Four-Point FFT Processor
An Asynchronous/Synchronous Comparison

Radhika Balasubramanian
Anil Kumar Reddy Palaparthi
Arun Tejusve Raghunath Rajan
Trent Rolf

Department of Electrical and Computer Engineering, University of Utah

radhika.balasubramanian@gmail.com anilreddy.syntel@gmail.com
tejus2005@gmail.com trent@rolfed.com

Abstract

Any periodic signal can be represented as a sum of sine and cosine waves of different frequencies. Hence, any periodic signal can be represented as a function of frequency. The representation of the original signal in terms of frequency or transformation of a signal into frequency domain is called Fourier Transform. The transformation of discrete periodic signal into frequency domain is called Discrete Fourier Transform(DFT). The Fast Fourier Transform (FFT) is an efficient algorithm to compute the Discrete Fourier Transform (DFT) and its inverse. It has several applications in the field of signal processing. The traditional butterfly FFT design requires needless computations and data storage which lead to unnecessary power consumption. The architecture used in this project simplifies and pipelines these computations, reducing power and enhancing speed. We implement both an asynchronous and a synchronous version of the 4-point FFT processor as a comparison of speed, power, and area. From this side-by-side comparison we decide which is a more efficient architecture for this application.

1.Introduction

The Fast Fourier Transform is derived from the Discrete Fourier Transform, which in its simplest mathematical form is defined as:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1$$

A straightforward hardware implementation of the DFT would result in at least N multiplications per output sample, storage of N coefficients, and inefficient data shifting and storage. The FFT was discovered to be a much more computationally friendly algorithm which recursively breaks down larger DFT's into smaller pieces that are easier to compute.[1] The particular model (loosely based on the Steven-Suter algorithm[2]) used in this project further enhances the efficiency of the FFT by pipelining computations which inherently shares computing resources and reduces overall power consumption. Both synchronous and asynchronous implementations have their own merits and demerits. Hence, we had built the synchronous and asynchronous

versions of the Steven-Suter algorithm in order to compare and test the performance of the design in terms of power, speed and area.

2.The FFT-4 Model

The block diagram of a 4-point FFT is shown (Figure 1). There are four inputs (x[0]-x[3]) and four outputs (X[0]-X[3]), each having a real and imaginary part.

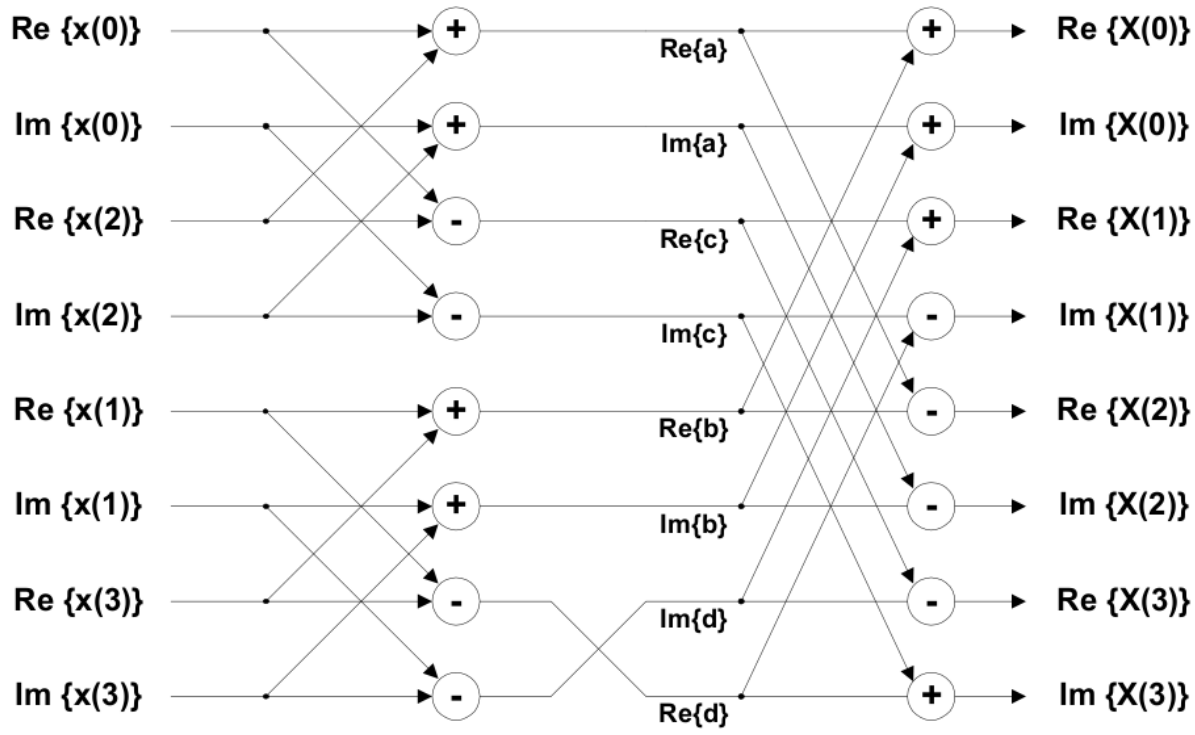

Figure 1: *FFT-4 Data flow diagram. [3]*

2.1.Flow of Data

Expansion of the matrix of 4-point FFT gives the following equations.

$X(0) = x(0) + x(1) + x(2) + x(3)$

$X(1) = x(0) - jx(1) - x(2) + jx(3)$

$X(2) = x(0) - x(1) + x(2) - x(3)$
$X(3) = x(0) + jx(1) - x(2) - jx(3)$

This can be separated into two stages of addition/subtraction.
In the first stage:
$a = x(0) + x(2)$
$b = x(1) + x(3)$
$c = x(0) - x(2)$
$d = x(1) - x(3)$,
In the second stage:
$Re\{X(0)\} = Re\{a\} + Re\{b\}$
$Im\{X(0)\} = Im\{a\} + Im\{b\}$
$Re\{X(1)\} = Re\{c\} + Im\{d\}$
$Im\{X(1)\} = Im\{c\} - Re\{d\}$
$Re\{X(2)\} = Re\{a\} - Re\{b\}$
$Im\{X(2)\} = Im\{a\} - Im\{b\}$
$Re\{X(3)\} = Re\{c\} - Im\{d\}$
$Im\{X(3)\} = Im\{c\} + Re\{d\}$

This breakdown of the FFT-4 computation is used directly in our implementation in behavioral Verilog HDL.

2.2.Advantages of FFT-4

The simplicity of 4-point FFT lies in the fact that it requires only 16 individual add or subtract operations and no complex multiplications. By only using hardware resources for addition and subtraction, we have more freedom to implement power-efficient arithmetic that will share computing resources among pipeline stages.

2.3.A "Golden Model" Software Implementation

The golden model of the 4-point FFT has been implemented in C++ language. The golden model generates four random integer inputs (separate real and imaginary values) and computes the FFT using the arithmetic presented above.

The golden model is implemented to mimic the hardware in such a way that it waits for data dependencies to resolve before it continues. As a result, 10 clock cycles pass before valid data appear at the final output (corresponding to the first input sample). The golden model results are then used to compare against the synthesized circuit.

3.Synchronous Implementation

With the software model of the design complete, we moved on to the hardware implementation of the circuit. Our strategy was to first implement the design as a clocked circuit, then remove the clock and replace it with a handshaking protocol to create an asynchronous design. The advantages of synchronous design is global control over the designed circuit and the entire design can be synchronised over a clock signal.

3.1.Synchronous Architecture

At the heart of the design is the mathematical model of the FFT-4 with its efficient add/ subtract-only structure as described above. Equally important, however, is the careful consideration required to convert the serial stream of input samples into segments of four parallel samples that the model depends on. Figure 2 shows a diagram of the top-level design.

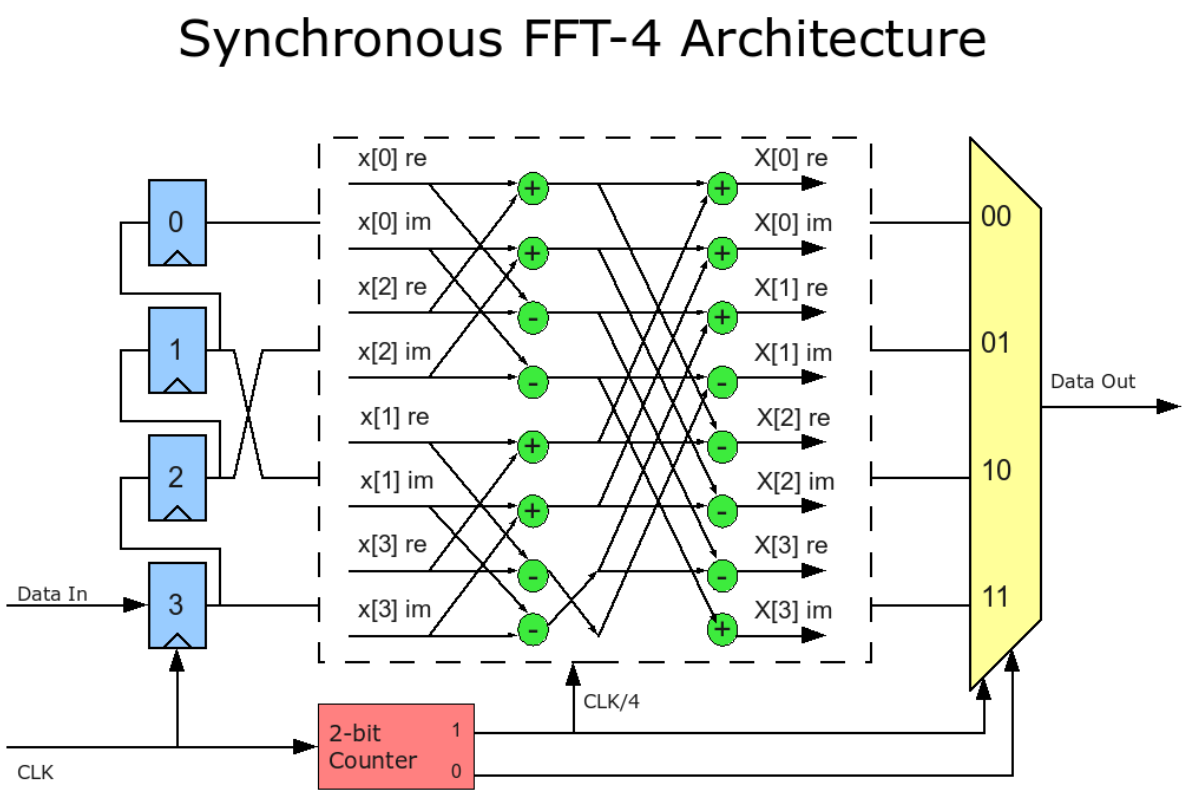

Figure 2: *Top-level synchronous design.*

The data enters the module one sample at a time (consisting of a real and imaginary part) and is latched into the first of four shift registers. Each value is a total of 32 bits wide (16 real, 16 imaginary). After three more clock cycles the input registers 0-3 contain the first four samples of data and the first actual FFT-4 computation can begin. A 2-bit counter acts as a clock divider providing a clock at 1/4 of the original frequency. The 1/4 frequency clock feeds the FFT-4 module.

Inside the FFT-4 module, the data bus expands to 20 bits (from 16) during the arithmetic stages to avoid computational overflow. Besides the adders, there are also buffer registers that exist to allow the synthesizer to re-time the circuit. The bus is truncated back to 16-bits at the final FFT-4 module stage.

At the top-level output stage, the parallelization operation is reversed using a multiplexer that selects the correct FFT result based on the 2-bit counter output. The data stream is serial once

again with the correct Fourier transform output appearing at a fixed latency from the corresponding input.

3.2.Synchronous Simulation Results

ModelSim simulation tools were used to simulate the design for a comparison against the golden model. The synthesized Verilog design generated the correct outputs as compared to the golden model, however the latency between input and corresponding output did not match the model exactly. The latency for the golden model was 10 samples while the hardware design had a latency of 22 samples. This discrepancy is probably due to Design Compiler adding re-timing registers and other data buffers.

Here are a few waveforms that demonstrate the correct operation of the FFT-4 design:
Figure 3 shows the simulated input samples, with one new value appearing on each clock cycle. Notice the top waveform is the real part, and the next waveform is the imaginary part.

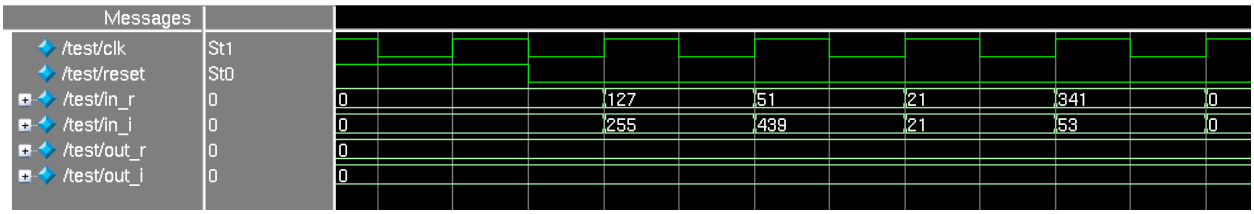

Figure 3: *Simulated input samples.*

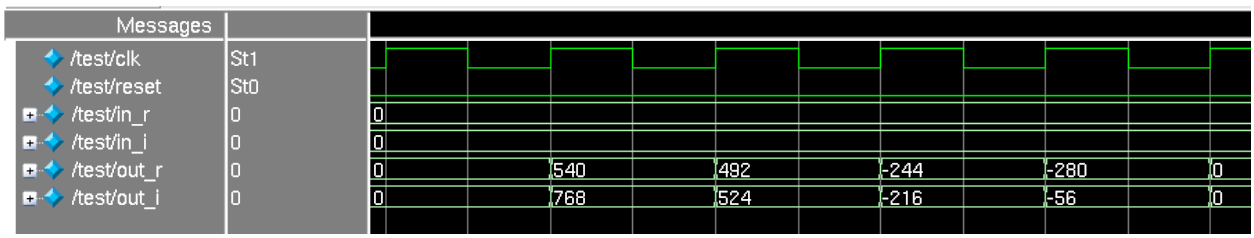

Figure 4: *Resulting output samples (arriving 22 clock cycles later).*

Figure 4 shows the resulting output samples which arrive at the output port 22 clock cycles after the corresponding input. By comparison, here is same data from the golden model:

```
input vector = [127+255i 51+439i 21+21i 341+53i]
output vector = [540+768i 492+524i -244-216i -280-56i]
```

4.Asynchronous Implementation

The Asynchronous Implementation is very much similar to that of the Synchronous implementation. The only difference being that there is no clock in the asynchronous version. The data transfer is based on the basic handshake protocol between different modules of the design. The advantages of asynchronous design are low power consumption where the computations and data transfer occurs only when request signal is asserted unlike in clocked version where the computations and data transfer occurs at every rise edge of clock, composability where a large design can be divided into simpler and smaller modules and can be connected easily, multi frequency implementation where different modules can run at different frequencies and elimination of clock skew.

4.1.Asynchronous Architecture

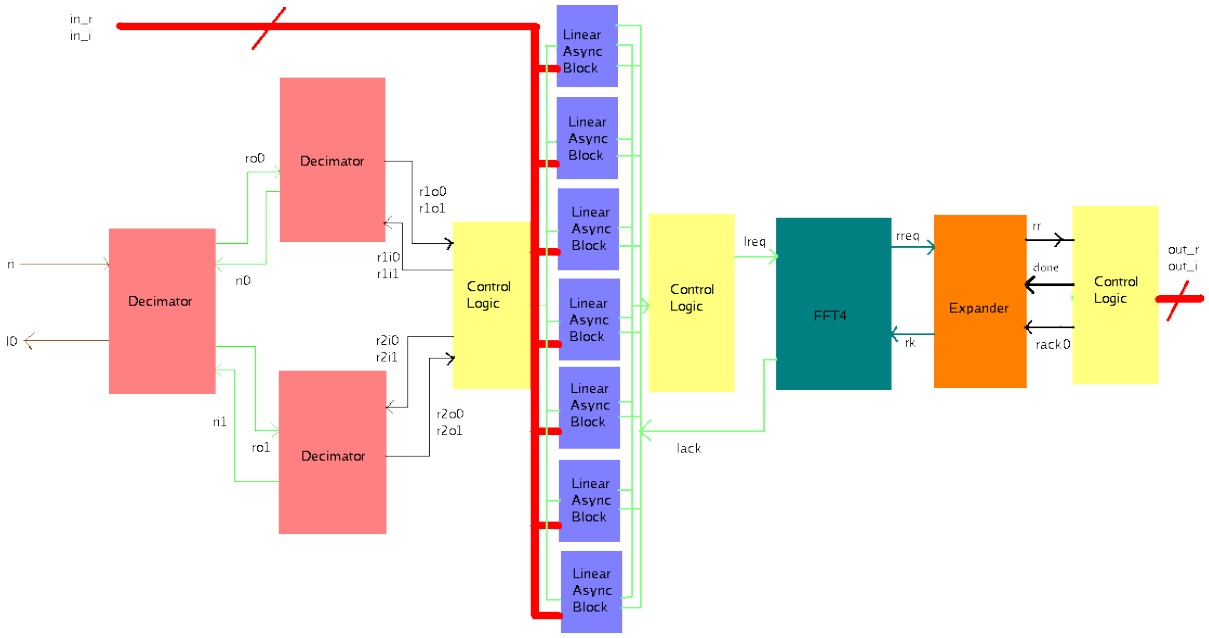

Figure 5: Aynchronous top level block diagram

The top-level block diagram of the Asynchronous version of the FFT4 design is as given in above figure. The design mainly consists of:

1. Decimator
2. Linear Asynchronous block
3. Logic Control
4. FFT-4 module, and
5. Expander.

The input data to the FFT4 comes in serially. Hence, the computation of the Fast Fourier Transform should be done after receiving all the four input signals. Hence, the FFT4 module should be operated at a frequency four times slower than the input data frequency. The Decimator that is designed halves the frequency of operation. Hence, two stages of Decimator implementation is required. The Decimator operates on hand shake protocol. The logical implementation of Decimator is as given below.

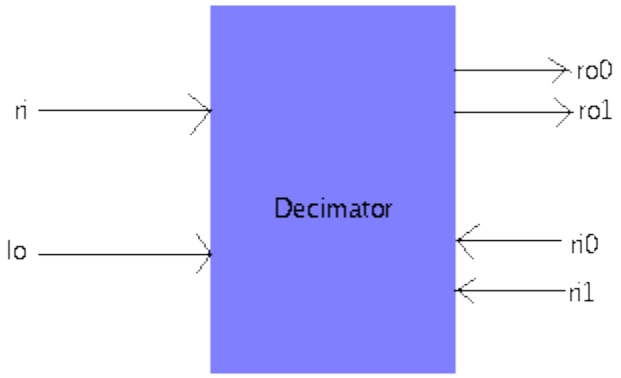

Figure 6: Block Diagram of Decimator

1 li+ | ro0+
2 ri0+ | ro0- lo+
3 li- ri0- | lo-
4 li+ | ro1+
5 ri1+ | ro1- lo+
0 li- ri1- | lo-

Each Decimator generates two request signals when li(lreq) signal goes high and asserts the lo(lack) when the data on the next stage is latched i.e when the ri(rack) signal goes high. The first Decimator sends the request signals (ro0,ro1) to the next stage of Decimators which in turn controls the next stage of linear asynchronous blocks. The async blocks are required to latch the data. Since the data signals are complex valued, eight linear async blocks are required to latch the data. The linear async latches the data as soon as it receives the request signal from the decimator blocks and sends back an ack signal to the decimator. The rreq signal from all the linear async blocks are anded together to generate a request signal to the FFT4 module.

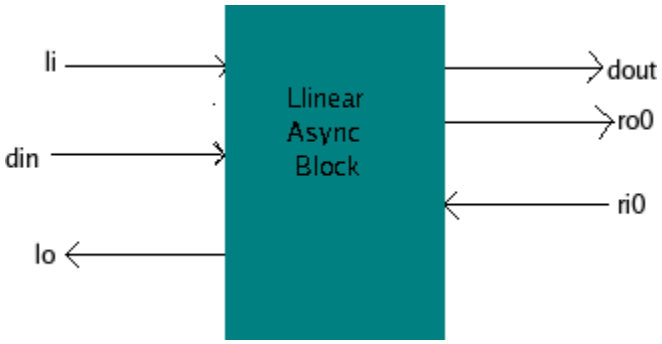

Figure 7: Block Diagram of Linear Asynchronous module

FFT4 module computes the fourier transform of the input data samples. Since the request signals from all the linear async blocks are anded together and it goes high only when all the input data is latched, the request signal to the FFT4 block is generated after all the four data signals are attained. The 4-point FFT computation involves two stages of additions and subtractions. The intermediate and final values are latched using the linear async blocks. The FFT4 module sends a request signal to the expander block which sends the data at the rate of the input data with the help of the counter.

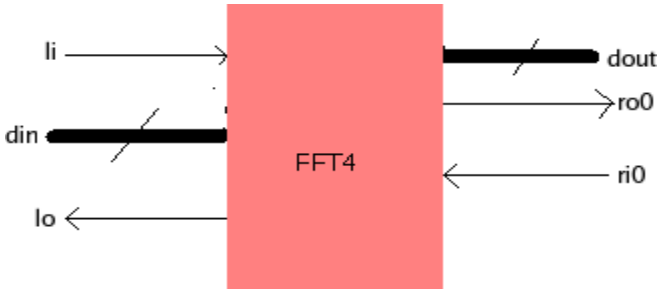

Figure 8: Block Diagram of FFT4 module

The expander functionality is similar to that of decimator in the reverse order. The expander sends the data out at the rate of the input data. The input and the output data run at the same

frequency but the computation of the FFT is performed at a rate four times slower than the input signal, the design is low power and efficient.

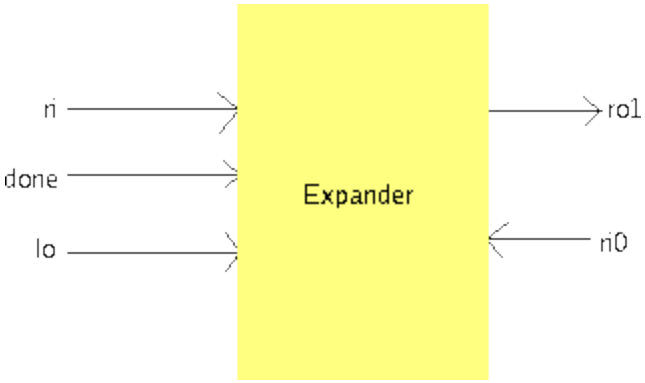

Figure 9: Block Diagram of expander

4.2.Asynchronous Simulation Results

For all our module designs, the logic was written in the form of a state diagram and the "3D" tool was used to generate the logic functions for each module. These generated logic functions, were mapped on to the UofUDigitalv1_1 library cells and were implemented in verilog. Each and every module was tested and verified with the functionality. All the simulations were run using "MODELSIM".

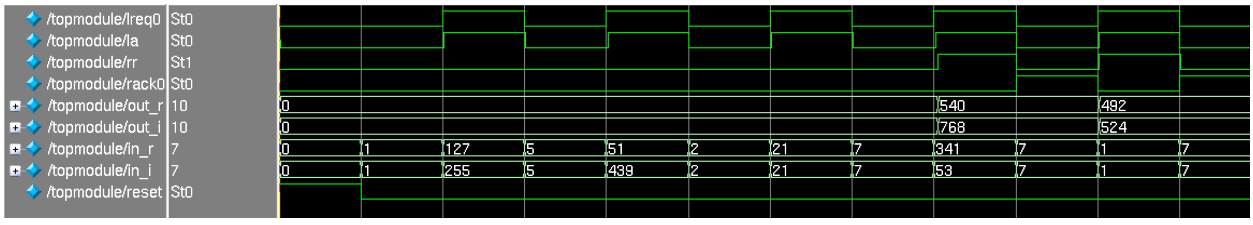

Figure 10: Input data xin = [127+255i 51+439i 21+21i 341+53i]

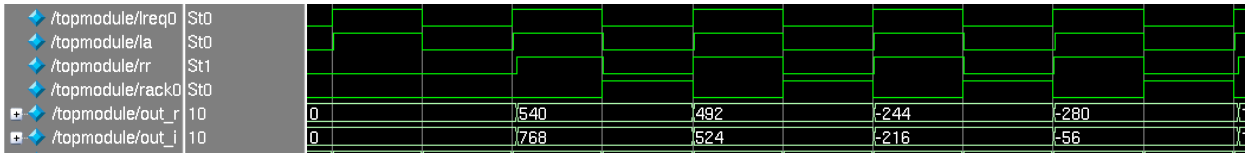

Figure 11: Output data xout = [540+768i 492+524i -244-216i -280-56i]

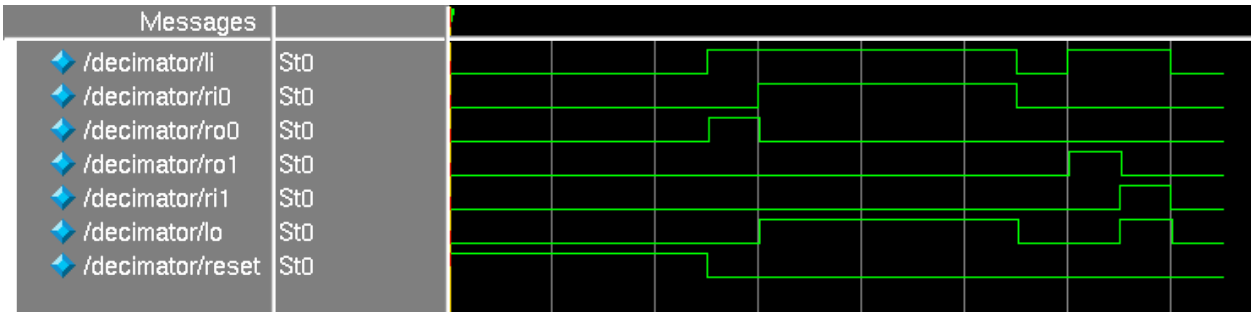

Figure 12: The behavior of the decimator

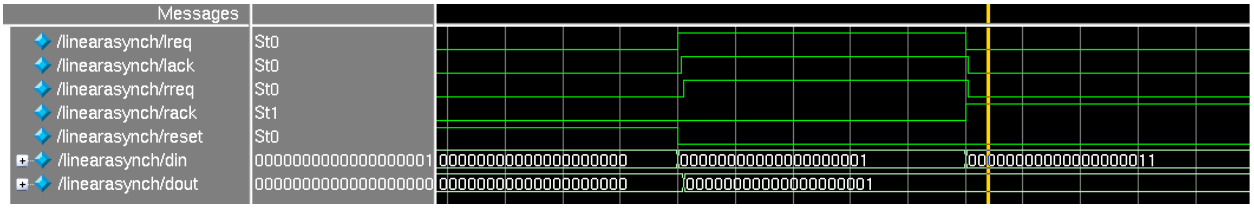

Figure 13: The behavior of linear async block

5.Conclusions

In Conclusion, the comparison of the 4-point FFT processor in asynchronous and synchronous implementations has led us to learn various concepts in asynchronous as well as synchrous designing. However, we could not really compare the designs due to the issues with multiple frequency clocks and the complexity in implementing the asynchronous design which led to our incapability of synthesizing the circuits. However, through our understanding of the design and

the concepts of synchronous and asynchronous designing we could say that asynchronous implementation is better in terms of power consumption. As a whole, the implementation of low power, high throughput FFT design has given us lot of scope in understanding the designing issues with synchronous and asynchronous versions.

6.Acknowledgements

We would like to thank Dr. Ken Stevens for his help revising and fine-tuning our design, as well as sharing his expertise in asynchronous circuit design. We would also like to thank David James Barnhart for sharing his thesis research for our benefit.

7.References

1. T. Welch, C. Wright, M. Morrow. *Real-time Digital Signal Processing*. Taylor & Francis Publishers. 2006.
2. B.W. Hunt, K.S. Stevens, B.W. Hunter, D.S. Gelosh. "A Single Chip Low Power Asynchronous Implementation of an FFT Algorithm for Space Applications". Air Force Institute of Technology.
3. Barnhart, David James. "An Improved Asychronous Implementation of a Fast Fourier Transform Architecture for Space Applications". 2001.