VWR2A: A Very-Wide-Register Reconfigurable-Array Architecture for Low-Power Embedded Devices

Benoît W. Denkinger
EPFL, Switzerland

Miguel Peón-Quirós
EPFL, Switzerland

Mario Konijnenburg
IMEC, Netherland

David Atienza
EPFL, Switzerland

Francky Catthoor
IMEC, and KU Leuven, Belgium

ABSTRACT

Edge-computing requires high-performance energy-efficient embedded systems. Fixed-function or custom accelerators, such as FFT or FIR filter engines, are very efficient at implementing a particular functionality for a given set of constraints. However, they are inflexible when facing application-wide optimizations or functionality upgrades. Conversely, programmable cores offer higher flexibility, but often with a penalty in area, performance, and, above all, energy consumption. In this paper, we propose VWR2A, an architecture that integrates high computational density and low power memory structures (i.e., very-wide registers and scratchpad memories). VWR2A narrows the energy gap with similar or better performance on FFT kernels with respect to an FFT accelerator. Moreover, VWR2A flexibility allows to accelerate multiple kernels, resulting in significant energy savings at the application level.

KEYWORDS

programmable cores, accelerators, CGRA, reconfigurable architecture, low power, embedded systems

ACM Reference Format:
Benoît W. Denkinger, Miguel Peón-Quirós, Mario Konijnenburg, David Atienza, and Francky Catthoor. 2022. VWR2A: A Very-Wide-Register Reconfigurable-Array Architecture for Low-Power Embedded Devices. In *Proceedings of Design Automation Conference '22 (DAC '22).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Moving the computational load towards the edge reduces communication energy but requires high-performance energy-efficient embedded devices. Hardware architecture exploration for such devices is an active area of research, as we still need to obtain higher energy efficiencies to enable long-lasting operation between battery recharges. The inclusion of hardware accelerators in the computing platform has become a standard in recent years. These accelerators can execute repetitive operations that account for a significant amount of the processing time (i.e., computational kernels) in a more efficient way than a general purpose CPU or generic GPU.

The development of accelerators follows two main trends: custom accelerators, or ASIC, and flexible (programmable) cores, usually considered as DSIP [8]. Fixed-function accelerators are often the most efficient way of implementing a particular functionality for a given set of constraints. They are, in general, not programmable and are thus focused on a single task or a small family of related tasks. One example of a custom accelerator is the FFT accelerator included in the MUSEIC platform [12], which can execute FFTs of different sizes much more efficiently than the platform ARM Cortex-M4 core. However, it is possible to build programmable cores tailored specifically for algorithms from a given application domain (i.e., DSIP) without becoming general purpose processors. Traditional knowledge says that these flexible cores are less efficient than fixed-function ones and that the latter should be preferred to improve the energy efficiency of the systems.

In this paper, we show how to move in the flexibility-performance trade-off and build flexible domain-specific cores that close the efficiency gap in terms of energy and performance with respect to custom ones for individual computation kernels. Moreover, as these cores still have an instruction-set which covers a broad domain, they hence can be used in more parts of the application. As consequence, they achieve larger improvements when the complete application — rather than individual kernels — is taken into account.

We started from the basis of a coarse-grained reconfigurable array (CGRA) [7], which offers a high computation density versus control logic, and improved it with a memory architecture based on very-wide registers (VWRs) and wide scratchpad memories (SPMs). To demonstrate the benefits of our architecture, we considered a biosignal-processing embedded system meant for wearable devices [12]. Then, we evaluated two typical kernels used in biosignal applications, FFT and FIR filters, and a biosignal application.

The rest of this paper is organized as follows. First, we discuss related architectures for low-power computing in Section 2. We then present our proposed architecture for ultra-low power programmable accelerators in Section 3. In Section 4, we describe the experimental setup used to evaluate our proposals, whereas in Section 5 we analyze the results obtained. Finally, in Section 6 we draw the main conclusions of our study.

2 RELATED WORK

CGRAs have often been used for DSIP architectures as they provide good flexibility with reasonable energy overhead compared to ASICs. One example is MorphoSys [11], a system-on-chip composed

This work was supported in part by the Swiss NSF ML-Edge Project under Grant Agreement (GA) no. 200020_182009, in part by the ReSoRT Project funded by Botnar Foundation under GA no. REG-19-019, in part by the ERC Consolidator Grant COMPUSAPIEN under GA no. 725657, and in part by a joint research grant for ESL-EPFL by IMEC.

DAC '22, July 10–14, 2022, San Francisco, CA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

of a TinyRISC processor and a CGRA. The processor controls the execution flow of the code while the CGRA executes the most computationally intensive kernels. The concept of VWR2A is similar to MorphoSys in that it runs besides a low-performance processor. The goal is to reduce the overall system energy to the minimum.

However, according to the authors of [9], such architecture is not optimal because the control-intensive code left to the processor significantly reduces the speed-up benefit of the CGRA. To address this issue, they proposed the ADRES framework [9]: an architectural template composed of a VLIW processor and a CGRA. The VLIW allows efficient execution of control-intensive code, and the CGRA still accelerates the computation-intensive kernels. It is focused on increasing the performance, but not on the low energy consumption. Whereas in VWR2A, we propose to improve the typical CGRA architecture to execute the intensive-control code. This allows complete applications to be executed on VWR2A, removing the need for an advanced general-purpose processor like the VLIW of the ADRES platform, thus reducing the overall system energy consumption.

VWR2A integrates low-power memory structures. The first one is a dedicated SPM tuned for our biosignal kernel target. VWR2A is meant to be integrated inside a platform and needs to have access to the system memory, usually through the system bus. Therefore, the performance of algorithms with many data accesses is dependent on the system bus latency and bandwidth, which can negatively impact the overall performance. Moreover, data access through the system bus is costly in energy and can be reduced with a judicious memory hierarchy design. A typical approach is to use caches, like in MorphoSys, but they incur a significant energy penalty due to their inherent control overhead. SPMs offer similar performance at lower energy as the control is moved to the software side [1].

Second, we propose the use of VWRs in replacement of the traditional register file for data. VWRs have been introduced in the FEENECS template of [10] and [2] as a better alternative, in terms of energy, to standard multi-ported register files. The first reason is that the cells of the VWRs are single-ported, while those of the register files are multi-ported. Second, their wide interface still allows multiple words to be loaded at once, which leads to a lower overall energy per word access than traditional register files. To fully benefit from the VWRs, the background memory — the SPM in our case — needs to match the VWR width, allowing to fill the VWR in one cycle. This effectively makes the VWR interface asymmetric with respect to the SPM side and the datapath side. The authors of [10] and [2] also show that such design is better for place and route, as both memories (i.e., the SPM and the VWR) can be aligned, and the wire length of the most active connection is reduced to a minimum. Indeed, only the outputs of the multiplexers switch in every cycle, not the outputs of the VWRs themselves, reducing the energy consumption.

Several previous works on CGRAs focus on the interconnection scheme of the reconfigurable cells (RCs), for example, to provide communication with distant neighbors. This paper does not consider advanced interconnections between the RCs because they represent a significant energy overhead. Limiting the interconnection to the neighbor cells reduces the wire length, which is essential for an ultra-low power architecture using scaled technologies, without significantly impacting performance.

In this paper, we focus on the novel architectural features of VWR2A. We integrated it into a specific platform to demonstrate its performance, but it could be integrated into any platform. CGRA compilers are also an active area of research, but we did not adapt any existing solution to our specific architecture. We have currently mapped the code manually on VWR2A.

3 ARCHITECTURE FOR ULTRA-LOW POWER PROGRAMMABLE CORES

Fig. 1 shows the VWR2A architecture diagram. We used the CGRA architecture as a basis as it offers a high computation density versus control logic. The new features and design choices proposed in this paper have been driven by two criteria: extended application code coverage and low power design.

3.1 Reconfigurable array

VWR2A contains a 4x2 array of reconfigurable cells. To reduce control overhead, the RCs are grouped in two columns, where all the RCs of a column are synchronized through a shared program counter (PC). The columns are independent, allowing two kernels to run in parallel. Each RC has a program memory of 64 words, which is enough to execute most kernels. The configuration words are stored in the configuration memory and loaded to the RCs' local program memory when a kernel execution starts. The CGRA architecture offers a high computation density because the bits of the configuration words (i.e., "instructions") correspond directly to the control signals in the cell datapaths, without an actual decoding process. There is evident parallelism between this architecture, where the RCs of a column share a program counter, and a VLIW in which all the execution slots are equivalent. Indeed, the instructions of the different RCs can be fused and considered as a wide (predecoded) instruction word.

Each RC of the reconfigurable array contains a small register file (two 32-bit entries) and a 32-bit ALU that can execute typical operations: signed addition, subtraction and multiplication, logical bitwise operations, and logical/arithmetic bit shift. All operations happen in one clock cycle. The multiplier has two working modes: a standard mode, where the lowest 32 bits are kept, and a fixed-point mode, where the lower 16 bits are discarded, and the next 32 bits are kept. This enables single-cycle fixed-point multiplication in 16.15 format and good performance for algorithms that require decimal representation, such as the FFT. The ALU operands have multiple sources: the VWRs, the SRF (see Sec. 3.2), the RC local register file, and the previous-cycle results of neighboring RCs. All the operators in the ALUs implement operand isolation [5] to minimize energy consumption.

3.2 Ultra-low energy memory organisation

VWR2A contains a dedicated 32 KiB SPM shared by all the columns. The SPM has a double interface: on the system side, it has the system bus width. On the accelerator side, it has the same width as the VWRs. A DMA performs the data transfers between the SPM and the system memory, while the LSU moves the data between the SPM and the VWRs. The VWRs act as a buffer between the SPM and the RCs, which can access the elements in the wide registers word by word. The RCs and the VWRs are connected through a

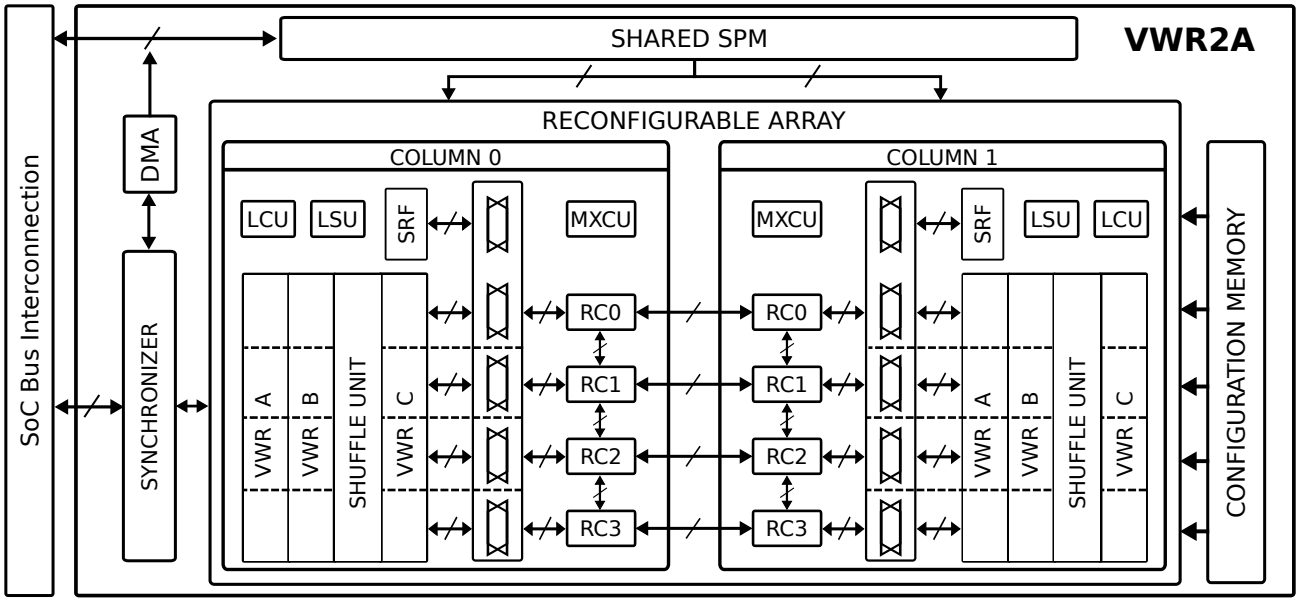

Figure 1: VWR2A architecture diagram

network of multiplexers, allowing multiple RCs to work in parallel on different sections of a VWR.

The size of the VWRs depends on the target domain, biosignal kernels in our case, and the datapath width consuming the data words. On the one hand, the VWRs need to be large enough to minimize the frequency at which new data have to be loaded. On the other hand, wider VWRs have higher leakage. In VWRA2, the VWRs and the SPM have a 4096 bitwidth allowing 128 words of 32 bits to be transferred in a single cycle. This width is a good tradeoff considering that the four RCs can access one-fourth of a VWR (32 words). To be single-ported, multiple VWRs are needed to store the different operands. After testing different implementations, we found out that 3 VWRs represent a good compromise between performance and energy efficiency.

The scalar register file (SRF) has 8 32-bit entries used for scalar values that are kernel-dependent, such as addresses for the SPM, masking values for the VWRs index computation, or loop parameters for the kernel execution control. The SRF is single-ported, allowing one access at a time from the different units (RCs, LSU, MXCU, and LCU).

3.3 Specialized slots

We borrow from the VLIW architecture the concept of specialized slots by introducing an LSU, an LCU, and an MXCU on top of the RCs of each column, as shown in Fig. 1. They all have their own instruction stream (see Table 1), synchronized with the RCs in the column via the common PC.

3.3.1 LSU: Load-Store Unit. Controls the data transfers between the SPM and the VWRs or the SRF. The data arrays are allocated to VWRs while scalar values (e.g., loop size) are stored in the SRF. The LSU also controls the shuffle unit that is an essential component of VWR2A. As each RC is accessing only one-fourth of a VWR, data reordering is needed to move the data inside the full VWR. Such reordering is possible through the RCs connection matrix, but it is highly inefficient in terms of performance and energy, whereas the shuffle unit enables fast and energy-efficient data reordering. It takes as input the data contained in the VWRs A and B, applies a hardcoded shuffle operation on the data, and stores the result in the VWR C (Fig. 1). The available shuffle operations are:

- *Words interleaving*: VWR A and B words are interleaved. The result is twice the size of a VWR, and the upper or lower half can be selected as the output.
- *Even or odd index pruning*: prunes the even/odd elements of VWRs A and B, and outputs the remaining elements (of both A and B).
- *Bit-reversal*: applies bit-reversal shuffling to the concatenation of VWRs A and B. The result is twice the size of a VWR, and the upper or lower half can be selected as the output.
- *Circular shift*: the concatenation of VWRs A and B is shifted by 32 words up in a circular manner (i.e., the upper 32 words are moved to the lower 32 words). The result is twice the size of a VWR, and the upper or lower half can be selected as the output.

3.3.2 MXCU: MultipleXer-Control Unit. Controls the multiplexers that connect the VWRs outputs to the RCs. Each RC has access to 1/4 of the VWRs width. To limit the number of control bits, all the RCs access the same address of their slice. This address is also used to write the data back to any of the VWRs. Although this structure adds some constraints to the kernel mapping, they can be solved with careful data placement and proper use of the shuffling unit.

3.3.3 LCU: Loop-Control Unit. Generates the branches and jumps for the program counter and notifies the synchronizer at the end of a kernel. It increases the code coverage by allowing the execution of loops with any nest depth and control-intensive code to be

Table 1: VWR2A instruction flow example

PC	LCU	LSU	MXCU	RC0-3
...				
3	NOP	LOAD A	k=0	NOP
4	i=0	LOAD B	NOP	NOP
5	i++	NOP	NOP	VWRC=VWRA+VWRB
6	BLT PC=5	NOP	k++	VWRA=VWRA-VWRB
...				
14	j++	STORE A	NOP	NOP
15	NOP	STORE C	NOP	NOP
16	BLT PC=3	NOP	NOP	NOP
17	EXIT	NOP	NOP	NOP

efficiently executed on VWR2A. When multiple columns work in parallel, their respective PCs are synchronized.

3.4 Kernel mapping

The FFT kernel mapping is discussed here to illustrate the use of the architecture. This kernel uses the common in-place radix-2 FFT algorithm [4], which reduces the computation complexity from $O(N^2)$ to $O(N \log N)$, where N is the number of points. The radix-2 algorithm divides the computation into k stages, where $2^k = N$. All the stages execute the same flow of operations; the only changes are the coefficients and the data ordering. The shuffle unit applies the "*words interleaving*" shuffling to create the correct data layout for the next stage. Both columns are used to execute the FFT kernel. The output of the kernel is in bit-reversed order, and the shuffle unit is again used to reorder the data.

An optimized version is used for real-valued FFTs (i.e., the imaginary part is null). The sequence of N real values is transformed into an $N/2$ complex sequence. Then, the complex FFT kernel presented above is used. This technique reduces the computations of the FFT kernel but requires some additional operations, also executed on VWR2A, to recover the correct output. The overall gain is approximately a factor of 2 compared to a complex FFT of size N where the imaginary part is zero.

These steps are mapped into instructions for the different units (i.e., LCU, LSU, MXCU, and the RCs). Table 1 shows a sample of the instruction flow of the different units for the FFT kernel, where k corresponds to the VWRs address that is accessed by the RCs. The RCs' instructions are grouped for simplicity, but they can all execute a different operation.

4 EXPERIMENTAL SETUP

4.1 Biosignal processing ultra-low power embedded platform

To demonstrate the advantages of our architecture, we have integrated VWR2A in an ultra-low power SoC intended for biomedical signal acquisition and processing [12]. This platform features an ARM Cortex M4F processor, 192 KiB of static random access memory (SRAM) (divided into six banks that can be individually power gated), an analog front end for the acquisition of biosignals (e.g., electrocardiogram (ECG), photoplethysmogram (PPG)), a DMA, and several fixed-function accelerators. The FFT accelerator is used for comparison with our VWR2A because this platform is also oriented to biosignal processing. It computes FFTs and inverse FFTs up to 4096 points, with an optimized flow for real-valued inputs. The FFT weights are stored in internal ROMs, whereas a dual-port memory is used to store the data. To avoid overflow, this custom FFT accelerator uses an internal representation of 18 bits with dynamic scaling. The SoC elements (e.g., accelerators, memories, processor) are connected through the AMBA-AHB bus interface. The SoC has multiple power domains that can be turned on and off during execution to optimize energy consumption further.

4.2 Integration of our programmable core

We connected our VWR2A to the AMBA-AHB bus interface, precisely like the other hardware accelerators, to have a fair comparison within the original SoC design. VWR2A has one master port, controlled by its DMA, to transfer data between the SoC SRAM and the VWR2A SPM. The kernel acceleration and DMA transfer requests from the CPU are received through an additional slave port. VWR2A informs the processor when a kernel execution, or a DMA transfer, is finished through an interrupt line. VWR2A is included in the same power domain as the other accelerators and can therefore be power gated.

4.3 Performance and energy characterization

We synthesized the complete SoC, including VWR2A, with the TSMC 40 nm LP CMOS technology at 80 MHz (the original SoC frequency) and ran post-synthesis simulations to compare the performance and the energy of both implementations (i.e., custom accelerator versus programmable core). This allows cycle accurate simulations from which we extract the cells switching activity that we use for power estimation with Synopsys PrimePower [13].

4.4 Software benchmarks

4.4.1 Standalone kernels. We first compared our VWR2A with the SoC FFT accelerator using a standalone FFT kernel, which is a typical kernel used in biomedical applications. We implemented different FFT sizes for both complex and real-valued sequences. The SoC FFT accelerator uses a mixed radix-2 and radix-4 implementation. For our VWR2A, we used the radix-2 algorithm presented in Section 3.4. The second kernel is a FIR filter with 11 taps. We used three different input sizes to compare our VWR2A with the CPU. Our mapping uses two columns of the reconfigurable array that work on different slices of the input array.

4.4.2 Biosignal application. To study the impact at the application level of our programmable architecture, we considered the MBioTracker application, which measures cognitive workload [6]. This application is divided into four steps: preprocessing, delineation, features extraction, and prediction. First, the preprocessing applies a FIR filter over the raw input data. Second, the delineation detects the maximums and minimums of the filtered signal to extract inspiration and expiration times. Third, these values are used for extraction of time features (mean, median, and RMS values), while the FFT of the filtered signal is employed for frequency features extraction. Finally, the cognitive workload is estimated using an SVM algorithm.

Table 2: FFT kernel performance comparison for various sizes

	CPU	FFT ACCEL		VWR2A	
Complex-valued	cycles	cycles	speed-up	cycles	speed-up
512	47 926	7099	6.8×	7125	6.7×
1024	84 753	13 629	6.2×	12 405	6.8×
2048	219 667	31 299	7.0×	30 217	7.3×
Real-valued					
512	24 927	3523	7.1×	3666	6.8×
1024	62 326	8007	7.8×	7133	8.7×
2048	113 489	16 490	7.4×	14 427	7.9×

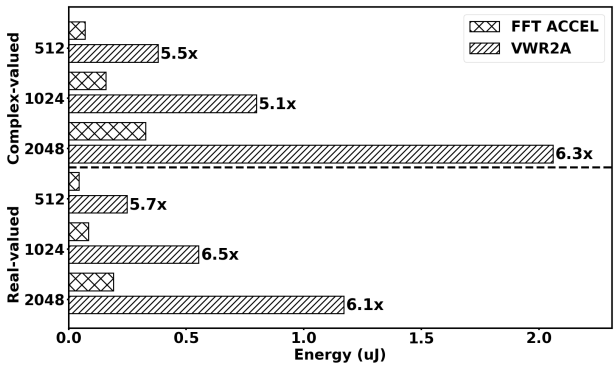

Figure 2: FFT kernel energy comparison for various sizes. Even if the performance of VWR2A is equivalent to that of the custom FFT accelerator, as expected, the gap in energy consumption is still significant in the case of isolated kernels.

5 EXPERIMENTAL RESULTS

5.1 Performance on standalone kernels

5.1.1 FFT kernel. The performance and energy consumption results for various complex and real-valued FFT sizes are reported in Table 2 and Fig. 2, respectively. These results show that both the FFT accelerator (FFT ACCEL) and VWR2A have similar performance and are 7.4× faster than the ARM Cortex-M4 (CPU) on average. VWR2A is less performant for small FFT sizes because the programming of the DMA transfers and the kernel parameters has a slightly larger overhead than the FFT accelerator programming. As expected, Fig. 2 shows that the FFT accelerator is more energy-efficient than VWR2A when the specific kernel it was designed for is considered in isolation. Nevertheless, our goal was to narrow as much as possible the energy gap between both implementations.

The FFT accelerator uses a mixed radix-2 and radix-4 implementation that depends on the actual FFT size, resulting in different performance and power consumption, while the VWR2A mapping is identical for all FFT sizes. This explains the variation of the energy consumption ratio in Fig. 2. Finally, this figure only considers the accelerator energy consumption. If the complete SoC is considered, the energy difference is between a factor 4× to 5×. In any case, compared to an FFT using only the Cortex-M4 processor and the CMSIS-DSP library with 16-bit data in q15 format, both the FFT accelerator and our architecture produce energy savings, of 86.0 % and 40.8 %, respectively.

Table 3 presents the power consumption breakdown per subcomponent, i.e., for the FFT accelerator and VWR2A. The main contributors in our architecture are the memories and the datapath (i.e., the RCs). This means that the overhead of the instruction control, which is non-negligible in typical instruction-set processors [2], is removed in VWR2A. The *Memories* category contains the VWR2A SPM (32 KiB) and the VWRs (3 KiB), accounting for 46 % and 54 % of the total power, respectively. In contrast, the FFT accelerator has 17 KiB of memory in total. To build the SPM wide interface, smaller memory macros of the width supplied by the technology provider were concatenated. The VWRs were built using latches of the standard cell library. A custom design for these memories will undoubtedly reduce power consumption. Regarding the *Datapath* category, the available optimizations are more limited because the FFT accelerator is specialized for FFTs, with an 18-bit-wide datapath, while our CGRA has a more general-purpose 32-bit ALU. One solution could be to have a 16-bit mode with two simultaneous 16-bit operations instead of one 32-bit operation.

Table 3: FFT accelerator and VWR2A power breakdown while executing a 512-point real-valued FFT

	FFT ACCEL		VWR2A		
Instance	Power (mW)	%	Power (mW)	%	ratio
DMA	1.07×10^{-2}	1 %	9.47×10^{-2}	2 %	8.9
Memories	6.68×10^{-1}	68 %	3.49×10^{0}	64 %	5.2
Control	6.25×10^{-2}	6 %	1.00×10^{-1}	2 %	1.6
Datapath	2.42×10^{-1}	25 %	1.72×10^{0}	32 %	7.1
Total	9.83×10^{-1}	100 %	5.41	100 %	5.5

Compared to the Ultra-Low Power Samsung Reconfigurable Processor (ULP-SRP) [3], a recent instantiation of the ADRES template that uses the TSMC 40 nm LP technology, VWR2A exhibits significant performance and energy gains. The authors reported an execution time of 839.1 µs and an energy consumption of 19.9 µJ for a 256-Point FFT,[1] while VWR2A executes that same kernel in 35.6 µs and consumes 0.3 µJ. These numbers correspond to a factor 23× improvement in performance and a factor of 66× in terms of energy. It is important to note that post-layout simulation has been done for the ULP-SRP, while we ran post-synthesis simulation, which can explain part of the significant difference in energy.

5.1.2 FIR filter kernel. Table 4 reports the experimental results for three different input sizes for the FIR filter with 11 taps. We compared the performance of the processor (CPU) with that of our VWR2A. The processor uses the CMSIS-DSP library with 16-bit data (q15 format), while our solution uses 32-bit data. Table 4 shows that our accelerator is on average 14.9 times faster and consumes 71.3 % less energy than the processor.

[1]The authors do not specify if they implement a complex-valued or a real-valued FFT. We considered a 256-Point complex-valued FFT, corresponding to the worst case for VWR2A.

Table 4: FIR filter kernel performance and energy comparison for different numbers of points and 11 taps

	CPU		VWR2A		GAIN	
	Cycles	Energy (µJ)	Cycles	Energy (µJ)	Cycles speed-up	Energy savings
256 pts	24 747	0.37	1849	0.11	13.4	69.9 %
512 pts	49 253	0.73	3260	0.21	15.1	71.7 %
1024 pts	98 283	1.45	6091	0.40	16.1	72.4 %

5.2 Performance on biosignal application

Table 5 reports performance and energy consumption for the different steps of the application. The complete application has been ported on VWR2A and the processor only manages the high-level control of the application. This table supports the central claim of this paper and shows that larger savings can be obtained from a reconfigurable architecture with respect to a custom accelerator when complete applications — rather than individual kernels — are considered. The significant gains in performance and energy presented below are due to the parallel computing power of VWR2A (i.e., 8 RCs and the specialized slots) and its low power consumption.

5.2.1 Preprocessing. The version using our VWR2A is 13.2 times faster than the Cortex-M4 (CPU), which translates into energy savings of 64.7 %. The CPU+FFT ACCEL version is equivalent to the CPU version because no code can be accelerated by the FFT accelerator (which remains power-gated), hence showing the advantage of a programmable architecture. This shows the potential benefits that can be obtained by using a programmable architecture that can execute a larger proportion of the overall application.

5.2.2 Delineation. This step is a typical example of control-intensive code. The computation load is low but there are a lot of *if* conditions used to detect the valid minimums and maximums. General purpose CPUs are very inefficient at executing such code, while VWR2A can take advantage of its more powerful ILP capabilities. This translates into a 94.1 % gain in performance and 82.9 % savings in energy. As before, the CPU+FFT ACCEL version is equivalent to the CPU version.

5.2.3 Features extraction and SVM prediction. The FFT accelerator computes a real-valued 512-Point FFT, which translates to an 9.3 % gain in energy compared to the CPU version. However, the FFT represents only a portion of the application code, of which the custom accelerator cannot execute anything else. In contrast, VWR2A can execute all the code of the feature extraction step and the SVM prediction, with a corresponding energy saving of 56.0 % compared to the CPU version. This result is 6 × better than the CPU+FFT ACCEL version. VWR2A benefits from its large code coverage, limiting the data transfers between the system's main memory and its SPM. For example, the FFT uses the filtered data loaded into the SPM during the preprocessing step and keeps the results inside the SPM. It copies only the estimated state by the SVM back to the system's main memory, while the FFT accelerator has to copy back the 512 FFT output values.

Table 5: Biosignal application performance and energy comparison

	CPU	CPU + FFT ACCEL		CPU + VWR2A	
Cycles			savings		savings
Preprocessing	49 760	49 760	0.0 %	3763	92.4 %
Delineation	46 268	46 268	0.0 %	2723	94.1 %
Feat. extraction	70 639	54 255	23.2 %	8627	87.8 %
Total	166 667	150 283	9.8 %	15 113	90.9 %
Energy (µJ)					
Preprocessing	0.74	0.74	0.0 %	0.26	64.7 %
Delineation	0.74	0.74	0.0 %	0.13	82.9 %
Feat. extraction	1.1	0.98	9.3 %	0.47	56.0 %
Total	2.6	2.5	3.9 %	0.86	66.3 %

6 CONCLUSION

In this paper, we have shown the feasibility of using a domain-specific programmable core as an accelerator instead of a fixed-function accelerator and achieving similar, or better, performance. Regarding energy consumption, fixed-function accelerators typically perform better on specific tasks, but when complete applications are considered, our VWR2A has better energy consumption. The reason is that programmable architectures can accommodate application-specific optimizations that are not possible with custom accelerators. In addition, more code is also eligible for acceleration with a flexible architecture, which leads to better overall performance and energy efficiency, as long as the energy gap with respect to the fixed-function accelerator has been sufficiently reduced.

REFERENCES

[1] R. Banakar et al. 2002. Scratchpad memory: a design alternative for cache on-chip memory in embedded systems. In *Proc. of IEEE CODES*. 73–78.
[2] Francky Catthoor et al. 2010. *Ultra-Low Energy Domain-Specific Instruction-Set Processors* (1 ed.). Springer Netherlands, Chapter An asymmetrical register file: the VWR, 199–222.
[3] Kim Changmoo et al. 2014. ULP-SRP: Ultra Low-Power Samsung Reconfigurable Processor for Biomedical Applications. *ACM TRETS* 7, 22 (2014), 1–15. Issue 3.
[4] James W. Cooley and John W. Tukey. 1965. An Algorithm for the Machine Calculation of Complex Fourier Series. *Math. Comp.* 19, 90 (1965), 297–301.
[5] Anthony Correale. 1995. Overview of the Power Minimization Techniques Employed in the IBM PowerPC 4xx Embedded Controllers. In *ISLPED'95*. ACM, 75–80.
[6] Fabio Dell'Agnola et al. 2021. MBioTracker: Multimodal Self-Aware Bio-Monitoring Wearable System for Online Workload Detection. *IEEE TBioCAS* 15, 5 (2021), 994–1007.
[7] Loris Duch et al. 2017. HEAL-WEAR: An Ultra-Low Power Heterogeneous System for Bio-Signal Analysis. *IEEE TCAS-I* 64, 9 (Sept. 2017), 2448–2461.
[8] Robert Fasthuber et al. 2013. *Energy-efficient communication processors.* Springer, New York.
[9] B. Mei et al. 2005. Architecture exploration for a reconfigurable architecture template. *IEEE Design Test of Computers* 22, 2 (2005), 90–101.
[10] Praveen Raghavan et al. 2007. Very Wide Register: An Asymmetric Register File Organization for Low Power Embedded Processors. In *DATE*.
[11] H. Singh et al. 2000. MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications. *IEEE Trans. Comput.* 49, 5 (2000), 465–481.
[12] Shuang Song et al. 2019. A 769 μW Battery-Powered Single-Chip SoC With BLE for Multi-Modal Vital Sign Monitoring Health Patches. *IEEE TBioCAS* 13, 6 (2019), 1506–1517.
[13] Synopsys. Q-2019.12. PrimePower. https://www.synopsys.com