Verilog Final Project

FFT Butterfly Diagram

EE 47200

Professor Pekcan

Chaim Frankel

Introduction:

An N-point signal, x[n], will have an N-point Fourier Transform, X[k], where N is a constant integer, usually chosen as a power of 2. The formula for X[k] is

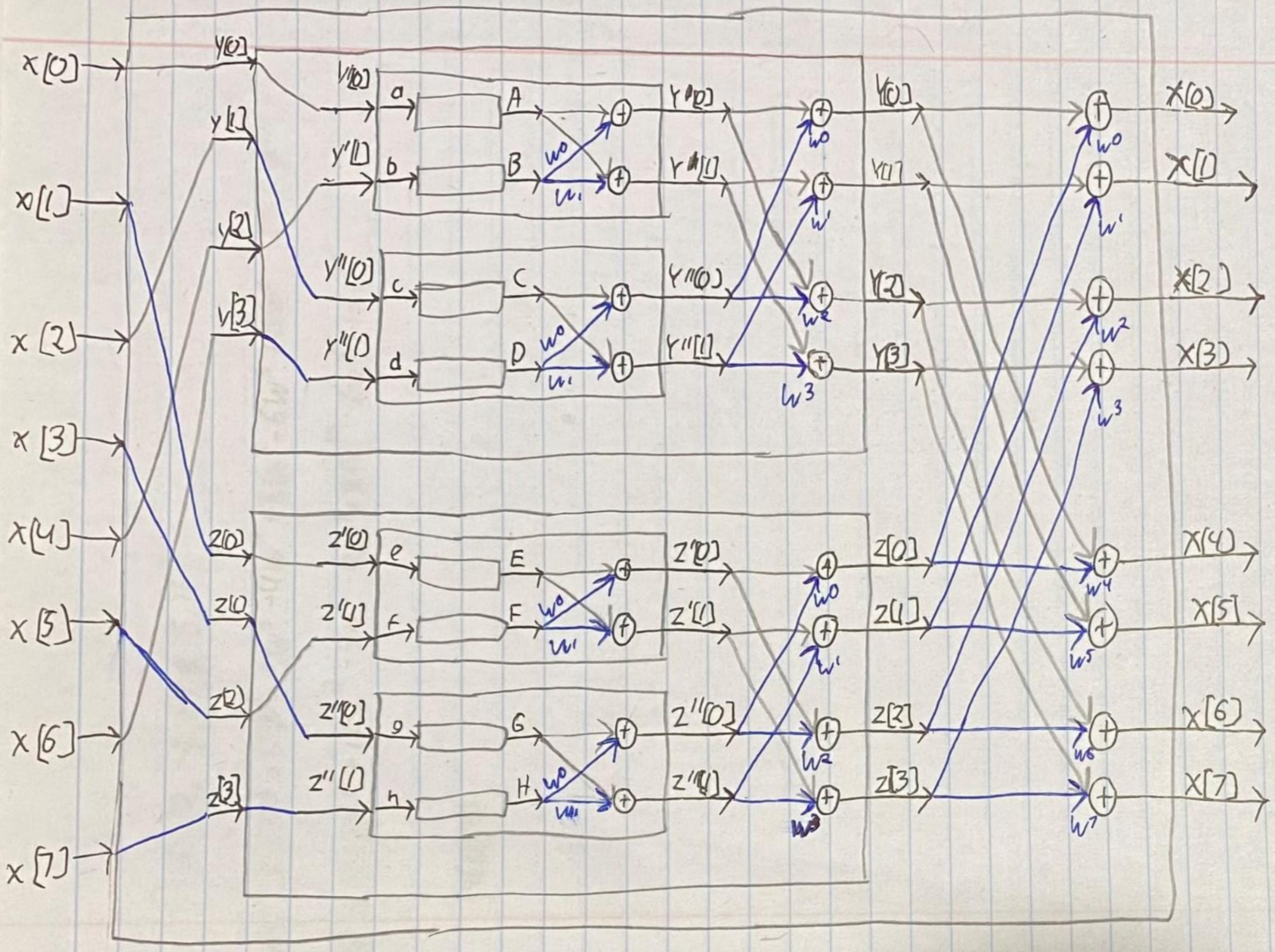
$$X[k] = \sum_{n=0}^{N-1} W_N^{kn} x[n]$$

where $W_N = e^{-j2\pi/N}$. However, this formula requires N^2 multiplications and $N^2 - N$ additions. If the sole goal is to get the Fourier Transform, we don't really care about how many computer operations it takes. However, in classes like these, we are interested in reducing the number of computer operations. Doing so saves us time, hardware, and money.

Enter the Cooley-Tukey algorithm and butterfly diagram. This algorithm breaks the input x[n] of length N, into 2 smaller inputs, y[n] and z[n], of length N/2. You can view it as 2 blocks, both of which are half the size of the first block.

y[n] and z[n] are further split into blocks of size N/4. This procedure continues until the signal length is 1, i.e. until there are N blocks, each of which has a single input (and therefore a single output). Likewise, the output, X[k], of length N, branches from outputs Y[k] and Z[k] of length N/2. Y[k] and Z[k] can be traced to the N aforementioned blocks containing a single input and output. It should be noted that the outputs are multiplied by W_N^k. See my attached butterfly diagram on the next page. Note that there are 3 columns of W's. In the rightmost column, N = 8, so $W_8 = e^{-j2\pi/N} = e^{-j\pi/4}$. In the middle column, N = 4, so we have $W_{N/2} = W_4$. Note from the formula of W_N that $W_{N/2} = W_N^2$, so $W_4 = e^{-j\pi/2} = -j$. Finally, in the leftmost column, we have N = 2, so $W_2 = W_4^2 = e^{-j\pi/2} = -1$.

This algorithm requires $N*\log_2 N$ additions and multiplications. If you're dealing with a signal of large N, say N = 1024, the first will require over 10^6 additions and multiplications. The Cooley-Tukey algorithm requires just over 10^4 additions and multiplications, so it is over 100 times faster! This is why the algorithm is known as the Fast Fourier Transform.



Ws should all be labeled W_N

Project:

The wiring structure in Verilog follows the diagram above. There are 8 output registers, representing the 8 capital X's, 8 input wires representing the 8 lowercase x's, as well as an input wire for the clock and active LOW reset. There are many internal registers declared inside the Verilog module, which represent the wiring inside the block. As in the butterfly diagram, the Verilog module contains arrays for y, z, y$, y$$, z$, z$$, Y, Z, Y$, Y$$, Z$, and Z$$. (Note that the character ' in the diagram is swapped for the $ symbol in Verilog, as the $ symbol is valid for variables in Verilog, while the ' symbol isn't.) The full code, with comments, is attached separately. Here are some bullet points about the code:

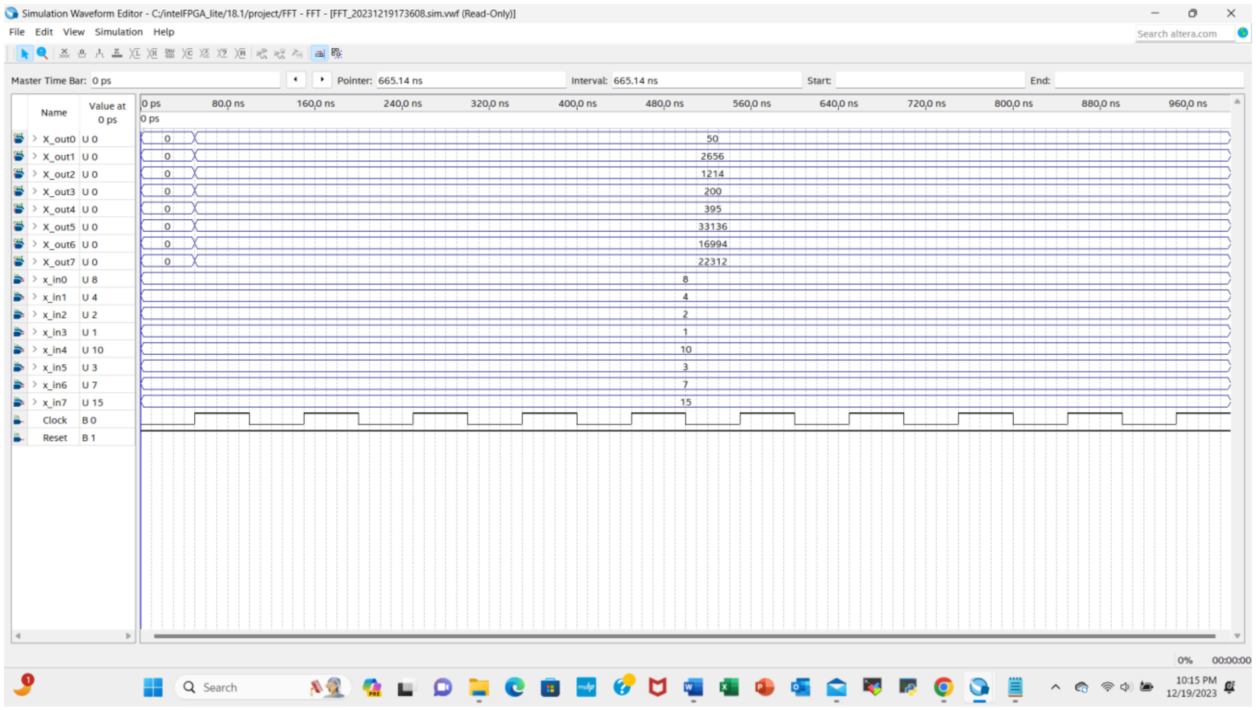
- In real life, the Cooley-Tukey algorithm only works when $W_N = e^{-j2\pi/N}$ (this is due to the periodicity of $e^{-j2\pi/N}$). This is a complex number whose real and imaginary parts are typically floating decimal points, depending on the value of N. For N = 8, $W_N = 0.7071 - 0.7071j$. However, Quartus only supports integers, not reals. To write this program properly W_N must be broken up into values of two parts: real and imaginary. The real and imaginary values must further be broken into integers of two parts, mantissa and exponent. In the interest of keeping this project simpler, W_N

is an integer, with $W_8 = 2$. Although the math will yield different results, the wiring and algorithm remain the same.

- I have included C++ code which implements the algorithm correctly, and returns the correct results. I defined a class called ComplexNumber which has two variables of type double, one for real and one for imaginary. This class also contains functions which allow complex numbers to be added, multiplied, and raised to a power. Although C++ has a complex library, I made my own class.
- The input variables begin with lowercase letters, while the output variables begin with their corresponding capital letters. In the C++ code, the input variables are of type int, while the output variables are of type ComplexNumber.
- In Verilog, the input wires are of length 4 bits, while the output registers are of length 26 bits (I would’ve done 32 bits, but Quartus said there isn’t enough space. Perhaps it would require a computer with more RAM). Since the outputs can be to the order of 2^{20}, it is necessary that the output registers be of a large length.
- Since there are 8 inputs and 8 outputs, it would make most sense to make the inputs and outputs an array of size 8. However, the inputs and outputs

contain multiple bits, and Verilog won't allow 2-D arrays in I/O ports. Instead, I created 8 separate inputs of length 4, and 8 separate outputs of length 26. I then created 2-D arrays as follows: reg [3:0] x [0:7] and reg [25:0] X [0:7]. For the inputs, I set x[0] = 1st input, x[1] = 2nd input, … , x[7] = 8th input. For the outputs, I set 1st output = X[0], 2nd output = X[1], … , 8th output = X[7].

Here is a screenshot of a sample waveform of the project.


Simulation Waveform Editor - C:/intelFPGA_lite/18.1/project/FFT - FFT - [FFT_20231219173608.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com
Master Time Bar: 0 ps
Pointer: 665.14 ns
Interval: 665.14 ns
Start:
End:
0 ps 80,0 ns 160,0 ns 240,0 ns 320,0 ns 400,0 ns 480,0 ns 560,0 ns 640,0 ns 720,0 ns 800,0 ns 880,0 ns 960,0 ns
Search
10:15 PM
12/19/2023

```
Name     Value at 0 ps
X_out0   U 0    0 | 50
X_out1   U 0    0 | 2656
X_out2   U 0    0 | 1214
X_out3   U 0    0 | 200
X_out4   U 0    0 | 395
X_out5   U 0    0 | 33136
X_out6   U 0    0 | 16994
X_out7   U 0    0 | 22312
x_in0    U 8    8
x_in1    U 4    4
x_in2    U 2    2
x_in3    U 1    1
x_in4    U 10   10
x_in5    U 3    3
x_in6    U 7    7
x_in7    U 15   15
Clock    B 0
Reset    B 1
```

<u>Conclusion:</u>

After some debugging, I was able to get the Verilog module working. There was one bug, however, that I couldn't fix. The waveform would not return large values, i.e. 2^{20}, even though the outputs are each 26 bits long. See the commented for loop at the bottom of the code. As a student with an interest in the field of Digital Signal Processing, this project gave me some insight into the hardware aspect of DSP.