A High-level Implementation of a High Performance Pipeline FFT on Virtex-E FPGAs

S Sukhsawas, K Benkrid
School of Computer Science, Queen's University Belfast, United Kingdom
{s.sukhsawas, k.benkrid}@qub.ac.uk

Abstract

This paper presents a high-level implementation of pipeline FFT. The design has been coded in Handel-C language and targeted Xilinx Virtex-E FPGA series. It is fully implemented and tested on real hardware using Celoxica RC1000-PP prototyping board. The implementation results show that our implementation outperforms other implementations of FFT on the same series of FPGA. An implementation of 1024-point FFT on Virtex-E can run at a maximum clock frequency of 82 MHz leading to a 82 MS/s throughput compared with the Xilinx core of the same size that can run at a maximum clock frequency of 83 MHz but with only 21 MS/s throughput. The design is parameterizable in terms of input wordlength, output wordlength, Twiddle factors wordlength and processing wordlength. It is scalable in terms of number of stages which means that a 16-point, 64-point, 256-point, 1024-point or higher power-of-4 complex-point FFT can be synthesized from the same code. The paper reports the fastest 1024-point FFT implementation on Virtex-E FPGA platform.

1 Introduction

Fast Fourier Transform (FFT) is one of the most utilized operations in Digital Signal Processing and Communications. The FFT and its inverse transform-IFFT are a key component in modern communication systems. Application Specific Integrated Circuit (ASIC) approaches have been used to achieve the high performance demands which software or general purpose DSP implementations fail to deliver. Recently FPGAs has become a valid alternative as the technology has matured greatly. Nowadays FPGAs play an important role in many areas due to their direct hardware solution performance as well as their inherent reprogrammability feature. Using FPGAs for FFT processing has now become feasible in real-time applications.

Development of the FFT in hardware is usually categorized into that of high throughput and that of low power. In the search for high performance FFT, this paper presents a pipelined architecture called Radix-2^2 Single Path Delay Feedback (R2^2SDF) [5] and its implementation on FPGAs using a high-level C-to-hardware programming language for rapid hardware prototyping called Handel-C [3]. The design is parameterizable and scalable and has been implemented and tested on real hardware using Celoxica RC1000-PP prototyping FPGA board [3] which is equipped with a Xilinx Virtex-E XV2000eBG560-6 FPGA [9].

In the following sections, detailed algorithm and architecture of the chosen R2^2SDF will be reviewed. The implementation method will then be described. After that, implementation results will be given and compared with other implementations both academic and commercial. Finally conclusion will be drawn.

2 Architectures

2.1 Radix-2^2 DIF FFT Algorithm

A useful state-of-the-art review of hardware architectures for FFTs was given by He et al. [5]. In [5], different approaches were put into functional blocks with unified terminology.

From the definition of DFT of size N:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \qquad 0 \le k < N \tag{1}$$

where W_N denotes the primitive N^{th} root of unity, with its exponent evaluated modulo N, He [5] applied a 3-dimensional linear index map,

$$\begin{aligned} n &= \ < \tfrac{N}{2} n_1 + \tfrac{N}{4} n_2 + n_3 >_N \\ k &= \ < k_1 + 2k_2 + 4k_3 >_N \end{aligned} \tag{2}$$

and Common Factor Algorithm (CFA) to derive a set of 4 DFTs of length $N/4$ as,

$$\begin{aligned} &X(k_1 + 2k_2 + 4k_3) \\ &= \sum_{n_3=0}^{\frac{N}{4}-1} \left[H(k_1, k_2, n_3) W_N^{n_3(k_1+2k_2)} \right] W_{\frac{N}{4}}^{n_3 k_3} \end{aligned} \tag{3}$$

Proceedings of the IEEE Computer Society Annual Symposium on VLSI Emerging Trends in VLSI Systems Design (ISVLSI'04)
0-7695-2097-9/04 $20.00 © 2004 IEEE

$$H(k_1,k_2,n_3)=\overbrace{\underbrace{\left[x(n_3)+(-1)^{k_1}x(n_3+\frac{N}{2})\right]}_{\text{BF I}}+(-j)^{(k_1+2k_2)}\underbrace{\left[x(n_3+\frac{N}{4})+(-1)^{k_1}x(n_3+\frac{3}{4}N)\right]}_{\text{BF I}}}^{\text{BF II}} \quad (4)$$

where $H(k_1,k_2,n_3)$ is expressed in Eqn. (4).

Eqn. (4) represents the first two stages of butterflies with only trivial multiplications in the SFG, as BF **I** and BF **II**. Full multipliers are required after the two butterflies in order to compute the product of the decomposed twiddle factor $W_N^{n_3(k_1+2k_2)}$ in Eqn. (3). Note the order of the twiddle factors is different from that of radix-4 algorithm.

Applying this CFA procedure recursively to the remaining DFTs of length $N/4$ in Eqn. (3), the complete radix-2^2 DIF FFT algorithm is obtained. The corresponding FFT flowgraph for $N = 16$ is shown in Fig. 1 where small diamonds represent trivial multiplication by $W_N^{N/4} = -j$, which involves only real-imaginary swapping and sign inversion.

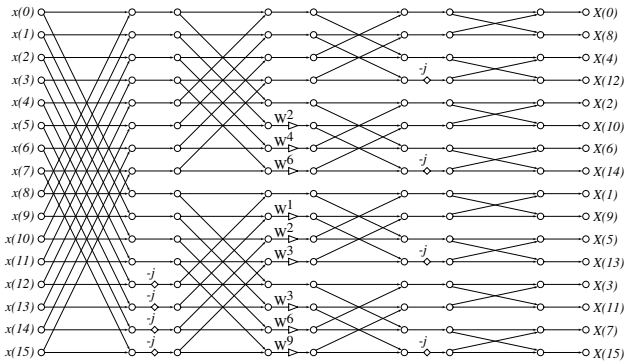

Figure 1. Radix-2^2 DIF FFT flow graph for $N = 16$

Radix-2^2 algorithm has the feature that it has the same multiplicative complexity as radix-4 algorithms, but still retains the radix-2 butterfly structures. The multiplicative operations are in a such an arrangement that only every other stage has non-trivial multiplications. This is a great structural advantage over other algorithms when pipeline/cascade FFT architecture is under consideration [5].

2.2 R2^{2}SDF Architecture

Mapping radix-2^2 DIF FFT algorithm to the R2SDF architecture [5] , an architecture of Radix-2^2 Single-path Delay Feedback (R2^{2}SDF) approach is obtained.

Fig. 3 outlines an implementation of the R2^{2}SDF architecture for $N = 1024$. The implementation uses two types of butterflies, as shown in Fig. 2(i)-2(ii) respectively. Due to the spatial regularity of the radix-2^2 algorithm, the synchronization control of the processor is very simple. A simple $(log_2 N)$-bit binary counter serves two purposes: synchronization controller and address counter for twiddle factor reading in each stages.

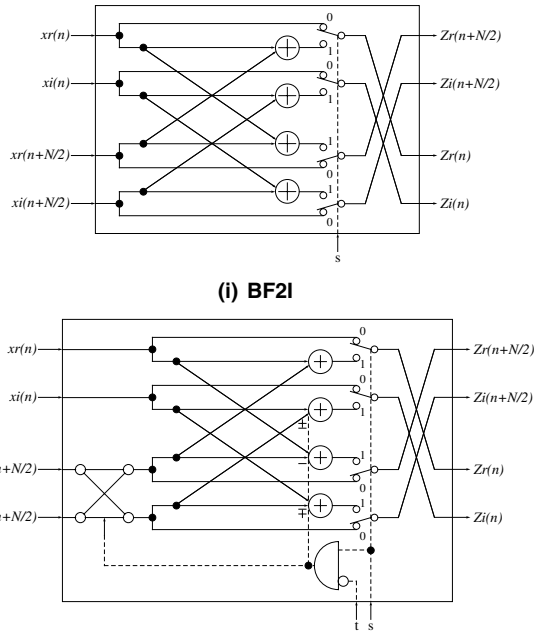

Figure 2. Butterfly structure for R2^{2}SDF FFT

In order to speed up the implementation, pipeline registers can be inserted between each multiplier and butterfly stage. This would be done at the expense of extra $3(log_4 N - 1)$ cycles in latency.

3 Implementation in Handel-C

The R2^{2}SDF presented above has been fully coded in a direct-C-to-hardware language called Handel-C [3]. This ANSI C-like language is a subset of C, extended with CSP parallelism and communication primitives. Once the design is coded in Handel-C, the Handel-C compiler generates an FPGA configuration in EDIF netlist format. The latter then

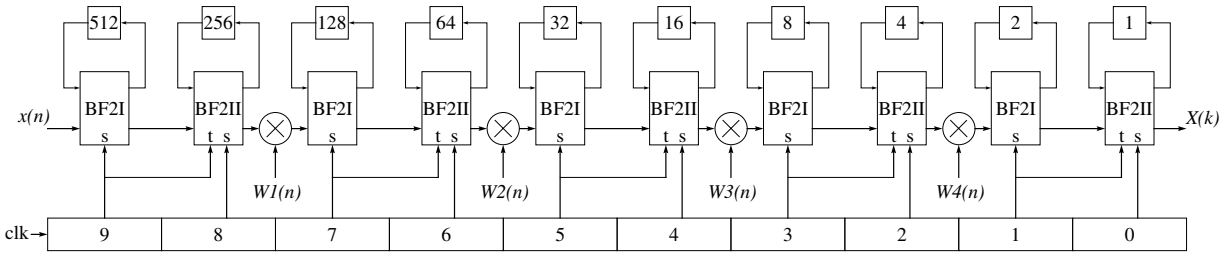

Figure 3. R2^{2}SDF pipeline FFT architecture for $N = 1024$

goes through the FPGA vendors specific Place And Route (PAR) tools to generate bitstream ready to be downloaded to the FPGA on RC1000-PP prototyping board.

From the architecture of R2^{2}SDF in Fig. 3, the butterfly blocks BF2I and BF2II are described as building blocks in the Handel-C code. Booth multiplication algorithm for signed binary numbers is used for complex multipliers. Thus, the overall latency of the real implementation varies as the processing wordlength changes. LUT-based RAMs and Flip-Flops are used to implement feedback memory of the very last stages whereas Virtex-E Block RAMs are used for the rest of stages. Similarly, LUT-based ROMs are used to implement Twiddle ROMs of the very last stages whereas Block RAMs are used for the rest of stages. The FFT is heavily pipelined to achieve as highest clock frequency as possible. Twiddle factors are generated by an external program and embedded to the Handel-C code.

The circuit description is scalable and parameterizable in terms of input wordlength, output wordlength, Twiddle factors wordlength, processing wordlength of each butterfly stage, scaling factor after each multiplier stage, and number of stages.

The design was simulated conveniently at the Handel-C source level. Hardware is generated automatically from Handel-C code by Celoxica DK software suite [3] together with Xilinx tools. Xilinx Foundation 5.1i was used for place and route. The maximum clock frequencies given were that reported by the P&R tool and without any manual floorplanning.

4 Implementation Result

Table 1 gives sample performance figures of the FFT with various complex sizes.

Only a few academic FFT implementations on FPGAs can be found from the open literature. Most of them were either implemented on old series of FPGAs or without complete speed/throughput reported. Pérez-Pascual [6] has implemented butterflies for HIPERLAN/2 on Virtex-E with a throughput of 15.6 MHz for radix-2 and 18.7 MHz for radix-4. Sansaloni [8] has implemented DA-based radix-2 DIF butterfly for large N on XC4085xl-1 with a throughput of 25.69 MHz. Shaditalab has implemented on two XC4013E-3 FPGAs a single radix-2 butterfly structure by using self-sorting algorithm. 5120 clock cycles are needed to compute 1024-point input data in 256.450μs at a clock frequency of 20 MHz. Dick [4] has implemented an FFT based on Systolic array architectures by mapping 1D input data set into a 2D array and use a pseudo 2D transform to compute the 1D DFT. Operated at 15.3 MHz on XC4010PG191-4 FPGA, the FFT computed 1000-point input data in 51.45ms on one FPGA, 5.15 ms using 10 FPGAs. The PEs were implemented using CORDIC arithmetic. Bahl [2] has implemented a pipeline FFT with an SDF architecture similar to ours but using radix-2^3 algorithm on Virtex-E FPGAs with the performance of 32 MHz.

As for commercial FFTs, most of them are implemented on Xilinx Virtex-II or Virtex-II Pro which provide Dedicated High Speed Multiplier support thus giving a much faster implementation speed (compared to Virtex-E implementations).

Table 2 gives a summary of performance comparison with selected implementations of FFT. Figures of Amphion core CS2411XV [1], Sacet SCT2201 [7], Xilinx core version 2 for Virtex-E [10] and Bahl's [2] are given together with our FFT. Sanet and Bahl's FFTs are not 1024-point so we predicted the figures only for comparison purpose.

The resulting figures show that our implementation outperforms the others. Its speed nearly matches that of the Xilinx core but its throughput is more than 3 times higher due to its pipeline nature. Although our implementation consumes more hardware, it actually achieves the highest throughput thanks to its pipelined architecture. Handel-C has proved to be a very useful language in rapid prototyping. It accelerates the design cycle time drastically. Design verification is easier and quicker with high-level C-like debugging mode.

5 Conclusion

This paper presented a high-level implementation of an efficient pipeline FFT algorithm R2^{2}SDF on Virtex-E

Table 1. Implementation Result

Point Size	Input Data Width	Phase Factor Width	Slices	Block RAMs	Maximum Clock Frequency (MHz)	Latency (cycles)	Transform Time		Throughput (MS/s)
							Clock (cycles)	Time (μs)	
16	8	12	1466	4	90	37	16	0.18	90
64	8	12	3017	16	88	103	64	0.73	88
256	8	12	5299	24	84	313	256	3.05	84
1024	8	12	9209	42	82	1099	1024	12.49	82

Table 2. Performance Comparison

FFT Design	Point Size	Input Data Width	Phase Factor Width	Slices	Block RAMs	Maximum Clock Frequency (MHz)	Latency (cycles)	Transform Time		Throughput (MS/s)
								Clock (cycles)	Time (μs)	
Amphion	1024	13	13	1639	9	57	5097	4096	71.86	14.25
Sacet	256	16	16	1318	8	100	1024	1024	10.24	25.00
Sacet*	1024	16	16	????	??	100	4096	4096	40.96	25.00
Xilinx	1024	16	16	1968	24	83	4096	4096	49.35	20.75
Bahl	512	16	16	7225	??	32	????	512	16.00	32.00
Bahl*	1024	16	16	????	??	32	????	1024	32.00	32.00
Ours	1024	16	16	7365	28	82	1099	1024	12.49	82.00

* Predicted figures

FPGAs. The architecture used consumes the minimum required amount of multipliers and storage. The implementation has been realized using a C-to-hardware language called Handel-C and was fully tested on Celoxica's RC1000-PP prototyping FPGA board. A maximum frequency of 82 MHz has been achieved for an implementation of 1024-point FFT with 16-bit input and Twiddle factors wordlength. The design is parameterizable in terms of input wordlength, output wordlength, Twiddle factors wordlength and processing wordlength. It is scalable in terms of number of stages which means 16-point, 64-point, 256-point, 1024-point or higher power-of-4 complex-point FFT can be synthesized from the same code. The paper reported the fastest 1024-point FFT implementation on Virtex-E FPGA platform.

Handel-C has proved to be a very useful language in rapid prototyping. It accelerates the design cycle time drastically. Design verification is easier and quicker with high-level C-like debugging mode. However this is not to be taken as a complete alternative to the well-estabished HDLs as these offer more control over the underlying hardware. In our opinion, a combination of the two might be the most fruitful solution.

Future work includes improving the rounding method, SNQR analysis and a study of employing CORDIC arithmetic. The FFT will also be expanded and used in 2D FFT for image processing operations. We will also target Virtex-II FPGA platforms which feature, among other things, high performance embedded multipliers. This should result in even higher performance.

References

[1] Amphion Semiconductor Ltd. *1024 Point Block Based FFT/IFFT*. Apr. 2002. http://www.amphion.com/signal.html.

[2] S. K. Bahl. A hardware efficient architecture for fast Fourier transform. In *Proc. GSPx and International Signal Processing Conference (ISPC'03)*, Dallas, Texas, Apr. 2003.

[3] Celoxica Ltd. http://www.celoxica.com/.

[4] C. H. Dick. FPGA based systolic array architectures for computing the discrete Fourier transform. In *IEEE International Symposium on Circuits and Systems (ISCAS'96)*, volume 2, pages 465–468, May 1996.

[5] S. He and M. Torkelson. A new approach to pipeline FFT processor. In *10th International Parallel Processing Symposium (IPPS'96)*, pages 766–770, 1996.

[6] A. Pérez-Pascual, T. Sansaloni, and J. Valls. FPGA-based radix-4 butterflies for HIPERLAN/2. In *IEEE International Symposium on Circuits and Systems (ISCAS'02)*, volume 3, pages 277–280, May 2002.

[7] Sacet. *64/256-Point Complex FFT/IFFT*. Jan. 2002. http://www.sacet.com/.

[8] T. Sansaloni, A. Pérez-Pascual, and J. Valls. Distributed arithmetic radix-2 butterflies for FPGA. In *The 8th IEEE International Conference on Electronics, Circuits and Systems (ICECS'01)*, volume 1, pages 521–524, Sept. 2001.

[9] Xilinx, Inc. http://www.xilinx.com/.

[10] Xilinx, Inc. *High-Performance 1024-Point Complex FFT/IFFT V2.0*. San Jose, CA, July 2000. http://www.xilinx.com/ipcenter/.