IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
ISSN: 2278-2834, ISBN: 2278-8735. Volume 3, Issue 2 (Sep-Oct. 2012), PP 14-19
www.iosrjournals.org

Efficient Implementation of Fast Fourier Transform Using NOC

[1]Lalitha Bhavani.Maddipati, [2]D. Nataraj
[1]M.Tech student, [2]Associate professor Pragati Engineering College, Surampalem,

Abstract: *In this paper, improved algorithms for radix-8 FFT are presented. Various schemes have been proposed for computing FFT. It has Different target domains of applications and different tradeoffs between flexibility and performance. Typically, they need reconfigurable array of processing elements .The applications have been restricted to domains based on floating arithmetic. We introduce floating-point Arithmetic which is based on processing elements. After developing the FFT design we present a routing Algorithm and use topology to reduce power dissipation. These modified radix-8 algorithms provide savings of more than 33% in the number of twiddle factor evaluations*

I. Introduction

An Orthogonal frequency division multiplexing (OFDM) signal consists of a sum of subcarriers that are modulated by using Phase Shift Keying (PSK) or Quadrature Amplitude Modulation (QAM). These days, OFDM technique is widely used for high-speed digital communications, such as xDSL, DAB, DVB-T/H, and WLAN. In OFDM system, Discrete Fourier Transform (DFT)/Inverse-DFT are used and it is a very important operation. Since DFT/IDFT computation requires a large amount of arithmetic operations, we need an efficient FFT algorithm which can reduce the number of arithmetic operations to meet real time computation in OFDM systems.

There are many kinds of FFT architectures used in OFDM systems. They are mainly categorized as three types: The Parallel architecture, the Pipeline architecture and the Shared memory architecture. The Parallel and Pipeline architectures have more buttery processing units to achieve high performance but they consume larger area than the Shared memory architecture. On the other hand, the Shared memory architecture requires only one buttery processing unit and has the advantage of area efficiency. But the Shared memory architecture has a drawback of low throughput and requires a complex circuit design of memory address controller. Fortunately, the lower throughput of the shared memory architecture increases dramatically if the high radix algorithm is used. But the high radix algorithm has defects of more complex memory scheme and limitation that FFT length (N) must be only powers of radix- r (rn).

We focus on the Shared memory architecture for area efficiency and hardware simplicity which are required to make small OFDM receivers. One radix-8 buttery processing unit is used and it has the pipeline structure in order to realize high throughput. However, the FFT computation is restricted to N points which are 8n. We propose the structure which can perform the radix-4 or radix-2 FFT algorithm in the radix-8 buttery processing unit to permit the FFT computation of all points which are powers of 2. Because of choosing N points are a power of the radix-r, the N-point DFT is decomposed into a set of recursively related r-point transforms. Efficient memory assignment and addressing are proposed to reduce the complexity of memory scheme. The ROM-based lookup table storing twiddle factors consumes large area in case of long-length FFT computation. To solve the problem, the twiddle factor generator is replaced with the ROM-based lookup table.

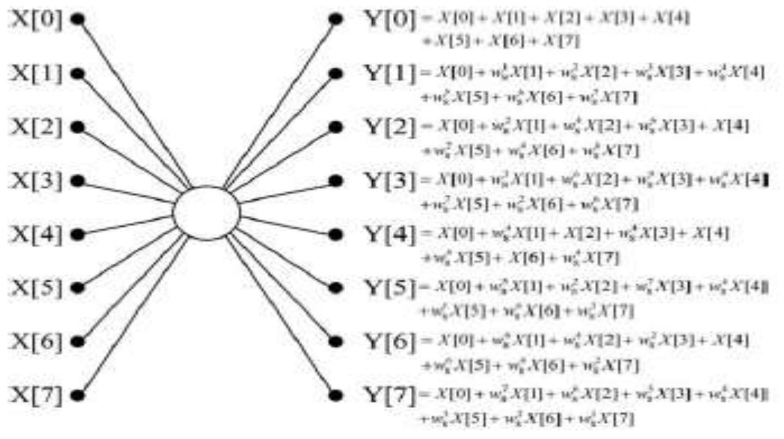
X[0]
X[1]
X[2]
X[3]
X[4]
X[5]
X[6]
X[7]
Y[0] = X[0] + X[1] + X[2] + X[3] + X[4] + X[5] + X[6] + X[7]
Y[1]
Y[2]
Y[3]
Y[4]
Y[5]
Y[6]
Y[7]

Fig 1. Twiddle factors involved in FFT

II. Proposed System

The N-point Discrete Fourier Transform (DFT) of a sequence x(n) is denoted as

$$X[k] = \sum_{n=0}^{N-1} x(n) W_N^{nk}, k = 0, 1, ..., N-1$$

Where Wn is exp(j2=N). To compute X[k] directly, N^2 multiplication and N (N logN) addition are needed. If above X[k] is represented in the matrix form like the following equation,

$$W = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w_N^1 & w_N^2 & \cdots & w_N^{N-1} \\ 1 & w_N^2 & w_N^4 & \cdots & w_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w_N^{N-1} & w_N^{2(N-1)} & \cdots & w_N^{(N-1)^2} \end{pmatrix}$$

A primitive 8th root of unity in R. If R contains the element p2/2, then it can be expressed by

$$\phi = \frac{\sqrt{2}}{2} + \mathrm{I} \cdot \frac{\sqrt{2}}{2}$$

Let R be the field of complex numbers for the remainder of this section. We will assume that a multiplication in C requires 4 multiplications and 2 additions in R, the real numbers.

$$\phi \cdot (A + \mathrm{I} \cdot B) = \frac{\sqrt{2}}{2} \cdot (A - B) + \mathrm{I} \cdot \frac{\sqrt{2}}{2} \cdot (A + B).$$

As introduced in , a radix-8 algorithm can be constructed using the transformation

$$\begin{aligned} R[x]/(x^{8m} - b^8) & \rightarrow & R[x]/(x^m - b) \\ & \times & R[x]/(x^m + b) \\ & \times & R[x]/(x^m - \mathrm{I} \cdot b) \\ & \times & R[x]/(x^m + \mathrm{I} \cdot b) \\ & \times & R[x]/(x^m - \phi \cdot b) \\ & \times & R[x]/(x^m + \phi \cdot b) \\ & \times & R[x]/(x^m - \phi^3 \cdot b) \\ & \times & R[x]/(x^m + \phi^3 \cdot b). \end{aligned}$$

The radix-8 algorithm can be developed by duplicating the steps used to create the radix-2 or radix-4 algorithm at this point. This analysis will produce a transformation matrix given by

$$\begin{pmatrix} +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 \\ -1 & +1 & -1 & +1 & -1 & +1 & -1 & +1 \\ -\mathrm{I} & -1 & +\mathrm{I} & +1 & -\mathrm{I} & -1 & +\mathrm{I} & +1 \\ +\mathrm{I} & -1 & -\mathrm{I} & +1 & +\mathrm{I} & -1 & -\mathrm{I} & +1 \\ -\phi^3 & -\mathrm{I} & -\phi & -1 & +\phi^3 & +\mathrm{I} & +\phi & +1 \\ +\phi^3 & -\mathrm{I} & +\phi & -1 & -\phi^3 & +\mathrm{I} & -\phi & +1 \\ -\phi & +\mathrm{I} & -\phi^3 & -1 & +\phi & -\mathrm{I} & +\phi^3 & +1 \\ +\phi & +\mathrm{I} & +\phi^3 & -1 & -\phi & -\mathrm{I} & -\phi^3 & +1 \end{pmatrix}$$

Which will be used to implement the reduction step

$$F = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w_8^1 & w_8^2 & w_8^3 & w_8^4 & w_8^5 & w_8^6 & w_8^7 \\ 1 & w_8^2 & w_8^4 & w_8^6 & 1 & w_8^2 & w_8^4 & w_8^6 \\ 1 & w_8^3 & w_8^6 & w_8^1 & w_8^4 & w_8^7 & w_8^2 & w_8^5 \\ 1 & w_8^4 & 1 & w_8^4 & 1 & w_8^4 & 1 & w_8^4 \\ 1 & w_8^5 & w_8^2 & w_8^7 & w_8^4 & w_8^1 & w_8^6 & w_8^3 \\ 1 & w_8^6 & w_8^4 & w_8^2 & 1 & w_8^6 & w_8^4 & w_8^2 \\ 1 & w_8^7 & w_8^6 & w_8^5 & w_8^4 & w_8^3 & w_8^2 & w_8^1 \end{pmatrix}$$

It can be shown that the number of operations needed to implement the classical version of the radix algorithm is governed by the recurrence relations

$$\begin{aligned} M(n) &= 8 \cdot M\left(\frac{n}{8}\right) + \frac{9}{8} \cdot n, \\ A(n) &= 8 \cdot A\left(\frac{n}{8}\right) + 3 \cdot n, \end{aligned}$$

where M(1) = 0 and A(1) = 0, M(2) = 1, A(2) = 2, M(4) = 3, and A(4) = 8.We must also subtract multiplications to account for the cases where j = 0. This does not appear to be an improvement compared to the radix-4 algorithms, but we have not yet accounted for the special multiplications by the primitive 8th roots of unity. The recurrence relation

$$M_8(n) = 8 \cdot M_8\left(\frac{n}{8}\right) + \frac{1}{4} \cdot n$$

Modeling an addition in C by 2 additions in R, a multiplication in C by 4 multiplications and 2 additions in R, and a multiplication in C by a primitive 8th root of unity with 2 multiplications and 2 additions in R, the total number of operations in R needed to implement the radix-8 algorithm for an input size which is a power of eight is given by

$$\begin{aligned} M_{\mathbb{R}}(n) &= \frac{4}{3} \cdot n \cdot \log_2(n) - 4 \cdot n + 4, \\ A_{\mathbb{R}}(n) &= \frac{11}{4} \cdot n \cdot \log_2(n) - 2 \cdot n + 2. \end{aligned}$$

This represents a savings of 1/6 · n · log2 (n) additions in R compared to the radix-4 algorithms. The savings for other input sizes are close to the above results, but not quite as attractive. The operation counts for the twisted radix-8 FFT algorithm are the same as the classical radix-8 algorithm.

METHODOLOGIES

Here three types of modules are used.

MODULES

1. ALU
2. Registers
3. Processing Element

III. Arithmetic units involved in FFT computation.

Briefly, architecture consists of an array of processing elements, memory; interconnect structures, I/O ports, and synchronization and reconfiguration mechanisms.

A. *Processing Element*: Each Processing Element consists of resources for functional operations. This can be either one or more dedicated functional units or one or more arithmetic logic units (ALUs). In case of an ALU we have to consider if this unit is configured in advance or during a reconfiguration phase, or if this ALU can be programmed with an instruction set. In case of programmability it has to be considered if a local program is only modulo sequentially executed by a sequencer or if the instruction set includes also conditional branches.

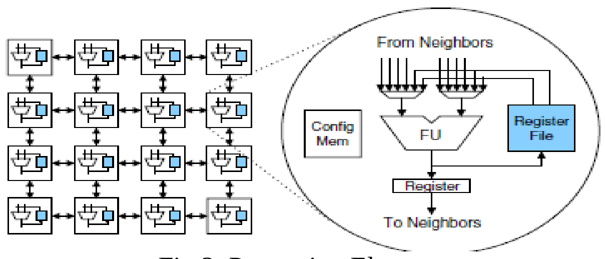

Fig 2: Processing Element

B. *Array***:** The size of the processing array, how many processing elements are present in the array and how are they aligned, as one line of processing elements, several lines of processing elements, or one array of NX N processing elements. Is it connected with the nearest neighboring PEs top, bottom, left, and right. The size of the array can be optimized to a specific application domain. In Fig 3, the area-critical functional units are located outside thePEs and shared among a set of PEs .Each area-critical functional unit is pipelined to curtail the critical path delay,and its execution is initiated by scheduling the area-criticaloperation on one of the PEs that share this area-critical resource. Thus, each PE can be dynamically reconfigured either to perform arithmetic and logical operations with its own arithmetic logic unit (ALU) in one clock cycle, or to perform multiply or division operations using the shared functional unit in several clock cycles with pipelining.

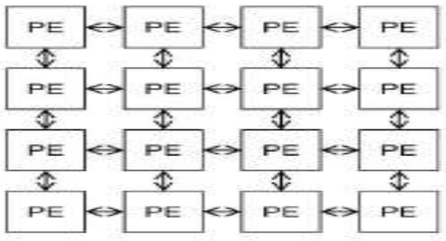

Fig 3: Integer PE Array (4 X 4)

C. *Memory***:** Memory can be divided into local memory in the form of register files inside each processor element and into memory banks with storage capacities in the range from hundreds to thousands of words. The alignment and the number of such memory banks are important for the data mapping. Furthermore, knowledge of several memory modes is helpful, e.g., configuration as FIFO.

D. *Interconnect***:** Here, the structure and number of communication channels is of interest. Type of interconnects used, buses or point-to-point connections used. How are these channels aligned, vertically, horizontally, or in both directions? How long can point-to-point connections be, without delay, or how many cycles have to be taken into account when communicating data from processing element to processing element. Additionally, similar structures are required to handle the control flow.

E. *I/O-ports***:** The maximum bandwidth is defined by the number and width of the I/O-ports. The placement of these I/O-ports is important, since they are responsible for feeding data in and out. Furthermore, it has to be considered if the I/O-port is a streaming port or an interface to external memory.

F. *Reconfiguration***:** Here, the configuration time and the number of configuration contexts have to be considered.In addition; possibilities of partial and dynamical reconfiguration during the execution have to be considered.

i. Loop Unrolling

Loop unrolling is the process of reusing the loop code to include more than one iteration of the old code, in a single pass with the new one. Loop unrolling works by replicating the body of a loop some number of times and scheduling the resulting code as a single basic block. Replicating the loop body has a couple of performance advantages

1. Producing a larger loop body provides a larger block of instructions for the scheduler to work with, which gives the scheduler more options when positioning operations.

2. Combining multiple iterations allows induction variable computations to be combined. These performance improvements are traded against the potential penalty caused by increased I-cache misses on the larger loop body. Loop unrolling is used to minimize stalls that may be encountered inside loops, and also to get rid of the Overhead of running unnecessary branch conditionals.

*ii***.** *Arithmetic Operations*

Unlike normal operations, floating point operations are multi-cycle operations executed on a pair of integer PEs. Normally, each PE fetches a new context word from the configuration cache every cycle to execute the operation corresponding to the context word. However, if the fetched context word is for a multi-cycle operation such as floating-point operation, the control is passed over to the FSM.

1) Arithmetic/logical operations: A PE can execute ALU operations in one clock cycle.

2) Load/store operations: A PE can execute load/store operations in several clock cycles. These operations are executed by dedicated functional resources located outside the PE array. Since the functional resources are pipelined, they can start a new computation every clock cycle.

3) Floating-point operations: A pair of PEs can execute floating-point operations taking several clock cycles.These operations are also multicycle operations like multiply/divide/load/store operations. However, they cannot be pipelined since they are executed directly on the PEs. Among the floating-point operations, however, some operations such as floating-point multiplication and division utilize the dedicated outside integer multiplier or divider. Both operands of a floating-point operation must be of floating-point type since we do not support mixed-type inputs or type casting in our current implementation.

iii. Resource Pipelining

The pipelining is used to achieve high computation throughput. The Loop engine array processor architecture use pipeline execution to map loop onto PE array and the loop program is discomposed to multi-parts which is implemented by some PEs:

1. Loop Finite State Machine is corresponding to the control conditions of loop.
2. Processing Element array is corresponding to the loop body.
3. Loop finite state machine control read/write data from memory.
4. Processing Element array handle the execution of the input data from memory.

The pipeline execution of loop depends on the mPEs and cPEs. mPEs provide flexible storage scheduling and cPEs provide powerful computation capacity. The index value of loop control variable is changed step by step in mPE and the data are read from LM according to the suffix of loop. These data are put to cPE for computing and the results are saved to the address that mPE specifies.

The process of synthesizing an RTL structure from the functional description during the high level synthesis involves three phases:

1. Allocation: Determining the number of instances of each resource needed.
2. Binding: Assignment of resources to computational operations.
3. Scheduling: Timing of computational operations.

IV. Fast Fourier Transform

In this section we present several methods for computing the FFT efficiently.

$$X(K)=\sum_{n=0}^{N-1} x(n)W_N^{nk} \quad \text{For } k=0, 1\ldots N-1$$

$$\alpha(k)=\sum_{m=0}^{7} x(m)W_8^{mk}$$

$$=\left\{\sum_{m=0}^{7}\left[x_{RE}(m)\cos(mk\pi/4)+x_{IM}(m)\sin(mk\pi/4)\right]\right.$$

$$\left.+j\sum_{m=0}^{7}\left[x_{RE}(m)\sin(mk\pi/4)-x_{IM}(m)\cos(mk\pi/4)\right]\right\}$$

A direct realization of this algorithm leads to N^2 multiplications and N (N-1) additions. Of course, a direct implementation is not realistic. Fortunately, the Cooley- Tukey FFT algorithm reduced the order of complexity from N^2 operations down to NlogN operations. 128 -point FFT radix-8 can be implemented by using two 64- point FFT according to Cooley -Tukey FFT algorithm. In view of the importance of the FFT in various digital signal processing applications, such as linear filtering, correlation analysis, and spectrum analysis, its efficient computation is a topic that has received considerable attention by many mathematicians, engineers, and applied scientists.

V. Network-on-a-Chip

Network-on-Chip or Network-on-a-Chip (NOC) is an approach to design the communication subsystem between IP cores in a System-on-a-Chip (SOC). NOCs can span synchronous and asynchronous clock domains or use unclocked asynchronous logic. NOC applies networking theory and methods to on-chip communication and brings notable improvements over interconnections.Here we use bus topology to interconnect the IP’s to minimize the complexity.

VI. Simulation And Implementation

VERILOG is frequently used for two different goals: Simulation of electronic designs and synthesis of such designs. Synthesis is a process where a VERILOG is compiled and mapped into an implementation technology such as an FPGA or an ASIC. Many FPGA vendors have free tools to synthesize VERILOG for use with their chips, where ASIC tools are often very expensive.

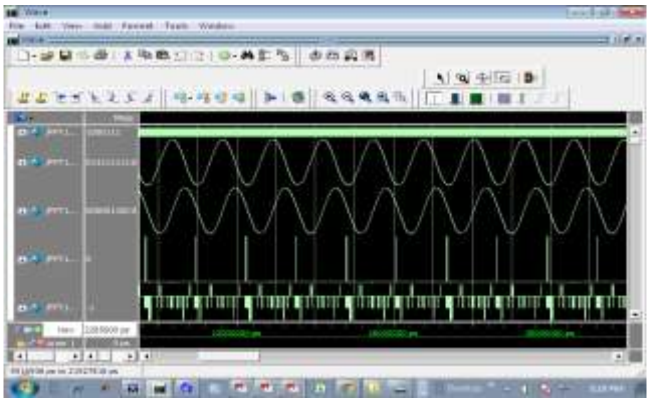

Fig 4.Modelsim Output

VII. Conclusion

We presented a effective implementation of Fast Fourier Transform on FPGA.In this paper we developed both integer and floating point operations. After completing the arithmetic design we carried out a Fast Fourier Transform (FFT) based on radix-8 techniques that perform pipelining, which achieves drastic performance improvement. For randomly generated test examples, we showed that the proposed method compute FFT in a effective way to achieve maximum speed of computation .Finally we carried out NOC with the help of Design partitioner tool to reduce the complexity and give considerable power reduction.

References

[1] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E.M. C. Filho, “Morphosys: An integrated reconfigurable system for dataparalleland computation-intensive applications,” IEEE Trans. Comput.,vol. 49, no. 5, pp. 465–481, May 2000.
[2] PACT XPP Technologies [Online]. Available: http://www.pactxpp.com
[3] B. Mei, S. Vernalde, D. Verkest, H. D. Man, and R. Lauwereins,“ADRES: An architecture with tightly coupled VLIW processor andcoarse-grained reconfigurable matrix,” in Proc. FPLA, 2003, pp. 61–70.
[4] Chameleon Systems, Inc. [Online]. Available:http://www.chameleonsystems.com
[5] T. J. Callahan and J. Wawrzynek, “Instruction-level parallelism forreconfigurable computing,” in Proc. IWFPL, 1998, pp. 248–257.
[6] W. Lee, R. Barua, M. Frank, D. Srikrishna, J. Babb, V. Sarkar, and S.P. Amarasinghe, “Space-time scheduling of instruction level parallelismon a RAW machine,” in Proc. ASPLOSV, 1998, pp. 46–57.
[7] B. Mei, S. Vernalde, D. Verkest, H. D. Man, and R. Lauwereins,“DRESC: A retargetable compiler for coarse-grained reconfigurablearchitectures,” in Proc. ICFPT, Dec. 2002, pp. 166–173.