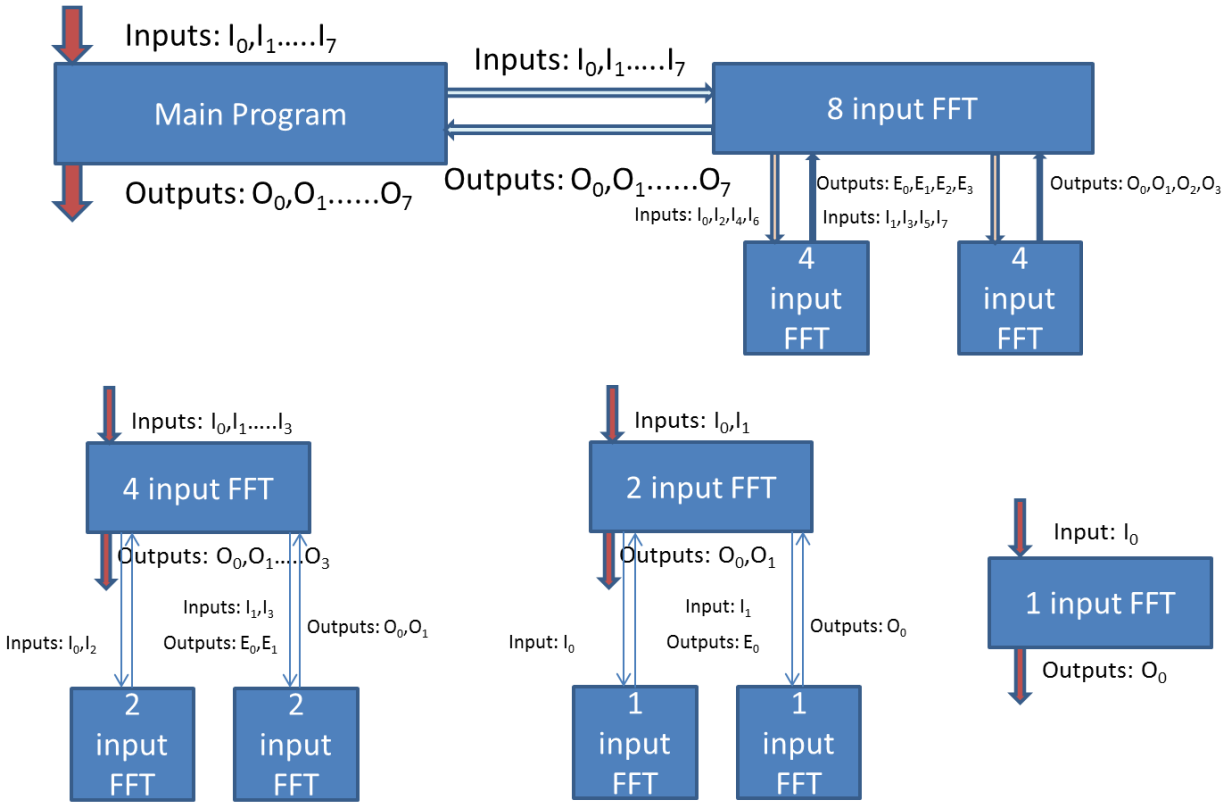
CS 288 Project Report

Group 18

Ayush Kanodia (# 110050049)
Dhanesh Kumar (# 110050021)
Mridul Garg (# 110050030)
Nishit Bhandari (# 110050026)
Rajlaxmi Sahu (# 110050087)

This is the report / documentation for the CS 288 Project.

FFT Block Design



The above was the initial block design submitted. The implementation has been identical to the design submitted. The only point of departure has been that the one input FFT block has been implemented within the 2 input FFT block as that block really does no calculation. Also, the butterfly components within the individual 8, 4, and 2 point FFT inputs have not been shown for simplicity

Submitted Components

- ➔ Input Output Module
- ➔ Point 8 FFT Module
- ➔ Point 4 FFT Module
- ➔ Point 2 FFT Module
- ➔ User Constraint file
- ➔ Test Bench (for the Point 8 FFT Module). This can be used if only simulation needs to be tried.

Description / Documentation of individual components:

The Input Output Buffer: This component takes all the inputs sequentially. It then invokes the 8 pt FFT module to find the answer. It then sequentially shows the output on the LEDs, and after this, goes back to the first stage to begin taking inputs.

Meaning of inputs and outputs:

For each input number of the FFT, we take a 16 bit number (in two steps).

1. The first eight bits represent the real part of input, and the next eight bits represent the imaginary part of input.
2. For the real part of input, we take eight bits as input.
3. The first five bits represent the number input before the decimal point.
4. The next three bits represent the number input after the decimal point.
5. The number in input is assumed to be in **two's complement** format.
6. Hence, if the first bit of input is 1, the number is negative, else it is positive.
7. The exact same convention is followed for the complex part of input.

For a particular input, we take only 8 bits of input for real and 8 bits for the imaginary part. Now, we convert this to 16 bits for real and 16 for the imaginary part. This is done to eliminate the possibility of overflows. Then the 8 pt FFT module works with these 32 bits for each input. Since all numbers, inputs and outputs, are represented in 2's complement format, we perform all calculations in VHDL using the provided **SIGNED** library. The conversion of the real part from 8 bits to 16 bits is as follows

1. The first bit of the eight bit input becomes the first four bits of the 16 bit input. This is because we want to conserve "sign".
2. The next four bits of the eight bit input become the next four bits (bit 5 to bit 8) of the 1 bit input. This conserves the part before the decimal point.
3. The last three bits of the eight bit input become the next three bits (bit 9 to 11) of the 1 bit input. This conserves the part after the decimal point.
4. The last 5 bits of the 16 bit input are set to 0, as their value must be 0 (The last 5 bits of the after the decimal point).

Hence, we achieve the conversion of the input from 8 bits to 16 bits for the real part. We do the same for the complex part.

After this is achieved, we combine the real and imaginary parts into one input vector which is 32 bits long. The first 16 bits of this are real, and the last 16 bits imaginary.

We do this for all 8 inputs. **The outputs are obtained in the exact same format.** The ways these will be displayed on the board have been detailed further.

Inputs:

clk : This is the input for the onboard 100 Mhz clock

Input : This is the eight bit input sequence which will be taken from the board, from the DIP switches.

In_tick: This is the input which is mapped to a button on the board. We use these button presses to mark instants in time where inputs are sampled and out displays change

Outputs:

Output: This is the eight bit output sequence which will be displayed on the LEDs on the board.

Procedure:

This module acts like the input output buffer. We first take an input, wait for an intick button press, take another input, wait for an intick button press, and so on. Two intick button presses amount to one input taken. This is because, we first take eight bits for the real part of the input, and the next eight bits for the imaginary part of the input. Thus, to take all inputs, we need to press the intick 16 times. Thus we use the following procedure.

1. Place the real part of the first input on the DIP switches
2. Press Intick
3. Place the complex part of the input on the DIP switches
4. Press Intick
5. Repeat Steps 1 to 4 for each of the next seven inputs

This way we take all 8 input vectors as input. **In this process, at all times, the 8 LEDs display the value of the last input that was taken, in Binary.**

Now, once we do this, we temporarily reach a state in which the 8 LEDs display all ones. This state means that on the next INTICK press, we will start displaying the outputs. Now, to see the output,

1. Press INTICK
2. You will now see the **8 Most significant bits** of the **real** part of the first input (before the decimal point) on the LEDs
3. Press INTICK
4. You will now see the **8 Least significant bits** of the **real** part of the first input (after the decimal point) on the LEDs
5. Press INTICK
6. You will now see the **8 Most significant bits** of the **imaginary** part of the first input (before the decimal point) on the LEDs
7. Press INTICK

8. You will now see the **8 Least significant bits** of the **imaginary** part of the first input (after the decimal point) on the LEDs

Now, to see the output of all the eight vectors, you need to repeat the above procedure 8 times. At the end of this, the LEDs will again show all 1s, which means that it is now ready to perform the entire set of calculations again, for a fresh set of inputs.

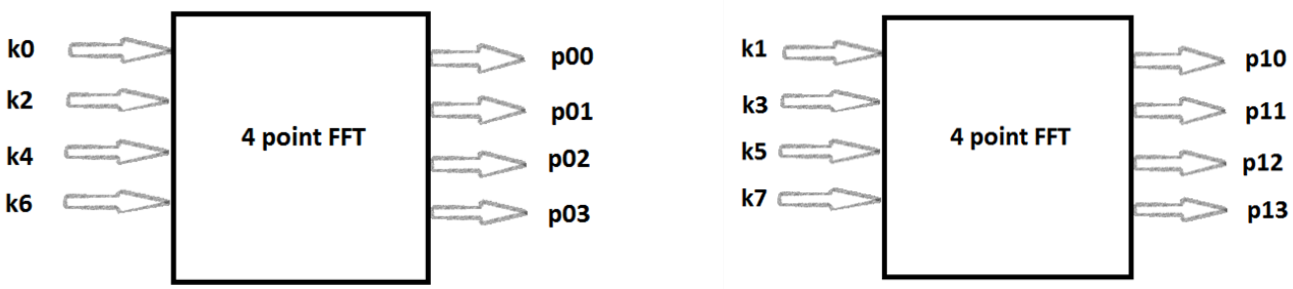
You now start with the input procedure again.

Next, we explain the individual FFT modules.

We have created three modules for calculating FFT namely **pt8fft**, **pt4fft** and **pt2fft**.

1. **pt8fft** – It accepts 8 inputs each of size 32 bits. The even indexed inputs and odd indexed inputs are sent to different pt4fft. The pt4fft module returns the fft outputs of the even indexed and odd indexed inputs correspondingly. The output of pt8fft is now generated in accordance to the following equation : (note that pij is the (j+1)th output of (i+1)th pt4fft)
    - first output : p00 + twiddle_factor$_0$ * p10 , where twiddle_factor$_0$ = e^{i*2*pi}
    - second output : p01 + twiddle_factor$_1$ * p11 , where twiddle_factor$_1$ = $e^{-i*pi/4}$
    - third output : p02 + twiddle_factor$_2$ * p12, where twiddle_factor$_2$ = $e^{i*pi/2}$
    - fourth output : p03 + twiddle_factor$_3$ * p13 , where twiddle_factor$_3$ = $e^{i*3*pi/4}$
    - fifth output : p00 - twiddle_factor$_0$ * p10
    - sixth output : p01 - twiddle_factor$_1$ * p11
    - seventh output : p02 - twiddle_factor$_2$ * p12
    - eighth output : p03 - twiddle_factor$_3$ * p13

It may be noted that during multiplication, when we multiply two 16 bit vectors, **we get a 32 bit output**. For this, we take the **middle 16 bits** as our answer. We can do this without any loss in accuracy, because, to begin with, in the first place, we had started out by increasing the size of input to the **size of its square** or **double the original number of bits** (8 bits to 16 bits for the real as well as imaginary parts individually), which **eliminates** all possibilities of overflow errors.

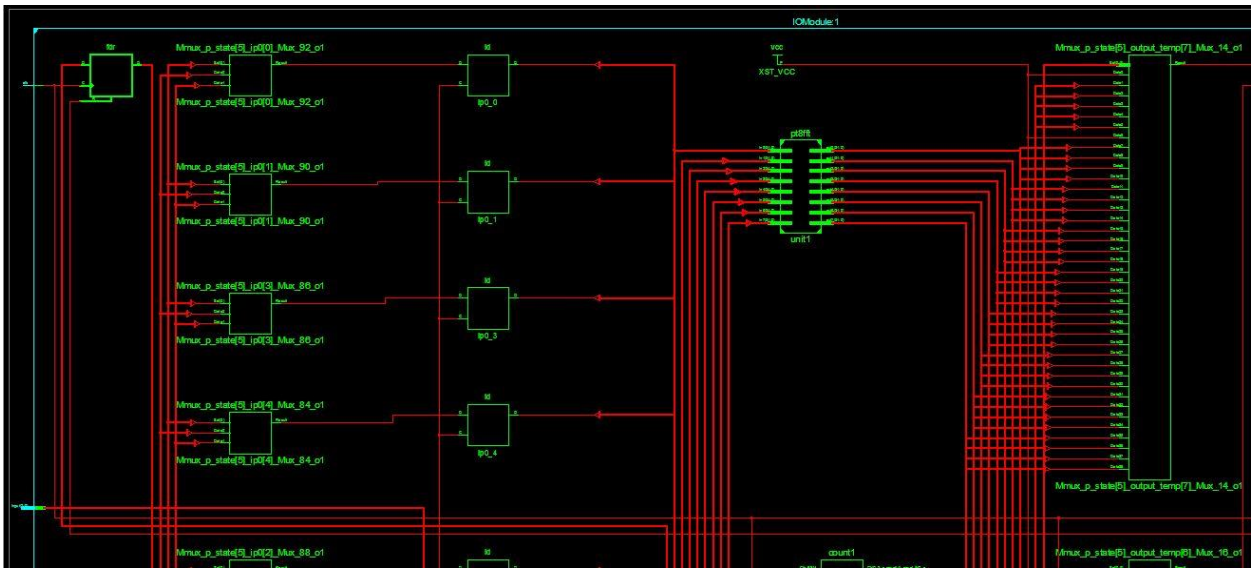


2. **pt4fft** – It accepts 4 inputs each of size 32 bits. The even indexed inputs and odd indexed inputs are sent to different pt2fft. The pt2fft module returns the fft outputs of the even indexed and odd indexed inputs correspondingly. The output of pt4fft is now generated in accordance to the following equation : (note that tji is the (j-1)th output of ith pt2fft)
    - first output : t01 + twiddle_factor$_0$ * t11 , where twiddle_factor$_0$ = e^{i*2*pi}
    - second output : t02 + twiddle_factor$_1$ * t12 , where twiddle_factor$_1$ = $e^{-i*pi/2}$
    - third output : t01 - twiddle_factor$_2$ * t11
    - fourth output : t02 - twiddle_factor$_3$ * t12

    The block diagram for pt4fft is similar to the above block diagram
3. **pt2fft** –
    - first output : first input + second input
    - second output : first input - second input

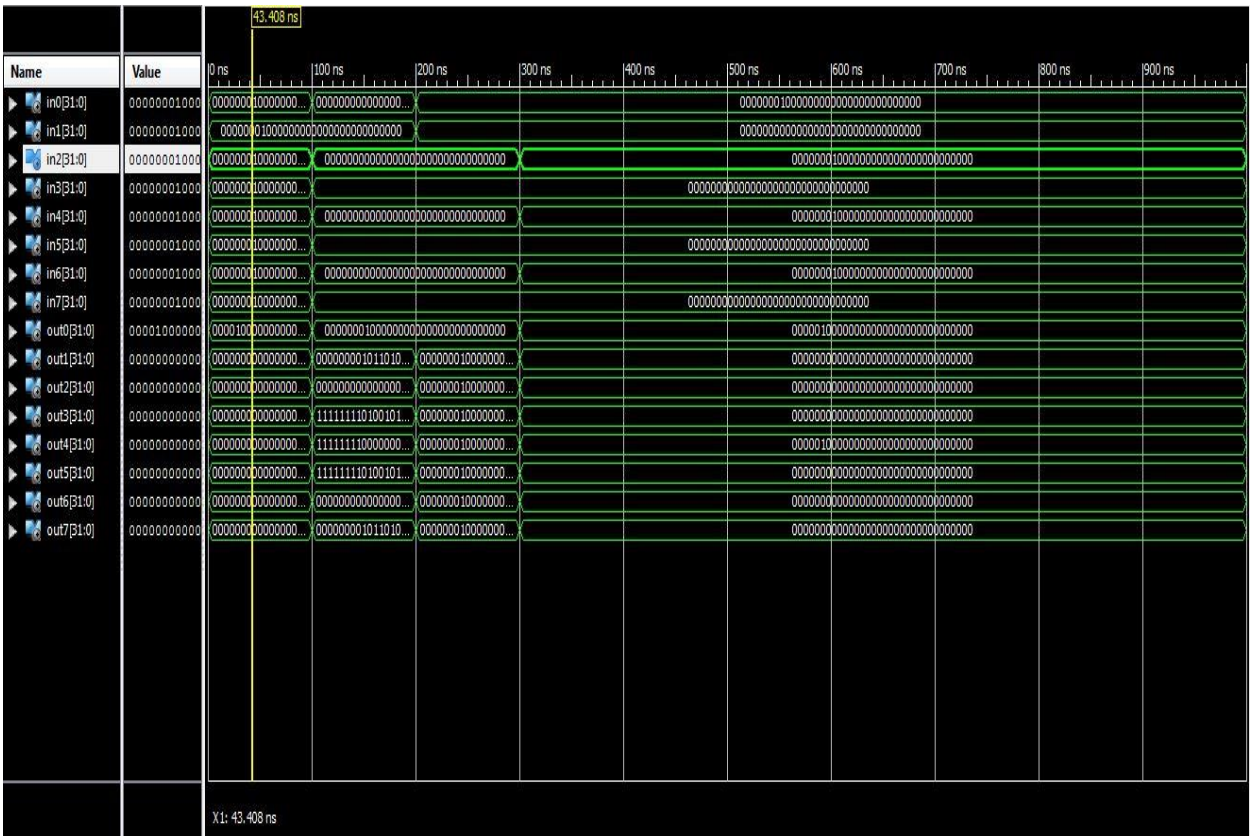
A snapshot of the RTL Block Diagram

Here we give a snapshot of the auto generated RTL block diagram by ISE.



The project simulation

A snapshot of the testbench simulation



Testbench

We have given 4 inputs and taken their outputs in the attached test bench.

First set of inputs (In decimal) :

- in0 : 1
- in1 : 1
- in2 : 1
- in3 : 1
- in4 : 1
- in5 : 1
- in6 : 1
- in7 : 1

Its fft output :

- out0 : 8
- out1 : 0
- out2 : 0
- out3 : 0
- out4 : 0
- out5 : 0
- out6 : 0
- out7 : 0

Second set of inputs (in decimal) :

- in0 : 0
- in1 : 1
- in2 : 0
- in3 : 0
- in4 : 0
- in5 : 0
- in6 : 0
- in7 : 0

Its fft output :

- out0 : 1
- out1 : .7070 + i *-.7070
- out2 : i
- out3 : -.7070 + i * -.7070
- out4 : -1
- out5 : -.7070 + i * .7070

- out6 : -i
- out7 : .7070 + i * .7070

Third set of inputs (in decimal) :

- in0 : 1
- in1 : 0
- in2 : 0
- in3 : 0
- in4 : 0
- in5 : 0
- in6 : 0
- in7 : 0

Its fft output :

- out0 : 1
- out1 : 1
- out2 : 1
- out3 : 1
- out4 : 1
- out5 : 1
- out6 : 1
- out7 : 1

Fourth set of inputs (in decimal) :

- in0 : 1
- in1 : 0
- in2 : 1
- in3 : 0
- in4 : 1
- in5 : 0
- in6 : 1
- in7 : 0

Its fft output :

- out0 : 4
- out1 : 0
- out2 : 0
- out3 : 0
- out4 : 4
- out5 : 0

- out6 : 0
- out7 : 0