http://ijea.jctjournals.com/ ISSN(ONLINE):2320 – 0804

Volume-11, Issue-05, May, 2022
International Journal ofEngineering Associates (IJEA)
Page Number: 01-08

VLSI ARCHITECTURE OF COMPLEX VALUED PIPELINE FAST FOURIER TRANSFORM

Ajay Kumar Tiwari , Dr. Neha Singh
[1]M.Tech Scholar ,[2]Assistant Professor
Department of Electronics and Communication (ECE)
IASSCOM Fortune Institute Of Technology, Bhopal. M.P. INDIA
ajayktiwari79@gmail.com, nsbaghelit@gmail.com

Abstract- **This study presents an energy-efficient serial pipelined architecture of fast Fourier transform (FFT) to process real-valued signals. A new data mapping scheme is presented to obtain a normal order input–output without the requirement of a post-processing stage. It facilitates reduction in the computational workload on the hardware resources which is confirmed through mathematical derivations. Further, the proposed design involves a novel quadrant multiplier with relatively lower hardware complexity. It performs the quarter operation of a complex multiplier in one clock cycle, and thereby consumes relatively lower power. Moreover, in the last stage, a merged unit for butterfly computation and data re-ordering is also proposed which performs either a half-butterfly operation or interchanges data, and thereby reduces the hardware usage. Application specific integrated circuit synthesis and field programmable gate array results show that for a 1024-points FFT computation, the proposed architecture offers 10.26% savings in area, 20.83% savings in power, 16.98% savings in area-delay product and 26.76% savings in energy-per-sample, 7.79% savings in sliced look-up tables, and 11.93% savings in flip-flops over the best existing design.**

Index Terms -FFT,LTI, Power, Pipeline, VLSI .

I. INTRODUCTION

Fourier Transform is the basis of many signal processing and communication applications. The Fourier transform has many applications, in fact, any field of physical science that uses sinusoidal signals, such as engineering, physics, applied mathematics, and chemistry, will make use of Fourier series and Fourier transforms.

Fourier transform is essential for converting any information signal from time to frequency domain. The Fourier transform is one of several mathematical tools that is useful in the analysis and design of LTI systems. This type of signal representation basically involves the decomposition of the signals in terms of sinusoidal (or complex exponential) components. With such a decomposition, a signal is said to be represented in the frequency domain. Fourier series is used for the class of periodic signals and Fourier transform is used for the class of finite energy signals. These decompositions are extremely important in the analysis of LTI systems because the response of an LTI system to a sinusoidal input signal is sinusoid of the same frequency but of different amplitude and phase. The conversion is necessary for spectrum analysis because in almost all the applications processing of signals and analysis of any system is done in the frequency domain. The basic motivation for developing the frequency analysis tools is to provide a mathematical and pictorial representation for the frequency components that are contained in any given signal. The process of obtaining the spectrum of a given signal using the basic mathematical tools is known as Spectral analysis.

$$\mathbf{X_k} = \sum_{n=0}^{N-1} \mathbf{x_n} e^{-\frac{2\pi i}{N} nk}, \quad \textbf{for k = 0: N-1} \quad \textbf{... (1.1)}$$

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} \mathbf{X_k} e^{\frac{2\pi i}{N} nk}, \quad \textbf{for n = 0: N-1} \quad \textbf{... (1.2)}$$

Where xn denotes input sequence in discrete time domain and Xk denotes the output sequence in frequency domain. Note that the index term m is unit-less. This is the primary difference between the DFT and the discrete-time Fourier transform (DTFT). The DTFT is a special case of the DFT, in which the input sequence is assumed to be defined in the time domain. The DTFT will always use the input index term t, for time.

II. LITERATURE REVIEW

The memory accessing problem is common to both pipeline and in-place designs. Every butterfly needs two

inputs and gives two outputs. Two data are need to be accessed at a time but a memory can provide only one data at a time. Memory accessing problem is severe in in-place because a single memory is used for all stages. Commonly used solutions include partitioning the memory into several banks.

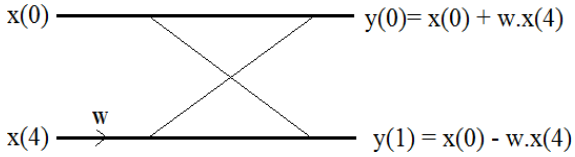


Fig 2 Radix-2Butter fly accessing two data at a time

A bank is a sub-section of memory. A large memory is divided into several banks of smaller size so that more number of data can be given to a system and processed. For accessing two different data at a time, RAM memory is divided into two banks, say Bank 1 and Bank2. At first stage, Bank 1 has data x(0), x(4), x(2), x(6) and Bank 2 has x(1),x (5),x(3), x(7). So now two data can be accesses data time for the butterfly operation.

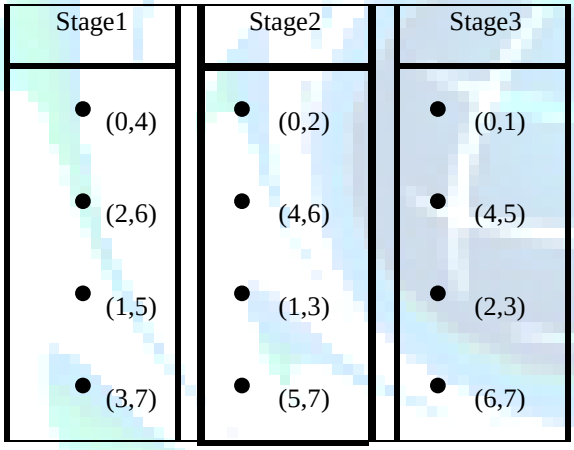
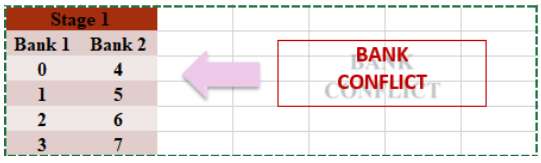


Table I. Order of data at different stages

Bank Conflict: In FFT structure, jumbling of data occurs at subsequent stages and every time two data to be accessed should always be in different banks to avoid memory accessing problem. If different banks are accessed in a single cycle then no conflict; each data can be accessed from different banks. But if banks are less; say two for 8-point then two data to be accessed may lie in the same bank. Because of data jumbling, two data may lie in the same bank at any arbitrary stage and conflict occurs. Hence at every stage bank assignment has to be changed accordingly. This is known as Bank Conflict.



As shown in Table I the sequence of the data to be accessed varies from stage to stage. So if the data is arranged in two banks according to stage 1 as shown above in Table II then at stage 2 conflict occurs if above arrangement is followed. So, bank conflict occurs almost at every stage. The number of banks has to be increased for subsequent stages which may require large memory. Hence, bank conflict is a serious issue in pipeline and in-place design.

Memory accessing problem was first resolved by Pease. Let (s, t) represent the butterfly operation:

$$X'(S) = X(s) + w * X(t) \quad 2.1$$

$$X'(t) = X(s) - w * X(t) \quad 2.2$$

Pease showed that for every butterfly two indices involved differ in their parity (the parity of X is defined as zero if the number of ONES in the binary representation of X is even, and one otherwise). Hence it is possible to organize the N-point memory needed in two banks, according to the parity of the addresses, such that in any cycle two points X(s) and X(t) are accessed in parallel, since s and t differ in parity, and X(s) and X (t) are stored in different banks.

E.g. Data = 4 i.e. 100; parity = 1 → Bank 1 Data = 6 i.e. 110; parity =0 → Bank 0

Cohen used a specific memory access pattern for implementation of radix 2 DIF. The design uses barrel shifter to shift address bits at subsequent stages after input is applied. The pattern accessed at 1st stage is (0, 1) (2, 3) (4, 5) (6, 7) at 2nd stage (0, 2) (4, 6) (1,3) (5,7) and so on. The reason for this order is that the j th butterfly in the i th iteration is (s, r) where:

s = **ROTATEn** (2j, i)

n = **log2N**, for i = 0, 1….n-1

t = **ROTATEn** (2j+1, i),

for j= 0, 1…N/2-1 Where ROTATEn(X, m) is the value of X rotated left, by m bits, within n bits, e.g., ROTATE4(13, 3) = 14 and ROTATE4(5, 2) = 5.

13: 1101- shift 3 times left- 1011, 0111, 1110.

Ma's scheme shortens the delay of address generation with comparable hardware cost. A common drawback of these two schemes is that two memory banks of size $2n-1$ are made active for each butterfly. Generally, if the number of ROM access can be reduced, then power consumption in the ROM can be lowered. In addition, since precharge of bit-lines and charge/discharge of word lines are two major power consumption contributors in a dynamic ROM, power is consumed even for the same subsequent coefficient access. Therefore, a significant amount of power can be saved by using a circuit [address-transition detection (ATD)] to enable

a coefficient access only if an address change appears on the address bus.

The drawback of Cohen's method is the address generation delay (barrel shifters and address parity check). Ma's method needs only barrel shifters and avoids parity check, resulting in a reduced address generation delay. In Ma's scheme active memories are reduced by using 4 banks hence lower power is consumed.

For the single butterfly implementation, a basic problem arises is how to efficiently arrange memory read/write accesses for the purposes of increasing its throughput rate. The commonly used solutions include: (1) Use the high-radix implementation to reduce the total number of memory accesses at the expense of increasing the arithmetic complexity, i.e., the hardware requirement of a high-radix butterfly unit. (2) Partitioning the memory into several banks to allow concurrent accesses of multiple data at the price of using a more complicated addressing scheme, which might correspond to a higher routing area.

III. MATHEMATICAL FORMULATION AND BLOCK DIAGRAM OF FFT PROCESSOR

The full transform requires

1) An address generator,
2) A "butterfly" operator to do the complex multiply / add,
3) Memories roots-of-unity (twiddle factor) generator

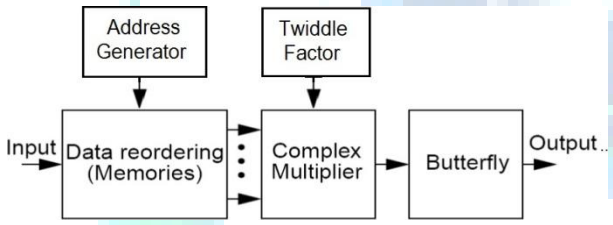


Fig.3 Block Diagram of FFT Processor

3.1.1 Description:

- The address generator provides the locations for the fetch and store operations to and from memory.
- The butterfly operator is the heart of the FFT. It provides transform at various stages
- The memory is needed to store the intermediate results as the transform runs.
- The twiddle factor generator is used to generate twiddle factors at various stages.

IV. HARDWARE DESCRIPTION LANGUAGE

Digital logic design starts from system requirement specifications. Then system specifications are converted into logic diagrams. The digital logic circuit is a combination of combinatorial logic and sequential logic blocks. The complex requirements are converted into state diagrams. The sestate diagrams represents various states of digital systems as per requirement specifications. This state diagram also represents inputs, outputs and transitions between states based on input events. This state diagram with complete details then converted into finite state machine (FSM) with additional combinational logic. Digital arithmetic logic circuits such as adders, sub tracters, multipliers and dividers are used to perform arithmetic operations. Digital logic has a special type of circuits called registers, which are used to store the logic values of a digital systems. The sequential circuits uses a special signal called clock as a reference signal. All the sequential circuits are synchronized to clock signals. The example of synchronized sequential circuits are shift registers, counters and finite state machines. Using hardware description languages such as Verilog HDL [8] or VHDL [17] it is possible to simulate, synthesize and implement digital logic as integrated circuits.

The digital simulation is used to verify functionalityofdigitallogic.Thesynthesisprocessinvolvestranslating HDL language constructs into suitable digital logic blocks. The implementation process involves programming in case of FPGA devices and designing Integrated Circuits (ICs) in case of Application Specific Integrated Circuits(ASICs). Digital logic value high, low, unknown and high impedance are used to represent bit values. Based on the bit position in a single bit or group of bits, and number interpretation, binary value of bit or byte is described. In digital design bits can be interpreted as an unsigned number, signed number, 2's complement number, fixed point number and floating-point number systems. Digital logic design used to build digital electronic systems such as FIR filters, FFTs, processors, memories and other control systems [2]. It involves converting systems specification into data path and control circuits. The data path consists of logic gates and registers. The control circuits consists of finite state machines, counters and microprocessors.

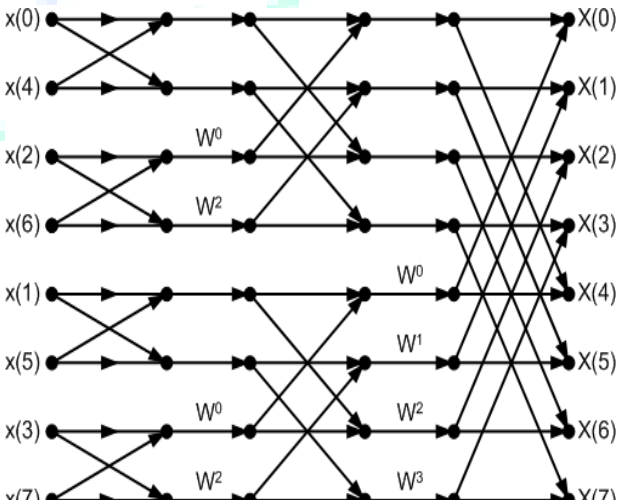


Fig 4. Architecture of 8 point FFT

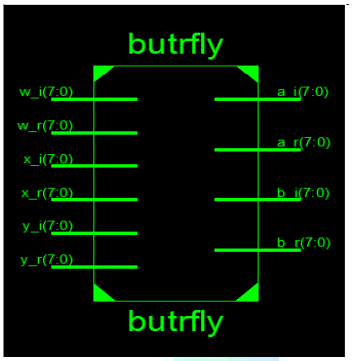


Fig. 5 RTL Schematic for entity butterfly

Scaling by 100: Initially scaling by 100 is used. Input data is scaled down by 100. For example, 1 is scaled down as 0.01, and 0.01 in 8 bits can be written as (7F and 1I (for sign)) = 00000010. This 8 bit representation actually leads to 0.0078 when we add in butterfly say -0.00828, we should get 0.0029 but we get 0.0078-0.00828= -0.0039. This result has been checked manually. This error occurred because of insufficient bits for representation of very small numbers. Also because of scaling by large factor, after multiplication with fractional twiddle factor, data becomes more smaller and 8 bits will be insufficient for them, e.g.0.00465 = 0.00 (00000000-11001111); step size is 1/ 27 and 0.0046 < 1/27 so the number of bits is smaller to represent smaller number. So solution can be smaller scaling factor or 16 bits representation on fraction side (data size increased).

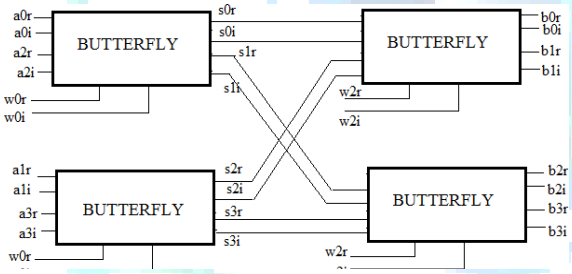


Fig. 6.4-point FFT Structure

8-point FFT Structure :-The inputs are selected as 2I and 8F.In 8-point FFT structure, butterfly is used as a component and twelve butterflies are used in the structure; the final as well as intermediate results are also of 2I and 8F (same as operands). Ten set of data normalized input in range of -0.5 to 0.5 is taken using random values from MATLAB. These data are read one by one from input file and 8-point FFT is performed successively and stored in output file. The 8 point FFT unit is designed by structural modeling. There are total 16 inputs (real and imaginary), 8 twiddle factors (real and imaginary) and 16 outputs (real and imaginary). The total number of butterflies are 12 hence component butterfly is declared in architecture and 12 butterflies are called for subsequent operations. Intermediate results are used with the help of 32 signals of 11 bits. Signals s0r, s0i, s1r and s1i store result temporarily of first butterfly at first stage similarly g0r,g0i, g1r and g1i store result of first butterfly at stage 2. The final stage results are directly provided as output. Outputs are of same size of operands.

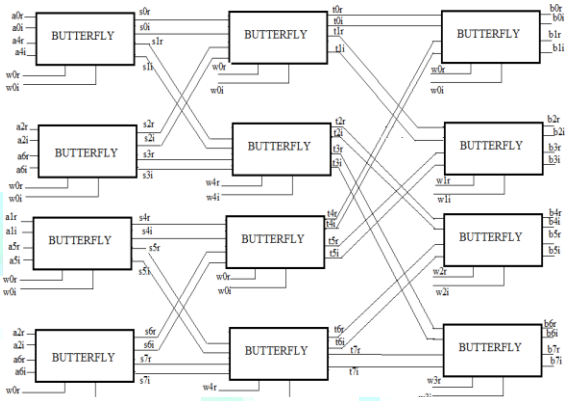


Fig.7 8-pointFFTStructure

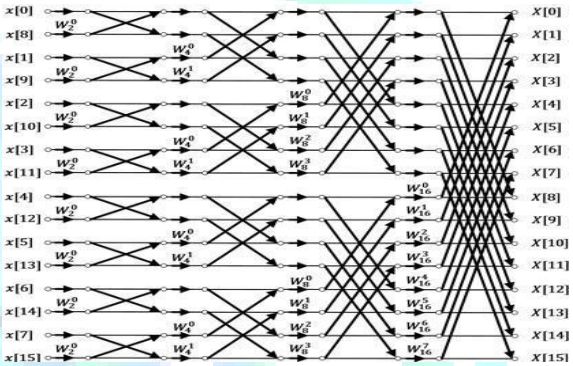


Fig.4.2.4 16-point FFT Structure

IV. MODELING& ANALYSIS OF THE DUAL HYDROFORMING PROCESS

The Incase of the VHDL implementation on codes have been written and run successfully for complex FFT using radix2. Radix2 parallel structure was implemented for 4 point, 8 point and 16 point whereas radix 2 pipeline structure was implemented for 8point. The results obtained from VHDL have been compared with results obtained with MATLAB. In MATLAB ,direct commands are available for FFT computation. MATLAB commands for various FFT structures are:

Table III. MATLAB Commands for FFT

FFT structure	Command
4-point FFT	fft(x,4)
8-point FFT	fft(x,8)
16-point FFT	fft(x,16)

Simulation 1: Radix-2 Butterfly Implementation for various scaling Techniques: In radix-2 butterfly, four inputs

are read from text file xr, xi, yr, yi and twiddle factors wr, wi are defined in the test bench. The butterfly operation is performed and the four outputs ar, ai, br and bi are written in the text file in order after the simulation. An example showing result for radix-2 butterfly is shown in Table 5.1.2 and result is compared with MATLAB results. We have taken inputs in the range of (-1 to 1) and the results have been compared for various scaling factors in Table 5.1.3. For input 0.1443, -0.1213, 0.3115, 0.0328, outputs are:

Table IV. Comparison between MATLAB and VHDL for Radix-2 butterfly

MATLAB	VHDL
0.4555	**0.4531**
-0.0886	**-0.0938**
-0.1669	**-0.168**
-0.1541	**-0.1523**

V. CONCLUSION AND FUTURE SCOPE

From In this thesis work, we have tried to resolve the memory accessing problem and bank-conflict in the pipeline architecture. The various schemes have used SRAM and divided it into banks according to size of FFT. SRAM is quite expensive to be included in hardware chips and used for small and portable devices. Hence we try to resolve the issue by using registers for to store the intermediate results obtained after subsequent horizontal foldings. The other issue is the product size obtained from twiddle factor and data multiplication. The product size increases two fold at every stage so in order to reduce memory consumption and latency due to large size multiplication operations, we apply truncation to obtained product at each stage. The input-data are scaled suitably to reduce the truncation error. The truncation error calculated as mean square error is estimated through simulation and found to be 0.0104% for N=8 and 0.0169% for N=16.

The Direct FFT computation architecture with N sample input needs a single clock cycle to compute FFT algorithm. This architectures requires more hardware resources (N.log2 *N* Butterflies). The Pipelined FFT computation architecture with N sample input needs (1.5*N) number of clock cycles (approximately) to compute FFT algorithm. This architecture requires moderate hardware resources (log2 N Butterflies and log2 N Switches). The direct FFT and pipelined FFT architectures can be used for WLAN OFDM 802.11a implementation. Because these four FFT architectures support input signal bandwidth greater than 20 MHz By using faster FPGAs it is possible to use all the six FFT architectures for WLAN OFDM 802.11a implementation. The methods used in this thesis, can be used to implement radix 4 DIT, radix 8 DIT and DIF algorithms. It is also possible to apply methods discussed in this thesis to the higher order FFTs, such as 128, 256, 512, 1024, 2024, 4096, 8192 and so on.

REFERENCES

[1] J.G.Proakis, D.G .Manolakis, ”DSP Principles, Algorithms and Applications”, PHI Publications,4th edition,2007.

[2] W. W. Smith and J. M. Smith. Handbook of Real-Time Fast Fourier Transforms. IEEE, 1995. …radix

[3] J. W. Cooley and J. W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. Mathematics of Computation, 19(90):297–301, 1965...DIT

[4] M. C. Pease, “Organization of large scale Fourier processors,” JACM, VOI. 16, pp. 474-482, July 1969.

[5] Cohen, D, “Simplified Control of FFT Hardware”. IEEE Transactions on Acoustics, Speech, Signal Processing, vol. 24, pp. 577–579, December 1976.

[6] Ma.Y, “An Effective Memory Addressing Scheme for FFT Processors”. IEEE Transactions on Signal Processing, vol. 47, no. 3, pp. 907–911, March 1999.

[7] H. F. Lo, M. D. Shieh, and C. M. Wu, “Design of an efficient FFT processor for DAB systems,” in Proc. IEEE Int. Symp. Circuits Systems (ISCAS’), 2001, pp. 654 –657

[8] Manohar Ayinala, Yingjie Lao, and Keshab K. Parhi, “An In-Place FFT Architecture for Real-Valued Signals”, IEEE Transactions On Circuits And Systems—Ii: Express Briefs, Vol. 60, No. 10, October 2013

[9] M. Garrido, K. K. Parhi, and J. Grajal, “A pipelined FFT architecture for real-valued signals,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 56, no. 12, pp. 2634 –2643, Dec. 2009.

[10] onlinelibrary.wiley.com/doi/ 10.1002/0471200727.app3/pdf

