To buffer, or not to buffer? A case study on FFT accelerators for ultra-low-power multicore clusters

Author(s):
Bertaccini, Luca; Benini, Luca; Conti, Francesco

Publication date:
2021

Permanent link:
https://doi.org/10.3929/ethz-b-000506206

Originally published in:
https://doi.org/10.1109/ASAP52443.2021.00008

To Buffer, or Not to Buffer? A Case Study on FFT Accelerators for Ultra-Low-Power Multicore Clusters

Luca Bertaccini
ETH Zurich
Zurich, Switzerland
lbertaccini@iis.ee.ethz.ch

Luca Benini
ETH Zurich and University of Bologna
Zurich, Switzerland
lbenini@iis.ee.ethz.ch

Francesco Conti
University of Bologna
Bologna, Italy
f.conti@unibo.it

***Abstract*—Hardware-accelerated multicore clusters have recently emerged as a viable approach to deploy advanced digital signal processing (DSP) capabilities in ultra-low-power extreme edge nodes. As a critical basic block for DSP, Fast Fourier Transforms (FFTs) are one of the best candidates for implementation on a dedicated accelerator core; however, their peculiar memory access patterns make direct integration of an FFT accelerator with a core cluster challenging. In this paper, we compare two different approaches for cluster-coupled FFT accelerators: one with a large internal buffer to store and shuffle partial results; and a buffer-less accelerator sharing all memory with the cluster cores. Both versions can work on complex data with 8/16/32-bit real and imaginary parts. We show that, thanks to a newly proposed scheme to reorder data access and exploit full bandwidth also for sub-word FFTs, the buffer-less accelerator can be made as fast as the buffered one at only 0.26× the area cost. We report post-layout performance and power results showing that the buffer-less accelerator can provide up to 4/2/1 butterfly/cycle performance, with an average power consumption of 4.1/5.5/6.8 mW @ 350 MHz, 0.65 V operating point in 22 nm CMOS technology, respectively for complex data with 8/16/32-bit real and imaginary part. The buffer-less accelerator is 8× faster than an optimized multicore software implementation working on 16-bit data and compares favorably with FFT accelerators presented in the recent literature.**

***Index Terms*—Fixed-point FFT, cluster-coupled HW accelerator, reordering scheme, energy-efficient architecture, buffer-less accelerator.**

I. Introduction

The coming of age of the internet-of-things (IoT) has increased the demand for small ultra-low-power edge devices with wireless interfaces. To reduce on-air time and bandwidth requirements and achieve higher energy efficiencies, more and more processing is performed on the edge, requiring advanced digital signal processing (DSP) capabilities. Heterogeneous systems augmented with domain-specific hardware accelerators can enhance IoT systems-on-chip to achieve higher energy efficiencies and meet tight timing constraints at a lower area cost than purely homogeneous, software-based ones [1]. This can also be seen as a side effect of the slowdown of Moore's law: power and performance improvements driven by technology scaling are becoming less cost-effective, and parallelism, domain specialization, and heterogeneity are needed to keep the pace of progress required by applications [2], [3] and improve silicon utilization [4], [5].

In most heterogeneous architectures, hardware accelerators achieve massive speedup by exploiting dedicated memories for internal buffering, which end up dominating area and power consumption [1]. Many hardware-accelerated tasks are active only for a fraction of an algorithm's execution time, often intermixed with software-based operations. This may lead to poor utilization of silicon-expensive memory resources, and requires marshaling and data transfers every time the execution flips from hardware-accelerated to software-based.

In this paper, we investigate a possible avenue for optimization: replacing internal buffers in domain-specific accelerators with resources already allocated in the system, and shared with other devices such as software cores, to improve area and energy efficiency. We concentrate our study on hardware-accelerated multicore clusters for ultra-low-power extreme edge nodes, and in particular on fast Fourier transforms (FFTs) – a critical and frequently used block in most modulation schemes for communication, DSP, and audio processing frontend [6]. FFTs are often supported by hardware accelerator engines. Several sensor applications require high-precision FFTs, using 16-bit to 32-bit number formats, while lower-precision 8-bit FFTs may be employed as a pre-processing step for machine learning algorithms such as quantized neural networks [7].

FFT acceleration is usually implemented either with accelerators decoupled from the cores, requiring data copy, or supported by instruction extensions within the pipeline of processor cores, which do not require data copies, but have lower potential for acceleration, due to the limited bandwidth of the cores' load/store interface. Cluster-coupled hardware processing engines (HWPEs) have been recently proposed to get the best of both worlds [8], [9]. In this scenario, all cores – both general-purpose and HWPEs – cooperate by directly sharing data through a low-latency multi-banked L1 data memory called tightly-coupled data memory (TCDM). Contrary to traditional accelerators, HWPEs can do without an internal buffer and rely entirely on the TCDM. With this design choice, the area efficiency of the HWPE can be significantly boosted, but questions arise on a potential bottleneck at the accelerator interface with the TCDM. This is a major concern for the specific case of FFT, which is well known to have complex and critical memory access patterns.

To study this specific point, in this paper we focus on a

practical FFT use case. In FFT accelerators, the presence or absence of an internal buffer is particularly relevant because partial results have to be iteratively shuffled. This fact is well known and has been extensively studied in the past. To exploit data locality, various FFT accelerators [10]–[12] have been designed with dedicated internal memories. Baas [13] proposed an intermediate solution, including a smaller cache in the accelerator, while other architectures, such as Texas Instruments tightly-coupled accelerator [14], share the memory with cores.

The FFT algorithm introduces bit-reversed reordering in the last stage of a decimation-in-time (DIT) FFT and leads inevitably to the presence of banking conflicts in accessing the multi-banked TCDM – most FFT accelerators have to be designed around these issues. To achieve a conflict-free FFT algorithm, Cohen [15] and Ma [16] propose different reordering schemes of the butterfly sequence, which allow the two butterfly inputs to be always in different memory banks. However, they target single butterfly engines and do not consider sub-word FFTs. In this work, we present:

1) Two novel designs for cluster-coupled FFT HWPEs: a buffer-less architecture and a buffered one implemented, in $22\,\mathrm{nm}$ CMOS at $0.65\,\mathrm{V}$. The former shares a 16-bank TCDM with 8 cores in the cluster, reducing the hardware footprint; the latter stores the FFT samples and intermediate results internally, reducing memory traffic. Both HWPEs can work at different fixed-point precisions with $8/16/32$-bit real and complex parts.
2) A novel reordering scheme, for the buffer-less design, that boosts the HWPE performance allowing for conflict-free accesses to the shared TCDM when computing sub-word FFTs, with no compromise in terms of performance.

Our experimental results show that the buffer-less accelerator offers a favorable area, performance, and energy design with respect to the buffered one, and it outperforms state-of-the-art accelerators [10]–[12], [14], [17].

II. Related Work

As the most widespread algorithm in DSP applications, FFTs are often computed on dedicated hardware blocks. FFT accelerators have been extensively investigated both by academia and industry, exploring numerous fully parallel and iterative implementations. Fully parallel architectures contain all the butterfly units necessary for an N-point FFT, while iterative accelerators work on a smaller datapath. The former provides high performance, like the 2018 fastest FFT architecture by Garrido *et al.* [18], but results in a large area for high values of N. For this reason, iterative architectures are usually adopted for embedded applications.

Iterative accelerators are typically memory-based, employing internal buffers to store and shuffle partial results, which dominate the area and power of such architectures. Chen *et al.* [11] include two processing engines in their design, each containing two butterfly units and $16\,\mathrm{KiB}$ of dedicated memory. Wang *et al.* [10] introduced a reconfigurable $2^n3^m5^k$ FFT accelerator implemented in $16\,\mathrm{nm}$ technology and working on

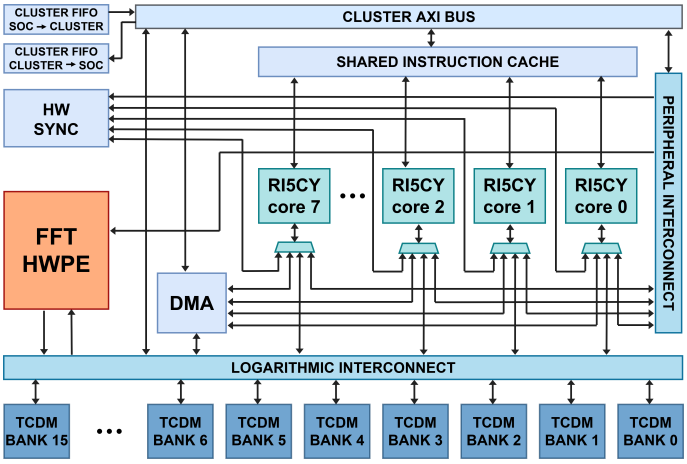

Fig. 1. PULP cluster with FFT HWPE.

24-bit precision. The accelerator occupies $0.37\,\mathrm{mm}^2$ and has been integrated with a RISC-V core. Their results show how the dedicated internal memory dominates both the area and the power consumption of the FFT accelerator. The NXP MAPLE-B [19], a multi-radix decoupled accelerator, includes an I/O data buffer, and the FFT module in the Infineon advanced driver assistance subsystems [20], part of the AURIX™ family - TC23xLA, also contains a RAM buffer for storing calculation coefficients.

A different approach is followed by Texas Instruments HWAFFT [14], an FFT accelerator tightly coupled to a DSP core. The accelerator is located outside the core but shares the core memory bandwidth. Such a design choice avoids the need for an internal buffer but provides the accelerator with a narrower memory bandwidth preventing high degrees of parallelization. TI HWAFFT contains only a single radix-2 butterfly unit working on 16-bit precision.

The recently proposed cluster-coupled HWPEs [8] represent a further alternative, directly sharing data with cores through a low-latency multi-banked data memory. Nevertheless, the absence of an internal buffer and FFT peculiar memory access patterns require a reordering scheme to avoid systematic conflicts when accessing the TCDM. Many reordering schemes have been proposed in the past two decades. Some well-known schemes such as Cohen [15], Johnson [21], and Ma [16] only targeted architectures including single butterflies units. Baek and Choi [22] introduced an address generation scheme for FFT architectures containing multiple radix-2 butterflies. Nonetheless, their proposal, as well as Sorokin and Takala's [23] and Xia *et al.*'s [17] methods, requires the input samples not to be initially stored inside the memory in natural order. This is not ideal since reordering data before the computation is costly. However, this overhead can be mitigated when employing an internal buffer since buffered architectures require explicit copy-in and copy-out phases that can be exploited to shuffle the samples.

Due to the large variety of applications requiring FFTs, different precision and range have been targeted. Bo *et al.* [24]

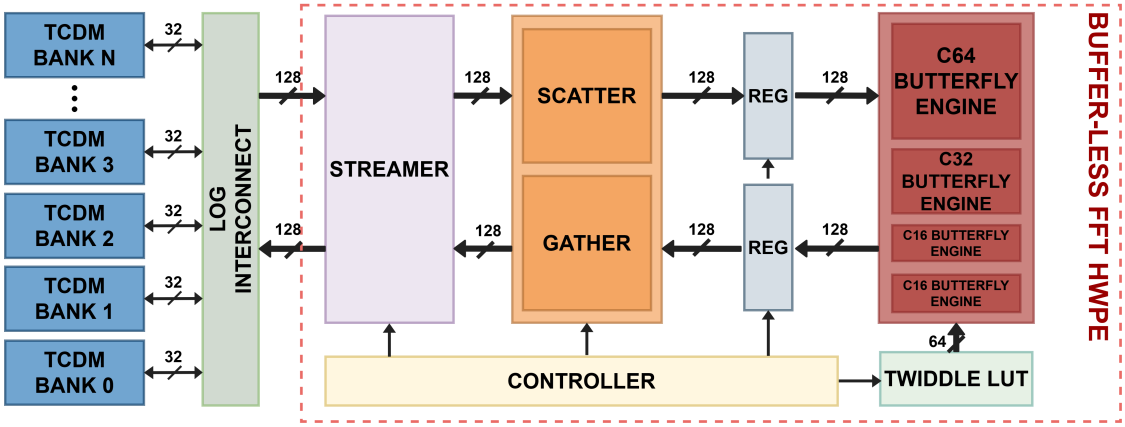

Fig. 2. Buffer-less FFT HWPE architecture.

proposed an application-specific instruction-set processor using 8-bit or 16-bit precision, Wang *et al.* [10] employed 24-bit fixed-point data, the FFT accelerator contained in the Infineon FFT accelerator [20] works on 16-bit or 32-bit values, while Chen *et al.* [11] used floating-point data. However, to the best of our knowledge, an accelerator supporting low, mid, and high precision is still missing.

In this work, we present two cluster-coupled FFT accelerators, one including a large internal buffer and one sharing the memory with the cores. The buffer-less accelerator employs a newly proposed reordering scheme to efficiently make use of the external memory. Both accelerators can work with $8/16/32$-bit precision.

III. Architecture

The architecture template we build upon is based on the open-source multicore PULP cluster [25], augmented with cluster-coupled HWPE[1].

A. PULP Cluster

The PULP cluster, shown in Fig. 1, contains eight RISC-V cores enhanced with DSP instruction set architecture extensions [26], sharing a 4 KiB instruction cache and 64 KiB of TCDM divided into 16 banks, implementing word-level interleaving to reduce the probability of contention. The cluster also includes a DMA, a peripheral interconnect, an AXI bus, and provides support for a software-controlled tightly-coupled hardware accelerator. The cluster cores, the DMA, and the HWPE are connected to the memory banks through a high-bandwidth logarithmic interconnect and can access the TCDM with a 1-cycle latency. The HWPE and the cluster cores communicate through the peripheral interconnect for control plane interactions.

An open-source example of HWPE[2] provides a template to build hardware accelerators conceived to be integrated into a PULP cluster. The fixed environment around the accelerator allows various internal modules to be reused for different HWPEs, speeding up the design of new accelerators. Such units only require to be tailored to each particular implementation and contain the interface towards the TCDM and the cluster cores. The rest of the accelerator is intrinsically application-dependent, requiring ad-hoc designs.

B. Buffer-less FFT HWPE Architecture

The proposed FFT accelerator is organized in a pipelined architecture and supports different bit-widths. It can compute up to $512/1024/2048$-point FFT, respectively for complex data with $8/16/32$-bit real and imaginary parts. Let us define for conciseness these formats as $C16/C32/C64$. We show the structure of the buffer-less HWPEs in Fig. 2. The accelerator implements the radix-2 Cooley-Tukey FFT algorithm [27] in its DIT flavor and consists of several submodules.

Butterfly Unit: the computational module of the accelerator. It is fully combinational and contains one $C64$, one $C32$, and two $C16$ butterfly engines. The HWPE reuses larger engines to compute lower-precision butterflies. Therefore, the accelerator can compute either one $C64$, two $C32$, or four $C16$ butterflies each cycle. Each butterfly engine employs fixed-point arithmetic in the format $Q0.x$, requiring the inputs to be smaller than 0.5 and dividing the outputs by 2 (except in the last FFT stage) to prevent overflow, as presented for example in [28].

Twiddle Factor lookup tables (LUTs): the twiddle factors corresponding to the maximum number of FFT points are stored in LUTs. An N-point FFT requires $N/2$ different twiddle factors. However, when N is a power-of-two, only $(N/8+1)$ twiddle factors are necessary, as the remaining can be obtained by exploiting trigonometric identities. For this reason, we only store $M/8$ twiddle factors in the LUT, where M is the maximum number of supported FFT points. Since we need up to four twiddle factors each cycle and the accelerator supports low, mid, and high precision, we include one $C64$ LUT containing 65 elements, one $C32$ LUT including 129 values, and two $C16$ LUTs with 257 twiddle factors. Higher-precision LUTs are still used when computing lower-precision FFTs, rounding their values to reduce the precision.

[1]https://github.com/pulp-platform/hwpe-doc
[2]https://github.com/pulp-platform/hwpe-mac-engine

Fig. 3. Buffer-less FFT HWPE pipeline. The example shows the beginning of a 64-point $C64$ FFT first stage. One $C64$ sample occupies one 64-bit register. In case of lower-precision FFTs, samples are packed inside 64-bit registers.

Streamer: a specialized DMA unit. This module manages the communication from/to memory. The buffer-less FFT HWPE contains eight 32-bit memory ports; four are used as output ports, and four as input ports. The communication with the TCDM is based on *ready/valid* handshakes.

Controller: the unit that controls the whole computation. It communicates and aligns all the modules in the accelerator. The controller contains an FSM that organizes the FFT computation, the logic to select each cycle the right twiddle factors from the LUT, and a set of registers software-programmed by the cores to communicate with the accelerator. In particular, a core will communicate to the HWPE when to start the FFT computation, the number of FFT points, the base address of the vector to be transformed, its bit-width, and whether or not to implement the final bit-reversed reordering. If the results are reordered, they are stored in the memory locations following the input vector, while non-reordered results overwrite the input samples. Non-reordered FFTs may be employed to speed up FFTs that require a higher number of points than the supported one. For example, all the stages of a $C32$ 2048-point FFT, except the last one, can be computed using two non-reordered $C32$ 1024-point FFT [14].

Butterfly Registers: two sets of four 64-bit registers located around the butterfly and used to shuffle inputs and outputs.

Gather-Scatter: the Scatter unit reorganizes the data coming from the memory in the set of registers before the butterfly unit, while the Gather unit reads processed data from the registers, rearranges them, and passes them to the Streamer.

C. Buffer-less FFT HWPE Reordering Scheme

To avoid the need for an internal buffer and to shuffle partial results, we propose a new scheme to reorder the butterfly sequence and the memory access patterns. Such a scheme considers having access to $N \geq 16$ memory banks and always loads/stores data at consecutive memory locations, preventing data races and conflicts between loads and stores, and allowing to access two $C16$ samples with one 32-bit memory port. Furthermore, it can be guaranteed that the load and store operations are not simultaneously trying to access the same memory banks by tuning the number of banks and the accelerator's pipeline latency. Due to its pipeline depth, the HWPE takes 6 cycles between a load request and a store request addressing the same memory locations. With 16 TCDM banks available, conflicts never happen. An example is provided in Fig. 3, where the first simultaneous requests are shown. The load requests the samples of indices $(6, 7)$, while the store request addresses the samples of indices $(0, 1)$. Since a $C64$ example is presented, each sample occupies two consecutive memory locations; thus, the load requires access to banks $(12, 13, 14, 15)$, while the store to banks $(0, 1, 2, 3)$.

Since the distance between the left wing of a butterfly and its right wing is fixed, the buffer-less FFT HWPE loads first a set of left wings at consecutive memory addresses, and then, during the following cycle, the right wings of the same butterflies. After loading new samples, the Scatter unit reorganizes the data, storing them into two of the four 64-bit registers in front of the butterfly unit. Once the four registers are full, both the left and right wings of some butterflies are stored in the registers, and the butterfly unit can start to fetch and process data. Once the registers are full, the Scatter decides each cycle where to store the new inputs, overwriting the samples just processed by the butterfly engines. An example is provided in Fig. 3, where the samples of indices $(2, 3)$ overwrite $(0, 32)$ after the first butterfly has been computed. After the butterfly unit, the situation is similar, partial results are stored into two of the other four 64-bit registers, and the Gather unit behaves complementary to the Scatter, rearranging the butterfly outputs and passing them to the Streamer.

In a DIT radix-2 FFT algorithm, the outputs are provided in bit-reversed order. Reordering data from natural to bit-reversed orders requires some additional techniques not to produce bank conflicts systematically. The new reordering scheme allows us to write two bit-reversed samples every cycle. During the last FFT stage, the inputs of a butterfly are at consecutive indices. Therefore, the HWPE loads the samples to compute butterflies whose bit-reversed outputs are consecutive two-by-two. E.g., for a 64-point $C32$ FFT, we can load first $(0, 1, 2, 3)$, and then $(32, 33, 34, 35)$. Now, we can compute the butterflies $(0, 1)$ and $(32, 33)$, whose bit-reversed output are $(0, 32)$ and $(1, 33)$. We cannot store all the results in one cycle since we would generate conflicts. We can exploit only half of the bandwidth, storing first $(0, 1)$, and then $(32, 33)$, stalling the pipeline for one cycle not to overwrite meaningful data inside the accelerator registers. For $C64$ FFTs, since we store two samples per cycle, we can exploit the full output bandwidth, while for $C16$ FFTs, only one-fourth of the bandwidth can be used. Bank conflicts can still arise between the load and store requests, but not systematically. Other conflicts can happen if the HWPE and the cores try to access the same banks simultaneously. All the residual conflicts are handled by the accelerator, delaying the load request by one cycle and stalling its pipeline.

D. Buffered FFT HWPE

The buffered accelerator, in Fig. 4, is structured similarly to the buffer-less HWPE, implementing only a couple of architectural modifications. The two sets of registers around the butterfly unit are replaced by a large internal memory. Such

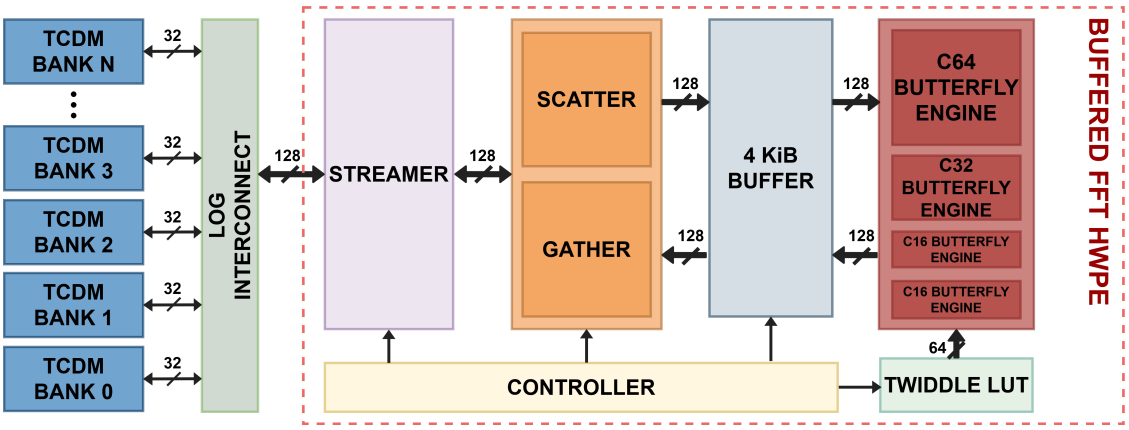

Fig. 4. Buffered FFT HWPE architecture.

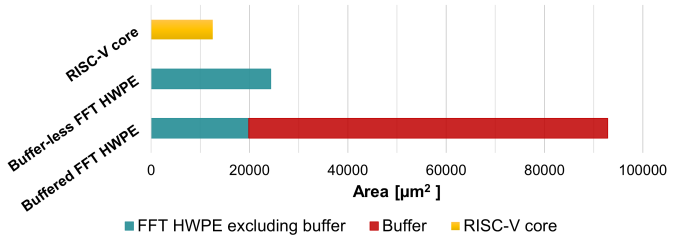

Fig. 5. Area comparison of buffered and buffer-less FFT HWPE synthesized at 200 MHz.

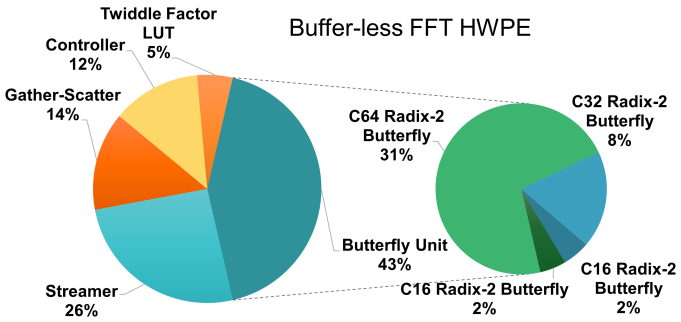

Fig. 6. Area breakdown buffer-less FFT HWPE synthesized at 200 MHz.

buffer consists of 4 KiB of standard cell memory [29], divided into four banks containing $C16$ words, each providing two input and two output ports. The buffer is organized in multi-ported banks to avoid conflicts when reordering and shuffling data. The internal buffer size limits the maximum number of FFT points supported by the buffered HWPE. A smaller set of registers is still kept around the butterfly unit to break the system's critical path. Furthermore, since data are never loaded and stored simultaneously, only four 32-bit memory ports are needed and used both as input and output ports. Consequently, the Streamer is much simpler than in the buffer-less case, while some additional logic is necessary to generate the sample addresses inside the buffer.

The buffered FFT HWPE first loads the samples from the TCDM to the internal buffer; then the computational phase start and, each cycle, data are read from the buffer, processed through the butterfly unit, and written back to the buffer; finally, data are read from the buffer and stored into the TCDM. During the initial loading and the final storing of the buffer, no computation is performed. In the buffered accelerator, we implement the bit-reversed reordering when loading the inputs. The buffer structure allows us to write two bit-reversed samples every cycle, similarly to the buffer-less case. However, due to the presence of the dual-ported buffer, reordering the butterfly sequence is not necessary for such implementation.

IV. Experimental Results

To evaluate the different versions of our FFT HWPEs, we integrated them in a PULP cluster [25], Fig. 1.

A. Area

We synthesized the PULP cluster first with the buffered FFT HWPE and then with the buffer-less accelerator, using Synopsys Design Compiler in GlobalFoundries 22FDX, with topographical synthesis. We targeted 200 MHz of clock frequency under worst-case conditions (SSG, 0.59 V, −40 °C). The buffer-less HWPE occupies 3.8× less area than the buffered design, and around 2× the area of a RISC-V core. We show the area comparison in Fig. 5, and the area breakdown of the buffer-less accelerator in Fig. 6. As can be noticed in the buffer-less HWPE, the area is dominated by the butterfly unit, where around 70% of the area is dedicated to the $C64$ butterfly engine. Furthermore, for the buffered accelerator, modifying the maximum number of supported FFT points, requires changing both the internal buffer size and the twiddle factor LUTs, while, for the buffer-less HWPE, only the LUTs have to be adapted. Since the current twiddle factor LUTs occupy only 5% of the buffer-less accelerator area, the maximum number of supported FFT points may be increased involving a low area overhead, highlighting another advantage of implementing buffer-less architectures. In the buffered case, the internal buffer already

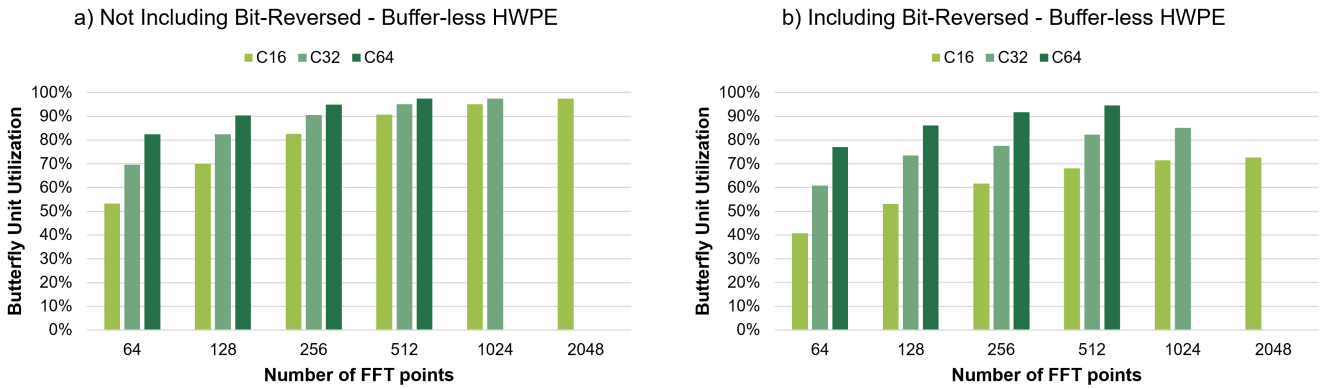

Fig. 7. Butterfly utilization of the buffer-less FFT HWPE for (a) FFTs excluding the bit-reversed reordering and (b) FFTs including the bit-reversed reordering. $C64$ 1024/2048-point and $C32$ 2048-point FFTs are not shown since not supported by the proposed architecture.

TABLE I
PERFORMANCE OF THE BUFFER-LESS AND BUFFERED HWPES FOR ALL THE SUPPORTED BIT-WIDTHS AND NUMBER OF FFT POINTS*

	Buffer-less FFT HWPE			Buffered FFT HWPE		
N [points]	**C16 FFT [cycles]**	**C32 FFT [cycles]**	**C64 FFT [cycles]**	**C16 FFT [cycles]**	**C32 FFT [cycles]**	**C64 FFT [cycles]**
64	118	158	249	115	171	282
128	211	305	520	221	349	605
256	415	661	1116	447	735	1311
512	845	1399	2434	929	1569	2549
1024	1791	3005	-	1955	3363	-
2048	3875	-	-	4133	-	-

*All the reported values refer to FFTs including bit-reversed reordering.

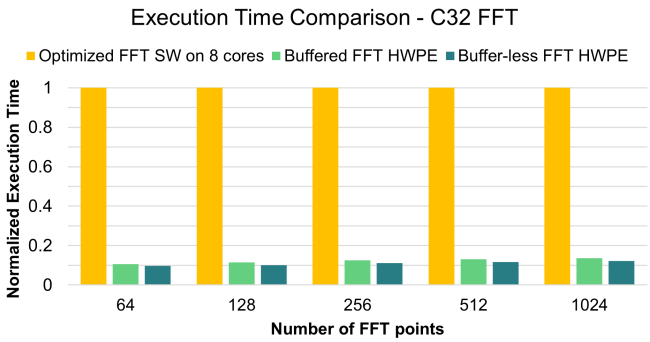

Fig. 8. Execution time of the optimized FFT SW running on 8 cores vs. the buffered and buffer-less FFT HWPEs.

counted for more than 75% of the HWPE area. Increasing the number of FFT points for such an architecture would then be highly costly.

B. Butterfly Unit Utilization

The butterfly utilization of the buffer-less HWPE for FFTs, including and excluding bit-reversed reordering, is shown in Fig. 7. Since the output bandwidth is not fully exploited when reordering $C16/C32$ results, $C16/C32$ FFTs without bit-reversed reordering present higher butterfly utilization.

When doubling the data precision, the number of available butterfly engines halves, thus doubling the number of butterfly cycles. Therefore, the HWPE reaches the same utilization for an N-point p-bit FFT and an $N/2$-point $2p$-bit FFT without the bit-reversed reordering. This is not true for FFTs including the bit-reversed reordering, where the final reordering prevents from using the whole output bandwidth for $C16/C32$ data, resulting in a different butterfly utilization. Despite reaching a lower percentage of butterfly utilization, thanks to the larger number of butterflies computed per cycle, the HWPE provides higher performance when less precision is required. For a reordered 512-point $C16/C32/C64$ FFT, the accelerator takes $835/1399/2434$ cycles.

In both bit-reversed reordered and non-reordered cases, the utilization grows when increasing the number of FFT points. For high numbers of FFT points, the control overhead and the time spent to fill and empty the pipeline decreases more and more with respect to the butterfly cycles. For $C16/C32/C64$ $2048/1024/512$ FFTs, the HWPE reaches over 97% of butterfly cycles, when excluding the bit-reversed reordering, and $73\%/85\%/95\%$ when including the bit-reversed reordering. The 2% utilization drop in the case of a $C64$ 512-point FFT is due to stalls introduced during the last stage to prevent conflicts between natural-order loads and bit-reversed stores. The buffered HWPE reaches a slightly lower utilization due to the time spent copying in/out the data. For a $C16/C32/C64$ $2048/1024/512$ FFT, including bit-reversed reordering, it achieves $68\%/76\%/90\%$ of butterfly cycles.

C. Performance

We report the performance of our FFT accelerators in Table I. The buffer-less version provides slightly better performance than the buffered HWPE since it always operates in a compute phase, not requiring the initial loading and final storing of the buffer when no butterflies are computed. We compared the performance of our HWPEs with a highly optimized fixed-point FFT software, running on the 8 cluster cores and exploiting their DSP custom instructions. The results are shown in Fig. 8.

TABLE II
COMPARISON OF FFT HARDWARE ACCELERATORS, STAND-ALONE (TOP SIX ROWS), AND IN A FULL-SYSTEM CONTEXT (BOTTOM THREE ROWS)

Design	Peak Performance [butterfly/cycle]	Technology	Voltage	Frequency	Area	Performance (1024-point)	Average Power (1024-point)	Energy Efficiency (1024-point)
Buffer-less FFT HWPE	4 $C16$ / 2 $C32$ / 1 $C64$	22 nm	0.65 V	350 MHz	0.024 mm^2	3005 cycles	5.5 mW (12.2 mW)[a]	47 nJ/FFT (105 nJ/FFT)[a]
Buffered FFT HWPE	4 $C16$ / 2 $C32$ / 1 $C64$	22 nm	0.65 V	350 MHz	0.093 mm^2	3363 cycles	12.6 mW (13.2 mW)[a]	121 nJ/FFT (127 nJ/FFT)[a]
Wang *et al.* [10]	$C48$ mixed-radix	16 nm	0.57 V	320 MHz[c]	0.37 mm^2	1905 cycles[b]	22.6 mW[c]	135 nJ/FFT[b]
Chen *et al.* [11]	4 FP $C64$	45 nm	0.9 V	1 GHz	2.4 mm^2	1380 cycles	91.3 mW	126 nJ/FFT
Xia *et al.* [17]	$C32$ mixed-radix	55 nm	1.08 V	122.88 MHz	1.063 mm^2	972 cycles[b]	26.3 mW[b]	208 nJ/FFT[b]
Guo *et al.* [12][d]	4 FP $C64$	65 nm	1 V	500 MHz	4.6 mm^2	1403 cycles	172.3 mW	483 nJ/FFT
Cluster w/ Buffer-less HWPE	4 $C16$ / 2 $C32$ / 1 $C64$	22 nm	0.65 V	350 MHz	0.44 mm^2	3005 cycles	18.5 mW	159 nJ/FFT
Cluster w/ Buffered HWPE	4 $C16$ / 2 $C32$ / 1 $C64$	22 nm	0.65 V	350 MHz	0.51 mm^2	3363 cycles	19.5 mW	187 nJ/FFT
TI HWAFFT [14]	1 $C32$	not available	1.3 V	100 MHz	not available	7315 cycles	38.5 mW	2820.6 nJ/FFT

[a] Values between parenthesis take into account both the HWPE and TCDM contributions.
[b] Results for 972-point FFTs (the authors do not report results for 1024-point FFTs).
[c] Wang *et al.* report a range of values. The highest result was considered for frequency and power.
[d] The results for Guo *et al.*'s accelerator [12] are taken from Chen *et al.*'s comparison table [11].

Our solutions are around $7\times$ (buffered) and $8\times$ (buffer-less) faster than the optimized software, while adding 22% of area to the PULP cluster in the buffered case and only 6% in the buffer-less case. The buffer-less architecture is then $4.25\times$ more area-efficient than the buffered one.

D. Power

We placed and routed the PULP cluster containing the buffer-less/buffered FFT HWPE using Cadence Innovus. We simulated FFT computations with Mentor Questasim, back-annotating the switching activity data. Then, we extracted the related average power consumption with Synopsys PrimeTime under typical conditions (TT, $0.65\,\mathrm{V}$, $25\,^{\circ}\mathrm{C}$), at $350\,\mathrm{MHz}$, the maximum frequency the designs could reach in this corner. Since the critical block for timing is the butterfly unit, the buffer-less and buffered accelerator achieve the same maximum frequency. The buffer-less accelerator consumes on average $4.1/5.5/6.8\,\mathrm{mW}$ at $350\,\mathrm{MHz}$, respectively for $C16/C32/C64$ $2048/1024/512$ FFT at full throughput; while the buffered consumes around 9% more power than the buffer-less one, even including the TCDM read/write power contribution. This is because the multi-ported structure of the buffer banks nullifies potential energy improvements related to the use of an internal buffer. For lower-precision FFTs, the buffer-less HWPE consumes less power mainly due to the final reordering, which allows using one-fourth, half, or the whole output bandwidth for $C16/C32/C64$ data. For higher-precision FFTs, the accelerator reaches a higher butterfly utilization, increasing its average power consumption.

E. SoA Comparison

We compared our solutions with state-of-the-art accelerators in Table II. Our performance, power, and energy results refer to a $C32$ application. First, we compared our HWPEs with four stand-alone FFT accelerators with internal buffers proposed by Wang *et al.* [10], Chen *et al.* [11], Xia *et al.* [17], and Guo *et al.* [12]. Chen *et al.* and Guo *et al.* include four $C64$ floating-point butterfly engines. Chen *et al.* reach an energy efficiency comparable to our buffered solution; while our buffer-less accelerator is 20% more energy-efficient than their implementation, and $4.6\times$ more efficient than Guo *et al.*'s implementation. Wang *et al.* and Xia *et al.* achieve higher performance than our HWPEs thanks to their mixed-radix butterfly engines. Since the authors did not report the results for 1024-point FFTs, we considered their results for 972-point FFTs. Wang *et al.* target $C48$ data, while Xia *et al.* use $C32$ values. Our buffer-less HWPE is $2\times$ and 28.5% more energy-efficient than Xia *et al.* and Wang *et al.*, respectively. Note that the energy efficiency gap would widen when considering the cost of copy-in and copy-out of data needed for the accelerators including internal buffers. The buffer-less HWPE provides the smallest-area solution, although part of the extra area needed for the considered state-of-the-art accelerators is dedicated to the larger floating-point or mixed-radix butterfly engines.

Finally, we compared the results obtained for the PULP clusters enhanced with the HWPEs with TI HWAFFT [14] since it reports energy efficiency numbers related to a full system. TI HWAFFT can only compute $C32$ FFT and contains one butterfly unit. While our HWPE also supports higher and lower precisions, addressing a wider range of applications. The PULP cluster enhanced with our buffer-less HWPE is $2.4\times$ faster for a 1024-point $C32$ FFT than TI HWAFFT, and around $18\times$ more energy-efficient. Since Texas Instruments reports the accelerator performance also when computing FFTs without the bit-reversed reordering, we were able to identify in the final reordering the responsible for our $2.4\times$ speed-up when computing a complete FFT. Without bit-reversed reordering, our HWPE is $2\times$ faster due to a double number of $C32$ butterfly engines. A 1024-point $C32$ FFT without reordering takes 5244 cycles on TI HWAFFT, while 2626 cycles on our buffer-less HWPE.

V. CONCLUSION

We presented a comparison between buffered and buffer-less multicore cluster accelerators, using FFT HWPEs as a use case, and introduced a new reordering scheme that allows conflict-free memory accesses also for sub-word FFTs. Our accelerators can work with complex data with $8/16/32$-bit

real and imaginary parts, addressing a wide range of applications. While being around $8\times$ faster than an optimized FFT software implementation running on the 8 cluster cores, the buffer-less implementation occupies only $0.26\times$ the area of the buffered HWPE and adds only 6% of area overhead to a PULP cluster. Furthermore, since the internal buffer size contributes to the constraint on the number of maximum FFT points, increasing the limit on the FFT length would be highly costly for the buffered implementation, while it can be achieved at a low area overhead for the buffer-less accelerator. The buffer-less HWPE consumes around $4.1/5.5/6.8\,\mathrm{mW}$ at $350\,\mathrm{MHz}$ when computing $2048/1024/512$-point FFTs. Also energy-wise, the buffer-less accelerator dominates the buffered accelerator, thanks to the lower execution time and the slightly lower power consumption. Finally, we compared our buffer-less implementation with state-of-the-art accelerators, showing energy efficiency improvements.

VI. Acknowledgment

This work was supported in part by the Croatian-Swiss Research Programme, project #180625, "Heterogeneous Computing Systems with Customized Accelerators" by the Swiss National Science Foundation.

References

[1] W. J. Dally, Y. Turakhia, and S. Han, "Domain-specific hardware accelerators," *Communications of the ACM*, vol. 63, no. 7, pp. 48–57, 2020.

[2] C. E. Leiserson, N. C. Thompson, J. S. Emer, B. C. Kuszmaul, B. W. Lampson, D. Sanchez, and T. B. Schardl, "There's plenty of room at the top: What will drive computer performance after moore's law?" *Science*, vol. 368, no. 6495, 2020.

[3] J. Shalf, "The future of computing beyond moore's law," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2166, p. 20190061, 2020.

[4] M. B. Taylor, "Is dark silicon useful? harnessing the four horsemen of the coming dark silicon apocalypse," in *DAC Design Automation Conference 2012*. IEEE, 2012, pp. 1131–1136.

[5] A. M. Rahmani, P. Liljeberg, A. Hemani, A. Jantsch, and H. Tenhunen, *The Dark Side of Silicon*. Springer, 2016.

[6] D. N. Rockmore, "The fft: an algorithm the whole family can use," *Computing in Science & Engineering*, vol. 2, no. 1, pp. 60–64, 2000.

[7] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[8] F. Conti, C. Pilkington, A. Marongiu, and L. Benini, "He-p2012: Architectural heterogeneity exploration on a scalable many-core platform," in *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*. IEEE, 2014, pp. 114–120.

[9] F. Conti, P. D. Schiavone, and L. Benini, "XNOR neural engine: A hardware accelerator IP for 21.6-fJ/op binary neural network inference," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2940–2951, 2018.

[10] A. Wang, B. Richards, P. Dabbelt, H. Mao, S. Bailey, J. Han, E. Chang, J. Dunn, E. Alon, and B. Nikolić, "A 0.37 mm 2 LTE/Wi-Fi compatible, memory-based, runtime-reconfigurable 2 n 3 m 5 k FFT accelerator integrated with a RISC-V core in 16nm FinFET," in *2017 IEEE Asian Solid-State Circuits Conference (A-SSCC)*. IEEE, 2017, pp. 305–308.

[11] X. Chen, Y. Lei, Z. Lu, and S. Chen, "A variable-size FFT hardware accelerator based on matrix transposition," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 10, pp. 1953–1966, 2018.

[12] L. Guo, Y. Tang, Y. Lei, Y. Dou, and J. Zhou, "Transpose-free variable-size fft accelerator based on-chip sram," *IEICE Electronics Express*, pp. 11–20 140 171, 2014.

[13] B. M. Baas, "A low-power, high-performance, 1024-point FFT processor," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 3, pp. 380–387, 1999.

[14] M. McKeown, "FFT implementation on the TMS320VC5505, TMS320C5505, and TMS320C5515 DSPs," *Texas Instruments Incorporated, White Paper SPRABB6B*, 2010.

[15] D. Cohen, "Simplified control of FFT hardware," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 24, no. 6, pp. 577–579, 1976.

[16] Y. Ma and L. Wanhammar, "A hardware efficient control of memory addressing for high-performance FFT processors," *IEEE transactions on signal processing*, vol. 48, no. 3, pp. 917–921, 2000.

[17] K.-F. Xia, B. Wu, T. Xiong, and T.-C. Ye, "A memory-based fft processor design with generalized efficient conflict-free address schemes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 6, pp. 1919–1929, 2017.

[18] M. Garrido, K. Möller, and M. Kumm, "World's fastest fft architectures: Breaking the barrier of 100 gs/s," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 4, pp. 1507–1516, 2018.

[19] "MAPLE Hardware Accelerator," https://www.nxp.com/files-static/training_pdf/vFTF09_AN149.pdf, NXP Semiconductors N.V., Jul. 2009.

[20] "Advanced driver assistance subsystem (ADAS)," https://www.infineon.com/dgdl/Infineon-AURIX_Advanced_Driver_Assistance_Subsystem-Training-v01_00-EN.pdf?fileId=5546d4626e651a41016e6522e0600003, Infineon Technologies AG, Jun. 2019.

[21] L. Johnson, "Conflict free memory addressing for dedicated FFT hardware," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 39, no. 5, pp. 312–316, 1992.

[22] J. Baek and K. Choi, "New address generation scheme for memory-based fft processor using multiple radix-2 butterflies," in *2008 International SoC Design Conference*, vol. 1. IEEE, 2008, pp. I–273.

[23] H. Sorokin and J. Takala, "Conflict-free parallel access scheme for mixed-radix fft supporting i/o permutations," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2011, pp. 1709–1712.

[24] Y. Bo, J. Han, Y. Zou, and X. Zeng, "A low power asip for precision configurable fft processing," in *Proceedings of The 2012 Asia Pacific Signal and Information Processing Association Annual Summit and Conference*. IEEE, 2012, pp. 1–4.

[25] F. Conti, D. Rossi, A. Pullini, I. Loi, and L. Benini, "PULP: A ultra-low power parallel accelerator for energy-efficient and flexible embedded vision," *Journal of Signal Processing Systems*, vol. 84, no. 3, pp. 339–354, 2016.

[26] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, 2017.

[27] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[28] P. Welch, "A fixed-point fast fourier transform error analysis," *IEEE Transactions on Audio and Electroacoustics*, vol. 17, no. 2, pp. 151–157, 1969.

[29] A. Teman, D. Rossi, P. Meinerzhagen, L. Benini, and A. Burg, "Power, area, and performance optimization of standard cell memory arrays through controlled placement," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 21, no. 4, pp. 1–25, 2016.