Abstract—The FFT plays a fundamental role in OFDM programmable digital baseband communication systems under the SDR context. The core nature of this algorithm marks it as a primary target for acceleration. Since long frame lengths of the FFT are desirable in order to achieve higher bitrates, the computational complexity becomes even more significant. In this paper we propose OpenCL FFT kernels for FPGAs and assess their performance with those obtained using GPUs at higher energy efficiency in the FPGA case.

I. INTRODUCTION

Mobile communication systems are adversely affected by multi-path, impulsive parasitic noise and interference effects, which need to be overcome in exiguous spectrum resources and by lower-powered devices. Bandwidth efficient systems which address such challenges are required, and effectively, Orthogonal Frequency Division Multiplexing (OFDM) is one of the most spectrum efficient multi-carrier modulation techniques employed nowadays. It has been adopted in several digital communication standards such as ETSI DVB-T2, -H, -T/H, -T2 and the IEEE 802.11a, to name a few.

OFDM encodes digital data on multiple carrier frequencies, by transforming a wideband channel into a number of narrowband slices which are then frequency multiplexed. This frequency division is achieved by baseband processing, relying on the orthogonality of the carriers base functions.

This paper is organized as follows: in Section II we present the motivation behind the development of Fast Fourier Transform (FFT) Field-Programmable Gate Array (FPGA)-based accelerators using Open Computing Language (OpenCL) [8]; in Section III the multicore-aware optimizations mapped onto the FPGA fabric are presented and hardware-aware optimizations are proposed; in Section IV the experimental results are presented and Section V closes the paper.

II. BACKGROUND AND MOTIVATION: A NEW APPROACH TO THE FFT ON RECONFIGURABLE DEVICES

In practice, a field-deployed wideband OFDM system requires a real-time FFT [1] processor which replaces the bank of modulators and demodulators for each sub-carrier [2]–[7], as seen in Figure 1. Typically, OFDM systems process such sub-blocks on Digital Signal Processor (DSP) and on reconfigurable substrates, such as FPGA devices. While DSPs are limited by their operating clock frequency, FPGA flexibility allows to get over frequency limitations through an increase of parallelism levels, for instance by adopting Single Instruction Multiple Data (SIMD) processing or several compute units, and increase the throughput. However, developing an FPGA system requires domain-specific knowledge for hardware developers, with VHDL and Verilog the most popular hardware description languages pursued when targeting FPGA substrates. In this paper, we propose a new approach for the calculation of the FFT on FPGAs, based on wide-pipeline accelerators generated through an high-level language synthesis tool from Altera that maps parallel kernels developed in OpenCL into gates. The main advantages of these approaches are: i) increased programming flexibility, allowed by OpenCL constructs, opens a faster design space exploration either for the FFT optimization/tuning or for its integration with other system sub-blocks on the FPGA; and ii) development effort since FPGA kernels are designed in a parallel programming language and model that reaches a wider programming community.

The computation of the DFT is best performed by the FFT algorithm. Whereas naive $N$-point DFT implementations demand too many arithmetic operations, $O(N^2)$, the FFT scales with $O(N \log_2(N))$ and allows for better numerical accuracy [1]. Notwithstanding, the FFT is a numerically intensive algorithm necessitating high arithmetic resources and is also communication intensive, requiring several data exchange operations. The latter are typically performed through storage locations, requiring capable methods that efficiently explore the memory hierarchy, for instance in the case of a General Purpose Processor (GPP), or that are able to efficiently overlap...
data exchanges with computation.

A. FFT Design Space on Multicores

The aim of a fast FFT implementation is to provide a collection of algorithms whose combination provides the optimal algorithm for the particular architecture on which execution will take place. Implementations are found in the literature for multicore devices, namely for multicore processors, CPUs and GPUs) [9]–[11], that address this. In this work, the Fast Fourier Transform in the West (FFTW) and the CUDA FFT (CUFFT) libraries will serve as benchmarks for the wide-pipeline accelerators herein developed. These libraries have been chosen as they convey very high computational performance on CPU and GPU platforms. Moreover, they do so by finding the optimal planner for the execution device [9]. Even though similar planner optimization techniques could be pursued for FPGA accelerators, it expands well beyond the scope of the work herein presented. Thus for the latter case, hard-coded accelerators for fixed radix-\( n \) FFTs of length \( N \) were developed.

In the particular case of GPU devices, the major challenges needed to be overcome to develop efficient FFT kernels [12]–[14] are the following:

A) use of an appropriate data layout in memory that allows coalesced accesses to global memory;
B) radix-\( n \) factorizations that drive the work-items granularity to appropriate GPU occupancy levels;
C) use of the local memory between FFT stages so work-items exchange data in a higher memory hierarchy level;
D) special strided accesses to local memory that minimize the number of bank conflicts due to local banks.

III. MAPPING THE DESIGN SPACE TO CIRCUITS

In this Section, we overview the wide-pipeline accelerator instantiated from OpenCL kernels and the design space exploration we performed for the FFT.

A. Altera OpenCL

The Altera OpenCL tool is a High-Level Synthesis (HLS) framework that allows generating a RISC-like processor from an OpenCL kernel [16]. This is achieved by translating the C99 OpenCL kernel description into a Register Transfer Level (RTL) project instantiating a wide-pipeline architecture in Verilog. This type of architecture is characterized by delivering high data processing rates through the pipelined execution of work-items, instead of processing them in parallel or in Single Instruction Multiple Thread (SIMT) fashion. This is illustrated in Figure 2 for the case of a typical parallel vector-add kernel. Whereas a SIMT processor would asynchronously schedule synchronous groups of work-items to perform the vector addition, by default, the instantiated wide-pipeline architecture translates the kernel instructions into a single core accelerator that is able to process a work-item in each stage of its pipeline in parallel. Consecutive work-items are processed within consecutive clock cycles in a processing rate of one work-item per clock cycle, which is also designated as Initiation Interval (II) [15]. The target clock frequency of the accelerator is 250 MHz and is limited by the arithmetic complexity of the OpenCL kernel and memory usage profile of the OpenCL-defined memory spaces [8].

In addition to the restrictions imposed by the arithmetic and memory access patterns of the kernel in the wide-pipeline accelerator operating frequency, the overall performance is also driven by the following aspects:

a) number of Compute Units (CUs) in the pipeline;
b) vectorized processing through \{2,4,8,16\}-way SIMD;
c) loop unroll explores Instruction Level Parallelism (ILP);
d) distinct DRAM banks to maximize global bandwidth;
e) explicit BRAM usage is limited to local memory;
f) barriers stall the pipeline.

The combination of a) and b), effectively lowers the II to less than a clock cycle per work-item as: a) each compute unit can dispatch a work-item; or b) each compute unit can vectorize several work-items; with a) and b) mutually non-exclusive. Also, when data dependencies are low or non-existing, loop unrolling increases the data throughput of the accelerator as ILP can be exploited by the compiler through c). Furthermore, as the global memory space is implemented in the dual-bank DRAM memory of the FPGA board, bandwidth peaks by deploying input and output buffers to distinct banks d).

The foremost critical aspect of the current possibilities of the Altera OpenCL is the inability to explicitly configure the usage of BRAM memory by the kernel e). BRAMs can be used as local memory buffers for data exchange among work-items.
However, this is preferable to the alternative of non-pipelined execution of OpenCL kernels for each stage, constrained to the use of global memory only where the whole execution grid must be flushed first so that the next stage may execute. Moreover, whenever the same local buffer memory location is utilized in different stages of the kernel, which in the FFT case means as many times as there are stages minus one, there is a time growth of the wide-pipeline critical path. We have tested two ways to avoid this are: the use of circular buffering so that the local memory buffer used prior to a stage is always different from the one used after it (only two are necessary); or we can define local memory buffers that are used only once. Both strategies are possible due to the availability of a large number of BRAM blocks, whereas in multicores local memory is typically limited to 48KB.

### IV. Experimental Results

The FFT has been run for lengths of $N = \{256, 1024\}$ on: Intel Nehalem i7 950 core using FFTW3 using the FFTW\_EXHAUSTIVE planner flag; Nvidia GeForce GTX 680 running CUFFT 5.0 in native FFTW compatibility (CUFFT\_COMPATIBILITY\_NATIVE) for highest performance [11]; Nullatech PCIe 385N-D5 Statrix V FPGA board using Altera OpenCL 13.0. The obtained results are shown in Table I and discussed herein, always for $N = \{256, 1024\}$ respectively, when results are stated in pairs, or for $N = 256$ singularly. It is worth noticing that while OpenCL is a cross-platform programming model, the FFT radix-4 design included in the Altera OpenCL SDK and the best performing proposed...
TABLE I

EXPERIMENTAL RESULTS FOR $N = \{256, 1024\}$ RADIX-4 FFT ALGORITHMS RUN ON FPGA, CPU AND GPU. THE THROUGHPUT IS EXPRESSED IN MEGA WORK-ITEM/S AND IN MEGA FFTS/S (M/S). I/R STANDS FOR INSUFFICIENT RESOURCES

<table>
<thead>
<tr>
<th>Device</th>
<th>FPGA</th>
<th>GPU/CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td>$N$ Radix</td>
<td>Total Logic (%)</td>
</tr>
<tr>
<td>Proposed 1 buffer</td>
<td>$256$ 4</td>
<td>71 31 43 12 215.14 108.95 1.387 3.32</td>
</tr>
<tr>
<td>Proposed circular buffer</td>
<td>$71$ 31 43 12 193.38 141.12 1.277 3.0</td>
<td></td>
</tr>
<tr>
<td>Proposed 3 buffers</td>
<td>$70$ 31 43 12 226.75 172.23 1.387 3.31</td>
<td></td>
</tr>
<tr>
<td>Proposed 3 buffers (with constant coefficients)</td>
<td>$60$ 26 43 3 193.34 172.23 1.287 3.33</td>
<td></td>
</tr>
<tr>
<td>Proposed 3 buffers (with manual SIMD of 2 FFTs)</td>
<td>$85$ 36 53 5 160.90 136.87 2.13 I/R</td>
<td></td>
</tr>
<tr>
<td>Altera [16]</td>
<td>$256$ 4</td>
<td>76 41 35 6 212.13 190.00 1.99 I/R</td>
</tr>
<tr>
<td>$1024$ N/A</td>
<td>87 40 45 8 204.7 190.00 1.27 I/R</td>
<td></td>
</tr>
<tr>
<td>CUFFT [11]</td>
<td>$256$ N/A</td>
<td>-</td>
</tr>
<tr>
<td>$1024$ N/A</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FFTW [9]</td>
<td>$256$ N/A</td>
<td>-</td>
</tr>
<tr>
<td>$1024$ N/A</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

solution do not compile due to lack of local memory resources on the GPU engine. These are tagged I/R, standing for Insufficient Resources.

The benchmark CPU FFTW computes 0.31 and 0.53 Mega transforms per second (MFFT/s), while the GPU CUFFT reference delivers 5.36 and 2.75 MFFT/s. This compares with the Altera SDK which is able to achieve 3.99 and 1.27 MFFT/s, while the optimal proposed algorithm peaks at 2.13 MFFT/s on the reconfigurable substrate. However, as previously said, none of the latter support cross-platform capabilities with GPUs.

Furthermore, analysis of the proposed FFTs shows that the use of only one local buffer is as efficient as defining single-use buffers, shown in Table I. The circular use of two local buffers penalized not only the delivered performance, 1.27 MFFT/s, but also the accelerator operating clock frequency, 193.38 MHz, while the former two are clocked at 215.14 and 226.75 MHz, respectively. All these solutions fail to be improved by defining SIMD units or more than one CU, due to high usage of FPGA logic resources.

When off-loading the twiddle coefficients to constant, a clock frequency reduction is experienced, down to 193.34 and the delivered throughput is 1.28 MFFT/s. The logic utilization saving is insufficient to drive the logic consumed to levels where 2-way SIMD or defining 2 CUs would be possible. However, vectorized processing may be defined manually by loading two sequences at once. This approach comes with the advantage of being possible to define SIMD-like processing as ILP is explored by the compiler using less logic resources than if the compiler were to generate a SIMD CU or several CUs. Inevitably, this lead to such consumption of local memory resources that GPU compilation of this kernel is not possible.

V. CONCLUSION

In this paper, we have shown that FPGA-based FFT accelerators can be realized by exploring the OpenCL programming model as a high-level synthesis hardware description. We have explored the mapping of the FFT kernel optimization on GPUs and the yielding results on FPGAs, revealing that multicore-aware optimizations can be exploited successfully for wide-pipeline accelerators. Our proposed solution explores optimization techniques maintaining cross-platform capabilities on GPUs and FPGAs, the latter at reduced power consumptions.

ACKNOWLEDGMENT

The authors acknowledge Instituto de Telecomunicações, Fundação para a Ciência e Tecnologia grant and scholarship PEst-OE/EEI/LA0008/2013 and SFRH/BD/78238/2011.

REFERENCES