Implementing an 8-point FFT using Verilog

Yellur Vishal Rao, 200907060, Electronics and Communication, Manipal Institute of Technology, Manipal, India

Dhruv Davuluri, 200907196, Electronics and Communication, Manipal Institute of Technology, Manipal, India

Twisha Awasthy, 200907202, Electronics and Communication, Manipal Institute of Technology, Manipal, India

Abhay H, 200907354, Electronics and Communication, Manipal Institute of Technology, Manipal, India

***Abstract*- In recent times, the significance of Digital Signal Processing (DSP) algorithms has seen a remarkable rise, with the Discrete Fourier Transform (DFT) and the Fast Fourier Transform (FFT) emerging as two crucial techniques. The Fast Fourier Transform (FFT) can be classified as an algorithm that efficiently computes the Discrete Fourier transform (DFT). The FFT is a key signal-processing algorithm employed in diverse applications, ranging from communication systems to audio and image processing. The implementation is tailored for Field-Programmable Gate Arrays (FPGAs), emphasising real-time processing capabilities. The SystemVerilog Hardware Description Language is implemented and simulated using the EDA playground.**

Keywords — Verilog, Fast Fourier transforms (FFT), Radix-2 DIT-FFT, Decimation in Time FFT (DIT – FFT), Finite State Machine(FSM), Datapath

I. INTRODUCTION

The evolution of DSP algorithms has become a cornerstone of today’s technological advancements. The DFT and FFT algorithms stand out as crucial techniques due to their various applications. DFT is used commonly in DSP tasks, such as convolution, correlation and filtering. It shows a relationship between a time–domain signal and its frequency–domain counterpart.

$$X(k) = \sum_0^{N-1} x(n)e^{-i2nk/N}, k = 0,1,\ldots N-1$$

Which can also be represented as:

$$X(k) = \sum_0^{N-1} x(n)W_N^{kn} \quad ,k = 0,1,\ldots.N-1$$

DFT doesn’t make use of the twiddle factor properties and therefore we are unable to compute the algorithm due to its inefficiency directly.

The main advantage of the FFT is its highly reduced computational complexity compared to the DFT algorithm. While the DFT has a time complexity of O(N)^2, the Cooley-Tukey FFT algorithm reduces this to O(NlogN), making it much more efficient for larger transform sizes. Computing FFT takes (N/2)log_2N multiplication calculations and Nlog_2N additions.

Decimation in time(DIT) and Decimation in Frequency(DIF) are the two forms of FFT that make use of the butterfly diagram for computations.

II. DESIGN METHODOLOGY

The Verilog code is structured following the Cooley-Tukey algorithm for the 8-point FFT. The design incorporates a single module for all butterfly computations done manually, twiddle factor values saved in a separate seat of registers, and overall an entirely combinational design was implemented. A function for the computation of imaginary number multiplication has also been specially made to enable this implementation. Special emphasis is placed on optimising resource utilisation and performance to meet real-time processing requirements.

Here, a DIT-FFT approach is used to implement the 8-point FFT. In DIT, an N-point transform is broken down into 2 N/2 point transforms, which are then further broken down into N/4 and so on and so forth. This computation is recursively performed until a 2-point DFT is obtained. This is illustrated in the butterfly diagram in Fig.1 as shown below.

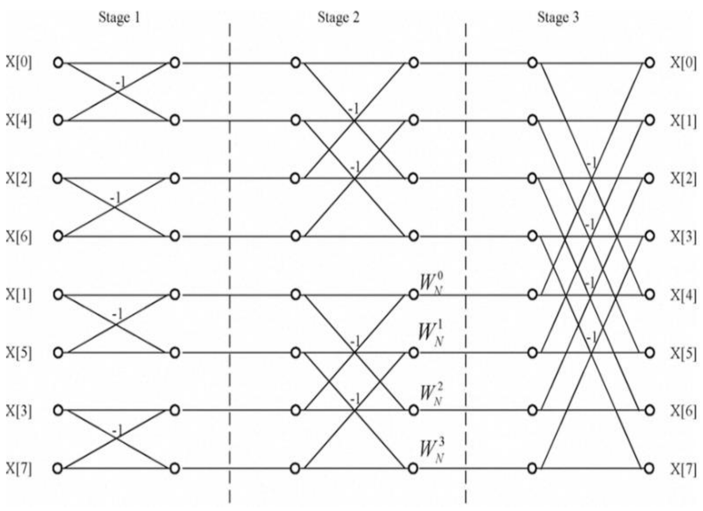


Fig.1 Butterfly diagram of an 8-point DIT-FFT

The signal decomposition process employed by the FFT algorithm in the time domain is illustrated in Fig.2 as shown below. In this example, the decomposition of a 16-point signal requires four stages. In the initial stage, the 16-point signal is split into two signals, each comprising 8 points. Subsequently, each of these 8-point signals undergo further decomposition into two signals, each comprising 4 points. This results in four 4-point signals, marking the second stage. This recursive decomposition continues iteratively until a single point is attained, as depicted. Throughout this decomposition, the signal is segregated into its odd and even samples.

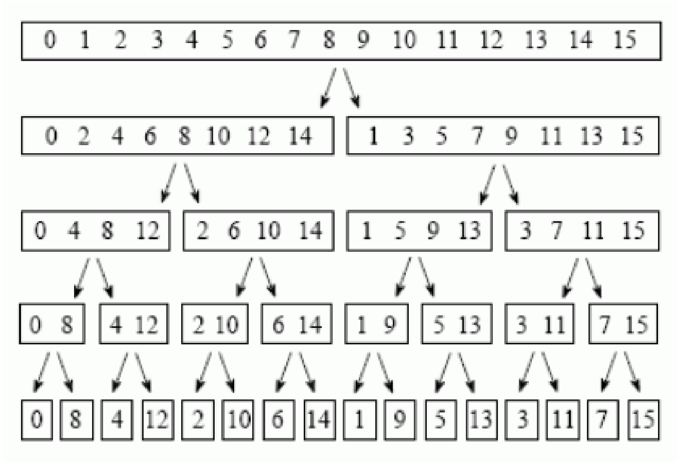
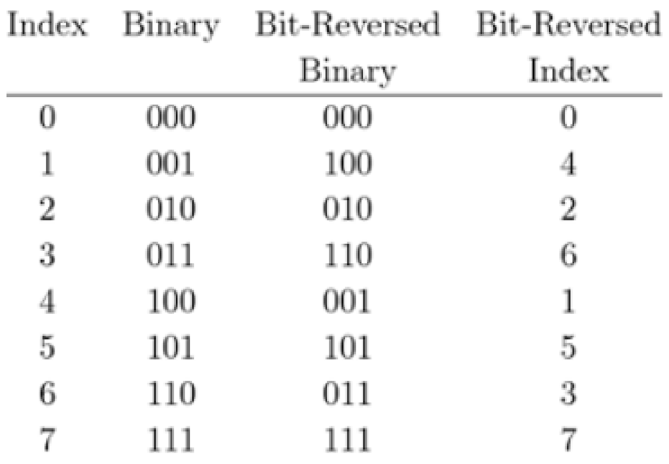


Fig. 2 Decomposition of time domain FFT signal

The described decomposition is essentially a rearrangement of the samples within a signal. The required pattern for FFT is as illustrated in Fig. 3 below. On the left side are the numbers corresponding to the original signal along with their binary representations. Meanwhile, on the right is its bit-reversed pattern necessary for FFT and is also accompanied by its binary representation.



Index	Binary	Bit-Reversed Binary	Bit-Reversed Index
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Fig. 3 Bit Reversing

III. IMPLEMENTATION

The proposed 8-point FFT is implemented in SystemVerilog with an architecture or datapath as shown in Fig. 4 below. Read_mem and sel are 3-bit input lines to select a register between X_o to X_7. Read_twd is a 2-bit line that is assigned to select twiddle factors from W_n^0 to W_n^3. A demux write_mem is used to update the values with the help of a buffer. This datapath is an exact representation of the butterfly diagram for the proposed 8-point FFT.

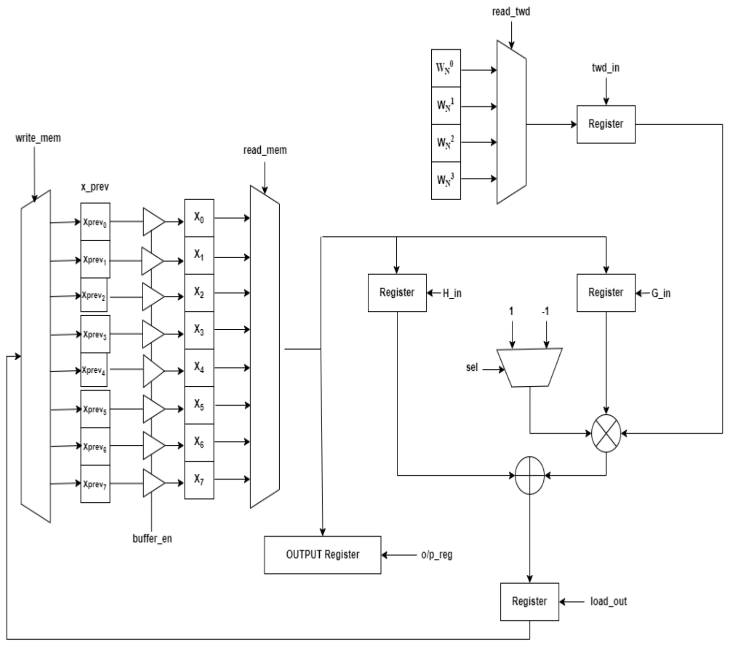
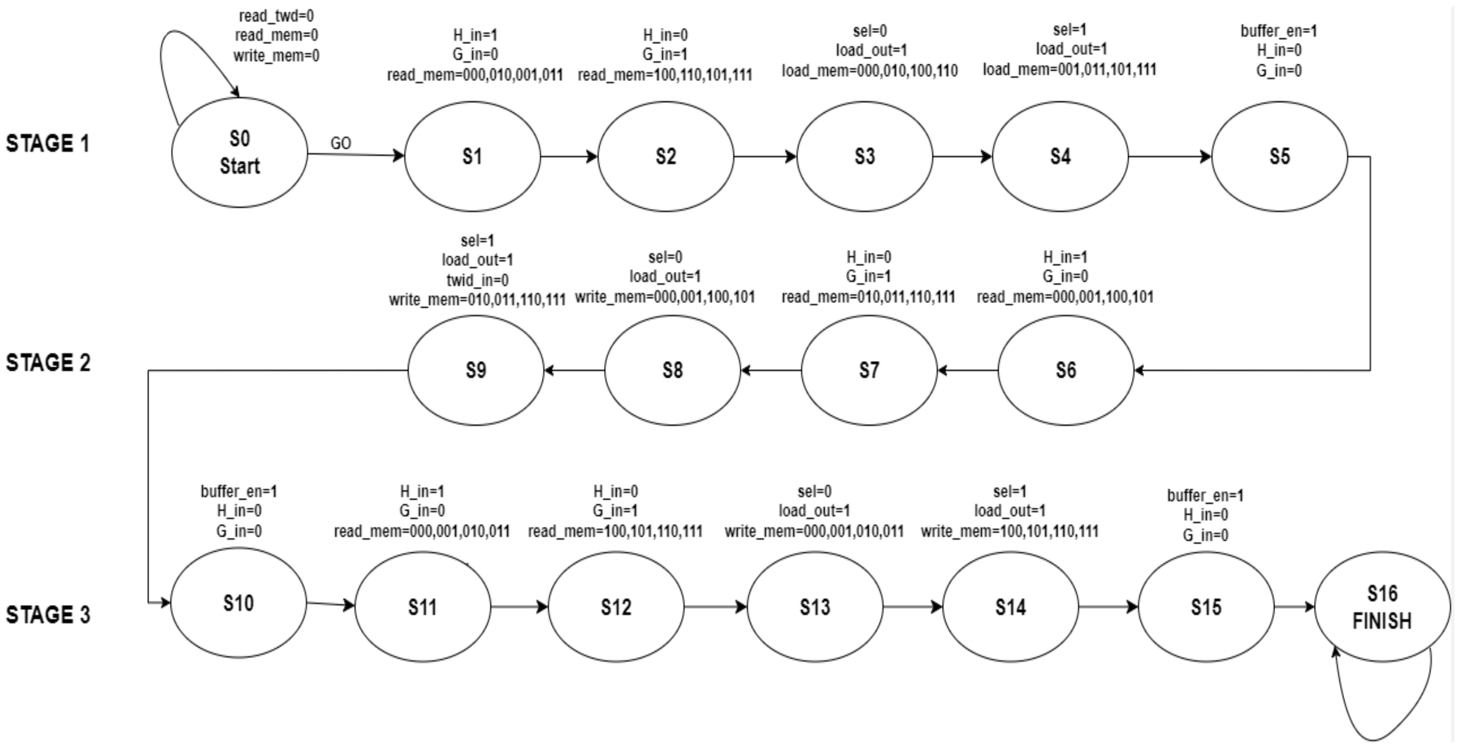


Fig.4 Datapath/Architecture of the 8-point FFT

A finite state machine (FSM) representation is used for modelling the datapath above to directly determine the system's behaviour and the various functionalities involved. The transitions between the various states as shown in Fig.5 below is used to define the sequence of events occuring in the datapath. This FSM is constructed using sixteen states and is divided into 3 stages, each stage corresponding to the 3 stages as discussed above in the butterfly diagram.

Fig.5 FSM for datapath implementation of 8-point FFT

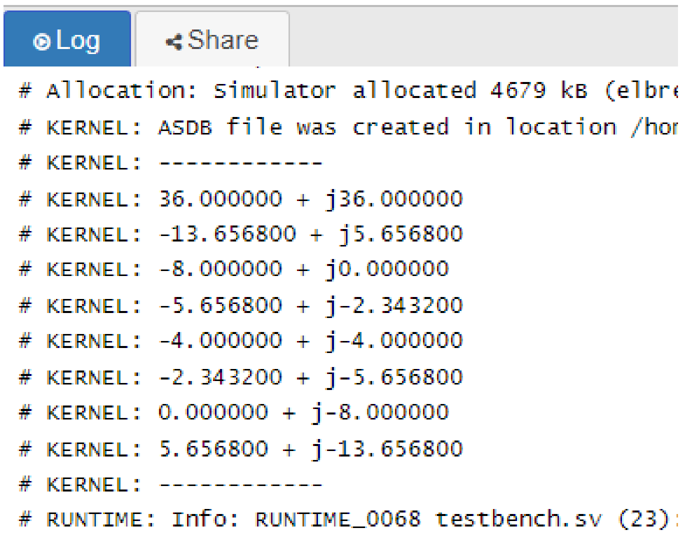


IV. VERILOG IMPLEMENTATION

The entire implementation of the specified design is available to run and simulate on EDA Playground, an online simulator here : https://edaplayground.com/x/aBxz

V. RESULTS

On simulation of a given set of real and imaginary inputs, the designed 8-point FFT will display the results obtained for that set of inputs in the output terminal as shown below in Fig.6, i.e, the values from X[0] to X[7]. As a demonstration in this case, the input stream is inp_real[0:7] = {1,2,3,4,5,6,7,8} and inp_imag[0:7] = {1,2,3,4,5,6,7,8}.



```
# Allocation: Simulator allocated 4679 kB (elbre
# KERNEL: ASDB file was created in location /hor
# KERNEL: ------------
# KERNEL: 36.000000 + j36.000000
# KERNEL: -13.656800 + j5.656800
# KERNEL: -8.000000 + j0.000000
# KERNEL: -5.656800 + j-2.343200
# KERNEL: -4.000000 + j-4.000000
# KERNEL: -2.343200 + j-5.656800
# KERNEL: 0.000000 + j-8.000000
# KERNEL: 5.656800 + j-13.656800
# KERNEL: ------------
# RUNTIME: Info: RUNTIME_0068 testbench.sv (23):
```

Fig.6 Generated output stream of real and imaginary parts

On simulation, the results show the accurate computation of the 8-point FFT. It also shows its feasibility for practical applications based on FFT.

VI. CONCLUSION

Making use of a minimalistic design of the DIT-FFT algorithm, complex operations that otherwise cannot be carried out by DFT are done efficiently. It helps to optimise resources and caters to real-time processing demands. The results validate the efficiency of the implementation and also help to leverage FFT algorithms within FPGA-based systems, allowing for efficient signal processing. This model can be extended and is designed to accept both real and imaginary inputs. It can also be implemented for larger point FFTs making it highly flexible and open to improvements.

VII. REFERENCES

[1] Qian, Z., & Margala, M. (2016, September). Low-Power Split-Radix FFT Processors Using Radix-2 Butterfly Units. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 24(9), 3008–3012.

[2] Reddy, N. A., Rao, D. S., & Suman, J. V. (2013, December 31). Design and Simulation of FFT Processor Using Radix-4 Algorithm Using FPGA. International Journal of Advanced Science and Technology, 61, 53–62.

[3] Patrikar, M., & Tehre, P. (2017, March). Design and Power Measurement of 2 And 8-Point FFT Using Radix-2 Algorithm for FPGA Implementation. IOSR Journal of VLSI and Signal Processing, 7(1), 44–48.

[4] Giri, N. K., & Sinha, A. (2012, May 31). FPGA implementation of a novel architecture for performance enhancement of Radix-2 FFT. ACM SIGARCH Computer Architecture News, 40(2), 28–32.