28-11-2023

`E0 217 Project Report (Aug 2023)

Project Title: Design, Implementation and Simulation of an 8-point FFT Circuit in 45nm CMOS

TABLE OF CONTENTS:

28-11-2023

E0 217 Project Report (Aug 2023)

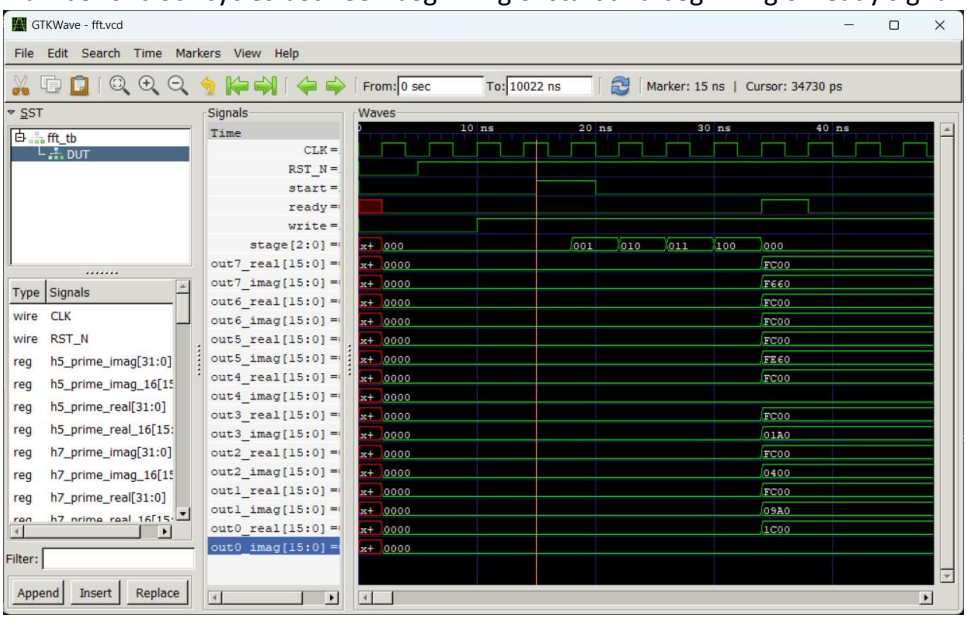
Names and IISc Email IDs of Group Members:

	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:

```
=== fft ===

   Number of wires:              9468
   Number of wire bits:          9468
   Number of public wires:       1736
   Number of public wire bits:   1736
   Number of memories:              0
   Number of memory bits:           0
   Number of processes:             0
   Number of cells:              7732
     AND2_X1                      350
     AND3_X1                       42
     AOI211_X1                     88
     AOI21_X1                     990
     AOI221_X1                      1
     AOI22_X1                       5
     DFF_X1                      1156
     INV_X1                       268
     MUX2_X1                      736
     NAND2_X1                     717
     NAND3_X1                     196
     NAND4_X1                       8
     NOR2_X1                      515
     NOR3_X1                       48
     NOR4_X1                        4
     OAI211_X1                      7
     OAI21_X1                     544
     OAI22_X1                     130
     OR2_X1                       108
     OR3_X1                        67
     OR4_X1                         2
     XNOR2_X1                     873
     XOR2_X1                      877

   Chip area for module '\fft': 13367.298000
```

- Area of synthesized design is **13367.298 μm^2**.

Max clock frequency supported by the synthesized design:

- **Typical corner, 25℃ and 1V**

```
Path Type: max

  Delay    Time   Description
---------------------------------------------------------
   0.00    0.00   clock CLK (rise edge)
   0.00    0.00   clock network delay (ideal)
   0.00    0.00 ^ _14567_/CK (DFF_X1)
   0.09    0.09 v _14567_/Q (DFF_X1)
   0.05    0.14 ^ _07894_/ZN (NOR2_X1)
   0.03    0.17 v _07895_/ZN (NAND2_X1)
   4.32    4.49 ^ _07896_/ZN (NOR2_X1)
   0.04    4.53 v _08158_/ZN (OAI21_X1)
   0.17    4.69 ^ _08159_/ZN (OAI22_X1)
   0.00    4.69 ^ _15401_/D (DFF_X1)
           4.69   data arrival time

   5.00    5.00   clock CLK (rise edge)
   0.00    5.00   clock network delay (ideal)
   0.00    5.00   clock reconvergence pessimism
           5.00 ^ _15401_/CK (DFF_X1)
  -0.05    4.95   library setup time
           4.95   data required time
---------------------------------------------------------
           4.95   data required time
          -4.69   data arrival time
---------------------------------------------------------
           0.26   slack (MET)


OpenSTA> S
```

- 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**

```
set_power_activity -global -activity 0.5
OpenSTA> report_checks
Startpoint: _14567_ (rising edge-triggered flip-flop clocked by CLK)
Endpoint: _15401_ (rising edge-triggered flip-flop clocked by CLK)
Path Group: CLK
Path Type: max

  Delay    Time   Description
---------------------------------------------------------
   0.00    0.00   clock CLK (rise edge)
   0.00    0.00   clock network delay (ideal)
   0.00    0.00 ^ _14567_/CK (DFF_X1)
   0.26    0.26 v _14567_/Q (DFF_X1)
   0.20    0.46 ^ _07894_/ZN (NOR2_X1)
   0.09    0.55 v _07895_/ZN (NAND2_X1)
  16.41   16.96 ^ _07896_/ZN (NOR2_X1)
   1.36   18.32 v _08158_/ZN (OAI21_X1)
   0.85   19.17 ^ _08159_/ZN (OAI22_X1)
   0.00   19.17 ^ _15401_/D (DFF_X1)
          19.17   data arrival time

  20.00   20.00   clock CLK (rise edge)
   0.00   20.00   clock network delay (ideal)
   0.00   20.00   clock reconvergence pessimism
          20.00 ^ _15401_/CK (DFF_X1)
  -0.16   19.84   library setup time
          19.84   data required time
---------------------------------------------------------
          19.84   data required time
         -19.17   data arrival time
---------------------------------------------------------
           0.66   slack (MET)
```

  - 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**

```
set_power_activity -input -activity 0.5

set_power_activity -global -activity 0.5
OpenSTA> report_checks
Startpoint: _14567_ (rising edge-triggered flip-flop clocked by CLK)
Endpoint: _15210_ (rising edge-triggered flip-flop clocked by CLK)
Path Group: CLK
Path Type: max

  Delay    Time   Description
---------------------------------------------------------
   0.00    0.00   clock CLK (rise edge)
   0.00    0.00   clock network delay (ideal)
   0.00    0.00 ^ _14567_/CK (DFF_X1)
   0.06    0.06 v _14567_/Q (DFF_X1)
   0.03    0.08 ^ _07894_/ZN (NOR2_X1)
   0.01    0.10 v _07895_/ZN (NAND2_X1)
   2.23    2.33 ^ _07896_/ZN (NOR2_X1)
  -0.01    2.32 v _09189_/ZN (NAND2_X1)
   0.11    2.43 ^ _09190_/ZN (OAI22_X1)
   0.00    2.43 ^ _15210_/D (DFF_X1)
           2.43   data arrival time

   5.00    5.00   clock CLK (rise edge)
   0.00    5.00   clock network delay (ideal)
   0.00    5.00   clock reconvergence pessimism
           5.00 ^ _15210_/CK (DFF_X1)
  -0.03    4.97   library setup time
           4.97   data required time
---------------------------------------------------------
           4.97   data required time
          -2.43   data arrival time
---------------------------------------------------------
           2.54   slack (MET)
```

  - 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:**

```
OpenSTA> report_power
Group                  Internal  Switching    Leakage      Total
                          Power      Power      Power      Power
---------------------------------------------------------------
Sequential             3.89e-03   6.85e-04   9.15e-05   4.67e-03  30.9%
Combinational          8.88e-03   1.36e-03   1.84e-04   1.04e-02  69.1%
Macro                  0.00e+00   0.00e+00   0.00e+00   0.00e+00   0.0%
Pad                    0.00e+00   0.00e+00   0.00e+00   0.00e+00   0.0%
---------------------------------------------------------------
Total                  1.28e-02   2.05e-03   2.76e-04   1.51e-02 100.0%
                          84.6%      13.6%       1.8%
OpenSTA>
```

  - The total power consumed at typical corner, 25℃ and 1V is **15.1mW**.

- **Power Consumption in SS corner,25℃ and 1V.**

```
OpenSTA> report_power
Group                  Internal  Switching    Leakage      Total
                          Power      Power      Power      Power
---------------------------------------------------------------
Sequential             7.12e-04   1.26e-04   6.29e-05   9.01e-04  36.7%
Combinational          1.19e-03   2.49e-04   1.18e-04   1.55e-03  63.3%
Macro                  0.00e+00   0.00e+00   0.00e+00   0.00e+00   0.0%
Pad                    0.00e+00   0.00e+00   0.00e+00   0.00e+00   0.0%
---------------------------------------------------------------
Total                  1.90e-03   3.75e-04   1.81e-04   2.45e-03 100.0%
                          77.4%      15.3%       7.4%
OpenSTA>
```

  - Power consumed at SS corner, 25℃ and 1V is **2.45mW**

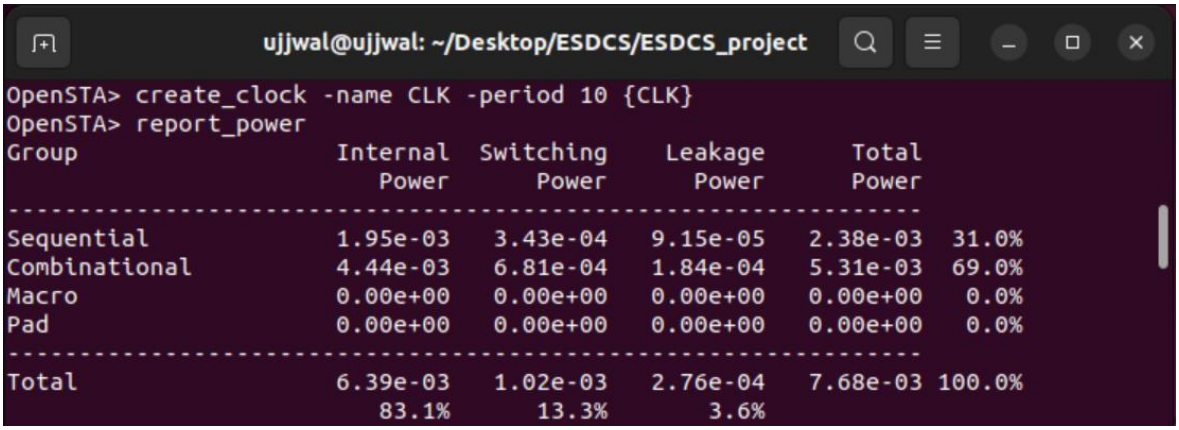
- **Power Consumption in FF corner,25℃ and 1V.**

```
OpenSTA> report_power
Group                  Internal  Switching    Leakage      Total
                          Power      Power      Power      Power
---------------------------------------------------------------
Sequential             5.46e-03   8.89e-04   3.20e-04   6.67e-03  24.5%
Combinational          1.82e-02   1.78e-03   6.13e-04   2.06e-02  75.5%
Macro                  0.00e+00   0.00e+00   0.00e+00   0.00e+00   0.0%
Pad                    0.00e+00   0.00e+00   0.00e+00   0.00e+00   0.0%
---------------------------------------------------------------
Total                  2.36e-02   2.66e-03   9.33e-04   2.72e-02 100.0%
                          86.8%       9.8%       3.4%
OpenSTA>
```

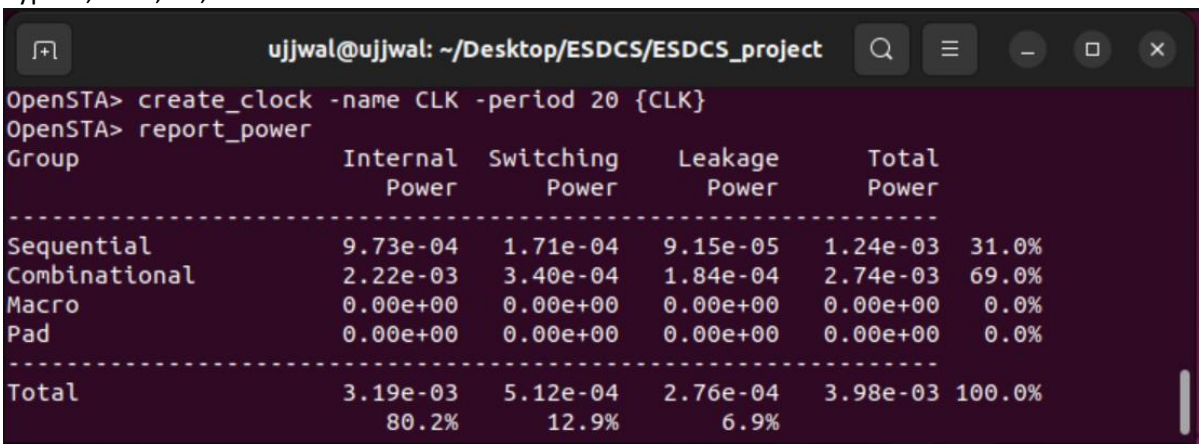
  - Total power consumption at FF corner, 25℃ and 1V is **27.2mW**.

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, $\sqrt{2}$ is considered as **1.414**.

1. **X(0) :**
   - Expected = 28 + j0
   - Obtained = $(1c.00 + j00.00)_H = 28 + j0$
   - Absolute Error = 0

2. **X(1)**
   - Expected = -4.0 + j 9.656
   - Obtained = $(fc.00 + j09.a0)_H = -4 + j9.625$
   - Absolute Error = $|(-4.0 + j\ 9.656) – (-4 + j9.625)| = 0.031$

3. **X(2)**
   - Expected = -4 + j4
   - Obtained = $(fc.00 + j04.00)_H = (-4 + j4)$
   - Absolute Error = 0

4. **X(3)**
   - Expected = -4 + j1.656
   - Obtained = $(fc.00 + j01.a0)_H = -4 + j1.625$
   - Absolute Error = $|(-4 + j1.656) – (-4 + j1.625)| = 0.031$

5. **X(4)**
   - Expected = -4 + j0
   - Obtained = $(fc.00 + j00.00)_H = -4 + j0$
   - Absolute Error = 0

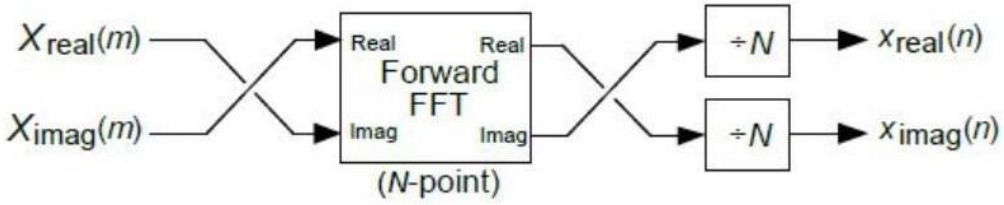
6. **X(5)**
   - Expected = -4 – j1.656
   - Obtained = $(fc.00 + jfe.60)_H = (-4 – j1.402)$
   - Absolute Error = $|(-4 – j1.656) – (-4 – j1.402)| = 0.254$

7. **X(6)**
   - Expected = -4.0 – j4.0
   - Obtained = $(fc.00 + jfc.00)_H = -4 - j4$
   - Absolute Error = 0

8. **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

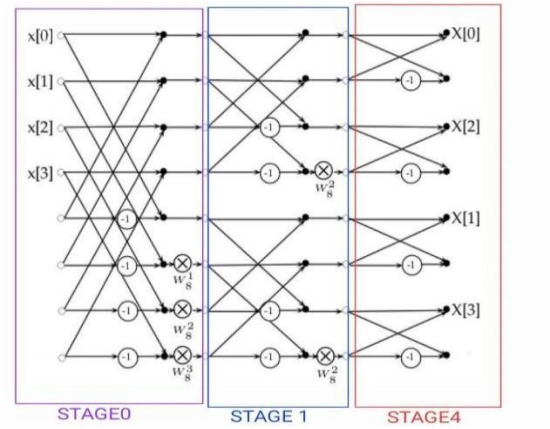
- Inverse fft can be computed using Data Swapping technique, as shown below:



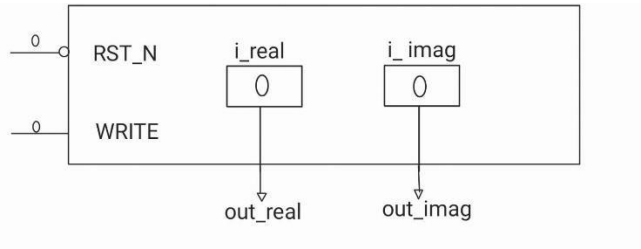
-
- The real part of FFT input is fed as imaginary input to the circuit and the imaginary part of FFT is fed as real input to the circuit.
- The real portion of the output gives imaginary part of expected time domain sequence scaled by N and the imaginary part of the output gives the real part of the 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:

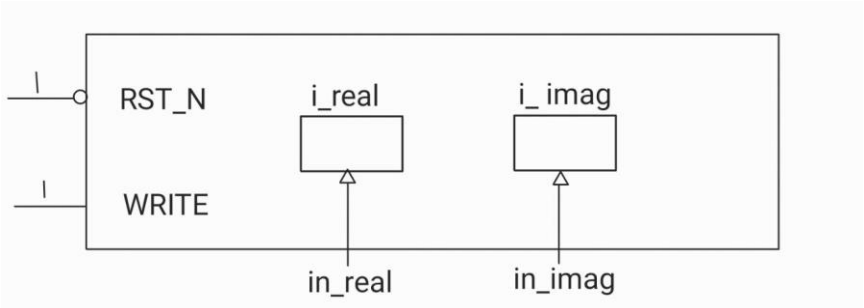


- Stage 2 and Stage 3 are used for intermediate variable computations in the code.
- But the overall FFT computation can be broadly classified into stages 0, 1 and 4.
- If the RST_N signal is low, the value 0 is written in the registers i0_real, i0_imag, i1_real,i1_imag, …… , i7_real, i7_imag.
- The 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.

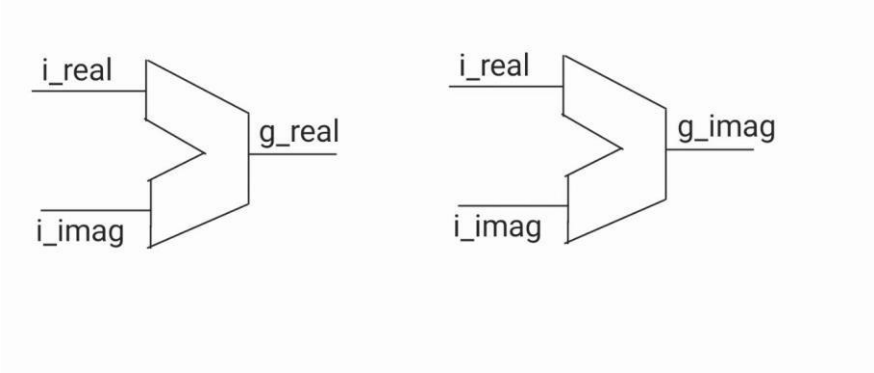
- **Storing Inputs in registers:**
  - The circuit stores inputs in registers when **write** signal is high.
  - When write signal is high, the registers i0_real ,i0_imag, i1_real,i1_imag,… ,i7_real, i7_imag will store the value held by the continuously driven wires in0_real,in0_imag,in1_real,in1_imag…..,in7_real,in7_imag 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.

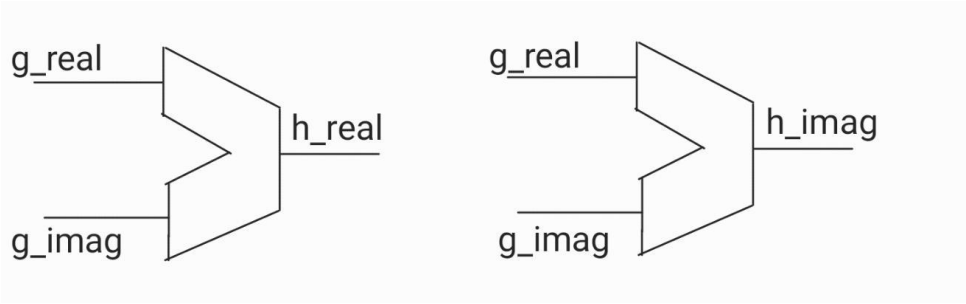
- Computation commences when **start** signal is high.
- The value stored by 2-bit register variable named **stage** indicates the stage of FFT computation.

- **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.

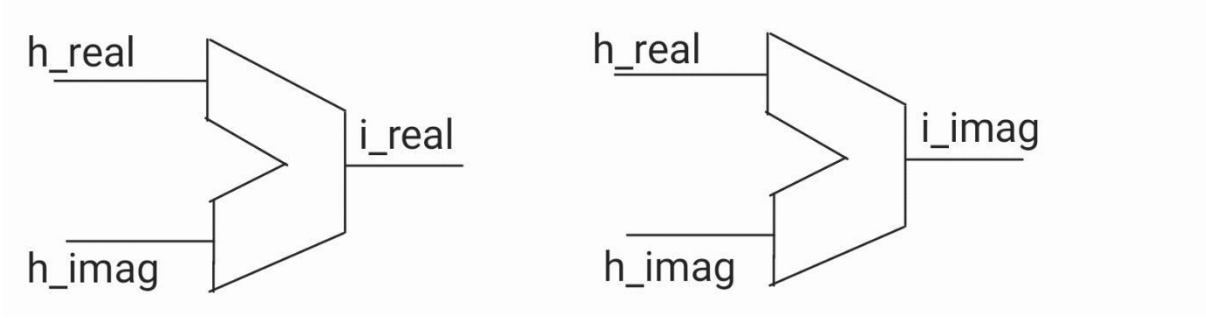
- **<u>Stage 1 computations</u>**
  - For stage 1 computations, the contents of the registers g_real[0], g_imag[0], g_real[1],g_imag[1]…..,g_real[7],g_imag[7] will be the inputs and the results are stored in the registers h_real[0],h_real[1],…..,h_real[7],h_imag[0],……,h_imag[7]



  - The computations are based on the equations obtained from the butterfly diagram.

- **<u>Stage 2 computations</u>**
  - 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.

- **<u>Stage 3 computations</u>**
  - In this stage, the contents of temp0, temp1, temp2, temp3 are multiplied with 0.707 (hard coded 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.

28-11-2023

Verilog Code to implement fft:

```
module fft(CLK, RST_N, write, start, in0_real , in0_imag, in1_real, in1_imag,
 in2_real  ,in2_imag, in3_real  ,in3_imag, in4_real  ,in4_imag, in5_real  ,in5_imag,
 in6_real  ,in6_imag, in7_real  ,in7_imag, ready, out0_real ,out0_imag, out1_real,
 out1_imag, out2_real ,out2_imag, out3_real ,out3_imag, out4_real ,out4_imag,
 out5_real ,out5_imag, out6_real ,out6_imag, out7_real ,out7_imag);

  input CLK, RST_N;
  input write, start;
  input wire signed [15:0]  in0_real  ,in0_imag;
  input wire signed [15:0]  in1_real  ,in1_imag;
  input wire signed [15:0]  in2_real  ,in2_imag;
  input wire signed [15:0]  in3_real  ,in3_imag;
  input wire signed [15:0]  in4_real  ,in4_imag;
  input wire signed [15:0]  in5_real  ,in5_imag;
  input wire signed [15:0]  in6_real  ,in6_imag;
  input wire signed [15:0]  in7_real  ,in7_imag;
  output reg ready;
  output wire signed [15:0] out0_real ,out0_imag;
  output wire signed [15:0] out1_real ,out1_imag;
  output wire signed [15:0] out2_real ,out2_imag;
  output wire signed [15:0] out3_real ,out3_imag;
  output wire signed [15:0] out4_real ,out4_imag;
  output wire signed [15:0] out5_real ,out5_imag;
  output wire signed [15:0] out6_real ,out6_imag;
  output wire signed [15:0] out7_real ,out7_imag;

  reg signed [15:0] i0_real, i0_imag;
  reg signed [15:0] i1_real, i1_imag;
  reg signed [15:0] i2_real, i2_imag;
  reg signed [15:0] i3_real, i3_imag;
  reg signed [15:0] i4_real, i4_imag;
  reg signed [15:0] i5_real, i5_imag;
  reg signed [15:0] i6_real, i6_imag;
  reg signed [15:0] i7_real, i7_imag;
  reg [2:0] stage;

  reg signed [15:0] g_real [0:7];   // Declare g_real as an array of registers to store real
part of stage 1 results
  reg signed [15:0] g_imag [0:7];   // Declare g_imag as an array of registers to store imag
part of stage 1 results
  reg signed [15:0] h_real [0:7];   // Declare h_real as an array of registers to store real
part of stage 2 results
  reg signed [15:0] h_imag [0:7];   // Declare h_imag as an array of registers to store imag
part of stage 2 results
  reg signed [15:0] i_real [0:7];   // Declare i_real as an array of registers to store real
part of stage 3 results
  reg signed [15:0] i_imag [0:7];   // Declare i_imag as an array of registers to store imag
part of stage 3 results
```

```
  reg signed [15:0] temp1,temp2,temp3,temp0;
  reg signed [31:0] h5_prime_real;  // 32 bit register to store real part of h[5] * (W8 ^ 1)
  reg signed [31:0] h5_prime_imag;  // 32 bit register to store imag part of h[5] * (W8 ^ 1)
  reg signed [31:0] h7_prime_real;  // 32 bit register to store real part of h[7] * (W8 ^ 3)
  reg signed [31:0] h7_prime_imag;  // 32 bit register to store imag part of h[7] * (W8 ^ 3)

  reg signed [15:0] h5_prime_real_16;  // 16 bit register to store real part of h[5] * (W8 ^
1)
  reg signed [15:0] h5_prime_imag_16;  // 16 bit register to store imag part of h[5] * (W8 ^
1)
  reg signed [15:0] h7_prime_real_16;  // 16 bit register to store real part of h[7] * (W8 ^
3)
  reg signed [15:0] h7_prime_imag_16;  // 16 bit register to store imag part of h[7] * (W8 ^
3)

  always @(posedge CLK) begin
    if (~RST_N) begin
      ready <= 1'b0;
      stage <= 2'b00;
      i_real[0] <= 0; i_imag[0]<=0;
      i_real[1] <= 0; i_imag[1]<=0;
      i_real[2] <= 0; i_imag[2]<=0;
      i_real[3] <= 0; i_imag[3]<=0;
      i_real[4] <= 0; i_imag[4]<=0;
      i_real[5] <= 0; i_imag[5]<=0;
      i_real[6] <= 0; i_imag[6]<=0;
      i_real[7] <= 0; i_imag[7]<=0;
    end
    else begin
      if (start) begin
        g_real[0] <= i0_real + i4_real;
        g_imag[0] <= i0_imag + i4_imag;
        g_real[1] <= i0_real - i4_real;
        g_imag[1] <= i0_imag - i4_imag;
        g_real[2] <= i2_real + i6_real;
        g_imag[2] <= i2_imag + i6_imag;
        g_real[3] <= i2_real - i6_real;
        g_imag[3] <= i2_imag - i6_imag;
        g_real[4] <= i1_real + i5_real;
        g_imag[4] <= i1_imag + i5_imag;
        g_real[5] <= i1_real - i5_real;
        g_imag[5] <= i1_imag - i5_imag;
        g_real[6] <= i3_real + i7_real;
        g_imag[6] <= i3_imag + i7_imag;
        g_real[7] <= i3_real - i7_real;
        g_imag[7] <= i3_imag - i7_imag;
        stage <= 3'b01;
      end
      else if (stage > 0 ) begin
```

```
        case(stage)
          3'b001:begin
            h_real[0] <= g_real[0] + g_real[2];        // h0_real = g0_real + g2_real
            h_imag[0] <= g_imag[0] + g_imag[2];        // h0_imag = g0_imag + g2_imag
            h_real[1] <= g_real[1] + g_imag[3];        // h1_real = g1_real - (j*g3_real)
            h_imag[1] <= g_imag[1] - g_real[3];        // h1_imag = g1_imag - (j*g3_imag)
            h_real[2] <= g_real[0] - g_real[2];        // h2_real = g0_real - g2_real
            h_imag[2] <= g_imag[0] - g_imag[2];        // h2_imag = g0_imag - g2_imag
            h_real[3] <= g_real[1] - g_imag[3];         // h3_real = g1_real + (j*g3_real)
            h_imag[3] <= g_imag[1] + g_real[3];        // h3_imag = g1_imag + (j*g3_imag)
            h_real[4] <= g_real[4] + g_real[6];        // h4_real = g4_real + g6_real
            h_imag[4] <= g_imag[4] + g_imag[6];        // h4_imag = g4_imag + g6_imag
            h_real[5] <= g_real[5] + g_imag[7];        // h5_real = g5_real - (j*g7_real)
            h_imag[5] <= g_imag[5] - g_real[7];        // h5_imag = g5_imag - (j*g7_imag)
            h_real[6] <= g_real[4] - g_real[6];        // h6_real = g4_real - g6_real
            h_imag[6] <= g_imag[4] - g_imag[6];        // h6_imag = g4_imag - g6_imag
            h_real[7] <= g_real[5] - g_imag[7];        // h7_real = g5_real + (j*g7_real)
            h_imag[7] <= g_imag[5] + g_real[7];        // h7_imag = g5_imag + (j*g7_imag)
          end
          3'b010:begin
            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]));
          end
          3'b011:begin
            h5_prime_real <= temp0 * 16'b0000000010110100;
            h5_prime_imag <= temp1 * 16'b0000000010110100;
            h7_prime_real <= temp2 * 16'b0000000010110100;
            h7_prime_imag <= temp3 * 16'b0000000010110100;
          end
          3'b100:begin
            i_real[0] <= h_real[0] + h_real[4];  // i0_real = h0_real + h4_real
            i_imag[0] <= h_imag[0] + h_imag[4];  // i0_imag = h0_imag + h4_imag
            i_real[1] <= h_real[1] + h5_prime_real_16;  // i1_real = h[1]_real + Re((W8 ^
1)*h[5])
            i_imag[1] <= h_imag[1] + h5_prime_imag_16;  // i1_imag = h[1]_imag + Im((W8 ^
1)*h[5])
            i_real[2] <= h_real[2] + h_imag[6];  // i2_real = h2_real - (j*h6_real)
            i_imag[2] <= h_imag[2] - h_real[6];  // i2_imag = h2_imag - (j*h6_imag)
            i_real[3] <= h_real[3] + h7_prime_real_16;  // i3_real = h[3]_real +
Re((W8^3)*h[7])
            i_imag[3] <= h_imag[3] + h7_prime_imag_16;  // i3_imag = h[3]_imag +
Im((W8^3)*h[7])
            i_real[4] <= h_real[0] - h_real[4];  // i4_real = h0_real - h4_real
            i_imag[4] <= h_imag[0] - h_imag[4];  // i4_imag = h0_imag - h4_imag
            i_real[5] <= h_real[1] - h5_prime_real_16;  // i5_real = h[1]_real -
Re((W8^1)*h[5])
            i_imag[5] <= h_imag[1] - h5_prime_imag_16;  // i5_imag = h[1]_imag -
Im((W8^1)*h[5])
```

```
            i_real[6] <= h_real[2] - h_imag[6];  // i6_real = h2_real + (j*h6_real)
            i_imag[6] <= h_imag[2] + h_real[6];  // i6_imag = h2_imag + (j*h6_imag)
            i_real[7] <= h_real[3] - h7_prime_real_16;  // i7_real = h[3]_real -
Re((W8^3)*h[7])
            i_imag[7] <= h_imag[3] - h7_prime_imag_16;  // i7_imag = h[3]_imag -
Im((W8^3)*h[7])
            ready <= 1'b1;
          end
        endcase

        if(stage == 3'b100) begin
          stage <= 2'b00;
        end
        else begin
          stage <= stage + 2'b01;
        end

      end
      else begin
        if (write) begin
          i0_real<= in0_real;  i0_imag <= in0_imag;
          i1_real<= in1_real;  i1_imag <= in1_imag;
          i2_real<= in2_real;  i2_imag <= in2_imag;
          i3_real<= in3_real;  i3_imag <= in3_imag;
          i4_real<= in4_real;  i4_imag <= in4_imag;
          i5_real<= in5_real;  i5_imag <= in5_imag;
          i6_real<= in6_real;  i6_imag <= in6_imag;
          i7_real<= in7_real;  i7_imag <= in7_imag;
        end
        ready <= 1'b0;
      end
    end
  end

always@(*) begin
  h5_prime_real_16 <= h5_prime_real[23:8];
  h5_prime_imag_16 <= h5_prime_imag[23:8];
  h7_prime_real_16 <= h7_prime_real[23:8];
  h7_prime_imag_16 <= h7_prime_imag[23:8];
end

  assign out0_real = i_real[0]; assign out0_imag = i_imag[0];
  assign out1_real = i_real[1]; assign out1_imag = i_imag[1];
  assign out2_real = i_real[2]; assign out2_imag = i_imag[2];
  assign out3_real = i_real[3]; assign out3_imag = i_imag[3];
  assign out4_real = i_real[4]; assign out4_imag = i_imag[4];
  assign out5_real = i_real[5]; assign out5_imag = i_imag[5];
  assign out6_real = i_real[6]; assign out6_imag = i_imag[6];
  assign out7_real = i_real[7]; assign out7_imag = i_imag[7];
```

```
endmodule
```

FFT testbench:

```
`include "fft.v"
`timescale 1ns/1ps

module fft_tb();

    parameter CLOCK_PERIOD = 5; // 10 MHz clock

    reg sys_clk, sys_rst_n;
    reg write, start;

    reg signed [15:0]  in0_real  ,in0_imag;
    reg signed [15:0]  in1_real  ,in1_imag;
    reg signed [15:0]  in2_real  ,in2_imag;
    reg signed [15:0]  in3_real  ,in3_imag;
    reg signed [15:0]  in4_real  ,in4_imag;
    reg signed [15:0]  in5_real  ,in5_imag;
    reg signed [15:0]  in6_real  ,in6_imag;
    reg signed [15:0]  in7_real  ,in7_imag;
    wire ready;

    wire signed [15:0] out0_real ,out0_imag;
    wire signed [15:0] out1_real ,out1_imag;
    wire signed [15:0] out2_real ,out2_imag;
    wire signed [15:0] out3_real ,out3_imag;
    wire signed [15:0] out4_real ,out4_imag;
    wire signed [15:0] out5_real ,out5_imag;
    wire signed [15:0] out6_real ,out6_imag;
    wire signed [15:0] out7_real ,out7_imag;

    fft DUT(.CLK(sys_clk), .RST_N(sys_rst_n), .write(write), .start(start),
.in0_real(in0_real) , .in0_imag(in0_imag), .in1_real(in1_real), .in1_imag(in1_imag),
 .in2_real(in2_real)  ,.in2_imag(in2_imag), .in3_real(in3_real)  ,.in3_imag(in3_imag),
.in4_real(in4_real)  ,.in4_imag(in4_imag), .in5_real(in5_real)  ,.in5_imag(in5_imag),
 .in6_real(in6_real)  ,.in6_imag(in6_imag), .in7_real(in7_real)  ,.in7_imag(in7_imag),
.ready(ready), .out0_real(out0_real) ,.out0_imag(out0_imag), .out1_real(out1_real),
 .out1_imag(out1_imag), .out2_real(out2_real) ,.out2_imag(out2_imag), .out3_real(out3_real)
,.out3_imag(out3_imag), .out4_real(out4_real) ,.out4_imag(out4_imag),
 .out5_real(out5_real) ,.out5_imag(out5_imag), .out6_real(out6_real) ,.out6_imag(out6_imag),
.out7_real(out7_real) ,.out7_imag(out7_imag));

    // VCD dump
    initial begin
        $dumpfile("fft.vcd");
        $dumpvars(0, DUT);
    end
```

```
    // System clock generator
    always begin
        #(CLOCK_PERIOD/2) sys_clk = ~sys_clk;
    end

    // Initial Condition
    initial begin
        sys_clk = 1'b0; sys_rst_n = 1'b0;
        write = 1'b0; start = 1'b0;
        in0_real=16'h00_00  ;in0_imag = 0;
        in1_real=16'h01_00  ;in1_imag = 0;
        in2_real=16'h02_00  ;in2_imag = 0;
        in3_real=16'h03_00  ;in3_imag = 0;
        in4_real=16'h04_00  ;in4_imag = 0;
        in5_real=16'h05_00  ;in5_imag = 0;
        in6_real=16'h06_00  ;in6_imag = 0;
        in7_real=16'h07_00  ;in7_imag = 0;
        #CLOCK_PERIOD sys_rst_n = 1'b1;
        #CLOCK_PERIOD write = 1'b1;
        #CLOCK_PERIOD start = 1'b1;
        #CLOCK_PERIOD start = 1'b0;
        #(CLOCK_PERIOD/2) $display("START OF SIMULATION (Time: %g ns)", $time);
        #10000;
        $finish;
    end

endmodule
```

---

synth.ys

```
# read design
read_verilog fft.v
hierarchy -check
hierarchy -top fft
#flatten

# high level synthesis
proc; opt; clean
fsm; opt clean
memory; opt; clean

# low level synthesis
techmap; opt; clean

# map to target architecture
dfflibmap -liberty stdcells.lib
abc -liberty stdcells.lib

# split larger signals
```

28-11-2023

splitnets -ports; opt; clean

write synthesis output
write_verilog synth.v
write_spice synth.sp

#print synthesis reports
stat
stat -liberty stdcells.lib

show -format svg -prefix /home/ujjwal/Desktop/ESDCS/ESDCS_project/syme fft

---

readme file:

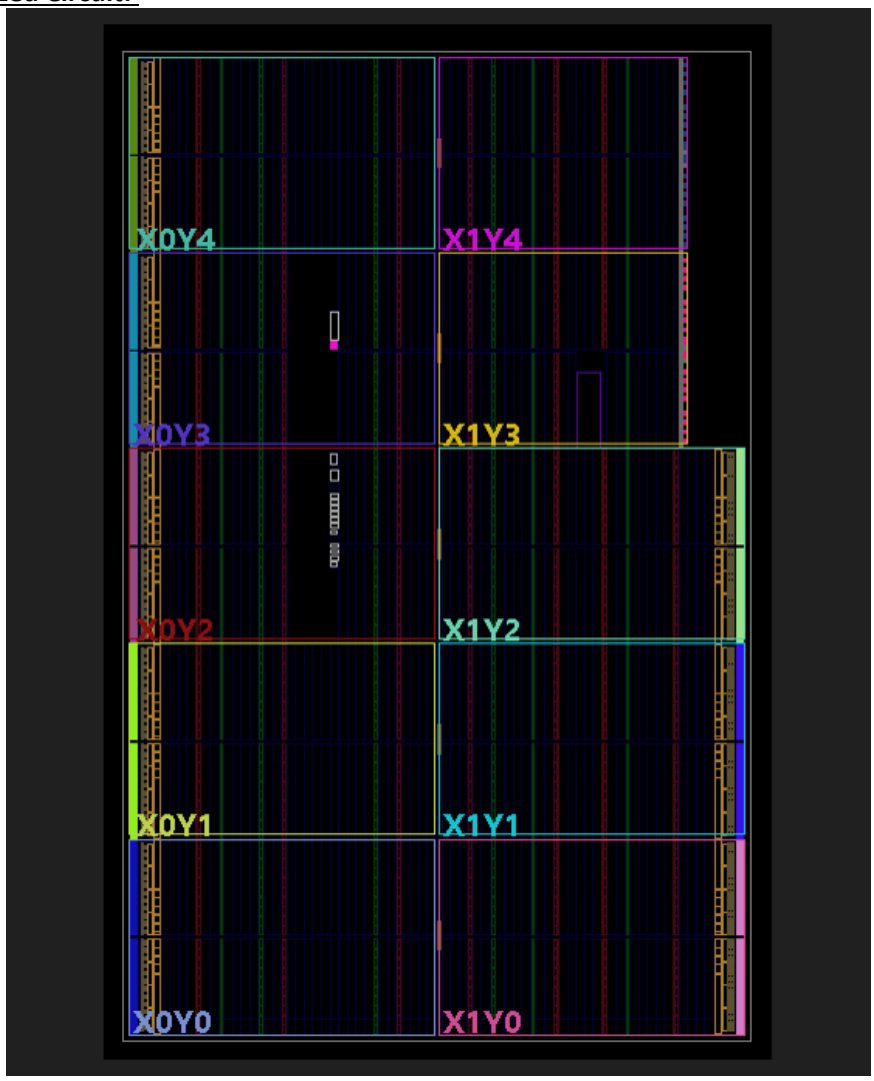
```
# ESDCS_FFT_Project
Project for the partial grading of Course: Efficient and Secure Digital Systems

## Project Objectives
1. Design a digital circuit for Fast Fourier Transform (FFT) in Verilog
2. Create an appropriate test bench in Verilog to test the design
3. Simulate the design using Icarus Verilog and visualize the simulated VCD waveforms using GTKWave to verify functionality
4. Synthesize the design using Yosys with the Nangate Open Cell 45nm standard cell library in the typical TT corner
5. Analyze timing and power of the implementation using OpenSTA

## Project Statement

- Input 𝑥 = {𝑥0, 𝑥1, ⋯ , 𝑥7}
- Output 𝑋 = {𝑋0, 𝑋1, ⋯ , 𝑋7}
- Output 𝑋 is the 8-point FFT of input x
- Each input element 𝑥𝑚 (for 𝑚 = 0,1, … , 7 ) is a complex number with real and imaginary parts each represented as a signed 16-bit fixed point quantity with 1 sign bit, 7 bits for decimal part and 8 bits for fractional part
- Each output element 𝑋𝑘 (for 𝑘 = 0,1, … , 7 ) is a complex number with real and imaginary parts each represented as a signed 16-bit fixed-point quantity with 1 sign bit, 7 bits for decimal part and 8 bits for fractional part
- Additional Signals
    - |CLK |Input| Clock| signal|
      |----|-----|------|-------|
      | RST_N | Input | Active-low | reset signal |
      | write | Input | Input | write signal |
      | start | Input | FFT computation | start signal |
      | ready | Output | FFT computation | done signal  |
```

Synthesized Circuit:



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