A Parallel FFT Architecture for FPGAs

Joseph Palmer and Brent Nelson

Department of Electrical and Computer Engineering,
Brigham Young University
Provo, UT 84602, USA
{palmer,nelson}@ee.byu.edu

1 Introduction

The inclusion of block RAMs and block multipliers in FPGA fabrics has made them more ammenable for implementing the FFT. This paper describes a parallel FFT design suitable for such FPGA implementations. It consists of multiple, parallel pipelines with a front end butterfly-like circuit to preprocess the incoming data and distribute it to the parallel pipelines. Implementation of the parallel FFT on Virtex II shows superlinear speedups for a range of FFT sizes.

1.1 The Parallel FFT

Figure 1 shows the standard data-flow graph for a 16-point, decimation in frequency FFT. The first stage is characterized by data communications between distant rows but at each new stage the dependencies move closer, and branches of independent data flow appear. After two stages the network has been divided into four independent partitions as denoted by the horizontal lines. This suggests the use of four independent processing elements to compute those partitions in parallel. This problem, known as the *parallel*

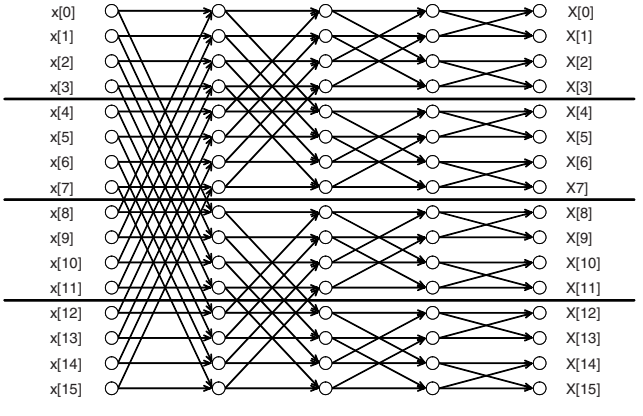

Fig. 1. 16-point FFT data-flow-graph

J. Becker, M. Platzner, S. Vernalde (Eds.): FPL 2004, LNCS 3203, pp. 948–953, 2004.

FFT, has been extensively studied in the parallel computing community, and a number of algorithms for it have been proposed [1]. One method is to map groups of the rows of the data-flow-graph (Figure 1) onto multiple processors (four processors in the case of Figure 2. However, because of the interprocessor data dependencies found in the first two stages of the computation, special mechanisms must be employed to handle this. In the Binary-Exchange Algorithm[1], data must be exchanged between the processors during these first stages. An alternative algorithm is called the Transpose Algorithm[1] where the processors cooperate to compute the first two stages *without interprocessor communication* but a transposition of the data in memory is required after the second stage.

1.2 Mapping the Parallel FFT to FPGAs

We are generating a custom hardware implementation and are thus not limited to using n identical processing elements and standard interprocessor communication. Our approach is to create a custom front-end circuit to accomplish the first two stages of the computation and then feed those results to the parallel pipelines which do the partitioned calculations.

The Discrete Fourier Transform (DFT), for sample frame of size N, is defined as

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

where $W_N = e^{-j2\pi/N}$, known as the *twiddle-factor*.

We want to split our output, $X(\omega)$, into four blocks, and the DFT for size N can be decomposed into

$$X(4\omega) = \sum_{n=0}^{N-1} x[n] W_N^{(4\omega)n} \tag{2}$$

$$X(4\omega+1) = \sum_{n=0}^{N-1} x[n] W_N^{(4\omega+1)n} \tag{3}$$

$$X(4\omega+2) = \sum_{n=0}^{N-1} x[n] W_N^{(4\omega+2)n} \tag{4}$$

$$X(4\omega+3) = \sum_{n=0}^{N-1} x[n] W_N^{(4\omega+3)n}. \tag{5}$$

Since the FFT will need to input four samples for every four outputs, (2)-(5) need to be in terms of four separate input blocks. Beginning with (2),

$$\begin{aligned} X(4\omega) &= \sum_{n=0}^{N-1} x[n] W_N^{(4\omega)n} \\ &= \sum_{n=0}^{N/4-1} x[n] W_N^{4\omega} + \sum_{n=N/4}^{N/2-1} x[n] W_N^{4\omega} + \end{aligned}$$

$$\sum_{n=N/2}^{3N/4-1} x[n]W_N^{4\omega} + \sum_{n=3N/4}^{N-1} x[n]W_N^{4\omega}.$$

Now, using variable substitution in the summations, it follows that

$$\begin{aligned} X(4\omega) = & \sum_{n=0}^{N/4-1} x[n]W_N^{4\omega} \\ & + \sum_{n=0}^{N/4-1} x[n+N/4]W_N^{4\omega n}W_N^{\omega N} \\ & + \sum_{n=0}^{N/4-1} x[n+N/2]W_N^{4\omega n}W_N^{2\omega N} \\ & + \sum_{n=0}^{N/4-1} x[n+3N/4]W_N^{4\omega n}W_N^{3\omega N}, \end{aligned}$$

and because $W_N^{Z\omega N} = 1$ and $W_N^{Z\omega n} = W_{N/Z}^{\omega n}$, where Z is some integer, the final solution becomes

$$\begin{aligned} X(4\omega) = & \sum_{n=0}^{N/4-1} (x[n] + x[n+N/4] + \\ & x[n+N/2] + x[n+3N/4])W_{N/4}^{\omega n}. \end{aligned}$$

The derivations for the other output blocks can be obtained in a similar fashion, resulting in

$$\begin{aligned} X(4\omega+1) = & \sum_{n=0}^{N/4-1} (x[n] - jx[n+N/4] + \\ & x[n+N/2] + jx[n+3N/4])W_{N/4}^{\omega n}W_N^n, \\ X(4\omega+2) = & \sum_{n=0}^{N/4-1} (x[n] - x[n+N/4] + \\ & x[n+N/2] - x[n+3N/4])W_{N/4}^{\omega n}W_N^{2n}, \\ X(4\omega+3) = & \sum_{n=0}^{N/4-1} (x[n] + jx[n+N/4] - \\ & x[n+N/2] - jx[n+3N/4])W_{N/4}^{\omega n}W_N^{3n}. \end{aligned}$$

Next, the following variables will be created,

$$\begin{aligned} a_0[n] = & (x[n] + x[n+N/4] + x[n+N/2] + \\ & x[n+3N/4]), \end{aligned} \tag{6}$$

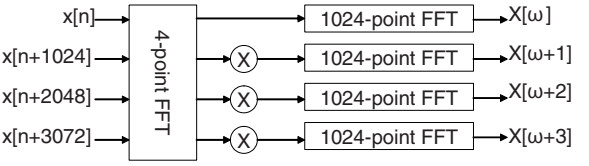

Fig. 2. 4096-point Quad-pipeline Radix-2^2 FFT

$$a_1[n] = (x[n] - jx[n + N/4] + x[n + N/2] + jx[n + 3n/4])W_N^n, \quad (7)$$

$$a_2[n] = (x[n] - x[n + N/4] + x[n + N/2] - x[n + 3n/4])W_N^{2n}, \quad (8)$$

$$a_3[n] = (x[n] + jx[n + N/4] - x[n + N/2] - jx[n + 3n/4])W_N^{3n}, \quad (9)$$

and substituting these into the derived block DFT equations produces

$$X(4\omega) = \sum_{n=0}^{N/4-1} a_0[n]W_{N/4}^{\omega n}$$

$$X(4\omega + 1) = \sum_{n=0}^{N/4-1} a_1[n]W_{N/4}^{\omega n}$$

$$X(4\omega + 2) = \sum_{n=0}^{N/4-1} a_2[n]W_{N/4}^{\omega n}$$

$$X(4\omega + 3) = \sum_{n=0}^{N/4-1} a_3[n]W_{N/4}^{\omega n}.$$

Note that these four results are each DFTs of length N/4, and share the same twiddle-factors of $W_{N/4}^{\omega n}$. Each can be computed by a separate FFT pipeline. The only additional hardware needed is to compute the set $\{a_1[n], a_2[n], a_3[n], a_4[n]\}$. The key observation is that this set of values can be computed using a conventional 4-point butterfly followed by modified twiddle factor mutliplications as shown in Figure 2 where the multiplication factors are the W's from equations (6) through (9).

This result is equivalent to that used for computing mixed radix FFT's [2]. The major contribution of this paper is to recognize that the method can be used to factor an FFT for parallel computation on an FPGA to achieve throughputs greater than 1. In addition, as will be show in the results below, the resulting parallel pipelines share significant logic and memory and achieve superlinear speedups (they use less than k times the hardware to achieve k-fold speedups). It should be obvious from the above discussion that breaking the FFT into four parallel pipelines is not the only choice — any k-way factorization (including non-power-of-two values) can be used followed by k parallel

pipelines. Obviously such an approach can result in extremely high throughputs, limited only by the chip resources available. It also provides flexibility to trade off chip area for throughput. Both fixed point and block floating point versions of the FFT shown in Figure 2 have been implemented using a module generator written in JHDL, simulated, and tested on a Virtex II 6000-4. The module generator uses the Radix-2^2 algorithm [3] for the parallel FFT pipelines but any constant I/O FFT pipeline algorithm could be used.

Implementation results are shown in Table 1. All FFT's in the table use fixed-point computations and an 18-bit word size. They are constant-I/O, meaning that the samples to be processed can be input continuously without any breaks between frames. Throughput, in samples per second, can be computed for these FFT's by multiplying the clock rate by the number of pipelines. The table clearly shows that the quad-pipeline FFT is only 2-3 times as large as a single-pipeline FFT but has a 1/4th the transform time at essentially the same clock rate.

Table 1. Fixed-point FFT's on the XC2V6000-4

Frame Size	Pipeline Style	Slices	Block RAMs	Block MULTs	Speed (MHz)	Latency (cycles)	Throughput Msps	Transform Time (μs)	Area $\times$ Transform Time Product
256	Single	2,233	6	9	163	547	163	1.57	3,506
	Quad	5,228	11	33	151	161	604	0.42	2,196
1K	Single	2,870	15	12	164	2,092	164	6.24	17,909
	Quad	7,656	27	45	151	554	604	1.70	13,015
4K	Single	3,838	33	15	155	8,245	155	26.4	101,323
	Quad	9,846	63	57	150	2,099	600	6.83	67,248

2 Comparison to Related Work and Conclusions

Multi-chip parallel FFT designs have been published [4] [5] but these focus on VLSI. At the time of the submission of this paper, two companies have released single-FPGA parallel FFT's — SiWorks and Pentek both advertise fixed-point FFT's with throughputs in the range of 400-500 Msps but provide few details on their internal architectures..

In contrast, this paper has proposed a Parallel FFT formulation, useful for creating parallel FFT's of essentially any size and throughput and suitable for custom hardware implementations using FPGA's. It exploits the characteristics of FPGA's (and custom hardware in general) as well as the structure of the FFT computation to significantly increase throughput.

Due to space constraints this paper has focused only on the parallel formulation of the FFT computation as completed in this work. This work also has included the development of block floating point FFT modules based on *convergent block floating point* [6] and provided an analysis of the cost and benefits of using block floating point. A parameterized module generator, written in JHDL, has been written which produces both fixed-point and block-floating-point FFT's of any size and parallelism desired (limited only by FPGA resources).

References

1. Ananth Gramam, Anshul Gupta, George Karypis, and Vipin Kumar, *Introduction to Parallel Computing, Second Edition*, Pearson Education, Harlow, England, 2003.
2. Winthrop W. Smith and Joanne M. Smith, *Handbook of Real-Time Fast Fourier Transforms*, IEEE, New York, 1995.
3. S. He and M. Torkelson, "A New Approach to Pipeline FFT Processor," *The 10th International Parallel Processing Symposium(IPPS)*, pp. 766-770,1996.
4. Tom Chen, Glen Sunada, and Jian Jin. COBRA: A 100-MOPS single-chip programmable and expandable FFT. *IEEE Transactions on VLSI Systems*, 7(2), Jun 1999.
5. Yu Tai Ma. A VLSI-Oriented Parallel FFT Algorithm. *IEEE Transactions on Signal Processing*, 44(2), Feb 1996.
6. E. Bidet, D. Castelain, C. Joanblanq and P. Senn, "A Fast Single-Chip Implementation of 8192 Complex Point FFT", *IEEE Journal of Solid-State Circuits*, vol. 30, March 1995.