Challenging the Limits of FFT Performance on FPGAs

(Invited Paper)

Mario Garrido, *Member, IEEE*, Miguel Acevedo, Andreas Ehliar and Oscar Gustafsson, *Senior Member, IEEE*

Department of Electrical Engineering

Linköping University, SE-581 83 Linköping, Sweden

E-mails: mario.g@isy.liu.se, migac630@student.liu.se, ehliar@isy.liu.se, oscarg@isy.liu.se

Abstract—This paper analyzes the limits of FFT performance on FPGAs. For this purpose, a FFT generation tool has been developed. This tool is highly parameterizable and allows for generating FFTs with different FFT sizes and amount of parallelization. Experimental results for FFT sizes from 16 to 65536, and 4 to 64 parallel samples have been obtained. They show that even the largest FFT architectures fit well in today's FPGAs, achieving throughput rates from several GSamples/s to tens of GSamples/s.

I. INTRODUCTION

In order to meet the high performance demands of current signal processing applications, multi-core systems [1], [2] have become very popular in the last years. They can carry out very fast computations of large amounts of data. This opens new possibilities for advanced algorithms in many fields. However, the current trend of increasing the number of cores also has drawbacks: A large number of processing units leads to higher power consumption and to a higher cost of the system.

In the current era, energy efficiency and sustainable development are critical variables to take into account. Thus, it must be studied when multi-core systems are actually needed and when we can indeed do the calculations in a single device, leading to savings in energy, money and natural resources.

With this perspective in mind, this paper analyzes which is the computational power that can be expected nowadays on a single field programmable gate array (FPGA). For this purpose, the paper studies the case of the fast Fourier transform (FFT), which is one of the most relevant algorithms for signal processing. It is used in countless applications in a wide range of fields, from communication systems, to medical applications and image processing.

In order to do the analysis, we have developed a tool that generates high-throughput FFT implementations on FPGAs. The tool allows for configuring multiple parameters such as FFT size, amount of parallelization, word length, radix, use of BRAM memories, etc. By varying these parameters, the tool generates FFT IP cores with various performance capabilities. These FFT IP cores are of the type feedforward or multi-path delay commutator (MDC) [3]–[6]. This has been shown to be the most efficient approach for high-throughput implementations [3].

The experimental results of this paper serve to identify the limits of the FFT performance on FPGAs. They give an estimation on when it is possible to do the calculations in a single device and when it is needed to think of multi-

Fig. 1. Flow graph of a 16-point radix-2² FFT.

core systems. The experimental results are compared to those provided by other high-throughput FFT architectures in the literature [7]–[9].

The paper is organized as follows. Section II gives an introduction to the FFT algorithm and architectures. Section III describes the tool that we have used for generating the FFT IP cores. Section IV presents the experimental results and analyzes the limits of the FFT performance on FPGAs. Finally, Section V summarizes the main conclusions of the paper.

II. BACKGROUND ON THE FFT

A. The FFT Algorithm

The N -point DFT of an input sequence $x[n]$ is defined as:

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

where $W_N^{nk} = e^{-j\frac{2\pi}{N}nk}$.

The DFT is mostly calculated by the FFT algorithm. The Cooley-Tukey version of the FFT [10] reduces the number of operations of the DFT from $O(N^2)$ to $O(N \log_2 N)$. Although other FFT sizes are possible, most implementations consider FFT sizes, N , that are powers of two.

TABLE I
CONFIGURATION PARAMETERS FOR THE FFT GENERATION TOOL

Parameter	Configurations
FFT Length (N)	Any power of two
Parallel inputs / outputs (P)	Any power of two
Radix	Radix-2 or Radix- 2^2
Data Word Length (WL)	Any integer Constant or Incremental
Number Representation	Fixed point Truncation or Rounding
Rotators	Distributed Logic or DSP Blocks
Coefficient Word Length	Any integer
Internal Memory	Distributed Logic or BRAM
External Memory	Not needed
I/O Data Order	Natural or Bit-reversed

Figure 1 shows the flow graphs of a 16-point FFTs, decomposed using decimation in frequency (DIF) [11]. The FFT is calculated in a series of $n = \log_\rho N$ stages, where ρ is the base of the radix, r , of the FFT, i.e. $r = \rho^\alpha$. The example in Fig. 1 corresponds to radix- 2^2 and, therefore, consists of $n = \log_\rho N = \log_2 16 = 4$ stages. At each stage of the graphs, $s \in \{1, \dots, n\}$, butterflies and rotations are calculated. The butterflies calculate additions in the upper edge and subtractions in the lower one. The rotations are indicated by the numbers ϕ in between the stages, and correspond to the twiddle factors $W_N^\phi = e^{-j\frac{2\pi}{N}\phi}$. Rotations corresponding to $\phi \in [0, N/4, N/2, 3N/4]$ represent complex multiplications by 1, $-j$, -1 and j , respectively. They are considered trivial rotations, because they can be performed by interchanging the real and imaginary components and/or changing the sign of the data.

B. The FFT Hardware Architectures

FFT hardware architectures are classified into three main groups: Memory-based, pipelined and direct implementation. Memory-based architectures [12], [13] calculate the FFT iteratively on data stored in a memory. Pipelined architectures [3]–[9], [14], [15] calculate the FFT in a continuous flow. Finally, a direct implementation maps each addition/multiplication in the FFT flow graph to an adder/multiplier in hardware.

The highest performance is achieved by parallel pipelined FFT architectures [3]–[9] and direct implementations. Parallel pipelined FFTs are classified into multi-path delay commutator (MDC) [3]–[6] and multi-path delay feedback (MDF) [7], [8]. In both cases, high performance is achieved by increasing the degree of parallelization, P , which corresponds to the number of parallel inputs and outputs of the FFT. The direct implementation of the FFT can be considered as a parallel pipelined FFT where the degree of parallelization reaches the FFT size, i.e., $P = N$.

III. FFT GENERATION TOOL

A. Overview of the Tool and Parameters

In order to obtain high throughput MDC FFT architectures, we have developed a computer tool to generate them. The tool is based on parameterizable VHDL code. The parameters and their different configurations are shown in Table I. The tool supports any FFT size, N , and parallelization, P , that

Fig. 2. Building blocks of an FFT architecture.

Fig. 3. Hierarchy of the FFT generation tool.

are powers of two. This allows for arbitrarily long FFTs and selectable throughput based on the parallelization. The tool supports radix-2 and radix- 2^2 , while other radices are expected for further versions. The tool allows to select any data word length, WL , with the possibility of maintaining this word length in all the stages or increasing the word length at every butterfly calculation in order to avoid rounding and truncation effects.

Rotators can be implemented using distributed logic or DSP Blocks. Likewise, either distributed logic or BRAM can be selected for the data memory. This provides flexibility to the implementation, which is particularly useful when the FFT is a part of a bigger system implemented on the FPGA or when it must be allocated in a small FPGA. The proposed architectures also have the advantage that they do not require any external memory, which could limit the performance.

B. Building Blocks and Programming Hierarchy

Figure 2 shows an example of MDC FFT architecture and highlights its building blocks. This example corresponds to $N = 16$, $P = 4$ and $r = 2^2$. As can be observed, the FFT consists of three main building blocks: Butterflies, rotators and shuffling circuits that carry out data permutations. Further examples and description of MDC FFT architectures are found in [3]. They vary in terms of FFT size, radix, parallelization, etc., but they are similar in the sense that they always consist of the same building blocks.

The programming hierarchy for the tool to generate MDC FFT architectures is shown in Fig. 3. This hierarchy is related to the building blocks of the MDC FFT. *FFT_Top* includes the entire architecture, which consists of various *Stages*. Thus, *FFT_Top* generates the different stages, and provides the interface between them. It also translates the FFT parameters in Table I to the specific parameters for each of the stages. For instance, P will be the same for all the stages, whereas the type of memory at each stage may differ depending on the configuration. Each *Stage* generates the corresponding

Fig. 4. Circuits for parallel data shuffling. (a) General structure. (b) Combining multiple buffers to a single memory.

TABLE II
FFT CONFIGURATION PARAMETERS USED IN THE EXPERIMENTS

Parameter	Configurations
FFT Length (N)	16, 64, 256, 1024, 4096, 16384, 65536
Parallel inputs / outputs (P)	4, 8, 16, 32, 64
Radix	Radix-2 ²
Data Word Length (WL)	16 (Constant)
Number Representation	Fixed point, Truncation
Rotators	DSP Blocks
Coefficient Word Length	16
Internal Memory	Decided by the synthesis tool
I/O Data Order	Bit-reversed

building blocks and creates the connections among them. These building blocks are *Permutation_Top*, *Butterfly_Top* and *Rotator_Top*. As *FFT_Top*, they are highlighted in darker gray in Fig. 3 to show that they may consists of various instances of the same element. For example, *Butterfly_Top* generates all the butterflies in a stage, whereas *Butterfly* is the instance of a single butterfly. This can be noted in Fig. 2, where each stage includes two butterflies in parallel. The same idea applies to the rotations. Note also that the type of FPGA resources used to implement the rotators can be selected among distributed logic or DSP blocks. Finally, *Permutation_Top* generates the components for the data shuffling circuits. Some FFT stages only require *Interconnections*, such as the first FFT stage in Fig. 2, while others include *Buffers* and *Multiplexers*. For the *Buffers*, different types of memories can be selected. Furthermore, the tool has the advantage of combining different buffers of the parallel structure in a single memory. This is shown in Fig. 4: Fig 4(a) shows a circuit for parallel permutation and Fig 4(b) shows the integration of $P/2$ buffers of length L and word length WL into a single buffer or memory of size L and word length $WL \times P/2$. This makes the design more compact, leading to more efficient resource allocation.

IV. EXPERIMENTAL RESULTS

The tool described in Section III has been used to generate multiple FFT architectures. This Section presents and analyzes the experimental results that have been obtained. The FFT parameters used for the experiment are shown in Table II.

TABLE III
AREA AND PERFORMANCE OF THE PROPOSED N -POINT P -PARALLEL RADIX-2² FEEDFORWARD FFT ARCHITECTURES FOR $WL = 16$ BITS.

FFT P	Area				f_{CLK} (MHz)	Lat. (μs)	Th. (GS/s)
	Slices	BRAM	DSP	$U(\%)$			
$N = 16$							
4	567	0	12	0.45	335	0.036	1.34
8	910	0	24	0.80	334	0.030	2.67
16★	1612	0	48	1.52	322	0.028	5.15
$N = 64$							
4	782	0	24	0.74	335	0.093	1.34
8	1391	0	48	1.42	332	0.069	2.65
16	2235	0	96	2.59	334	0.057	5.34
32	3838	0	192	4.89	333	0.051	10.66
64★	6590	0	384	9.30	414	0.039	26.49
$N = 256$							
4	924	4	36	1.05	240	0.358	0.96
8	1909	0	72	2.05	322	0.168	2.66
16	3105	0	144	3.77	297	0.128	4.75
32	6230	0	288	7.55	252	0.119	8.06
64	9085	0	576	13.59	198	0.131	12.67
$N = 1024$							
4	1351	12	48	1.52	227	1.256	0.91
8	2241	16	96	2.76	232	0.677	1.86
16	4186	16	192	5.22	221	0.421	3.54
32	8515	0	384	10.16	251	0.243	8.03
64	13257	0	768	18.64	200	0.225	12.80
$N = 4096$							
4	1824	20	60	2.02	230	4.609	0.92
8	2946	32	120	3.64	228	2.404	1.82
16	4918	48	240	6.67	229	1.275	3.66
32	10132	64	480	13.14	185	0.886	5.92
64	20994	64	960	25.95	170	0.588	10.88
$N = 16384$							
4	3418	32	72	3.06	208	19.899	0.83
8	4748	48	144	5.01	226	9.252	1.81
16	7081	80	288	8.77	229	4.659	3.66
32	12894	128	576	16.64	178	3.118	5.69
64	23753	192	1152	31.69	141	2.121	9.02
$N = 65536$							
4	7723	80	84	5.68	125	131.472	0.50
8	9808	96	168	8.17	148	55.689	1.18
16	12281	128	336	12.39	143	28.993	2.29
32	18940	192	672	21.60	143	14.671	4.58
64	30228	320	1344	39.11	115	9.339	7.36
-	74400	3192	2016	= Total FPGA resources			

*: Correspond to the fully parallel or direct implementation.

Table III shows experimental results for the different configurations on a Virtex-6 XC6CSX475T-1-FF1156 FPGA. The area and frequency figures in Table III are based on post place and route results where the default parameters for voltage and temperature were used. It is also assumed that no jitter is present on the system clock. The table includes sub-tables for the different FFT sizes, N . For all of them, the first column shows the degree of parallelization, P . Columns two to four show the area usage in terms of Slices, BRAM and DSP48E1. Column five shows the FPGA resource utilization, U . This is calculated by averaging the percentage of use of Slices, BRAMs and DSP blocks in the FPGA, i.e.,

$$U_{FPGA}(\%) = \frac{Slices(\%) + BRAM(\%) + DSP(\%)}{3} \quad (2)$$

where the percentage of use is calculated from the total shown in the last row of the table. It can be observed that the DSP

Fig. 5. Comparison of throughput vs FFT size.

Fig. 6. Throughput vs FPGA utilization on Virtex-6 XC6CSX475T.

Slices sets the limit of resources in the most demanding case of 64-parallel 65536-point FFT. Larger parallelization would not fit in the FPGA due to an excess of DSPs.

Column six shows the maximum clock frequency supported by the architecture, f_{CLK} . Column seven shows the latency of the FFT, which is equal to:

$$Lat = (N/P)/f_{CLK} + \text{Pipeline time} \quad (3)$$

Finally, column eight shows the throughput in gigasamples per second (GSamples/s), which is calculated as:

$$Th = P \cdot f_{CLK} \quad (4)$$

From the results in Table III, it can be observed that the designs achieve very high throughput. Most of them are in the range of GSamples/s or even tens of GSamples/s. Fig. 5 compares the throughput versus N , for different P . The figure shows that the throughput mainly dependent on P , and presents a certain decay with N , specially for the largest FFTs. The figure also includes results from other high-throughput FFTs on Virtex-6 [9], [16] and ASIC technology [7], [8].

Fig. 6 compares the throughput versus the FPGA utilization, $U(\%)$, for different N . The figure shows a proportionality between the throughput and the resource utilization. It is noticeable that all the designs, even the largest ones, fit well in the FPGA. Thus, not only high-throughput FFTs can be

implemented in a single FPGA, but also they provide room for implementing a bigger system. This highlights the feasibility of current FPGA technologies to implement complex and high-performance signal processing systems in a single chip.

Finally, the processing time of the proposed FPGA designs is from 10 to 100 faster than FFT implementations multi-core systems [1], where the fastest 2048-point FFT on 8 cores is calculated in 32 μs . This confirms the advantage of FPGAs when aiming for high performance, low power and low cost.

V. CONCLUSIONS

This paper has analyzed the performance limits of FFT on current FPGAs. Experimental results show that even as large as 65536-point FFTs can be calculated at GSamples/s. Furthermore, even large and high-throughput FFTs can be allocated in a single FPGA, leaving enough room for implementing other algorithms. Finally, compared to multi-core systems, the use of a single FPGA not only reduces the number of chips, but also increases the performance.

REFERENCES

- [1] C. Brunelli, R. Airoldi, and J. Nurmi, "Implementation and benchmarking of FFT algorithms on multicore platforms," in *Proc. IEEE Int. Symp. Syst. Chip*, Sep. 2010, pp. 59–62.
- [2] Z. Yu *et al.*, "A 16-core processor with shared-memory and message-passing communications," *IEEE Trans. Circuits Syst. I*, vol. 61, no. 4, pp. 1081–1094, Apr. 2014.
- [3] M. Garrido, J. Grajal, M. A. Sánchez, and O. Gustafsson, "Pipelined Radix-2^k Feedforward FFT Architectures," *IEEE Trans. VLSI Syst.*, vol. 21, pp. 23–32, Jan. 2013.
- [4] K.-J. Yang, S.-H. Tsai, and G. Chuang, "MDC FFT/IFFT processor with variable length for MIMO-OFDM systems," *IEEE Trans. VLSI Syst.*, vol. 21, no. 4, pp. 720–731, Apr. 2013.
- [5] M. Sánchez, M. Garrido, M. López, and J. Grajal, "Implementing FFT-based digital channelized receivers on FPGA platforms," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 44, no. 4, pp. 1567–1585, Oct. 2008.
- [6] S. He and M. Torkelson, "Design and implementation of a 1024-point pipeline FFT processor," in *Proc. IEEE Custom Integrated Circuits Conf.*, May 1998, pp. 131–134.
- [7] S.-N. Tang, J.-W. Tsai, and T.-Y. Chang, "A 2.4-GS/s FFT processor for OFDM-based WPAN applications," *IEEE Trans. Circuits Syst. II*, vol. 57, no. 6, pp. 451–455, Jun. 2010.
- [8] T. Cho and H. Lee, "A high-speed low-complexity modified radix-2⁵ FFT processor for high rate WPAN applications," *IEEE Trans. VLSI Syst.*, vol. 21, no. 1, pp. 187–191, Jan. 2013.
- [9] P. A. Milder, F. Franchetti, J. C. Hoe, and M. Püschel, "Computer generation of hardware for linear digital signal processing transforms," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 17, no. 2, pp. 15:1–15:33, Apr. 2012.
- [10] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, pp. 297–301, 1965.
- [11] A. V. Oppenheim and R. W. Schaffer, *Discrete-Time Signal Processing*. Prentice-Hall, 1989.
- [12] C.-M. Chen, C.-C. Hung, and Y.-H. Huang, "An energy-efficient partial FFT processor for the OFDMA communication system," *IEEE Trans. Circuits Syst. II*, vol. 57, no. 2, pp. 136–140, Feb. 2010.
- [13] P.-Y. Tsai and C.-Y. Lin, "A generalized conflict-free memory addressing scheme for continuous-flow parallel-processing FFT processors with rescheduling," *IEEE Trans. VLSI Syst.*, vol. 19, no. 12, pp. 2290–2302, Dec. 2011.
- [14] L. Yang, K. Zhang, H. Liu, J. Huang, and S. Huang, "An efficient locally pipelined FFT processor," *IEEE Trans. Circuits Syst. II*, vol. 53, no. 7, pp. 585–589, Jul. 2006.
- [15] A. Cortés, I. Vélez, and J. F. Sevillano, "Radix r^k FFTs: Matricial representation and SDC/SDF pipeline implementation," *IEEE Trans. Signal Process.*, vol. 57, no. 7, pp. 2824–2839, Jul. 2009.
- [16] Xilinx LogiCORE IP Fast Fourier transform v8.0, Jul. 2012, Online: http://www.xilinx.com/support/documentation/ip_documentation/ds808_xfft.pdf