`**E0 217 Project Report (Aug 2023)**

**TABLE OF CONTENTS:**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TITLE** | **PAGE NUMBER** |
| 1 | Introduction about group members and individual Contributions | 2 |
| 2 | Number of clock cycles for FFT computation | 2 |
| 3 | Area of synthesized design | 3 |
| 4 | Analysis of max clock frequency | 3 |
| 5 | Power Consumption analysis | 5 |
| 6 | Energy Consumption analysis | 6 |
| 7 | Error Computation | 6 |
| 8 | Modification of FFT circuit to compute inverse FFT | 8 |
| 9 | Design Details and explanation of implementation | 8 |
| 10 | Architectural Trade-offs | 11 |
| 11 | References | 11 |

**E0 217 Project Report (Aug 2023)**

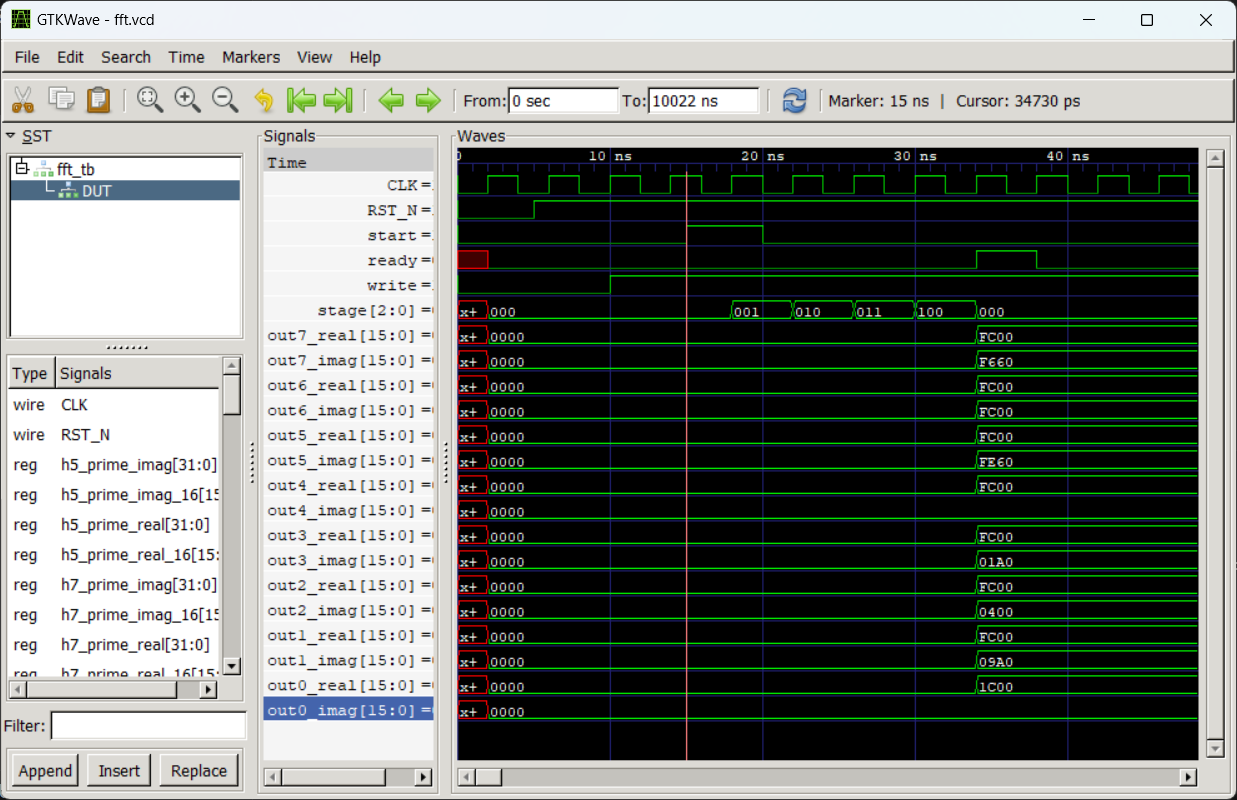
**Names and IISc Email IDs of Group Members:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Name** | **Sr.No** | **IISc Email ID** |
| 1. | **Guhan Rajasekar** | **22410** | **guhanr@iisc.ac.in** |
| 2. | **Ujjwal Chaudhary** | **22577** | **ujjwalc@iisc.ac.in** |

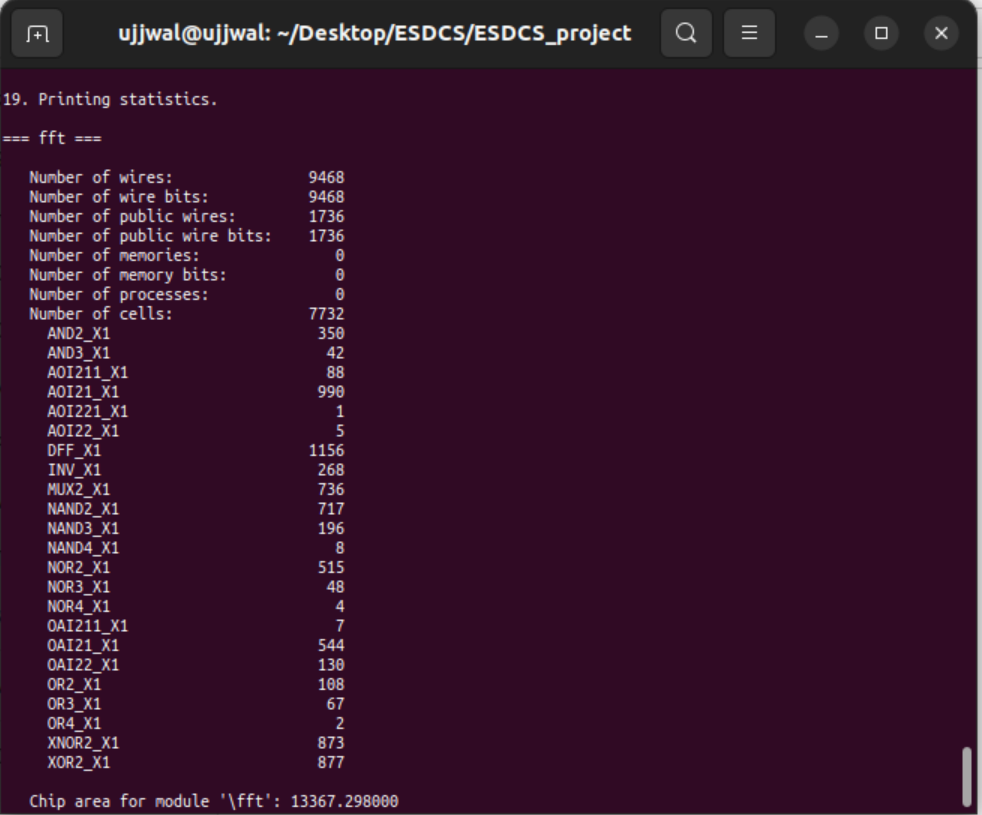
**Individual Contributions**

|  |  |  |
| --- | --- | --- |
|  | **Name** | **Contributions** |
| 1. | **Guhan Rajasekar** | * Implementation of FFT code (without control signals) * Error computation between obtained and desired results * Report compilation |
| 2. | **Ujjwal Chaudhary** | * Optimization of code (changing combinational blocks to sequential blocks) and making the code compatible with CLK signal and control signals like RST, WRITE, START, READY * Creation of test bench and simulation of design in GTK Wave. * Performed area, power and speed analysis. |

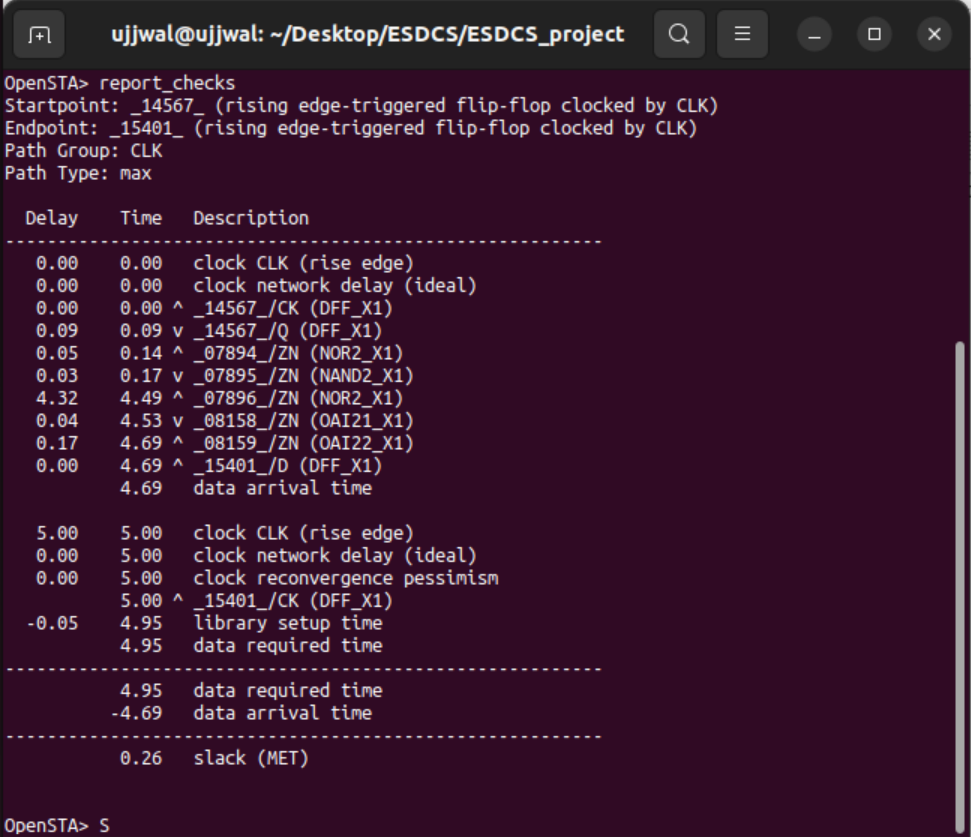
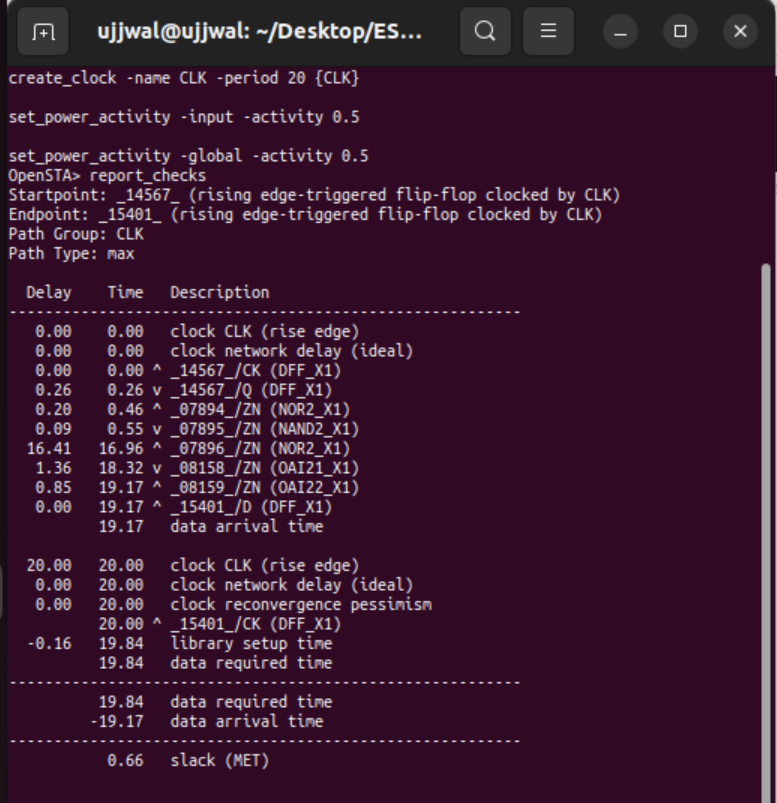
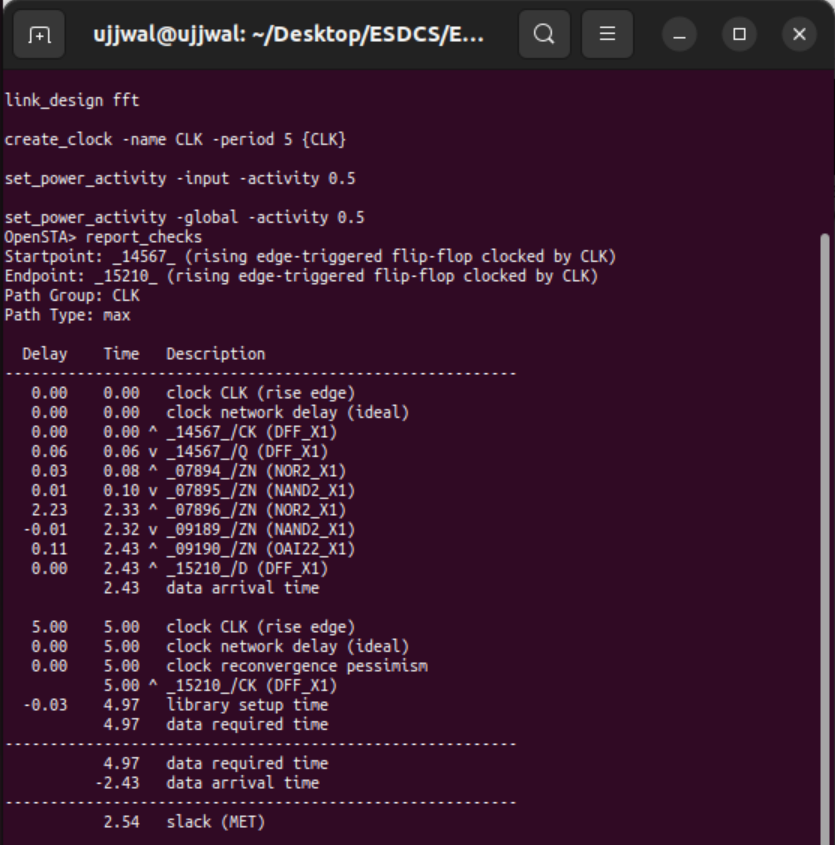
**Number of clock cycles per FFT computation:**

* Number of clock cycles between beginning of start and beginning of ready signal = **4.75**
* 

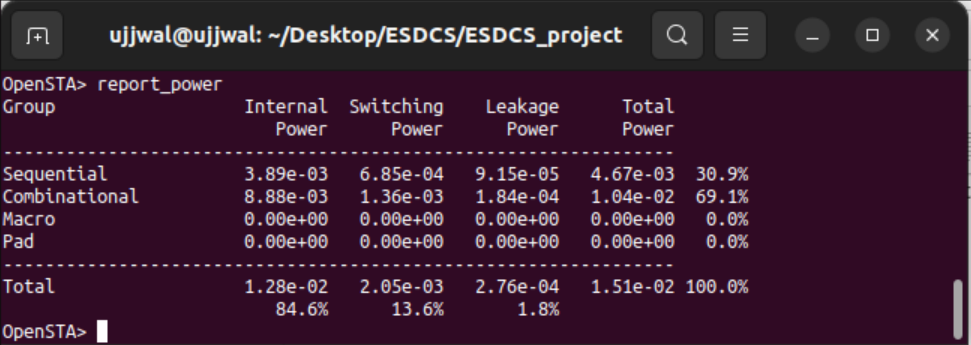
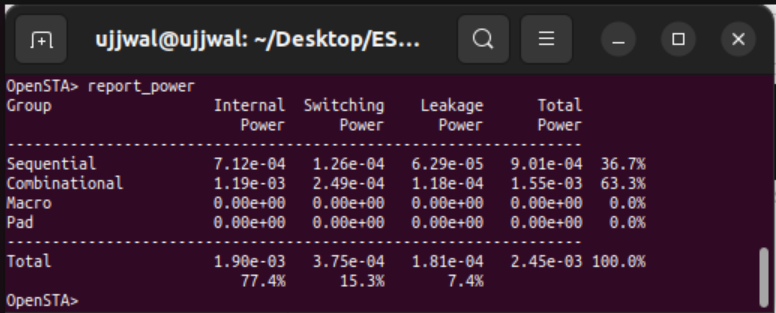
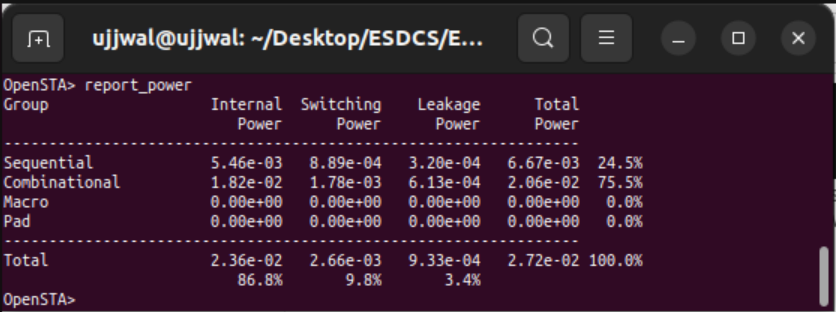
**Area of synthesized design:**

* 
* Area of synthesized design is **13367.298 µm2**.

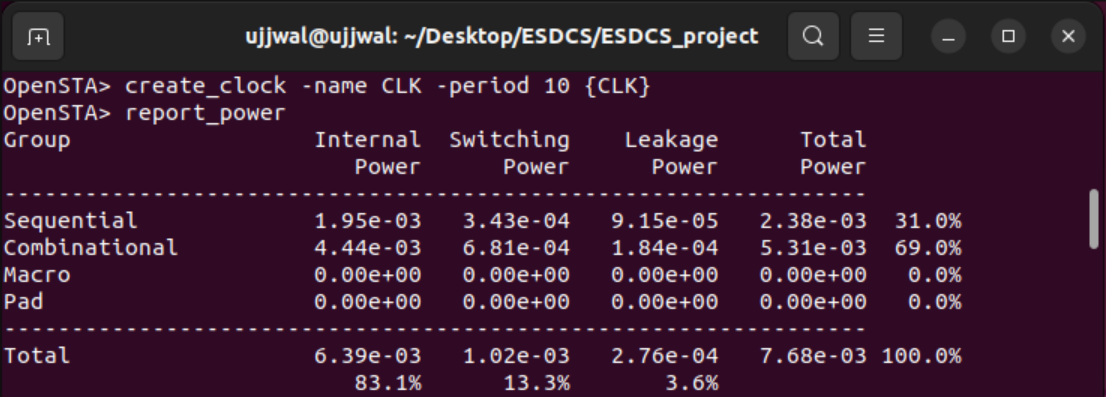
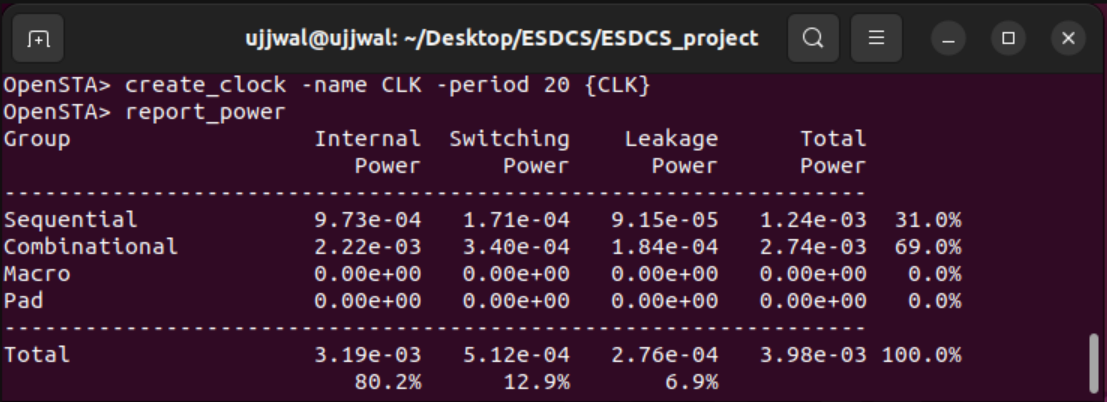
**Max clock frequency supported by the synthesized design:**

* **Typical corner, 25℃ and 1V**
* 
* This result is for typical corner, 25℃ and 1V.
* Input clock signal period is 5ns.
* Here the slack period is 0.26ns.
* Supported clock period is 5ns – 0.26ns = **4.74ns**
* Operating frequency is **210.97 MHz**.
* **SS corner, 25℃, 1V**
* 
* This is for SS corner, 25℃ and 1V.
* Input clock period is 20ns. (because this case does not support clock signal of 5ns)
* Slack period is 0.66 ns.
* Supported clock signal period is 20ns – 0.66ns = **19.34ns**
* Operating frequency = **51.706MHz.**
* **FF corner, 25℃, 1V**
* 
* Input clock signal period = 5ns.
* Slack period = 2.54 ns.
* Clock period supported by the design = **2.46ns.**
* Max operating frequency at FF corner = **406.504MHz**

**Power Consumption of the synthesized design:**

* **Power Consumption in typical corner, 25℃ and 1V:**
* 
* The total power consumed at typical corner, 25℃ and 1V is **15.1mW**.
* **Power Consumption in SS corner,25℃ and 1V.**
* 
* Power consumed at SS corner, 25℃ and 1V is **2.45mW**
* **Power Consumption in FF corner,25℃ and 1V.**
* 
* Total power consumption at FF corner, 25℃ and 1V is **27.2mW**.

**Energy Consumption of synthesized design:**

* Typical, 25℃, 1V, T=5ns (Image attached above)
* Typical, 25℃, 1V, T=10ns 
* Typical, 25℃, 1V, T=20ns 

|  |  |  |  |
| --- | --- | --- | --- |
| **Corner Case** | **Power Consumption (mW)** | **Frequency (MHz)** | **Energy Consumption (pJ)** |
| Typical, 25℃, 1V | 15.1 | 200 | 75.5 |
| Typical, 25℃, 1V | 7.68 | 100 | 76.8 |
| Typical, 25℃, 1V | 3.98 | 50 | 79.6 |

**Dependence of Energy Consumption of the synthesized design on the clock frequency:**

* From above mentioned table, we conclude that **energy consumption reduces as clock frequency increases.**

**Maximum absolute error when computing FFT of {0, 1, 2, 3, 4, 5, 6, 7}**

* Let the 8-point FFT result be denoted as X(0), X(1), X(2), X(3), X(4), X(5), X(6) and X(7)
* Here a **decimal point has been added in the obtained results to indicate there are 8 bits for the fractional part, 7 bits for the decimal part and MSB is the sign bit.** This holds true for both real and imaginary parts.
* In the following calculations, is considered as **1.414**.

1. **X(0) :**

* Expected = 28 + j0
* Obtained = (1c.00 + j00.00)H = 28 + j0
* Absolute Error = 0

1. **X(1)**

* Expected = -4.0 + j 9.656
* Obtained = (fc.00 + j09.a0)H = -4 + j9.625
* Absolute Error = |(-4.0 + j 9.656) – (-4 + j9.625) | = 0.031

1. **X(2)**

* Expected = -4 + j4
* Obtained = (fc.00 + j04.00)H = (-4 + j4)
* Absolute Error = 0

1. **X(3)**

* Expected = -4 + j1.656
* Obtained = (fc.00 + j01.a0)H  = -4 + j1.625
* Absolute Error = |(-4 + j1.656) – (-4 + j1.625) | = 0.031

1. **X(4)**

* Expected = -4 + j0
* Obtained = (fc.00 + j00.00)H = -4 + j0
* Absolute Error = 0

1. **X(5)**

* Expected = -4 – j1.656
* Obtained = (fc.00 + jfe.60)H = (-4 – j1.402)
* Absolute Error = |(-4 – j1.656 ) – (-4 – j1.402) | = 0.254

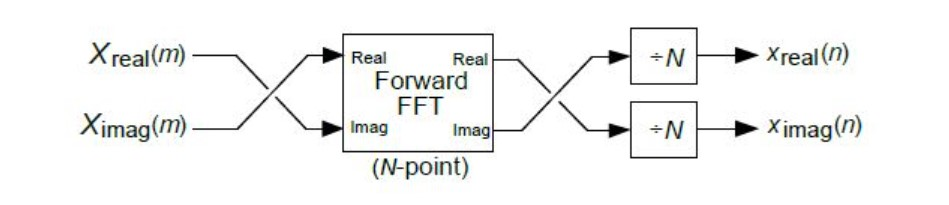
1. **X(6)**

* Expected = -4.0 – j4.0
* Obtained = (fc.00 + jfc.00)H = -4 - j4
* Absolute Error = 0

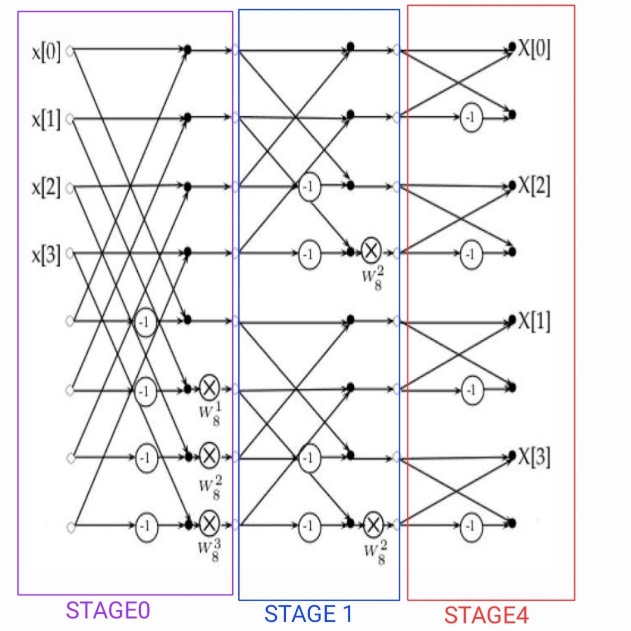
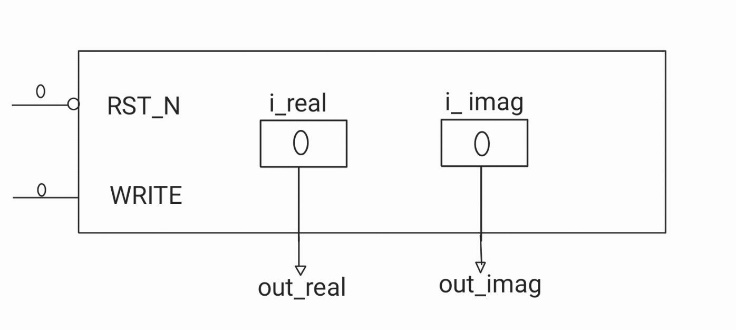
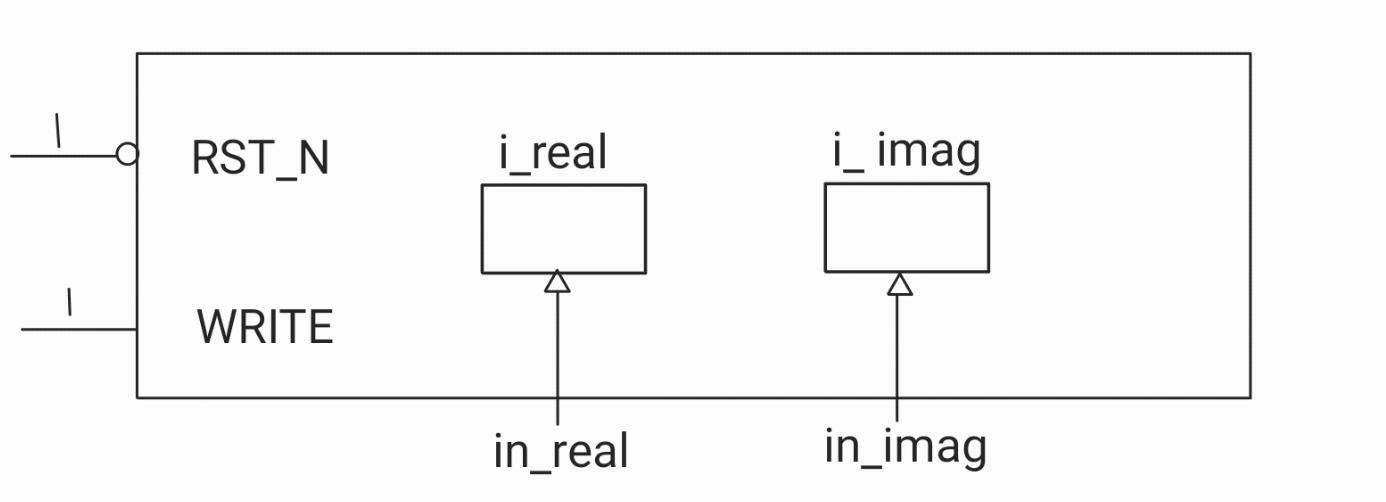
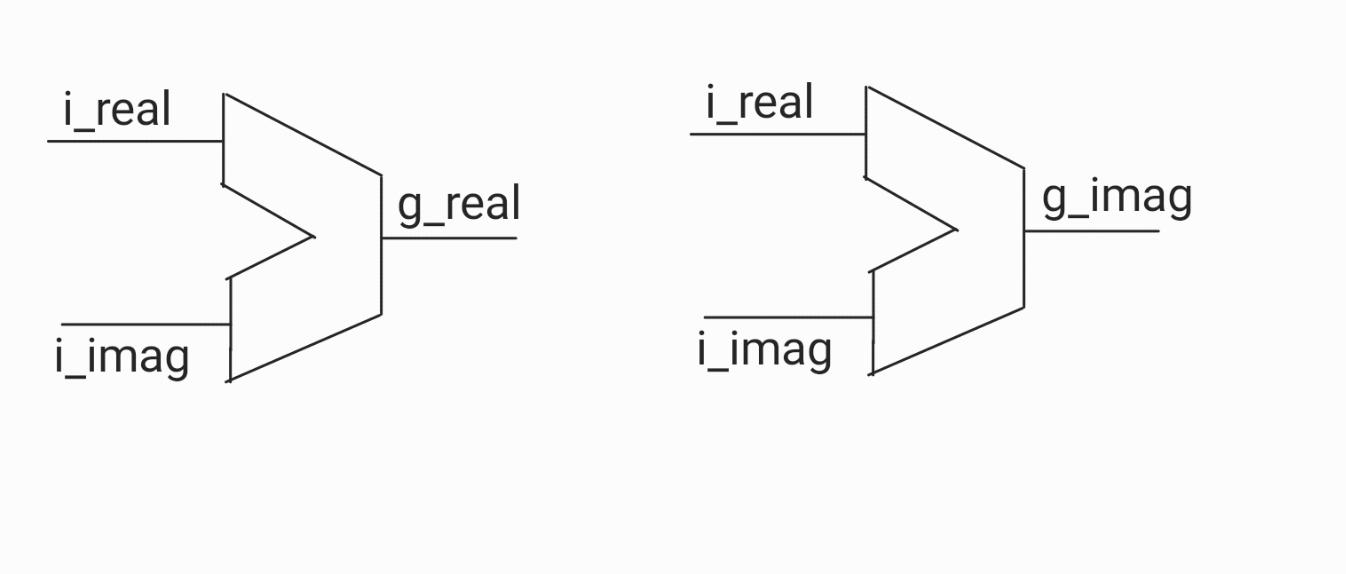
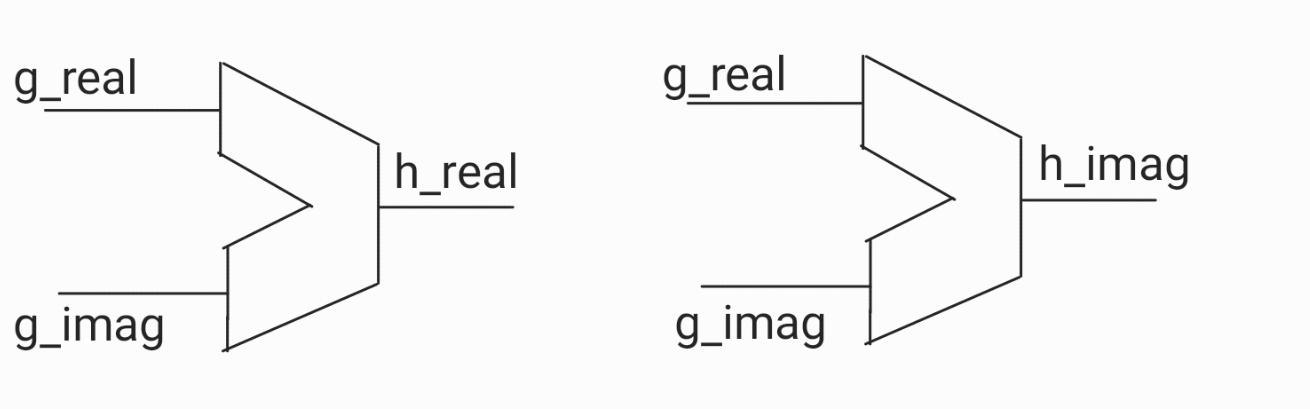
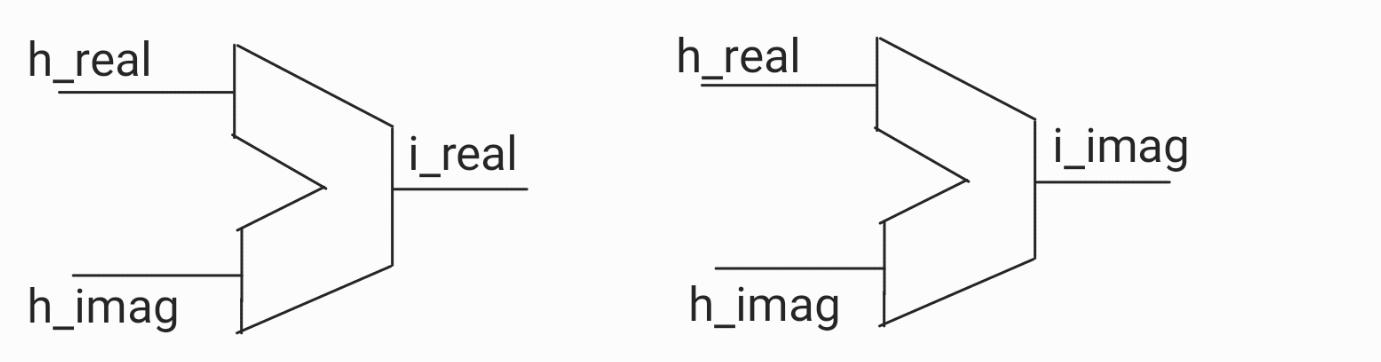
1. **X(7)**

* Expected = -4 – j9.656
* Obtained = (fc.00 + jf6.60)H = -4 – j9.625
* Absolute Error = | (-4 – j9.656 ) – (-4 – j9.625) | = 0.031

**Modifying the fft circuit to compute inverse fft**

* Inverse fft can be computed using Data Swapping technique, as shown below:
* 
* The real part of FFT input is fed as imaginary input to the circuit and the imaginary part of FFT is fed as real input to the circuit.
* The real portion of the output gives imaginary part of expected time domain sequence scaled by N and the imaginary part of the output gives the real part of the time domain sequence scaled by N. To get the desired outputs, we divide them by N.
* So, in addition to the existing hardware circuitry, extra circuitry will be required to scale by factor of (1/N) to get the desired time domain inverse FFT sequence.

**Design Details:**

* The entire FFT computation has been divided into five stages as follows:
* 
* Stage 2 and Stage 3 are used for intermediate variable computations in the code.
* But the overall FFT computation can be broadly classified into stages 0, 1 and 4.
* If the RST\_N signal is low, the value 0 is written in the registers i0\_real, i0\_imag, i1\_real,i1\_imag, …… , i7\_real, i7\_imag.
* The output wires out0\_real,out0\_imag, out1\_real,out1\_imag,…..out7\_real,out7\_imag are continuously driven by the data present in the registers as shown below.
* 
* In the above block, although only two registers are shown, there are total of 16 registers ( 8 for the real part and 8 for the imaginary part ) and each register holds 16 bit data.
* **Storing Inputs in registers:**
* The circuit stores inputs in registers when **write** signal is high.
* When write signal is high, the registers i0\_real ,i0\_imag, i1\_real,i1\_imag,… ,i7\_real, i7\_imag will store the value held by the continuously driven wires in0\_real,in0\_imag,in1\_real,in1\_imag…..,in7\_real,in7\_imag respectively as shown below:
* 
* Here as well only two signals in\_real and in\_imag are shown for brevity of block diagram.
* But there are 16 wires ( 8 for real and 8 for imaginary parts) that are continuously driven.
* Computation commences when **start** signal is high.
* The value stored by 2-bit register variable named **stage** indicates the stage of FFT computation.
* **Stage 0 computations:**
* The contents in i0\_real, i0\_imag,…….,i7\_real,i7\_imag act as inputs for stage 0 computations.
* The registers g\_real[0], g\_imag[0], g\_real[1],g\_imag[1]…..,g\_real[7],g\_imag[7] store the results of stage 0.
* 
* Once again for brevity, not all the registers of stage 0 are shown.
* There are 16 registers. 8 registers hold the real part of stage 0 results and 8 registers hold the imaginary part of stage 0 results .
* Stage 0 computations is based on the formula obtained from the butterfly diagram.
* Once stage 0 computations are done, the content of **stage register** is incremented by 1 to go to the next stage.
* **Stage 1 computations**
* For stage 1 computations, the contents of the registers g\_real[0], g\_imag[0], g\_real[1],g\_imag[1]…..,g\_real[7],g\_imag[7] will be the inputs and the results are stored in the registers
* 
* The computations are based on the equations obtained from the butterfly diagram.
* **Stage 2 computations**
* In this stage , the registers temp0, temp1 ,temp2 and temp3 are populated as follows:
* temp0 = h\_real[5] + h\_imag[5]
* temp1 = h\_imag[5] - h\_real[5]
* temp2 = h\_imag[7] - h\_real[7]
* temp3 = -(h\_imag[7] + h\_real[7])
* These calculations aid in the computation of the final result.
* Once the above calculations are done, contents of the stage register is incremented by 1 and we go to the next stage.
* **Stage 3 computations**
* In this stage, the contents of temp0, temp1, temp2, temp3 are multiplied with 0.707 ( hardcoded in binary in the code) .
* Then content of the stage register is incremented and we go to the final stage.
* **Stage 4 computations:**
* Here the contents of h\_real[0],h\_imag[0],h\_real[1],h\_imag[1]…,h\_real[7],h\_imag[7] act as inputs and the final output is stored in the registers i\_real[0],i\_imag[0],i\_real[1],i\_imag[1],…i\_real[7],i\_imag[7].
* 
* The wires out0\_real , out0\_imag, out1\_real, out1\_imag,…….out7\_real,out7\_imag are continuously driven by the contents in these registers. Hence at any point of time, the output at that instant is present in these output wires.

**Architectural Trade-offs**

* In this FFT implementation, focus is low latency.
* Hence the inbuilt multiplication function has been used with the ‘ \* ‘ symbol to get faster multiplication results that are required at the intermediate stages.
* This comes at the cost of more area and power consumption.
* As an addition, pipelining can be implemented if we want to improve throughput.

**References**

[1] Wikipedia – Fast Fourier Transform

[2] NPTEL – Hardware Modelling using Verilog by Prof. Indranil Sen Gupta

[3] “The Fast Fourier Transform (FFT): Most Ingenious Algorithm Ever?” Youtube.com by

Reducible

[4] “DSP Tricks: Computing inverse FFTs using the forward FFT” – embedded.com