IMPLEMENTATION OF 64 POINT FFT USING RADIX 8 ALGORITHM

C. SANTHI, Asst. Professor, ECE department, CMR Institute of Technology
G. ARCHANA DEVI, Asst. Professor, ECE department, CMR Institute of Technology
K. NARMADA REDDY, Asst. Professor, ECE department, CMR Institute of Technology

ABSTRACT

The Fast Fourier Transform (FFT) and its inverse (IFFT) are very useful algorithms in signal processing. Radix-8 FFT can accomplish more computing speed than its previous approaches and also achieves cost effective performance with less development time. For fast calculation, Twiddle Factor is introduced in the mathematical operation. The algorithm of radix 8 FFT is formulated depending on the basic approach involved in solving a radix-n FFT. By using radix 2 and Radix 4 algorithms more number of stages are required to design 64 point FFT, whereas the 64 point FFT can be developed by using only 2 stages of radix 8 algorithms which reduces the development time of FFT. The design can be verified using the simulation results.

Keywords – Radix 2, Radix 4, Radix 8, Twiddle Factors, FFT.

1. INTRODUCTION

Frequency analysis of discrete signal can be conveniently performed on digital signal processors. In order to perform such an analysis one has to transform the signal from time domain to frequency domain representation. FFT is itself not a transformation but just a computational algorithm to evaluate Discrete Fourier Transform (DFT). The fast Fourier transform (FFT) plays an important role in the design and implementation of discrete-time signal processing algorithms and systems. In recent years, motivated by the emerging applications in the modern digital communication systems and television terrestrial broadcasting systems, there has been tremendous growth in the design of high-performance dedicated FFT processors. Pipelined FFT processor is a class of real-time FFT architectures characterized by continuous processing of the input data which, for the reason of the transmission economy, usually arrives in the word sequential format. However, the FFT operation is very communication intensive which calls for spatially global interconnection. Therefore, much effort on the design of FFT processors focuses on how to efficiently map the FFT algorithm to the hardware to accommodate the serial input for computation.

Consequently, computation of the N-point DFT corresponds to the computation of N samples of the Fourier transform at N equally spaced frequencies $W_k = 2\Pi k/N$. Considering input x[n] to be complex, N complex multiplications and (N-1) complex additions are required to compute each value of the DFT, if computed directly from the formula given as

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \qquad 0 \le k \le N-1$$

$$W_N = e^{-j2\pi/N} \qquad \text{(Eq. 1)}$$

FFT computation requires $N*\log_2(N)$ operations, whereas DFT requires N^2 operations, so the FFT is more faster than DFT. Hence Fast Fourier Transform (FFT) becomes more important in applications which require fast signal processing. Using this transform time domain signal is converted into frequency domain; where filtering and correlation can be performed easily. The FFT algorithm is chosen to consider the execution speed, hardware complexity, flexibility and precision for many applications. However, for real time systems the speed of the execution is the main concern.

2. RADIX 8 ALGORITHMS

Radix-8 algorithm is more attractive since it requires less number of multiplication operations for FFT which reduces the complexity of computation. Modification of algorithm in term of mathematical simplification and the reducing of repeated computation has enhanced the power and active area consumption of the FFT design. To develop 8 point FFT using Radix 2 algorithm 3 stages are required. The same 8 point FFT can be developed by using Radix 4 algorithm and Radix 2 algorithm in 2 stages which reduces the time and area.

In the previous Radix Algorithms either the input or output will be in the bit reversal order and a resolver is used to get the output sequence in proper order. But by using the proposed method of Radix 8 FFT an 8 point FFT can be developed in a single stage which reduces the development time. Also the input and output both are in the proper order and no need of any resolver.

In this Radix 8 algorithm, the real and imaginary parts of twiddle factor are determined separately, which gives fast computation of FFT. The complexity is reduces and the development of higher Radix algorithms will become more easy with this approach.

The twiddle factor can be represented as W_N, where

$W_N=W_R+W_I$ (Eq.2)

And the input can be declared as X, where

$X=X_R+X_I$ (Eq.3)

When the above two equations are multiplied we obtain a set of outputs. Let the output obtained is Y, where

$Y=W_N*X$ (Eq.4)

$Y= (W_R+W_I)*(X_R+X_I)$ (Eq.5)

$Y= [(W_R*X_R) + (W_R*X_I) + (X_R*W_I) + (X_I*W_I)]$ (Eq.6)

Now, even the output can be obtained in two parts i.e., real and imaginary denoted by Y_R and Y_I. Therefore,

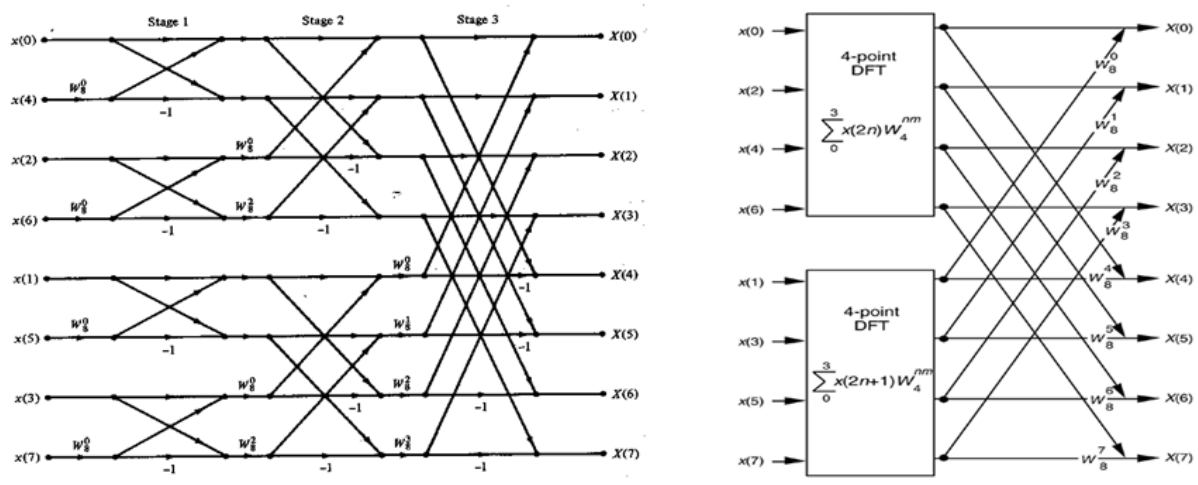
$Y=Y_R+Y_I$ (Eq.7)

Comparing the two equations of Y

We have,

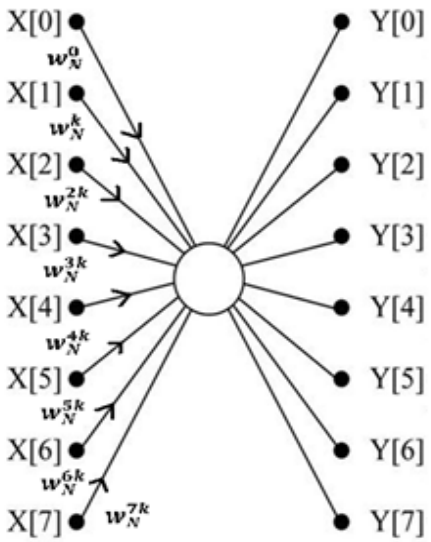
$Y_R= [(W_R*X_R) + (W_I*X_I)]$ (Eq.8)

$Y_I= [(W_R*X_I) + (X_R*W_I)]$ (Eq.9)



(a) (b)

Fig.1 8 point FFT using (a) Radix 2 and (b) Radix 4 Algorithms



$$Y[0] = X[0]+X[1]+X[2]+X[3]+X[4]+X[5]+X[6]+X[7]$$

$$Y[1] = X[0]+w_8^1X[1]+w_8^2X[2]+w_8^3X[3]+w_8^4X[4]+w_8^5X[5]+w_8^6X[6]+w_8^7X[7]$$

$$Y[2] = X[0]+w_8^2X[1]+w_8^4X[2]+w_8^6X[3]+X[4]+w_8^2X[5]+w_8^4X[6]+w_8^6X[7]$$

$$Y[3] = X[0]+w_8^3X[1]+w_8^6X[2]+w_8^9X[3]+w_8^4X[4]+w_8^7X[5]+w_8^2X[6]+w_8^6X[7]$$

$$Y[4] = X[0]+w_8^4X[1]+X[2]+w_8^4X[3]+X[4]+w_8^4X[5]+X[6]+w_8^4X[7]$$

$$Y[5] = X[0]+w_8^5X[1]+w_8^2X[2]+w_8^7X[3]+w_8^4X[4]+w_8^1X[5]+w_8^6X[6]+w_8^3X[7]$$

$$Y[6] = X[0]+w_8^6X[1]+w_8^4X[2]+w_8^2X[3]+X[4]+w_8^6X[5]+w_8^4X[6]+w_8^2X[7]$$

$$Y[7] = X[0]+w_8^7X[1]+w_8^6X[2]+w_8^5X[3]+w_8^4X[4]+w_8^3X[5]+w_8^2X[6]+w_8^1X[7]$$

Fig.2 Radix 8 Algorithm and output equations

The output equations are directly obtained by multiplying inputs with twiddle factors. No need of any bit reversal resolver at the output as the output is obtained in normal order even though the input is given in the normal order. The simulation code for real and imaginary parts of the output are as follows:

```
// for xr1
re1[1] = (xr1*wr1) - (xi1*wi1);
Im1[1] = (xi1*wr1) + (xr1*wi1);

// The real and Imaginary parts of outputs
r0 = xr0+xr1+xr2+xr3+xr4+xr5+xr6+xr7;
i0 = xi0+xi1+xi2+xi3+xi4+xi5+xi6+xi7;

r1 = xr0+re1[1]+re2[2]+re3[3]+re4+re5[5]+re6[6]+re7[7];
i1 = xi0+Im1[1]+Im2[2]+Im3[3]+Im4+Im5[5]+Im6[6]+Im7[7];

r2 = xr0+re1[2]+re2[4]+re3[6]+xr4+re5[2]+re6[4]+re7[6];
i2 = xi0+Im1[2]+Im2[4]+Im3[6]+xi4+Im5[2]+Im6[4]+Im7[6];

r3 = xr0+re1[3]+re2[6]+re3[1]+re4+re5[7]+re6[2]+re7[5];
i3 = xi0+Im1[3]+Im2[6]+Im3[1]+Im4+Im5[7]+Im6[2]+Im7[5];

r4 = xr0+re1[4]+xr2+re3[4]+xr4+re5[4]+xr6+re7[4];
i4 = xi0+Im1[4]+xi2+Im3[4]+xi4+Im5[4]+xi6+Im7[4];

r5 = xr0+re1[5]+re2[2]+re3[7]+re4+re5[1]+re6[6]+re7[3];
i5 = xi0+Im1[5]+Im2[2]+Im3[7]+Im4+Im5[1]+Im6[6]+Im7[3];

r6 = xr0+re1[6]+re2[4]+re3[2]+xr4+re5[6]+re6[4]+re7[2];
i6 = xi0+Im1[6]+Im2[4]+Im3[2]+xi4+Im5[6]+Im6[4]+Im7[2];
```

r7 = xr0+re1[7]+re2[6]+re3[5]+re4+re5[3]+re6[2]+re7[1];
i7 = xi0+Im1[7]+Im2[6]+Im3[5]+Im4+Im5[3]+Im6[2]+Im7[1];

The above realization gives direct method of radix-8 FFT calculation and reduces the design time and area.

3. 64 Point FFT Using Radix 8 Algorithm

The 64 point FFT can be developed by using different Radix algorithms. By using Radix 2 algorithm, 6 stages are required which has a large computation time. By using Radix 4 algorithm, 3 stages are required where as by using this proposed Radix 8 algorithm, only 2 stages are required, which reduces the complexity and computation time.

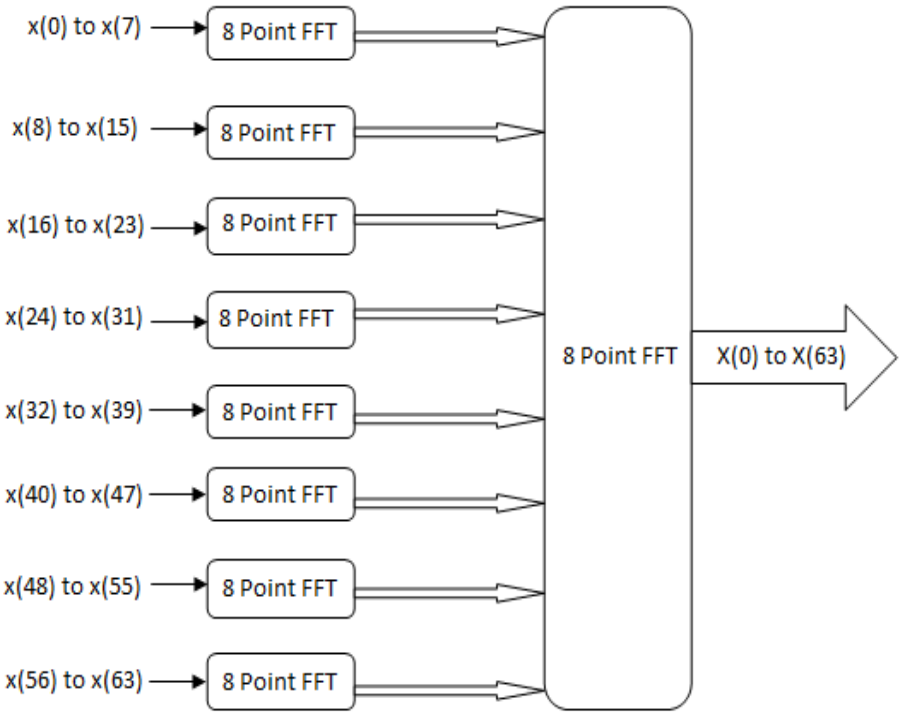
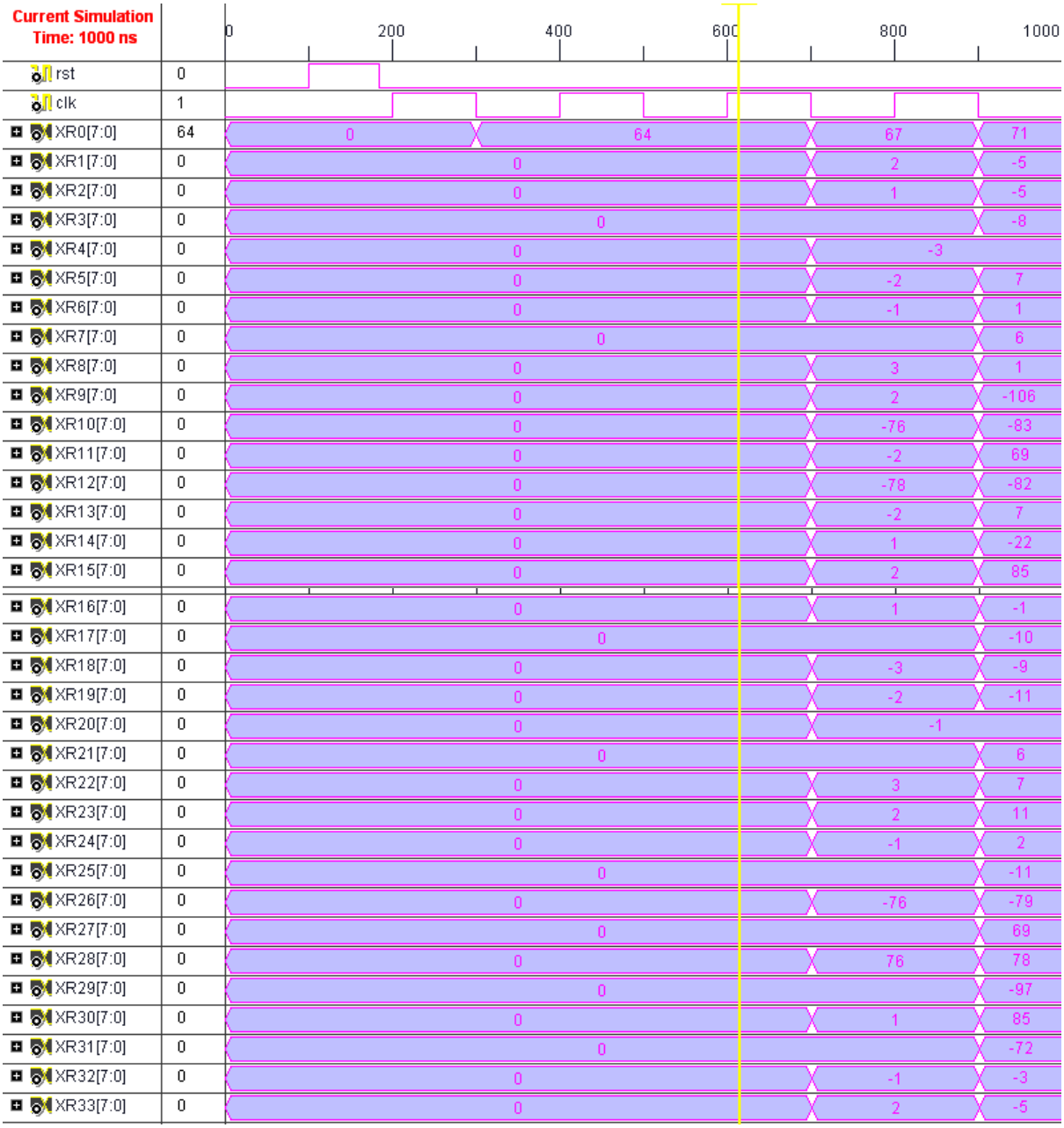
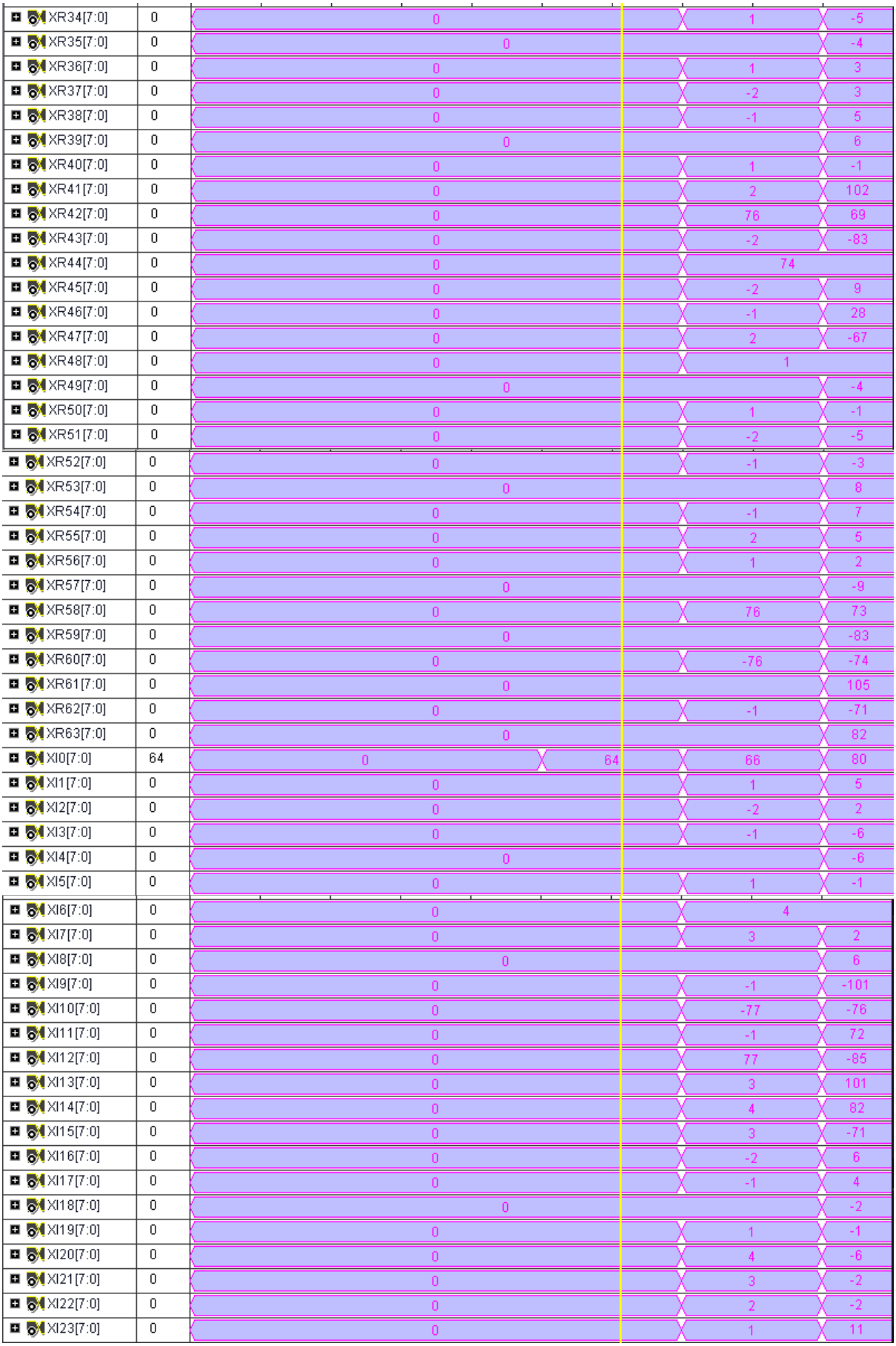
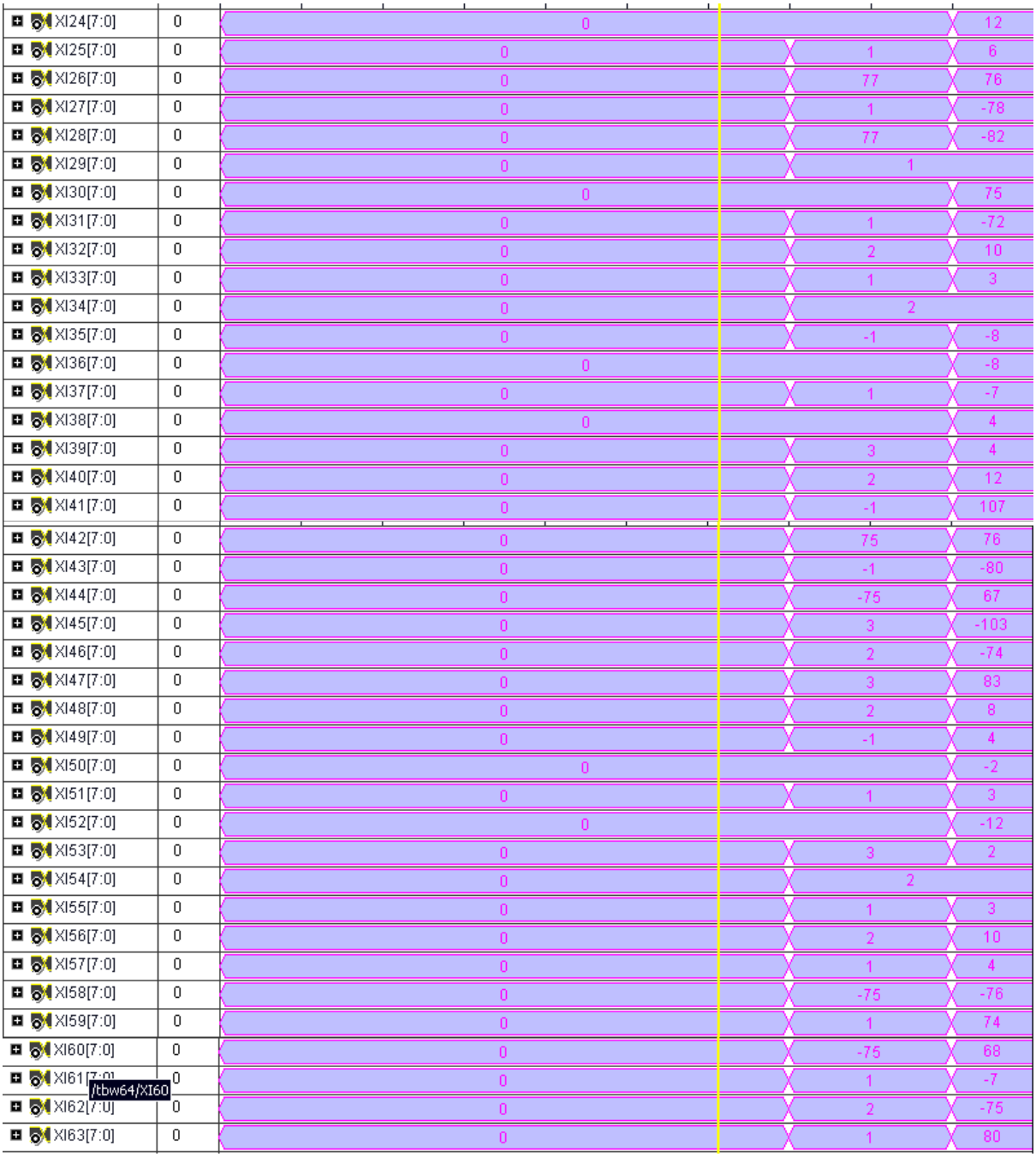


Fig.3 64 Point FFT using Radix 8 Algorithm

4. Simulation Results


Current Simulation Time: 1000 ns
0
200
400
600
800
1000
rst 0
clk 1
XR0[7:0] 64 0 64 67 71
XR1[7:0] 0 0 2 -5
XR2[7:0] 0 0 1 -5
XR3[7:0] 0 0 -8
XR4[7:0] 0 0 -3
XR5[7:0] 0 0 -2 7
XR6[7:0] 0 0 -1 1
XR7[7:0] 0 0 6
XR8[7:0] 0 0 3 1
XR9[7:0] 0 0 2 -106
XR10[7:0] 0 0 -76 -83
XR11[7:0] 0 0 -2 69
XR12[7:0] 0 0 -78 -82
XR13[7:0] 0 0 -2 7
XR14[7:0] 0 0 1 -22
XR15[7:0] 0 0 2 85
XR16[7:0] 0 0 1 -1
XR17[7:0] 0 0 -10
XR18[7:0] 0 0 -3 -9
XR19[7:0] 0 0 -2 -11
XR20[7:0] 0 0 -1
XR21[7:0] 0 0 6
XR22[7:0] 0 0 3 7
XR23[7:0] 0 0 2 11
XR24[7:0] 0 0 -1 2
XR25[7:0] 0 0 -11
XR26[7:0] 0 0 -76 -79
XR27[7:0] 0 0 69
XR28[7:0] 0 0 76 78
XR29[7:0] 0 0 -97
XR30[7:0] 0 0 1 85
XR31[7:0] 0 0 -72
XR32[7:0] 0 0 -1 -3
XR33[7:0] 0 0 2 -5


XR34[7:0] 0 0 1 -5
XR35[7:0] 0 0 -4
XR36[7:0] 0 0 1 3
XR37[7:0] 0 0 -2 3
XR38[7:0] 0 0 -1 5
XR39[7:0] 0 0 6
XR40[7:0] 0 0 1 -1
XR41[7:0] 0 0 2 102
XR42[7:0] 0 0 76 69
XR43[7:0] 0 0 -2 -83
XR44[7:0] 0 0 74
XR45[7:0] 0 0 -2 9
XR46[7:0] 0 0 -1 28
XR47[7:0] 0 0 2 -67
XR48[7:0] 0 0 1
XR49[7:0] 0 0 -4
XR50[7:0] 0 0 1 -1
XR51[7:0] 0 0 -2 -5
XR52[7:0] 0 0 -1 -3
XR53[7:0] 0 0 8
XR54[7:0] 0 0 -1 7
XR55[7:0] 0 0 2 5
XR56[7:0] 0 0 1 2
XR57[7:0] 0 0 -9
XR58[7:0] 0 0 76 73
XR59[7:0] 0 0 -83
XR60[7:0] 0 0 -76 -74
XR61[7:0] 0 0 105
XR62[7:0] 0 0 -1 -71
XR63[7:0] 0 0 82
XI0[7:0] 64 0 64 66 80
XI1[7:0] 0 0 1 5
XI2[7:0] 0 0 -2 2
XI3[7:0] 0 0 -1 -6
XI4[7:0] 0 0 -6
XI5[7:0] 0 0 1 -1
XI6[7:0] 0 0 4
XI7[7:0] 0 0 3 2
XI8[7:0] 0 0 6
XI9[7:0] 0 0 -1 -101
XI10[7:0] 0 0 -77 -76
XI11[7:0] 0 0 -1 72
XI12[7:0] 0 0 77 -85
XI13[7:0] 0 0 3 101
XI14[7:0] 0 0 4 82
XI15[7:0] 0 0 3 -71
XI16[7:0] 0 0 -2 6
XI17[7:0] 0 0 -1 4
XI18[7:0] 0 0 -2
XI19[7:0] 0 0 1 -1
XI20[7:0] 0 0 4 -6
XI21[7:0] 0 0 3 -2
XI22[7:0] 0 0 2 -2
XI23[7:0] 0 0 1 11



5. CONCLUSION

Fast Fourier Transform (FFT) techniques have revolutionized the Digital Signal Processing techniques in the past 30 years. It does not only provide a fast response but also logic thought to be un-realizable easily in the range to be realized. The parallel processing of FFT hence has been proposed to be used in multiprocessor algorithms. When an FFT is implemented in special-purpose hardware, errors and constraints due to finite word lengths are unavoidable. While deciding the word length needed for an FFT, these quantization effects must be considered. The algorithm for radix-8 FFT is studied for implementation. The HDL coding in DIT scheme is done and tested with some test vectors generated using MATLAB. A different implementation methodology could be used to reduce the quantization errors. Also, the quantization factor could be made floating, i.e. dependent on

the result rather only the inputs. This could provide different scaling levels from the same logic and have a large scope of application of the design.

6. Bibliography

- Digital signal processing book by Nagoor kani.
- Digital system design book by Roth C.H.
- www.dspguru.com
- dstarena.com
- tevatrontech.com