A Discussion of Using a Reconfigurable Processor to Implement the Discrete Fourier Transform

Michael J. White*
NASA/Goddard Space Flight Center
Signal Processing and Microelectronics Branch
Code 564 Greenbelt, MD 20771
Michael.J.White@nasa.gov

Abstract

This paper presents the design and implementation of the Discrete Fourier Transform(DFT) algorithm on a reconfigurable processor system. While highly applicable to many engineering problems, the DFT is an extremely computationally intensive algorithm. Consequently, the eventual goal of this work is to enhance the execution ~~time~~ speed of a floating-point precision DFT algorithm by off loading the algorithm from the computing system. This computing system, within the context of this research, is a typical high performance desktop computer with an array of field programmable gate arrays(FPGAs). FPGAs are hardware devices that are configured by software to execute an algorithm. If it is desired to change the algorithm, the software is changed to reflect the modification, then download to the FPGA, which is then itself modified.

This paper will discuss methodology for developing the DFT algorithm to be implemented on the FPGA. We will discuss the algorithm, the FPGA code effort, and the results to date.

Introduction

The DFT is a useful but computational intensive algorithm for engineering applications. To implement the DFT requires N^2 complex multiplications. For a 1024-point DFT, this represents 1,048,572 complex multiplies.[1] To enhance processing speed, we wish to move the DFT to an FPGA from the microprocessor[2]. That is, we desire to move the application from software to hardware.

In order to do this, the instruction set of the reconfigurable floating-point vector processor that has been developed allows the reuse of particular op-codes for different instructions that are loaded into the FPGA. This combination of microcomputer/ small instruction set provides the performance advantages of a reduced instruction set microprocessors as well as the benefits of a large instruction set offered by a complex

---

* Member NTA

instruction set microprocessors. A standard instruction set architecture is utilized and a methodology for mapping the digital signal processing algorithm onto the reconfigurable processor system is applied.

The instruction set architecture includes a very flexible data path that contains floating-point function cores that can be tailored for each application. A complex function core developed for the DFT application is presented, with the input data being either complex or real. It consists of several simple floating-point function cores including: a floating-point adder, a floating-point multiplier, and a floating point multiply-accumulate core. The reconfigurable processor uses a sine/cosine look-up table for computing the necessary trigonometric functions.

A data array can be a size other than 2^N but it is still desirable to use Fourier analysis, in such a case the DFT is obvious candidate. The desired algorithm should use an array less than 1024, however it is important that the processing be able to manipulate data that is not of size 2^N.

There are examples of Fourier transform being implemented on FPGAs.[3,4] These algorithms suffered from being implemented in a fixed-point format and/or restricted to only real input data format of sizes 2^N. Furthermore, there are examples of comparing a DSP processor to a FPGA.[5] Using the floating-point reconfigurable processor system, the best of both worlds, the DSP processor and the FPGA, is combined and demonstrated using the DFT algorithm.

The Field Programmable Gate Array

Field Programmable Gate Arrays (FPGAs) are logic devices that offer in-circuit hardware reconfigurability. The same integrated circuit can be used for an entirely different function at a later date. With this technology, we envision a single hardware unit that could be used for many common functions. The ability to do extensive image processing on-board a spacecraft is an example of the application of this technology.

Reconfigurable computing (RC) is an emerging technology that utilizes FPGAs to implement computation intensive algorithms at the hardware level. A reconfigurable computer, within the context of our research, is a general-purpose processor with a high-speed connection to one or more FPGAs. Since particular hardware architecture is implemented for each application, typical RC systems can achieve acceleration rates that are several orders of magnitude faster than current desktop computers. Furthermore, research has shown a reduction in computational time using RC technology, but very little has been done on using floating-point digital signal processing algorithm applying RC technology.

While using a reconfigurable computer can be effective in reducing overall application execution time, much of the process of algorithm development is manual and requires skills in both hardware design and software development. Hardware description

languages, i.e. VHDL* and Verilog, are typically used to model the hardware that is developed for each application. This is followed by extensive simulation of these models. Once the model is verified, the models are mapped to an FPGA using commercial tools for FPGA placement and routing. Finally, software is written to download the bitstream produced from placement and routing onto the FPGA as well as to initialize memory and manage overall execution of the application.

Our approach has been to develop a reconfigurable microcomputer instruction set architecture (ISA) that supports a small number of instructions that are tailored for each application. A large portion of the ISA is fixed to simplify compilation of a hardware description language(HDL) model of the system. This microcomputer architecture includes a very flexible data path containing unique function cores that execute floating-point vector instructions. Floating point data is used to facilitate system debugging and functional verification.

A function core is loaded into the ISA prior to program execution defining the instruction used for a particular op-code. Subsequently a different function core can be loaded into the ISA and the same op-code reused for a completely different instruction. Hence, there is a one-to- one mapping of op-codes to assembly language instructions. In this paper we present a function core that was developed for the DFT algorithm.

The next section will discuss the details of the DFT algorithm, a sample application for proving the concepts presented in this paper follows. The paper concludes with the presentation of the DFT results compared to a simulation.

The DFT Algorithm

The Discrete Fourier Transform takes a signal from the time domain to the frequency domain by the relationship given by equation 1 and is defined as for each output sample, X, as:

$$X(k) = \sum_{n=0}^{N-1} c(n) * \exp(-j * 2 * \pi * n * k/N)$$

Where k is the index of the output sample
n is the index of the input sample
c is the input sample
N is the total number of inputs
J= sqrt(-1)

Equation 1: Discrete Fourier Transform

The output of the magnitude of X(k) as a function of k will produce a spectrum between (-F) and +F.

* Very High Speed Integrated Circuits(VHSIC) Hardware Description Language(VHDL)

Using this definition, we wish to implement an RC system that uses floating-point arithmetic that fits on a single FPGA chip while enhancing software performance. Equation 2 is an expansion of Equation 1 to a form better for implementation.

$$X(k) = \sum (Xr+jXi)(\cos \theta_k + j \sin \theta_k) = \sum Xr \cos \theta_k + j\, Xr \sin \theta_k + j\, Xi \cos \theta_k - Xi \sin \theta_k$$
$$= \sum Xr \cos \theta_k - Xi \sin \theta_k + j\,(Xr \sin \theta_k + Xi \cos \theta_k)$$
$$= \sum Yr + \sum Yi = Xrealout + Ximagout$$

Where $\sum$,the sum, is over the range k=0 to N-1 for all cases and $\theta_k = nk/N$

Equation 2: Expansion of DFT Equation

The Processing Element

A simple model of the system is seen in figure 1. The FPGA or processing element(PE) consist of two sections, a control unit and data processing unit which contains the function core. The function core is the application being implemented which in this case is the DFT.

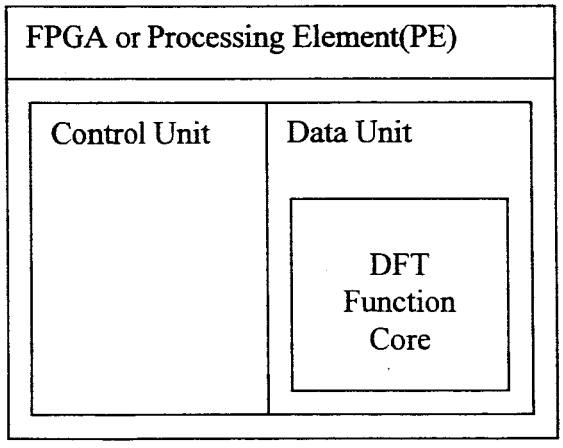

Figure 1: DFT Function Core

The Control unit, which handles processing for all hardware modules/ instructions, manages memory read/write transactions as well. This unit will also supervise instruction fetch, decode, and execution. Lastly, it will determine when instruction processing is completed and turns control back over to the Host/Memory interface.

The data unit contains several memory address registers and counters for indexing. Furthermore, the data unit contains a register file of 8 32-bit registers and counters for determining when vector instructions are completed. The data unit can

contain up to seven function cores. Each function core has one or more 32-bit inputs and simple control functions. The function core can be independent or made-up of other function cores. These function cores make up the floating-point functions.

FPGA Implementation of the DFT

A high level description of the DFT is shown in figure 2. Each 32 bit input is complex where Xrealin represent the real input value and Ximagin represents the imaginary component. K is the output index and is represented by 10 bits. DFT/IDFT tells the PE to execute the normal DFT or its inverse. We use –1 for DFT and 1 for Inverse DFT. To start processing, the Enable flag is set to 1 and processing ends when Empty flag has been set and saying the data buffer is depleted.

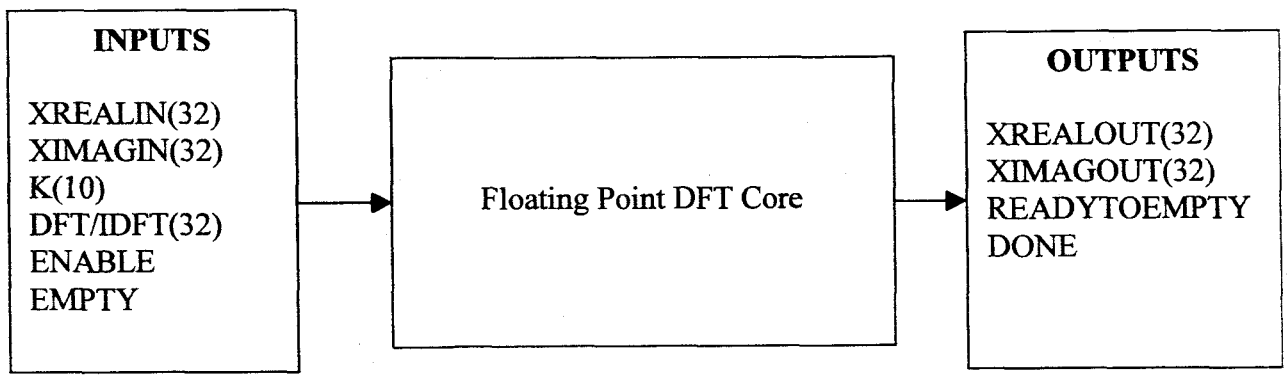

Figure 2: High Level Description of DFT

Similar to the input, the output is complex and denoted by Xrealout and Ximagout for the real and imaginary values of output, respectfully. Readytoempty flag indicates FPGA processing is done. Finally, the Done flag tells when the processing pipeline is completed or flushed.

In figure 3, DFT process is depicted. This algorithm is derived in Equation 2. Using the product of the output index, K and the sample number, n the algorithm generates the table look-up address. This address represents the sin/cosine angle, theta, we are interested in at this point. A maximum of 2^10 or 1024 angles can be generated by this table with each sin/cos value being 32 bits value. This represents the largest DFT we can process. The details of the complex multiply instruction is shown in figure 4.

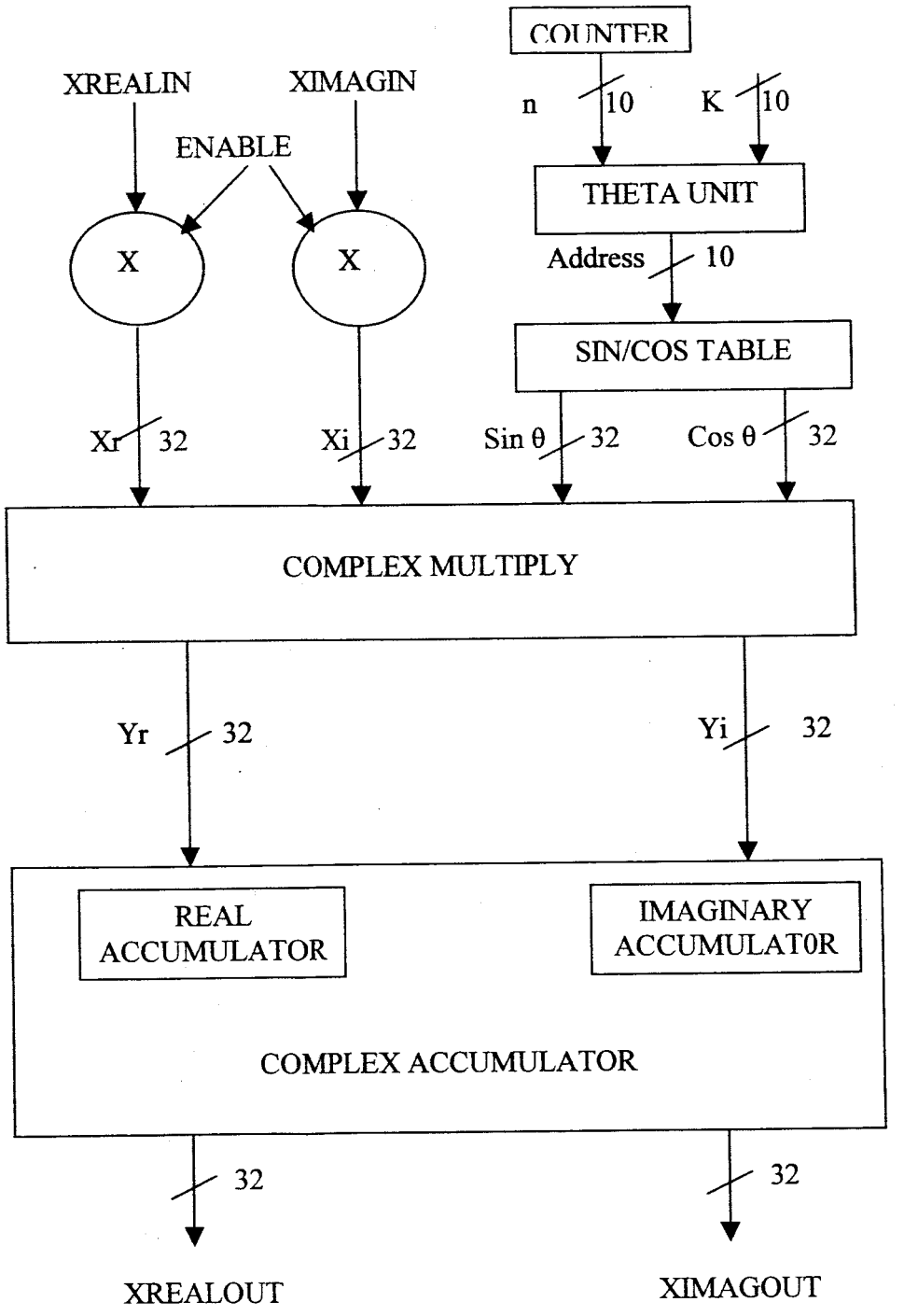

Figure 3: FPGA Processing of DFT

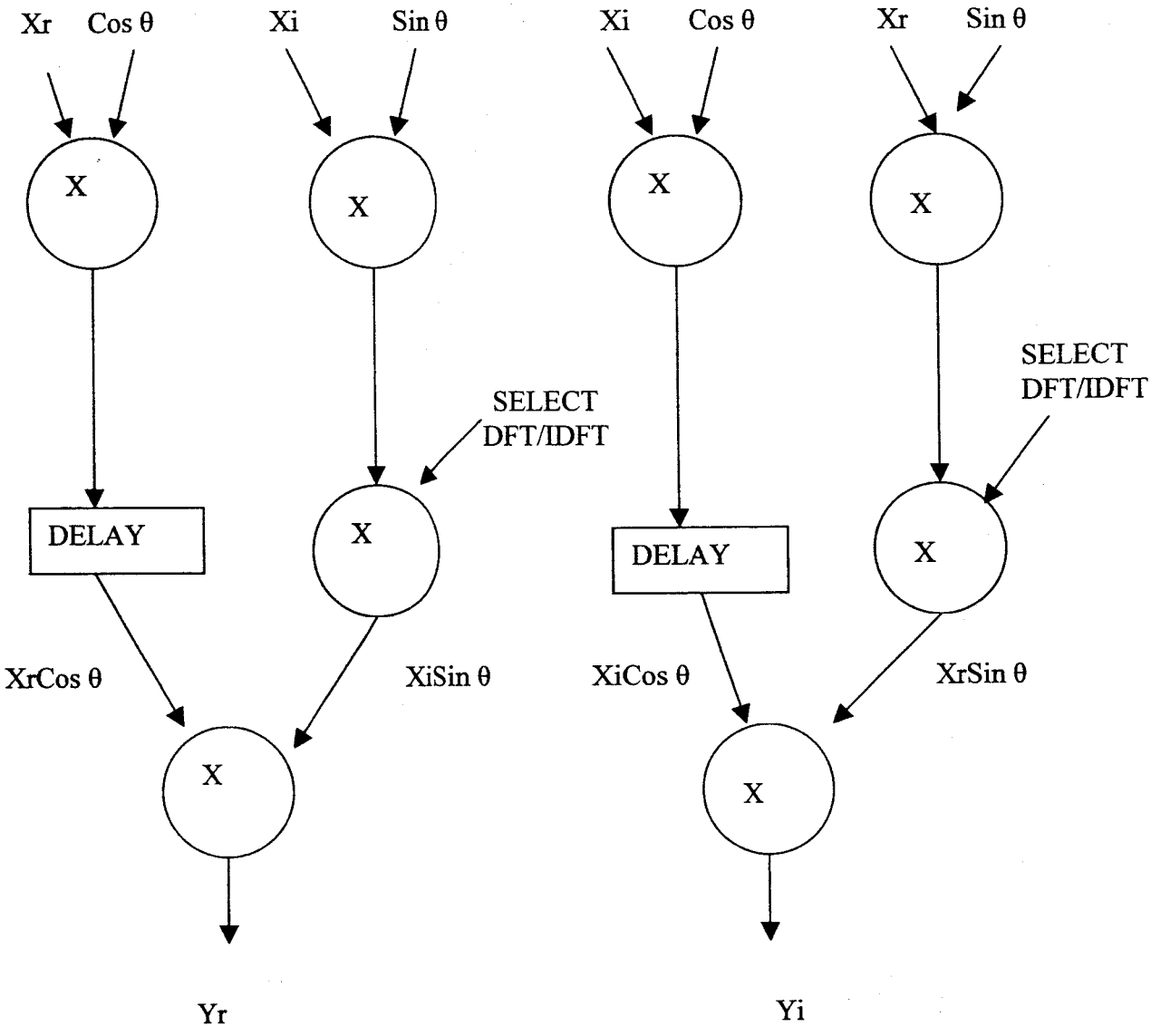

Figure 4: FPGA Complex Multiplication

This VHDL code was compiled and tested. Next, this code is synthesized. This process creates logic gate connections. Finally, the place and route is performed. In this step, the output of the synthesis is mapped physical to the FPGA part. The code can now be executed on the FPGA.

To test the algorithm, an input of a 20 Hz sin wave sampled at 1/128 times a second was generated. This gives a resolution of 12.8 Hz per cell. Depicted in Figure 5 is a comparison of the output of a 10-pt DFT using FPGA and comparing the results with the output of a simulation using the same data. Within reasonable error, the peak is as expected about 2 cells away from the center.

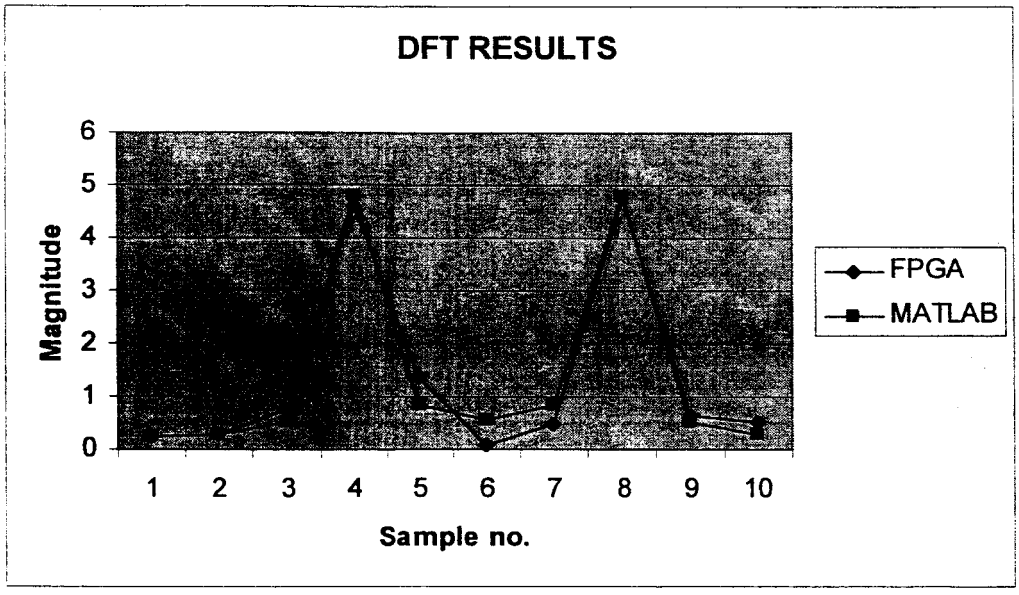

Figure 5: Graph of Comparison output of Simulation vs. FPGA DFT

Conclusion

Figure 5 depicts the output X(k) as a function k. As predicted, it shows a spectrum centered on f=0. This spectrum represents frequencies between +/- 64Hz. As seen in figure 5, there is a high correlation between the output generated by the FPGA and the simulation. The largest error can be observed at the center, where the frequency is equal to 0, at n between 5 and 6.

Future Work

We have successfully implemented the DFT algorithm on the FPGA with an accuracy equal or surpassing a commercially accepted package. We have three objectives for the future at this time. First of all, while there was little error between the FPGA calculation and the simulation, there is enough to generation interest as what is causing this discrepancy, particular at zero. The next step will be to expand to larger examples to determine the functionality of the FPGA under more computational stressful conditions. Finally, we will implement a floating-point FFT on the FPGA and determine its performance.

References:

[1] Proakis, John G. and Dimitris G. Manolakis, "*Digital Signal Processing: Principles, Algorithms, and Application, 3rd Edition.*", 1996, Prentice-Hall, Upper Saddle River, NJ, pg 459.

[2] Kizhner,Semion, "Goddard DFT Engine Specification", Internal Report, 2002.

[3] Botros, N and W. Zakhem, "FFT Processor Using Field Programmable Gate Arrays", Proceedings of the Fifth Annual IEEE International ASIC Conference and Exhibit, 1992, Sept 21-25, 1992, Rochester, NY, pages 115-118

[4] Szedo, G, V. Yang, and C. Dick, "High Performance FFT Processing using Reconfigurable Logic", Conference Record of the Thirty-Fifth Asilomar Conference on Signals, Systems, and Computers, 2001, Pacific Grove, CA, pages 1353-1356 vol.2.

[5] Bilsby, D.C.M, R.L.Walke, and R.W.M. Smith, "Comparison of a Programmable DSP and a FPGA to Real-Time Multiscale Convolution", IEE Colloquium on High Performance Architecture for Real-Time Processing, Dec. 12, 1998. London, UK, pages 4/1-4/6.