Abstract: In 1965 a technique called Fast Fourier Transform (FFT) was invented to find the Fourier Transform. This paper compares three architectures, the basic architecture/ non-reduced architecture of FFT, decomposed FFT architecture without retiming and decomposed FFT architecture with retiming. In each case, the adder used will be Ripple Carry Adder (RCA) and Carry Save Adder (CSA). A fast Fourier transform (FFT) calculates the discrete Fourier transform (DFT) or the inverse (IDFT) of a sequence. Fourier analysis transforms a signal from time to frequency domain or vice versa. One of the most burgeoning use of FFT is in Orthogonal Frequency Division Multiplex (OFDM) used by most cell phones, followed by the use in image processing. The synthesis has been carried out on Xilinx ISE Design Suite 14.7. There is a decrease in delay of 0.824% in Ripple Carry Adder and 6.869% in Carry Save Adder, further the reduced architecture for both the RCA and CSA architectures shows significant area optimization (approximately 20%) from the non-reduced counterparts of the FFT implementation.

Keywords : Area optimization, CSA, Decomposition, DFT, FFT, IDFT, Non-Reduced, OFDM, RCA

I. INTRODUCTION

The efficient and structured computation for the purpose of reducing the hardware requirements has been the field of interest of the designer from a very long time. Fourier transform converts a signal from time or space domain to frequency domain and vice versa. FFT is a way to evaluate the same result of DFT promptly, where DFT takes O(N^2) arithmetic operations for computation, FFT takes, only O(N log N) operations. The difference in speed can be humongous, especially where N may be in the thousands or millions.

The Discrete Fourier Transform (DFT) deals with a finite set of data and can be implemented in computers by algorithms or even dedicated hardware. DFT has been utilised across numerous fields such as image processing, radar, voice processing, data compression and sonar systems. [1]

The Fast Fourier Transform is an algorithm to find the DFT and IDFT. It produces the identical output precisely as computing the DFT, however FFT is much faster. [2] Cooley-Tukey algorithm, most commonly used, is a divide and conquer algorithm which breaks a DFT into smaller DFTs having twiddle factors.

The algorithm of two types, first - time-based (DIT) and, second - frequency-based (DIF) Fast Fourier Transform. Order of data in DIT FFT is from bit reversal in input to normal order in output, whereas DIF FFT is converse. In Radix-2 algorithm, a N point FFT is continuously split into smaller parts till two point FFT is obtained.[3]

II. OVERVIEW OF FFT ARCHITECTURE, RE-TIMING AND ADDERS USED

In Fast Fourier transform, a butterfly puts together the results of smaller DFTs into a larger DFT, or does the inverse. The name “butterfly” comes from the structure of the data-flow representation in the radix-2 case.[4]

![Figure 1: Signal Flow Graph with input a and output b](image)

According to radix-2 Cooley–Tukey algorithm, the output b0 and b1 can be written in terms of inputs a0 and a1 by using the following equations:

\[ b_0 = a_0 + a_1 \]

\[ b_1 = a_0 - a_1 \]

Where twiddle factors are not included.

Considering twiddle factor \( W_{nk} = e^{2\pi ik/n} \), the output b0 and b1 are:

\[ b_0 = a_0 + a_1 W_{nk} \]

\[ b_1 = a_0 - a_1 W_{nk} \]

Where \( k \) is an integer which depends on the part of the transform being computed.

Revised Manuscript Received on August 05, 2019.

Mallika Verma, Electronics and Communication, Jaypee Institute of Information Technology, Noida, Uttar Pradesh, India. Email: contact.mallikaverma@gmail.com

Ankur Bhardwaj, Electronics and Communication, Jaypee Institute of Information Technology, Noida, Uttar Pradesh, India. Email: ankur.bhardwaj@jiit.ac.in
Implementation of FFT Architecture using various adders

The adders used in butterfly computation are Ripple Carry Adder and Carry Save Adder. A carry save adder calculates the sum of three or more n bit numbers in binary. It produces two outputs – Sum and Carry, a sequence of partial sum bits and a sequence of partial carry bits respectively.[5,6] On the other hand, a ripple carry adder takes two inputs, A0 and B0 with Cin initially as 0 for a full adder, the carry generated is fed as an input to the next full adder with inputs A1 and B1. The carry gets rippled till n stages and the final Cout along with the sum generated in each stage is the final output. As each carry is fed as input to the next full adder or is rippled, it is called a Ripple Carry Adder.[6]

Re-timing is used to shift the memory elements, latches or registers, in such a way that it’s purpose remains the same in the circuit, thereby improving area, power characteristics and performance.[7] Re-timing is frequently used to minimise the clock period by searching for the minimum workable period by binary search. Further, it can be used to change a given synchronous circuit into a more effective circuit keeping different cost criteria in mind.[8]

III. SUGGESTED FFT ARCHITECTURE AND IMPLEMENTATION

Three FFT architectures build on Cooley-Tukey algorithm have been implemented.

A. Basic Butterfly / Non-Reduced Architecture

By using the divide and conquer approach, radix-2 FFT is obtained. Splitting the sequence x[n] into sequences A1 and A2, each of length \( \frac{N}{2} \), A1[n] = x[2n], samples are even A2[n] = x[2n + 1], samples are odd

For \( n = 0 \) to \( \frac{N}{2} - 1 \)

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

(1)

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

(2)

\[
x[k] = \sum_{n=0}^{N-2} x[2m]W_N^{k2m} + \sum_{m=0}^{N-1} x[2m + 1]W_N^{k(2m+1)}
\]

(3)

\[
x[k] = \sum_{m=0}^{N-1} s1[m]W_N^{km} + W_N^{k} \sum_{m=0}^{N-1} \sum_{m=0}^{2m+1} s2[m]W_N^{km}
\]

(4)

Because of recursion property

\[
s1[k] = \sum_{m=0}^{N-1} s1[n]W_N^{km} \\
\text{and} \\
s2[k] = \sum_{m=0}^{N-1} s2[n]W_N^{km}
\]

(5)

Output of each node is calculated by placing RCA and CSA at each step of node calculation to find the ultimate output.
B. (i) Reduced FFT Architecture without retiming

Let us take an 8-point real valued sequence, \( x(n) = \{x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\} \)
where \( x(n) \) can be expressed as, 
\( x(n) = \{m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8\} \)
Using the equations of DFT, we obtain the DFT of \( x(n) \) as \( X(K) \), 
\( X(K) = \{X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8\} \)
The architecture presented [9] reduces power and the area is optimized.

The outputs are Figure 6 are mapped as follows:

- \( X_1 = X_1R + iX_1I \)
- \( X_2 = X_2R + iX_2I \)
- \( X_3 = X_3R + iX_3I \)
- \( X_4 = X_4R + iX_4I \)
- \( X_5 = X_5R + iX_5I \)
- \( X_6 = X_6R + iX_6I \)
- \( X_7 = X_7R + iX_7I \)
- \( X_8 = X_8R + iX_8I \)

Where \( X_6R = X_4R \), and \( X_6I = -X_4I \)

Where \( X_7R = -X_3R \), and \( X_7I = -X_3I \)

And similarly,

- \( X_8R = X_2R \), and \( X_8I = -X_2I \)

Where twiddle factor X is: \( \pm 0.707 \) at consecutive stages

(ii) Reduced Architecture with Retiming

A clock is fed to the state machine where in a single butterfly computes all the stages of the butterfly architecture.

Using an overlapped structure, re-arranging the input and recreating the stage a state machine is designed whose input values, computed values and output values are stored in a register.

The state machine is made such that it is input controlled so that we can manipulate the inputs being fed to the butterfly to get the respective output without increasing the structural complexity.

In the following architecture clock, reset, state_machine_start and inputs are fed to obtain the fast fourier transform of the desired inputs. The architecture has a single butterfly structure which performs all the operations, thereby decreasing delay.

IV. RESULTS AND DISCUSSIONS

A. Basic Butterfly/ Non-Reduced Architecture

1. RCA

Table I: Delay

<table>
<thead>
<tr>
<th>Cell/in-out</th>
<th>fanout</th>
<th>Gate</th>
<th>Net</th>
<th>Delay</th>
<th>Logical Name</th>
<th>Net Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENBF1-&gt;0</td>
<td>5</td>
<td>1.213</td>
<td>0.000</td>
<td>0.000</td>
<td>a1_o_INST</td>
<td>a1_o_INST</td>
</tr>
<tr>
<td>LUT210-&gt;0</td>
<td>1</td>
<td>0.704</td>
<td>0.000</td>
<td>0.000</td>
<td>real/ab/Mesh_dout/0</td>
<td>(real)</td>
</tr>
<tr>
<td>MHCY1B-&gt;0</td>
<td>1</td>
<td>0.666</td>
<td>0.000</td>
<td>0.000</td>
<td>real/ab/Mesh_dout/0</td>
<td>(real)</td>
</tr>
<tr>
<td>MHCY1C-&gt;0</td>
<td>3</td>
<td>0.698</td>
<td>0.704</td>
<td>0.000</td>
<td>real/ab/Mesh_dout/0</td>
<td>(real)</td>
</tr>
<tr>
<td>LUT211-&gt;0</td>
<td>3</td>
<td>0.709</td>
<td>0.410</td>
<td>0.000</td>
<td>real/ftftreal/ree/27/28/cnt</td>
<td>(real)</td>
</tr>
<tr>
<td>LUT311-&gt;0</td>
<td>3</td>
<td>0.709</td>
<td>0.410</td>
<td>0.000</td>
<td>real/ftftreal/ree/27/28/cnt</td>
<td>(real)</td>
</tr>
<tr>
<td>LUT312-&gt;0</td>
<td>3</td>
<td>0.709</td>
<td>0.410</td>
<td>0.000</td>
<td>real/ftftreal/ree/27/28/cnt</td>
<td>(real)</td>
</tr>
<tr>
<td>LUT313-&gt;0</td>
<td>3</td>
<td>0.709</td>
<td>0.410</td>
<td>0.000</td>
<td>real/ftftreal/ree/27/28/cnt</td>
<td>(real)</td>
</tr>
<tr>
<td>GBEF1-&gt;0</td>
<td>3</td>
<td>0.272</td>
<td>0.000</td>
<td>0.000</td>
<td>X3_GOUT (X2-&gt;7)</td>
<td>(real)</td>
</tr>
</tbody>
</table>

Total: 16.059ns (12.059ms logic, 5.990ns route)

(66.9% logic, 33.1% route)
Implementation of FFT Architecture using various adders

Table II: Logic Utilization

<table>
<thead>
<tr>
<th>Logic Utilization</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of slices</td>
<td>142</td>
<td>4656</td>
<td>3%</td>
</tr>
<tr>
<td>No. of 4 input LUTs</td>
<td>261</td>
<td>9312</td>
<td>2%</td>
</tr>
</tbody>
</table>

Figure 10: Simulation Results

2. CSA

Table III: Delay

<table>
<thead>
<tr>
<th>Logic Utilization</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of slices</td>
<td>175</td>
<td>4656</td>
<td>3%</td>
</tr>
<tr>
<td>No. of 4 input LUTs</td>
<td>302</td>
<td>9312</td>
<td>3%</td>
</tr>
</tbody>
</table>

Figure 11: Simulation Results

B. Reduced FFT Architecture Without Re-Timing

1. RCA

Table V: Delay

<table>
<thead>
<tr>
<th>Logic Utilization</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of slices</td>
<td>115</td>
<td>4656</td>
<td>2%</td>
</tr>
<tr>
<td>No. of 4 input LUTs</td>
<td>213</td>
<td>9312</td>
<td>2%</td>
</tr>
</tbody>
</table>

Figure 12: Simulation

Table VI: Logic Utilization

<table>
<thead>
<tr>
<th>Logic Utilization</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of slices</td>
<td>115</td>
<td>4656</td>
<td>2%</td>
</tr>
<tr>
<td>No. of 4 input LUTs</td>
<td>213</td>
<td>9312</td>
<td>2%</td>
</tr>
</tbody>
</table>

Table VII: Delay

<table>
<thead>
<tr>
<th>Logic Utilization</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of slices</td>
<td>144</td>
<td>4656</td>
<td>3%</td>
</tr>
<tr>
<td>No. of 4 input LUTs</td>
<td>266</td>
<td>9312</td>
<td>2%</td>
</tr>
</tbody>
</table>

Figure 13: Simulation

Table VIII: Logic Utilization
C. Reduced FFT Architecture With Re-Timing

1. RCA

Table IX: Delay

<table>
<thead>
<tr>
<th>Logic Utilization</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of slice flipslops</td>
<td>83</td>
<td>9312</td>
<td>1%</td>
</tr>
<tr>
<td>No. of 4 input LUTs</td>
<td>292</td>
<td>9312</td>
<td>3%</td>
</tr>
<tr>
<td>No. of occupied slices</td>
<td>155</td>
<td>4656</td>
<td>3%</td>
</tr>
</tbody>
</table>

Table X: Logic Utilization

Table XI: Delay

<table>
<thead>
<tr>
<th>Logic Utilization</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of slice flipslops</td>
<td>90</td>
<td>9312</td>
<td>1%</td>
</tr>
<tr>
<td>No. of 4 input LUTs</td>
<td>312</td>
<td>9312</td>
<td>3%</td>
</tr>
<tr>
<td>No. of occupied slices</td>
<td>175</td>
<td>4656</td>
<td>3%</td>
</tr>
</tbody>
</table>

2. CSA

Table XII: Logic utilization

V. CONCLUSION

A Verilog HDL code has been successfully simulated and synthesized on Xilinx ISE Design suite 14.7 for 8-point FFT with and without retiming.

From the synthesis report is can be observed that there is a significant decrease in delay for FFT architecture with re-timing, while comparing with basic FFT Architecture and decomposed FFT Architecture without re-timing, for both RCA(55.752% decrease) and CSA(61.069% decrease). However, in terms of area, it increases for RCA and remains the same for CSA when compared with basic FFT architecture and increases when compared with decomposed FFT Architecture without Re-Timing.

On the other hand, the decomposed FFT architecture without re-timing shows a significant decrease in terms of area (approximately 20%) for both RCA and CSA and a decrease in delay of 0.824% for RCA and 6.869% for CSA when compared with basic FFT architecture. The synthesis has been carried out on Xilinx SPARTAN 3 Series of FPGA with XC3S500E Device series. The results are favourable in terms of area and delay.

Table XIII: Percentage Increase/Decrease in area and delay while comparing basic architecture and reduced architecture without re-timing

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Ripple Adder</th>
<th>Carry Adder</th>
<th>Save</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of slices</td>
<td>19.014% Decrease</td>
<td>17.714% Decrease</td>
<td></td>
</tr>
<tr>
<td>Number of 4 input LUTs</td>
<td>18.391% Decrease</td>
<td>16.875% Decrease</td>
<td></td>
</tr>
<tr>
<td>Delay</td>
<td>0.824% Decrease</td>
<td>6.869% Decrease</td>
<td></td>
</tr>
</tbody>
</table>

Table XIV: Percentage Increase/Decrease in area and delay while comparing basic architecture and reduced architecture with re-timing

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Ripple Adder</th>
<th>Carry Adder</th>
<th>Save</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of slices</td>
<td>9.155% Increase</td>
<td>No increase or decrease</td>
<td></td>
</tr>
<tr>
<td>Number of 4 input LUTs</td>
<td>11.877% Increase</td>
<td>2.5% Decrease</td>
<td></td>
</tr>
<tr>
<td>Delay</td>
<td>55.752% Decrease</td>
<td>61.069% Decrease</td>
<td></td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENT

It is author’s privilege to express the most sincere gratitude to Mr. Ankur Bhardwaj, Department of Electronics and Communications, Jaypee Institute of Information Technology, Noida, under whose supreme and enlightened guidance the author has been able to accomplish this work. The author gratefully acknowledge her sincere gratitude to his illuminating guidance, whole-hearted co-operation right from beginning to its write-up and completion.

REFERENCES

2. V. Mangayarkarasi, Dr. C.Kumar Charlie Paul, “Performance analysis between Radix2, Radix4, Mixed Radix4-2 and Mixed Radix8-2 FFT”, 2nd International Conference on Current Trends in Engineering and Technology, ICCTET’14

AUTHORS PROFILE

Mallika Verma was born at Delhi, India on July 26, 1997. She received her B. Tech degree from Jaypee Institute of Information Technology, Sector 62, Noida, Uttar Pradesh (June, 2019) and currently is pursuing her Masters degree in International Business from Queensland University of Technology, Brisbane, Australia.

Ankur Bhardwaj was born at Meerut, Uttar Pradesh, India on January 04, 1990. He received B.Tech degree from Gautam Buddh Technical University, Lucknow (June 2011), M.Tech degree from DTU Delhi (2013) and currently pursuing PhD degree from Jaypee Institute of Information Technology, Noida. He joined Jaypee Institute of Information Technology, Noida, in July 2013 as a lecturer. Presently he is an Assistant Professor in the Department of Electronics and Communication Engineering, Jaypee Institute of Information Technology, Sector 62, Noida.