A New VLSI Algorithm on Fast Fourier Transforms on the ARM Processor

Valiki Vijaya bhasker
Department of ECE,
Siddhartha Institute of Technology and Science,Narapally, Hyderabad, TS, India

ABSTRACT

Now a days in VLSI design the ARM Proessor controllers the design and analysis of the embedded softwares that can the study of the vlsi do the things the With the advent of new FPGA technology, one can embed more complex systems on a single chip, at this point design process become more critical and complex. This problem could be overcome with available latest CAD tools like design, simulation, synthesis (behavioral/RTL/logic) and implementation tools from various vendors. Design engineer can easily translate higherlevel design description (HDL) to lower levels. This way of designing (using CAD tools/HDL) is certainly a revolution in electronic industry. Applications based on Fast Fourier Transforms such as signal and image processing requires high computational power plus the ability to experiment with algorithms. Reconfigurable devices in the form of FPGAs have been proposed as a way of obtaining high performance at an economical price. At present however users must program FPGAs at a very low level and have detailed knowledge of the architecture of the device being used. To try to reconcile the dual requirements of high performance and ease of development, this project reports on the design and realization of a high level framework for the implementation of 1-D FFTs and 2-D FFTs for real time applications

I. Introduction

Applications based on FFT such as signal and image processing requires high computational power plus the ability to experiment with algorithms. Reconfigurable hardware devices in the form of FPGAs have been proposed as a way of obtaining high performance at an economical price. However users must program FPGAs at a very low level and have a detailed knowledge of the architecture of the device being used. They do not therefore facilitate easy development of, or experimentation with signal/image processing algorithms.

A wide range of FFT algorithms including radix-2, radix-4 transforms have been implemented on a common framework in order to enable the system designers to meet different system requirements. Most of the research to date for the implementation and benchmarking of FFT algorithms has been performed using general-purpose processors , Digital signal processors and dedicated FFT processor ICs. However as FPGAs have grown in capacity, improved in performance and decreased in cost, they have become viable solution for performing computationally intensive task (i.e. computation of FFT) with the ability to tackle applications for custom chips and programmable DSP devices. The FFT architectures have been designed using Handel-C language. Although the task could have been accomplished using traditional hardware description languages (HDL) such as VHDL or Verilog, Handel-C has been selected as we aimed to evaluate its rapid design capabilities and suitability for the design of IP cores. The target hardware for the implementation and verification of the architectures is Celoxica RC1000-PP PCI based FPGA development board equipped with a Xilinx XCV2000E Virtex FPGA. Reconfigurable hardware usually in the form of FPGAs has been touted as a new and better means of performing high performance computing. Reconfigurable computing systems are those computing platforms whose architecture can be modified by software to suit the application at hand. Reconfigurable computing involves manipulation of the logic within the FPGA at runtime. In other words the design of hardware may change in response to the demands placed upon the system while it is running.

Typically FPGA structures provide a reconfigurable hardware with flexible interconnections, with field programmable ability, which are widely used for prototyping of DSP and computer systems. Furthermore the recent advances in IC processing technology and innovations in their architectures have made FPGAs highly suitable alternatives to design powerful computing platforms. Direct computation of DFT requires on the order of N2 operations where N is the transform size. The FFT algorithm was first explained by Cooley and Tukey opened a new area in digital signal processing by reducing the order of complexity of DFT from N2 to N log2N. Since the early paper by Cooley and Tukey a large number of FFT algorithms have been developed such as radix-2 m algorithms, Winograd algorithm (WFTA) , prime factor algorithms (FPA) and Fast Hartley algorithm (FHT). Among these radix-2, radix-4 algorithms are the ones that have been most widely used for practical applications due to their simple structure, with constant butterfly geometry and the possibility of performing them ‘in-place’.

FFT Architecture The 1-D FFT architecture consists of a single DIF radix-2, radix-4 butterfly, two (since FHT is a real-valued transformthere is just one dual port) dual port RAMs and an address generator unit. The simplified architectural block diagram of 1-D FFT design is depicted in Fig 1.

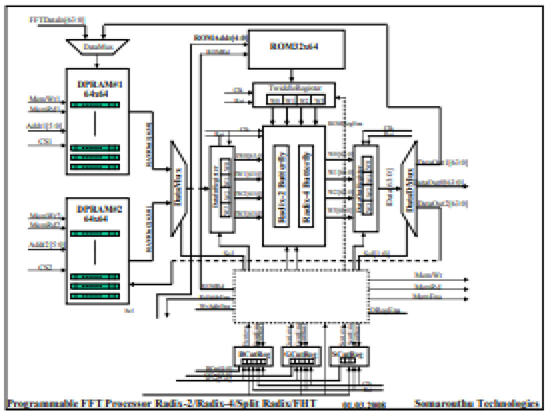


Figure 1: FFT Architecture

A.Input and Output data memory

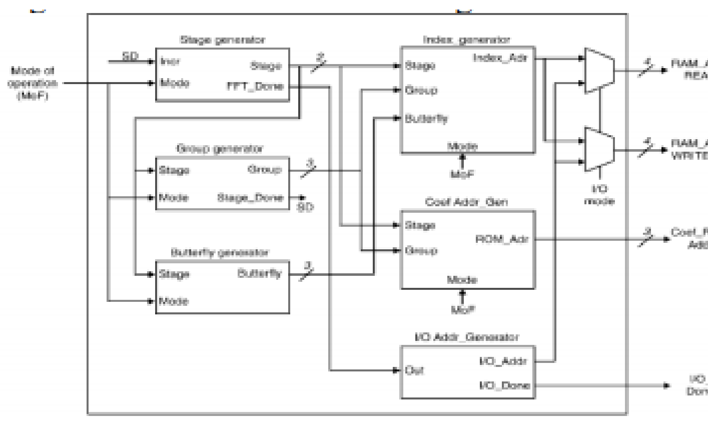
The 1-D FFT architecture supports two different memory modes, internal and external working modes. Internal memory mode: there are two different internal memory configurations- single memory and dual memory configuration which is also termed double-buffering method. In single memory configuration memory A is used as input, working and output memory, input data is loaded into memory A (data load phase) then FFT is started (computation phase) and when FFT is complete the result vector is read out of memory. In this mode memory B is not used. In dual memory configuration input, computation and output operations are overlapped so that FFT processor is never left in an idle state waiting for an I/O operation. During the execution of input data initially in memory A, a new vector of input data is loaded into memory B. When execution of input data loaded in memory A is completed, the FFT processor can start execution of next input vector which is ready in memory B. While FFT processor works with memory B, the result of previous input data is unloaded from memory A and then memory A is loaded with new input data. Data load, computation and output operations are overlapped, and no computation cycles are wasted due to I/O cycles. External memory mode: the real and imaginary parts of input and output data and the twiddle coefficients are stored in external on-board memory banks. The data transfer mechanism to and from external SRAM memory banks has been designed for use in wide variety of applications. The storage of Input/output data in external SRAM memory rather than internal memory allows transformations up to 1M point without the need for additional internal buffering.

B. Butterfly Unit

The butterfly operation is the heart of the FFT algorithm. It takes data words from memory and computes FFT. The results are written back to same memory locations since an in-place algorithm is used. The butterfly element is pipelined in order to compute a result every clock cycle. It consists of parallel L-bit multipliers and L-bit and (L+W) bit adders/subtractors. Internal data path word length of butterfly unit is (L+W+3) bits including three guard bits for input samples. Because radix-n butterfly can add up to three bits from input to output, bit growth must be monitored to avoid data overflow. Since FFT is recursive word length reduction to the output of butterfly takes place between stages by shifting output W bits to right. Also in computation of FFT scaling of intermediate results is necessary in order to prevent overflows. Each stage of FFT is scaled down by a factor of 2 and 4 in radix-2/FHT and Radix-4/split-radix algorithms respectively. In this way, N scales the final FFT output down.

C. Address Generation Unit

The purpose of this is to provide the I/O RAMs and twiddle coefficient look-up tables (LUTs) with correct addresses. Since address generation during data input, output and FFT computation process is different, it keeps track of mode of operation and generates required address. It is also responsible for unscrambling (bit-reversal) output data at the end of each FFT execution. An architectural block diagram of the AGU is shown in fig 2



Architectural block diagram of AGU. For clarity, the 16 point radix-2 FFT case is demonstrated

Figure 2: Address Generation Unit

D. Stage Generator

It keeps track of current stage in the FFT computation. It generates the 'stage' output signal which is a log2 (log 2 N)- bit or log2 (log 4 N)-bit counter, since there are s=log2N or s=log4N stages (indexed by S=0,1,……….s-1) in an N point radix-2 and radix-4 FFT algorithm respectively. The stage signal is fed into the 'group generator' block. When stage counter reaches log2N [or log4N] it generates the 'FFT done' signal in order to start the data output process. For a 16-point radix-2 FFT there are s=log2N=4 stages (indexed by S=0;1;2;3) therefore stage generator implements a basic 2-bit counter

E. Group Generator

It controls which group is being computed in a particular stage. The number of groups is Bs=2 s and Bs=4 s for radix-2 and radix-4 FFT algorithms respectively. Therefore it consists of log2N/2 bit or log4N/4 bit counter for radix-2 and radix-4 algorithms respectively. The 'group' signal is fed into the 'Butterfly generator' and 'Index generator' blocks where it is used for index address generation during FFT process. For 16-point radix-2 FFT , the number of groups is 2s where s is stage number(S=0;1;2;3) where maximum number of groups which is 8 occurs at the last stage (S=3). Therefore 'group' signal represents a 3-bit counter output

F. Butterfly Generator

It keeps track of which butterfly is being performed in a particular group. It is log2(N/2) bit or log4(N/4) bit counter since there are maximum of N/2 and N/4 butterflies per group in radix-2 and radix-4 algorithms respectively. This block generates two signals called 'butterfly' and 'stage done'. Butterfly is log2(N/2) bit [or log2(N/4)] bit counter output which is used along with 'group signal to generate index addresses for the RAMs during FFT process. The 'stage done' signal is generated when all butterfly computations have been performed in the stage. The stage generator uses this signal so that the stage value is incremented by one. For 16-point radix-2 FFT each group performs Bs=2 (s-S) butterflies and the 'butterfly' signal is 3- bit counter output.

G. Index Generator

It is responsible for generating addresses (RAM_Addr_Read and RAM_Addr_Write) for each butterfly operation during FFT computation mode. In order to generate FFT addresses, 'butterfly' and 'group' signals are used according to FFT algorithm selected. Index generator is used to generate index address values to address the I/O RAMs. The 2-bit MOF signal indicates the type of FFT algorithm being used so that correct index address values can be generated.

H. I/O address Generator

It is responsible for generating addresses for I/O RAMs during data input and output processes. Since I/O RAMS consist of N memory locations this block is a log2N bit counter. This block is also responsible for the generation of a 'bit-reversed' I/O RAM address during data output process. Since the bit-reversed address is selected by muxes, no extra hardware is required

I.Twiddle LUTs

The twiddle coefficients (sine and cosine values) used in the FFT computation are computed and stored in thetwiddle LUT ROMs during 'synthesis process of the code'. Since twiddle coefficients are represented using W bits fixed –point format each twiddle LUT consists of two N*W bit or (N/2)*W bit ROMs for radix-2 and radix-4 algorithms respectively. Address generation for twiddle LUTs is performed by 'coefaddr_gen' block. The 'stage' and 'group' signals are fed into this block where they are used for computation of necessary address

II. Fast Fourier Transforms

The DFT of an N-point discrete-time complex sequence x(n) indexed by n=0,1…..N-1; is defined by N-1 kn X (k) =∑ x (n) WN k=0, 1…N-1-------(1) n=0 And WN=e -j2∏/N and is referred to as the twiddle factor. The number of complex multiplies and adds operations required for the computation of an N-point DFT is of order N 2 . In fact, the excessively large amount of computations required to compute the DFT directly when N is large has prompted alternative methods for computing the DFT efficiently. This problem was alleviated with the development of special fast algorithms, collectively known as fast Fourier transforms

Where we have used the fact that WN 2 = WN/2. The computational procedure above can be repeated through decimation of the N/2-point DFTs X (2k) and X (2k+1). The entire process involves v = log2N stages of decimation, where each stage involves N/2 butterflies of the type shown in Figure. Consequently, the computation of the N-point DFT via the decimation-infrequency FFT requires (N/2) log2N complex multiplications and Nlog2N complex additions, just as in the decimation-in-time algorithm. For illustrative purposes, the eight-point decimation-infrequency algorithm is given in Figure

Conclusion

Due to the importance and use of FFT in many signal and image processing applications, an FPGA based library for the implementation of 1-D FFTs based on radix-2, radix-4 algorithms have been developed to enable the system designer to meet different system requirements such as hardware resources (chip area, memory etc) and computation time. Having implemented all the algorithms under a common framework, the performance of different 1-D FFT algorithms have been comprehensively evaluated for FFT lengths from 1K point to 256K point. It is worth mentioning that the radix-2 outperforms other algorithms in terms of area and system frequencyIt has been consistently seen in our evaluation that the radix-4 is by far the most efficient algorithm in terms of computation time. If the choice of algorithm is to be made solely based on memory requirements, the FHT algorithm is the best since the memory requirement of the FHT is exactly half of the other algorithms. It can be clearly seen that the architecture based on a radix-4 algorithm outperforms all implementations of other architectures .The performance of 1-D FFT architecture implementations has been discussed, investigated and compared with existing versions

REFERENCES

[1] Brigham, E.: 'The fast Fourier transform and its applications' (Prentice Hall, New York, 1988)

[2] Burrus, C., and Parks, T.: 'DFT/FFT and convolution algorithms' (Wiley, New York, 1985)

[3] Uzun, I., Amira, A., and Bensaali, F.: 'An FPGA based parametrisable System for parallel 2-d FFT implementation'. Proc. Int. Conf. Computer, Communication and Control Technologies (CCCT 2003), 2003

[4] Wino grad, S: 'On Computing the DFT', Math. Computation.1986, 32, pp. 175–199

[5] Dick, C.: 'Computing multidimensional DFTs using Xilinx FPGAs'. Proc.8th Int. Conf. on Signal Processing Applications and Technologies, 1998