ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Issue 7, July 2017

Design and Implementation of an Area efficient Split-Radix FFT Processors Using Radix-2 Butterfly Units

P. Venu M.Tech
VLSI Design, Dept of ECE
GRIET-Hyderabad.
venupoladi32@gmail.com

Dr.Mamatha Samson
Professor in Dept of ECE
GRIET-Hyderabad.
mamata2001@gmail.com

Abstract- Split-Radix Fast Fourier Transform (SRFFT) is an ideal candidate for the implementation of a low-power FFT processor, because it has the lowest number of arithmetic operations among all the FFT algorithms. In the design of such processors, an efficient addressing scheme for FFT data as well as twiddle factors is required. The signal flow graph of SRFFT is the same as radix-2 FFT, and therefore, the conventional address generation schemes of FFT data could also be applied to SRFFT. However, SRFFT has irregular locations of twiddle factors and forbids the application of radix-2 address generation methods. This project presents shared-memory low-power SRFFT processor architecture. The SRFFT is computed by using a modified radix-2 butterfly unit. The butterfly unit exploits the multiplier-gating technique to save dynamic power at the expense of using more hardware resources. In addition, two novel address generation algorithms for both the trivial and nontrivial twiddle factors are developed. Simulation results show that compared with the conventional radix-2 shared-memory implementations, the proposed design achieves over 20% lower power consumption when computing a 1024-point complex-valued transform.

Keywords—Address generation, low power, radix-2, split-radix fast Fourier transform (SRFFT), twiddle factors.

I. INTRODUCTION

Discrete Fourier Transform generates a predefined duration discrete frequency sequence that is obtained by sampling one period of Fourier Transform. Fast algorithms to determine the DFT are called as Fast Fourier Transform (FFT). FFT calculate the DFT and produces accurate results as same as that of the DFT calculated by using definition. The difference is that an FFT is much faster than normal computation. The split-radix FFT has minor complication than the radix-4 or any higher-radix power-of-two FFT.

Y. Chen [3] presented a new Dynamic Voltage and Frequency Scaling (DVFS) FFT processor for MIMO OFDM applications. MIMO OFDM systems had achieved better reliability and superior capacity, the power consumption also increases because of the unlimited difficulty for the multi-stream processing. Therefore, less power becomes a major target in design of MIMO OFDM devices, especially for portable applications. To save the power of the FFT processor cultivates proportionally the stream number as this consumes a large percentage of system power. The Dynamic Voltage and Frequency Scaling (DVFS) is an effective technique to achieve less power. Parallelized radix-2 FFT structure has been adopted to reduce the power consumption.

In 2005, Y. W. Lin shows that the pipelined FFT architecture, called as Mixed-Radix Multipath Delay Feedback (MRMDF) reduces power consumption and hardware cost. This processor uses higher radix FFT algorithm

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Issue 7, July 2017

with less memory and complex multipliers. A novel 128-point FFT/IFFT processor for OFDM-based UWB systems was proposed by J.Kwong and M.Goel [10]. In addition, the number of complex multiplications is less effectively reduced by using a higher radix algorithm.

In 2010 Shen-Jui Huang used eight-data-path 2048-point FFT processor that has been proposed with a transfer rate of 2.4 GS/s for OFDM-based gigabit WPAN application. Proposed work was based on split radix FFT architecture. FFT Computation can be done using different algorithm based on Radix-2, Radix-4, Radix-8, Split-Radix etc. The split-radix algorithm has lesser multiplicative complexity than both radix-2 and radix-4 algorithms. The split-radix FFT mixes radix-2 and radix-4 decompositions. The Split-Radix FFT has lower complication than the radix-4 or any higher-radix power-of-two FFT.

The rest of this brief is organized as follows. Section II provides a theoretical comparison of the number of complex multiplications between the radix-2 FFT and the SRFFT. Section III discusses the architecture of the proposed design. Section IV provides the implementation results and Section V concludes this brief.

II. COMPARISON OF SRFFT AND RADIX-2 FFT

Let us consider the computation of the $N = 2^S$ point DFT by the divide-and conquer approach. We split the N-point data sequence into two $N/2$-point data sequences $f_1(n)$ and $f_2(n)$, corresponding to the even-numbered and odd-numbered samples of $x(n)$, respectively, that is,

$$f_1(n) = x(2n) \quad (1)$$

$$f_2(n) = x(2n+1), \quad (2)$$

Thus $f_1(n)$ and $f_2(n)$ are obtained by decimating $x(n)$ by a factor of 2, and hence the resulting FFT algorithm is called a decimation-in-time algorithm.

Now the N-point DFT can be expressed in terms of the DFT's of the decimated sequences as follows:

$$X(K) = \sum_{n=0}^{N-1} x(n)\, W_N^{nK}, \qquad 0 \le K \le N-1 \quad (3)$$

$$= \sum_{n=0,even}^{N-1} x(n)\, W_N^{nK} + \sum_{n=0,odd}^{N-1} x(n)\, W_N^{nK} \quad (4)$$

$$= \sum_{r=0}^{N/2-1} x(2r)\, W_N^{2rK} + \sum_{r=0}^{N/2-1} x(2r+1)\, W_N^{(2r+1)K} \quad (5)$$

But $W_N^2 = W_{N/2}$. With this substitution, the equation can be expressed as

$$X(K) = \sum_{r=0}^{\frac{N}{2}-1} f_1(r)\, W_{\frac{N}{2}}^{rK} + W_N^K \sum_{r=0}^{\frac{N}{2}-1} f_2(r)\, W_{\frac{N}{2}}^{rK} \quad (6)$$

$$X(K) = F_1(K) + W_N^K\, F_2(K), \qquad K = 0,1,\ldots\ldots N-1 \quad (7)$$

where $F_1(K)$ and $F_2(K)$ are the $N/2$-point DFTs of the sequences $f_1(r)$ and $f_2(r)$, respectively.

Since $F_1(K)$ and $F_2(K)$ are periodic, with period $N/2$, we have $F_1(K+N/2) = F_1(K)$ and $F_2(K+N/2) = F_2(K)$. In addition, the factor $W_N^{K+N/2} = -W_N^K$. Hence the equation may be expressed as

$$X(K) = F_1(K) + W_N^K\, F_2(K), \qquad K = 0,1,\ldots\ldots, \frac{N}{2}-1 \quad (8)$$

$$X(K+N/2) = F_1(K) - W_N^K\, F_2(K), \qquad K = 0,1,\ldots\ldots, \frac{N}{2}-1 \quad (9)$$

The direct computation of $F_1(k)$ requires $(N/2)^2$ complex multiplications. The same applies to the computation of $F_2(k)$. Furthermore, there are $N/2$ additional complex multiplications required to compute $W_N^k F_2(k)$. Hence the computation of $X(k)$ requires $2(N/2)^2 + N/2 = N^2/2 + N/2$ complex multiplications. This first step results in a reduction of the number of multiplications

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Issue 7, July 2017

from N^2 to $N^2/2 + N/2$, which is about a factor of 2 for N large.

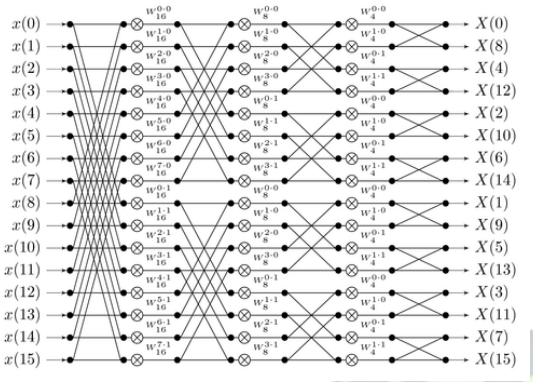

Fig.1: Signal flow graph for radix-2 FFT.

The Split-Radix FFT, alongside its varieties, long had the qualification of accomplishing the most minimal distributed number-crunching operation tally (add up to correct number of required genuine options and duplications) to figure a DFT of energy of-two sizes N. The number-crunching tally of the first split-radix calculation was enhanced in 2004 .In spite of the fact that the quantity of number juggling operations is not the sole factor (or even essentially the predominant factor) in deciding the time required to figure a DFT on a PC, the subject of the base conceivable check is of longstanding hypothetical intrigue. [7] proposed a system, this paper presents an effective field programmable gate array (FPGA)-based hardware implementation of a parallel key searching system for the brute-force attack on RC4 encryption. The design employs several novel key scheduling techniques to minimize the total number of cycles for each key search and uses on-chip memories of the FPGA to maximize the number of key searching units per chip. Based on the design, a total of 176 RC4 key searching units can be implemented in a single Xilinx XC2VP20-5 FPGA chip. Operating at a 47-MHz clock rate, the design can achieve a key searching speed of 1.07 x 107 keys per second. Breaking a 40-bit RC4 encryption only requires around 28.5 h.

The essential thought behind the SRFFT is the use of a radix-2 index guide to the even-list terms and a radix-4 map to the odd-list terms. For the even-list terms, it can be deteriorated as

$$X(2K) = \sum_{n=0}^{\frac{N}{2}-1} [x(n) + x\left(n+\tfrac{N}{2}\right)]\, W_{N/2}^{nK}, \quad K = 0,1,2\ldots\ldots\ldots\ldots,\tfrac{N}{2}-1 \qquad (10)$$

For the odd-record terms, it can be disintegrated as

$$X(4K+1) = \sum_{n=0}^{\frac{N}{4}-1} [x(n) - jx\left(n+\tfrac{N}{4}\right) - x\left(n+\tfrac{N}{2}\right) - x\left(n+\tfrac{3N}{4}\right)] W_N^n W_{N/4}^{Kn} \qquad (11)$$

$$X(4K+3) = \sum_{n=0}^{\frac{N}{4}-1} [x(n) + jx\left(n+\tfrac{N}{4}\right) - x\left(n+\tfrac{N}{2}\right) - x\left(n+\tfrac{3N}{4}\right)] W_N^{3n} W_{N/4}^{Kn} \qquad (12)$$

where $K = 0, 1, \ldots, N/4$. The formulas above result in the L-shaped split-radix butterfly structure, which can be found in [2] and the scheduling of the L-shaped butterfly is irregular. Assume that we have $N = 2^S$ point FFT, both SRFFT and radix-2 FFT require S passes to finish the computation, as shown in Figs. 1 and 2. For SRFFT, the total number of the L butterflies N_{SR} is given by

$$N_{SR} = \frac{[(3S-2)2^{S-1}+(-1)^S]}{9} \qquad (13)$$

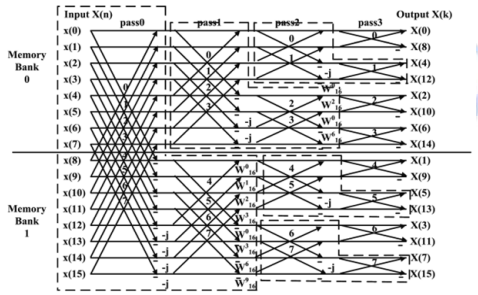

Fig.2: Signal flow graph for SRFFT

Each L butterfly contains two nontrivial complex multiplications, and therefore, the total number of nontrivial complex multiplications M_{SR} in SRFFT is

$$M_{SR} = \frac{2[(3S-2)2^{S-1}+(-1)^S]}{9} \qquad (14)$$

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Issue 7, July 2017

In the (S − 1) pass, the number of SR butterfly N_{S-1} is

$$N_{S-1} = \frac{[2+(-1/2)^{S-2}]N}{12} \quad (15)$$

However, in the $(S-1)$ pass, each L butterfly does not contain any nontrivial twiddle factors and hence, the total number of nontrivial multiplications M'_{SR} in SRFFT is

$$M'_{SR} = M_{SR} - 2N_{S-1} \quad (16)$$

For the conventional radix-2 FFT, the total number of complex multiplications M_{R2} is

$$M_{R2} = (S-1)2^{S-1} \quad (17)$$

III. HARDWARE IMPLEMENTATION

A. Shared-Memory Architecture

In computer systems, shared memory is that may be simultaneously accessed by multiple programs with intent to provide communication among them or avoid redundant copies. Shared memory is an efficient means of passing data between programs. Depending on context, programs may run on a single processor or on multiple separate processors. Using memory for communication inside a single program, e.g. among its multiple threads, is also referred to as shared memory.

The architecture of shared-memory processor is shown in Fig. 3. The FFT data and the twiddle factors are stored in the RAM and ROM banks, respectively. The flow graph of split-radix algorithm is the same as radix-2 FFT except for the locations and values of the twiddle factors and therefore, the conventional radix-2 FFT data address generation schemes could also be applied to SRFFT (RAM address generator). However, the mixed-radix property of SRFFT algorithm leads to the irregular locations of twiddle factors and forbids any conventional address generation algorithm (ROM address generator).

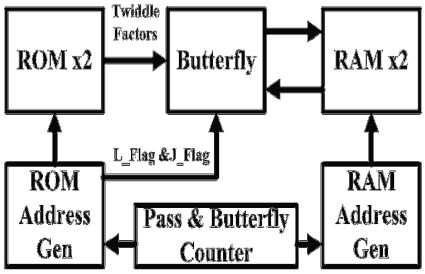

Fig.3 Shared memory architecture

B. Modified Radix-2 Butterfly Unit

The modified butterfly unit structure which is shown in Fig.4. The structure of this butterfly unit is determined by the fact that the SRFFT has multiplications of both upper and lower legs. To prevent unnecessary switching activity, we put the clock gating registers in the multiplier path and a few registers are placed at the address port of memory banks to synchronize the whole design. The key to use this architecture is knowing about which butterflies require no multiplications (the complex multipliers are then skipped), trivial multiplications (swapping), and nontrivial multiplications (using complex multipliers). In Section C, we present an efficient algorithm to solve this problem.

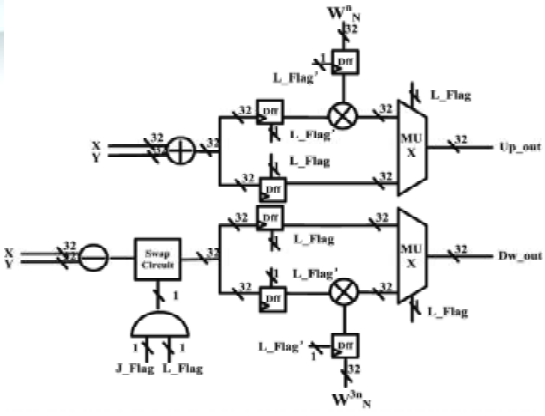

Fig.4: Modified butterfly structure

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Issue 7, July 2017

C. Address Generation of Twiddle Factors

The flow graph for the 16-point SRFFT is shown in Fig.2. There are two kinds of twiddle factors: j and W_n. For those multiplications involving j is called trivial multiplications, because these operations are essentially the swapping of the real and imaginary part of the multiplier, hence no multiplication is involved. For those multiplications involving W_n are called nontrivial multiplications, because complex multipliers are used to complete these operations. In Fig.2 each area surrounded by the dashed lines is called one L block which is formed by L butterflies in each pass and there are totally five L blocks for a 16-point SRFFT.

```
1: Initialization: set all the L_Flag to 1
2: for P = 0 to S − 1 do
3:   for B = 0 to 2^(S−1) do
4:     J_Flag ← b_(S−2−P)
5:     if L_Flag_b = 1 then
6:       if J_Flag = 1 then
7:         Multiply trivial twiddle factor j (swapping)
8:         ROM_Address ← don'tcare
9:         L_Flag_b ← 0
10:      else
11:        No Multiplication Required
12:        ROM_Address ← don'tcare
13:        L_Flag_b ← 1
14:      end if
15:    else
16:      Multiply non-trivial twiddle factor W_n
17:      L_Flag_b ← 1
18:      ROM_Address
19:      ← b_(S−2−P) b_(S−3−P) ... b_0 0 ... 0
20:    end if
21:  end for
22: end for
```

Fig. 5. Pseudocode for tracking trivial and nontrivial twiddle factors.

Given N = 2^S point FFT data, we first have the following definitions.

1) Butterfly Counter B: (S − 1)-bit counter that indicates, in each pass, which butterfly is currently under operation.

2) Pass Counter P: ([log2 S])-bit counter that indicates which pass is currently under operation.

3) L_Flag: A set of variables indicate if the butterfly under operation is in the L-shaped block. Each variable corresponds to one butterfly in each pass and the number of such variables is the same as the number of radix-2 butterflies in each pass. For example, in Fig. 2, each pass contains eight butterflies so eight L_Flags are required.

4) J_Flag: A variable indicates if the butterfly under operation should multiply the trivial twiddle factor j (swapping). In Fig. 2, we have made two observations. First, in the current pass, if the ith butterfly is not within the L block (L_Flag = 0), in the next pass, the same ith butterfly will be definitely within the L block. For example, butterflies 100, 101, 110, and 111 are not within the L block in pass 1 (because they belong to the L block in pass 0) and butterflies 100, 101, 110, and 111 are within the L block in pass 2.

The second observation is that if the butterfly is within the L block in this pass (L_Flag = 1), in the next pass, whether it will be within the L block is determined by J_Flag. If, in the current pass, the ith butterfly needs to multiply j (J_Flag = 1), then in the next pass, the same butterfly needs to multiply Wn. For example, butterflies 100, 101, 110, and 111 are within the L block in pass 0, and they need to multiply by j; in pass 1, the butterflies 100, 101, 110, and 111 need to multiply Wn. On the other hand, if, in the current pass, the ith butterfly does not need to multiply j (J_Flag = 0), then in the next pass, the same butterfly still belongs to L block and does not need to multiply Wn. For example, butterflies 000, 001, 010, and 011 are within the L block in pass 0 and they do not need to multiply by j (J_Flag = 0), in pass 1, the butterflies 000, 001, 010, and 011 still belong to the L block and do not need to multiply Wn.

The high level structure of the proposed algorithm is shown in Fig. 5. All L_Flag are set to one before the program starts,

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Issue 7, July 2017

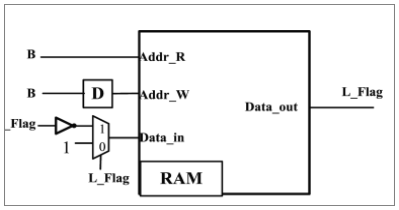

Fig.6.L_Flag structure

because all the butterflies in pass 0 are contained in the first L block.

L_Flags could be efficiently implemented as an RAM block, as shown in Fig. 6. Butterfly counter B is served as the read address of RAM and at each clock cycle, the corresponding L_Flag value for the current butterfly is provided. The updated L_Flag value for the next pass is written to this memory at next clock cycle. The size of this memory is 2^{S-1} bits, which equals to the number of butterflies in each pass. Such a size is trivial on the modern field-programmable gate array (FPGA). For example, for a 2048-point FFT only 1 kbit is required.

J_Flag is a combinational signal. The value of this variable depends on the butterfly counter B. In the Pth pass, J_Flag equals to b_{S-2-P} .

TABLE II
ADDRESS GENERATION TABLE OF THE PROPOSED ALGORITHM 1 FOR A 16-POINT SRFFT

Pass 00				
Butterfly	L_Flag	J_Flag	ROM0 Addr	ROM1 Addr
000	1	0	XX	XX
001	1	0	XX	XX
010	1	0	XX	XX
011	1	0	XX	XX
100	1	1	XX	XX
101	1	1	XX	XX
110	1	1	XX	XX
111	1	1	XX	XX
Pass 01				
Butterfly	L_Flag	J_Flag	ROM0 Addr	ROM1 Addr
000	1	0	XX	XX
001	1	0	XX	XX
010	1	1	XX	XX
011	1	1	XX	XX
100	0	0	00	00
101	0	0	01	01
110	0	1	10	10
111	0	1	11	11
Pass 10				
Butterfly	L_Flag	J_Flag	ROM0 Addr	ROM1 Addr
000	1	0	XX	XX
001	1	1	XX	XX
010	0	0	00	00
011	0	1	10	10
100	1	0	XX	XX
101	1	1	XX	XX
110	1	0	XX	XX
111	1	1	XX	XX
Pass 11				
Butterfly	L_Flag	J_Flag	ROM0 Addr	ROM1 Addr
000	1	0	XX	XX
001	1	0	XX	XX
010	1	0	XX	XX
011	1	0	XX	XX
100	1	0	XX	XX
101	1	0	XX	XX
110	1	0	XX	XX
111	1	0	XX	XX

In the last pass, L_Flag is set to one and J_Flag is set to zero. When nontrivial multiplication is required, twiddle factors need to be retrieved from the ROM banks. Unlike conventional method that stores all the Wn in one ROM bank, we organize Wn in two ROM banks: one stores Wn for the upper leg of the butterfly unit and the other stores those for the lower leg of the butterfly unit. Started in pass 1, in the Pth pass the address of each ROM bank is given by
$b_{S-2-P}\, b_{S-3-P} \ldots b_0 0 \ldots 0$ (following $(P-1)$ '0' s).

It is worth mentioning that in conventional implementations, the twiddle factors are required for each butterfly so ROM banks are always enabled, and in our implementation, the L_Flag signal can be used as the enable signal for the ROM banks, since that if the butterfly belongs to the L block, no multiplication is required. In each pass except for the last one,

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Issue 7, July 2017

J_Flag equals to b_{S-2} .

The address for each ROM bank is given by
$b_{S-2}b_{S-3} \ldots b_1$. (18)

IV. IMPLEMENTATION AND RESULTS

The simulation result of the shared memory 16-point SRFFT are shown in Fig.7. The total inputs to the 16-point SRFFT are 16 and each input is comprises of real and imaginary number so the outputs also 16 having both real and imaginary part. The FFT is synthesized under the constraint of 800MHz in Xilinx ISE 13.2 targeting for Virtex-6 XC6VLX760 device FF1760 package with a speed grade of -2. Power is measured by Xilinx XPower analyzer using the switching activity interchange format file recorded in a sufficient long simulation time. In the given architecture, when the FFT size increases, a larger RAM and ROM size is required, but the butterfly unit does not change. The limitation of the proposed design is the usage of large number of resources used in the butterfly unit. This limitation is removed by using different butterfly structures for additions and multiplications.

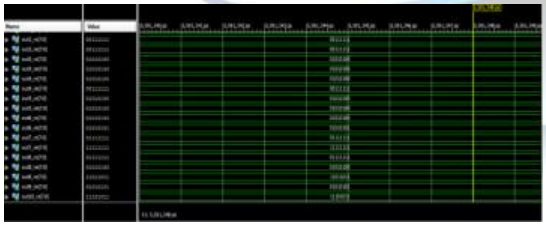

Fig.7: Simulation result of SRFFT (16-Point)

RTL Schematic:

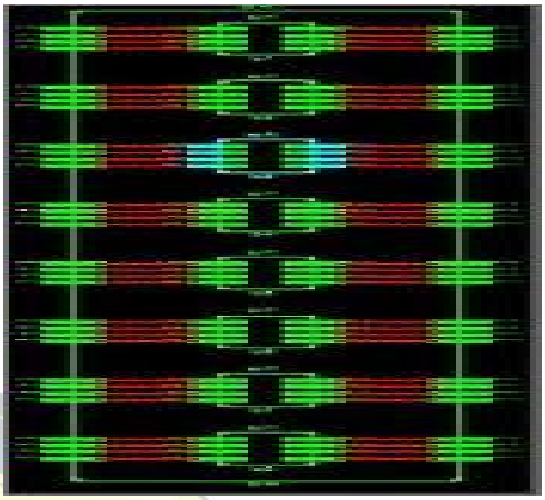

Fig.8: RTL schematic of SRFFT

Design Summary

Device Utilization Summary (estimated values)			[-]
Logic Utilization	Used	Available	Utilization
Number of Slices	128	4656	2%
Number of 4 input LUTs	256	9312	2%
Number of bonded IOBs	512	232	220%

Comparison results of FFT & SRFFT

Parameter	FFT (16-Point)	SRFFT (16-Point)
Delay (ns)	27.721	1.324
Memory(MB)	508.67	261.2
Power (mW)	8.245	4.472
Frequency(MHZ)	36	755

V.CONCLUSION

In this brief, a shared-memory-based low power SRFFT processor is designed, implemented and verified. The SRFFT processor is computed by using a modified radix-2 butterfly unit. The butterfly unit exploits the multiplier-gating technique to save dynamic

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Issue 7, July 2017

power. This method reduces the dynamic power consumption at the expense of more hardware resources. In addition, two novel address generation algorithms for both the trivial and nontrivial twiddle factors are developed. Simulation results show that compared with the conventional FFT implementation the shared-memory SRFFT design achieves 52% lower power consumption & 20% reduced latency when computing a 16-point complex valued transform. Since SRFFT has the minimum number of multiplications compared with other types of FFT, the results are optimal in the sense of floating point operations.

REFERENCES

[1] P. Duhamel and H. Hollmann, "'Split radix' FFT algorithm," Electron. Lett., vol. 20, no. 1, pp. 14–16, Jan. 1984.

[2] M. A. Richards, "On hardware implementation of the split-radix FFT," IEEE Trans. Acoust., Speech Signal Process., vol. 36, no. 10, pp. 1575–1581, Oct. 1988.

[3] J. Chen, J. Hu, S. Lee, and G. E. Sobelman, "Hardware efficient mixed radix-25/16/9 FFT for LTE systems," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 23, no. 2, pp. 221–229, Feb. 2015.

[4] L. G. Johnson, "Conflict free memory addressing for dedicated FFT hardware," IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process., vol. 39, no. 5, pp. 312–316, May 1992.

[5] D. Cohen, "Simplified control of FFT hardware," IEEE Trans. Acoust., Speech, Signal Process., vol. 24, no. 6, pp. 577–579, Dec. 1976.

[6] X. Xiao, E. Oruklu, and J. Saniie, "An efficient FFT engine with reduced addressing logic," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 55, no. 11, pp. 1149–1153, Nov. 2008.

[7] Christo Ananth, Muthamil Jothi.M, M.Priya, V.Manjula, "Parallel RC4 Key Searching System Based on FPGA", International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE), Volume 2, Special Issue 13, March 2016, pp: 5-12.

[8] A. N. Skodras and A. G. Constantinides, "Efficient computation of the split-radix FFT," IEE Proc. F-Radar Signal Process., vol. 139, no. 1, pp. 56–60, Feb. 1992.

[9] H. V. Sorensen, M. T. Heideman, and C. S. Burrus, "On computing the split-radix FFT," IEEE Trans. Acoust., Speech Signal Process., vol. 34, no. 1, pp. 152–156, Feb. 1986.

[10] J. Kwong and M. Goel, "A high performance split-radix FFT with constant geometry architecture," in Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE), Dresden, Germany, Mar. 2012, pp. 1537–1542.

[11] W.-C. Yeh and C.-W. Jen, "High-speed and low-power split-radix FFT," IEEE Trans. Signal Process., vol. 51, no. 3, pp. 864–874, Mar. 2003.