaking Protocol

Behavior of a two-phase handshaking protocol is illustrated in Figure 4.1. When the new data is ready by the sender to transmit, control circuit issues a transition on the request line. When receiver accepts the data, control circuit issues a transition on acknowledge line.

Two-phase protocol is also called a transition signaling protocol, because during transfer of each data from one block to another block control circuit makes transition in control circuit, so signal level has no significance [7]. Below stated, how the control signals change step-by-step.

1. Sender block place a data on bus then set Request=1
2. Receive gets the data from bus and then set Acknowledge=1
3. Sender puts a another data on bus and then set Request=0
4. Receiver gets the data from bus and then set Acknowledge=0

**4.1.2 Four-Phase Protocol Handshaking Protocol**

The signal transition graph of four phase handshaking protocol during the data transfer from sender to receiver is depicted in Figure 4.2. When the sender block place a new data on the data transmitting bus, the control circuit issues a low to high transition. After then when receiver block gets a new data from data bus, the control circuit issues a low to high transition on acknowledge line. Once a sender gets acknowledge then control circuit issues high to low transition on request line, which makes acknowledge lines to go low. This protocol is also called level signaling protocol because transmission of each data from sender to receiver block is depend upon the level of control signals.


How the control signal changes during transmission of data from sender block to receiver block is mentioned below:

1. The sender block place the data on bus and then control circuit set Request=1
2. The receiver block gets the data from bus and then control circuit set Acknowledge=1
3. Control circuit set request=0
4. Control circuit set request=0

For the each new data transfer control signal has to go zero which consume more power then 2- phase protocol but it use a level-sensitive latch so require less area then 2-phase protocol.

4.2 ELEMENTS OF ASYNCHRONOUS CIRCUIT DESIGN

Design of an asynchronous circuit involves two basic elements that are muller C-element for state holding and latch for data holding.

4.2.1 Muller C-Element

Design of asynchronous circuits involve a Muller C-element for state-holding. It is act as the AND element for the events. Muller C-element generates an output signal whenever both the input signals are changes. When both inputs are 0, the output of the muller element is set to 0 and when both the inputs are 1, the output is set to 1. A logical and dynamic implementation of the Muller C-element in CMOS is illustrated in Figure 4.3. The electrical capacitance of an internal mode is use as the storage element in Muller C-gate [8].

![Logical and dynamics representation of muller C-element.](image)

From the Table 4.1 we can say that if the two inputs of the Muller C-element are different then there will be no changes in the output and it hold a previous value. Each input of the Muller C-element must receive an event before an event is transmitting to its output.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>HOLD</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>HOLD</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### 4.2.2 Sutherland’s Capture and Pass Latch

Sutherland’s approach for 2-phase handshaking protocol design describes the use of a capture-pass latch as a data storage element. Figure 4.4 depicted a basic structure of a capture-pass latch. The capture-pass latch is remaining transparent until an event occurs on its Capture line (C). This will cause the latch to hold any data on its input line. When the capture operation is done, it will generate an event on Capture done (Cd). Now Dout represents new captured data, any change on Din will not change Dout. An event on Pass signal (P) will makes a latch transparent and ready to capture a next data value. The event Pass Done (Pd) signals the completion of the pass operation [8].

The behavior of an event controlled storage element is easy to describe. Assume that the event-controlled storage element is initially transparent. The capture and pass control signal events are always alternate. An event on capture control line flips the two switches to which the capture line is connected and causes the storage element to capture and hold the new data value then passing through it. This event isolates the output value of the capture pass element from changes as the element’s input changes. A following event on the pass control wire flips the other switch, and makes the element to work in transparent state and pass the next data value to output. A new output value appears after each event on the control wires of the capture pass element [8].

Chain of Capture-pass latch structures can be used to form a FIFO or micro pipeline design with the use of the Muller C-element to ensure correct operation of handshaking control or bundled data protocol. Sutherland’s capture pass latch structure is shown in Figure 4.5. It has four transmission gates compared to the single transmission gate in the Paul day’s latch.


**4.2.3 Paul Day’s Transparent Latch**

The latch structure used in 4-phase handshaking protocol is shown in Figure 4.6. The control signal Lt is used to set latch in transparent mode or capture mode. Logical high value
of $L_t$ makes latch transparent and it latch the new data value. The control signal $L_{td}$ is used to detect the completion of latch process.

Logical diagram of Paul Day’s transparent latch is shown in Figure 4.7. This latch requires 4-phase protocol as compared to the capture-pass latch approach, which requires a 2-phase protocol. The control signal $E_n$ and $nE_n$ are used to enable and disable the latch. Sutherland’s capture pass latch has four times total gate capacitance loading on $C,P,nC$ and $nP$ lines as compared to total gate capacitance loading on the $E_n$ and $nE_n$ lines of the Paul Day’s latch [9]. The Paul Day’s latch offers significant energy saving by switching half the capacitive load on its control wires for each cycle as compared with the Sutherland’s capture-pass latch.

![Figure 4.6. Data storage element for four-phase protocol.](image)

![Figure 4.7. Paul Day’s pass transistor transparent latch.](image)

4.3 Sutherland’s 2-Phase Micropipeline

Sutherland’s micro-pipelining without data processing is illustrated in Figure 4.8. Capture-pass latches and one-inverted-input C-elements are used to form a micropipeline. Initially all request and acknowledge line is set to logical zero in FIFO structure. When new data is placed on the input of the first register, the control circuit issued event on request line. One of the inputs of the Muller C-elements is inverted, so a logical level transition from zero to one on the input request line will fire the first Muller C-element then the first latch to become opaque and capture a new data value. A delayed event will then appear at the Capture done (Cd) pin of the latch which will send acknowledgment to original request and also fire the next C-element. The second latch responds this action by moving into an opaque state which followed by a delayed event on its Capture done (Cd) line. A delayed event on “Cd” of the second latch will cause event on Pass (P) of the first latch, which turns it in to transparent mode and also at the same time it fires a third Muller C-element. The request control signal continues to propagate down to the micropipeline structure with the data until an event occur on output request line at the opposite end which results in the fourth latch being opaque. Since the first output request was never acknowledged, additional requests on the input request line will eventually fill the FIFO.

To empty the FIFO, events on the output acknowledge line is used. If we place an event (a zero to one transition since this is the first output acknowledgment) on the output acknowledgment line, the last latch will turn in transparent mode and the data from third latch will propagate through this, because delayed event on the Pass done (Pd) pin of the third latch. Then fourth C-element will fire and latch the new data followed by a delayed event on its Capture done (Cd) pin. This delayed event on “Cd” will generates a new output request and becomes an acknowledge signal to the third latch. Then this cycle repeats as the initial output acknowledgment propagates back through the micropipeline and resulting in all of the data stored in the FIFO. A micropipeline structure of any data processing design can be created from the above FIFO by inserting combinatorial logic and associated delay elements between the latches as depicted in Figure 4.9.

![Figure 4.9. Micro-pipeline structure with data processing. Source: Johnson, D., Akella, V., and Stott, B., “Micropipelined asynchronous discrete cosine transform (DCT/IDCT) processor,” IEEE transaction on very large scale (VLSI) system, vol. 6, pp. 731-740, Aug. 1998.](image)

The control signal which declares the validity of the data should arrive after the data has arrived at the receiver for the correct operation of a system. This phenomenon is similar to the set-up time constraint in standard synchronous design. The one method to solve this problem is illustrated in Figure 4.9, which uses “bounded delays” to model the combinational logic stage’s worst-case delay. The difference between bounded-delaying asynchronous design and the constraints used in setting the clock period in a synchronous
system is that in the former case it is worst-case delay of the specific combinational logic block through which the data is passing while in the later one has to consider the worst-case path of the whole design.

4.4 Four-phase Asynchronous Design

A structure of 4-phase asynchronous FIFO is shown in Figure 4.10. Initially all the request and acknowledge line of the FIFO is set at low. When a new data is available at the first register, it issues a first event on the input request line (Rin). The event on Rin will cause Latch (Lt) signal to goes high, it makes first stage opaque and it capture the data Din. At the same time capture signal Lt will set Latch done (Ltd) signal to high, which fire the second stage Muller C-element. Second stage capture signal L set its Ltd to high, so it send acknowledge signal to first stage, which set Lt signal of the first stage to low. This action will cause first stage to become transparent and send the input data value to output line and also it set Ltd of the first stage to low so it reset the input acknowledge pin (Ain). Simultaneously second stage’s high Ltd signal fires the third stage’s Muller C-element.

CHAPTER 5

PROPOSED FFT ARCHITECTURE

This chapter discusses why we need to design new architecture and it introduce our proposed FFT architectures and describe how it better than the other existing architectures.

5.1 NEED FOR THE NEW FFT ARCHITECTURE

Globally asynchronous and locally synchronous (GALS) FFT design takes advantages of both the synchronously and asynchronously pipelined design as discussed in chapter 3. Input buffer, FFT computational block and output buffer blocks are globally works asynchronously and locally synchronously. FFT computation block is the most important block of the FFT, Which does major part of computation. Synchronously pipelined design of FFT computational block limits the speed of FFT processor. Proposed architecture of FFT processor is further divide FFT computation block in the GALS design, which helps to improve design latency.

FFT algorithm does lots of complex multiplications and additions, which limits the operating speed of the design. So we need to optimize multiplier and adder part. Proposed architecture used optimized multiplier which improves latency of FFT processor by taking advantage of trivial multiplication case.

5.2 PROPOSED FFT ARCHITECTURE 1

The proposed FFT architecture 1 is the similar as the GALS FFT architecture except it uses an asynchronous radix-2 butterfly instead of synchronous butterfly. The use of asynchronous butterfly reduces the latency and the power consumption in design. For 48-bit complex inputs and 16-bit twiddle factor synchronous butterfly will use 12 clock cycles to compute the output, while asynchronous butterfly will use 9 clock cycles.

We have used an asynchronous Radix-2 butterfly to improve the latency of the design. Initially all acknowledgment lines are set to zero. When output from the modified booth multiplier is ready, it makes Rin signal high. This will fire the first Muller element so it makes c1 to goes high and send acknowledgment signal Ain to input stage. When c1 is high,
L1 is work as a latch and it latch the new data. As the R1 becomes high, it will fire the
element and m2 will send an acknowledgment signal A1 to first stage, which makes first
latch transparent so it will transfer an output of modified booth multiplier to the input of
carry look ahead adder 1. Simultaneously m2 will fire third Muller element by sending a
request R2. Asynchronous control circuit requires bounded delay element in request line to
delay the request signal by the delay of the combinational logic. Delay should be equal to the
critical path delay of the combinational logic to ensure that the data is available at the input
of next latch before it gets the request signal high. The architecture of the 4- phase
asynchronous radix-2 butterfly is shown in Figure 5.1.

![Figure 5.1. Asynchronous radix-2 butterfly architecture.](image)

Asynchronous radix-2 butterfly use synchronous booth multiplier because
asynchronous booth multiplier requires more area and increase design complexity.
Synchronous Booth multiplier takes 8 cycles to calculate the output for 24-bits multiplicand
and 16-bits multiplier and one clock cycle is needed to perform the addition operations. So,
asynchronous radix-2 butterfly requires 9 clock cycles to perform the whole operation. For
the 8-point FFT, we have three stages of radix-2 butterfly. So, we requires $9 \times 3 = 27$ clock
cycles to compute all the stage and FFT output, where as if we use synchronous radix-2
butterfly it will takes $12 \times 3 = 36$ clock cycles. Use of asynchronous radix-2 butterfly in
proposed FFT architecture improves latency by $36 - 27 = 9$ clock cycles.
5.3 PROPOSED FFT ARCHITECTURE 2

Available control circuit for handshaking requires bounded delay element to delay the request signal by the delay of combination logic as discussed in section 5.1. This bounded delay element increase the design area and limits the design speed when the combinational path delay is less than its critical path delay. In our radix-2 butterfly design, modified booth multiplier has delay less than critical path delay in trivial multiplication case. To improve the speed we have proposed a control circuit for handshaking. The proposed control circuit for handshaking will not operate on a critical delay of the combinational block but actual operating speed of the combinational block.

The proposed modified iterative booth multiplier will generate a multiplier output for trivial multiplication in single clock cycle. Trivial multiplication is special case of the multiplier in which one or both of the multiplicand and multiplier are 0, 1 or -1. We remark that for the 8-point FFT computation first two stages have all the multiplication trivial. So by using a Proposed modified iterative booth multiplier, we can compute the first two stage of the FFT in just four clock cycle.

5.3.1 Proposed Control Circuit for Handshaking

Proposed control circuit is works same as the general structure but in this request signal to next block generated by the combination block rather by Muller element and we do not have to use bounded delay element for the request signal. When the output of the block is ready, it will send a request high signal to next block. In Figure 5.2 shows architecture of the radix-2 butterfly which use proposed handshaking control circuit.

![Figure 5.2. Radix-2 butterfly with proposed control circuit for handshaking.](image)
Request (R) signal is issued by the combinational block when its output is ready instead of muller-c element. Proposed control circuit for handshaking do not use bounded delay element to delay request signal by the critical delay of combinational logic. Combinational logic generate request signal when its output is ready. So the operating speed of proposed control circuit for handshaking is depend upon the operating speed of combinational logic not on its critical delay. Proposed control circuit for handshaking reduces the delay of the design when delay of the combinational logic is less than critical delay.

5.3.2 Proposed Architecture of the Iterative Booth Multiplier

We remark that first two stages of FFT computation have all trivial multiplications (i.e. ×1, ×-1, and ×0 from the coefficients). In the new architecture of the iterative booth multiplier we disable a power hungry multiplier in trivial multiplication case. In case of 24×16 bit trivial multiplication, multiplier generates output in only one clock cycle. So it reduces latency and power consumption of the design. The proposed architecture of the booth multiplier is depicted in Figure 5.3.

![Proposed architecture of booth multiplier.](image)
The proposed multiplier architecture has check logic at the starting, which determine the case of operation weather it is a trivial multiplication case or not. If it is a trivial case multiplication then it disable a power hungry booth multiplier and calculate the output in just one clock cycle.
CHAPTER 6

SIMULATION AND SYNTHESIS RESULTS

This chapter discusses the simulation and synthesis results for the proposed FFT architectures. We have implemented proposed architectures for the 8-point FFT. Code has been written in VHDL language.

6.1 SIMULATION RESULTS

Both the proposed FFT architectures have been simulated in Modelsim and Cadence Nanosim using VHDL test bench. VHDL test bench used input text file to give the 8-point inputs to design and results has also been written in output text file. See Appendix for VHDL test bench code. VHDL test bench generates the design’s output in form of waveform and HEX values which are written in text file. The VHDL coding for these designs has been done in Xilinx ISE 10.1.

Functionality of the designs has been verified by comparing the VHDL results with the MATLAB results for the 8-point FFT. Random input values have used to check the functionality of the designs. Compared the bit by bit results that are obtained from the hardware with the MATLAB outputs and 100% of accuracy is noticed for 8-point FFT.

6.2 SYNTHESIS RESULTS

Synthesis has been done using Leonardo spectrum and results have been analyzed for timing and area. For synthesis, TSMC 0.18um 1.8V technology is used. We have used Leonardo spectrum for synthesis because it gives area in terms of gates while Build Gates tool gives area in terms of unit cell.

We have synthesized the 8-point FFT design and its small part radix-2 butterfly separately to verify how asynchronous radix-2 butterfly help to improve design latency than the synchronous radix-2 butterfly. For the 8-point FFT design, maximum clock frequency is listed for clk1 and clk2 because we have use two separate clock for buffers and FFT computational part.
For the asynchronous radix-2 butterfly, we got 488.3 MHz maximum clock operating frequency and it required 12,463 gates. Asynchronous radix-2 butterfly requires 9 clock cycles to processed the data and produce the outputs. Maximum clock frequency of the clk1 and clk2 for the 8-point FFT is obtained 362.5MHz and 294.3 MHz respectively and it occupied 60,938 gates. The 8-point FFT takes 47 clock cycles to load the inputs, computes and send out the FFT outputs. All the synthesis results for the proposed-1 FFT architecture are listed in Table 6.1.

### Table 6.1. Performance Parameters of Proposed-1 FFT Architecture

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Radix-2 Butterfly</th>
<th>8-point FFT design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. frequency</td>
<td>488.3MHz</td>
<td>Clk1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clk2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>362.5MHz</td>
</tr>
<tr>
<td></td>
<td></td>
<td>294.3 MHz</td>
</tr>
<tr>
<td>Delay</td>
<td>2.04 ns</td>
<td>2.76 ns</td>
</tr>
<tr>
<td></td>
<td>3.39 ns</td>
<td></td>
</tr>
<tr>
<td>Area</td>
<td>12,463</td>
<td>60,938</td>
</tr>
<tr>
<td>Latency</td>
<td>9</td>
<td>45</td>
</tr>
</tbody>
</table>

The performance parameters of the proposed-2 FFT architecture are listed in Table 6.2. For the radix-2 butterfly, we got 391.73 MHz maximum clock operating frequency and it required 12,965 gates. Asynchronous radix-2 butterfly requires 9 clock cycles to process the data in general case and 3 clock cycles in trivial case.

### Table 6.2. Performance Parameter of Proposed-2 FFT Architecture

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Radix-2 Butterfly</th>
<th>8-point FFT design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. frequency</td>
<td>391.7MHz</td>
<td>Clk1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clk2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>362.5MHz</td>
</tr>
<tr>
<td></td>
<td></td>
<td>273.3 MHz</td>
</tr>
<tr>
<td>Delay</td>
<td>2.55 ns</td>
<td>2.76 ns</td>
</tr>
<tr>
<td></td>
<td>3.66 ns</td>
<td></td>
</tr>
<tr>
<td>Area</td>
<td>12,965</td>
<td>63,125</td>
</tr>
<tr>
<td>Latency</td>
<td>Trivial case</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Critical case</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>33</td>
</tr>
</tbody>
</table>
For the proposed-2 FFT architecture, we got maximum clock frequency of the clk1 and clk2 are 362.5MHz and 294.3 MHz respectively and it occupied 63,125 gates. 8-point FFT takes 33 clock cycles to load the inputs, computes and send out the FFT outputs.

From the Table 6.1 and 6.2, we can say that proposed-2 FFT architecture is the low latency FFT architecture. The radix-2 butterfly of the proposed-2 FFT architecture requires 3 clock cycles to compute its output in trivial case whereas radix-2 butterfly of the proposed-1 FFT architecture requires 9 clock cycles. We got this improvement in latency because our proposed-2 FFT architecture used asynchronous radix-2 butterfly with proposed booth multiplier and proposed handshaking control circuits. Extra logic in proposed booth multiplier increased area of proposed-2 FFT design by 3.58% but our latency of the 8-point FFT design is improved by 26.66% as compared to proposed-1 FFT design.
CHAPTER 7

CONCULSION AND FUTURE WORK

As the need for faster and low power digital signal processing product growing, the need of the faster FFT processor increases. A lot of research has been done to design low latency FFT processor, while keeping area reasonable. The synchronous FFT processor has many difficult challenges to overcome in terms of latency, clock skew and power. These challenges have made an asynchronous FFT processor design an increasingly practical alternative. Asynchronous designs have many advantages over synchronous designs that are speed and power consumption. But fully asynchronous design is difficult to design, test and also require more area for designing control circuit. GALS FFT design is a globally asynchronous and locally synchronous pipelined design, which takes advantages of both the synchronously and asynchronously pipelined design.

7.1 CONCLUSION

Our goal is to design a low latency GALS FFT processor which is faster than exiting FFT processor. We have achieved competitive results with our proposed GALS FFT architecture which uses asynchronous butterfly with proposed four-phase handshaking control circuit and proposed iterative modified booth multiplier. Proposed iterative modified booth multiplier can calculate a multiplier output in one clock cycle for trivial case. Delay of available asynchronous control circuit is depends upon a critical delay of a combinational block while delay of proposed 4-phase handshaking control circuit is depend upon operating speed of the combinational logic. Proposed architecture of asynchronous control circuit and multiplier helps to improve a latency of an asynchronous design.

We have implemented 8-point FFT for proposed-1 and proposed-2 FFT architecture. Simulated and synthesized all three designs using Leonardo spectrum for AMI 0.18um technology and compare our results with existing FFT architecture. We got competitive results for our proposed FFT design.

GALS FFT architecture is used synchronous FFT computational block. It used four radix-2 butterflies which are working in parallel. To compute each stage it takes 12 clock
cycles and 8-point FFT has three stages. So, overall latency of the synchronous FFT computational block is $12 \times 3 = 36$ clock cycles.

Our Proposed-1 FFT architecture is further divide FFT computational block in to GALS design. The GSLA FFT computational block in Proposed-1 FFT design requires 12,965 gates and having a delay of 2.04 ns. Each stage of computation takes 9 clock cycles to compute the output. So, overall latency of the GSLA FFT computational block is $9 \times 3 = 27$ clock cycles.

Our Proposed-2 FFT architecture is used the GSLA FFT computational block. We have proposed modified booth multiplier and four-phase handshaking control circuit which help to reduce design latency in trivial case. In 8-point FFT, 45% of multiplications are trivial, with the proposed-2 FFT architecture we had reduce the design latency. Proposed four-phase handshaking protocol avoids the use of delay element in request line, so we can save the area. The GSLA FFT computational block in Proposed-2 FFT design requires 12,463 gates and having a delay of 2.55 ns. So it used % more area as compared to proposed-1 architecture but the latency is significantly improved. Each stage of computation takes 9 clock cycles to compute the output in normal case and 3 clock cycles in trivial case. First two stages of the 8-point FFT have all trivial multiplication. So, overall latency of the GSLA FFT computational block is $3 + 3 + 9 = 15$ clock cycle.

The trade off with proposed FFT architectures is area and the extra resources. The control infrastructure of the asynchronous FFT architectures requires more area than the synchronous FFT architectures. Verification of the designs needs extra efforts as the data is processed asynchronously in the proposed FFT architectures.

### 7.2 Future Work

Future works that can be done to improve the proposed architecture is listed below

1. Optimizing this processor for area is the most important future work.
2. Designing a fully asynchronous FFT processor is another challenging future work.
3. This thesis ends with the FPGA synthesizable code for the GALS FFT design. VLSI layout of the GALS processor can give more accurate results of power consumption and delay.
4. Design for test (DFT) is increasing in importance due to the increasing size of designs. Adding design for test methodology to the project could be interesting future work.
REFERENCES


APPENDIX

VHDL CODE AVAILABLE ON CD-ROM