Acceleration of Control Intensive Applications on Coarse-Grained Reconfigurable Arrays for Embedded Systems

Benoît W. Denkinger, Miguel Peón-Quirós, Mario Konijnenburg, David Atienza, *Fellow, IEEE*, Francky Catthoor, *Fellow, IEEE*

Abstract—Embedded systems confront two opposite goals: low-power operation and high performance. The current trend to reach these goals is toward heterogeneous platforms, including multi-core architectures with heterogeneous cores and hardware accelerators. The latter can be divided into custom accelerators (e.g., ASICs) and programmable domain-specific cores (e.g., DSIPs). VWR2A [1] is a programmable architecture that integrates high computational density and low power memory structures. The flexibility of VWR2A allows a large portion of applications to be covered, resulting in better performance and energy efficiency than ASICs and general-purpose processors. However, while this has been well studied for data-intensive kernels, this is not the case for control-intensive kernels —code with complex if-else and nested loop structures. Traditionally, control-intensive code is left to be executed by the host processor. This situation unnecessarily restricts the potential impact of energy-efficient acceleration, especially at the application level.

In this paper, we evaluate the performance and energy consumption of VWR2A for control-intensive code and compare it with an ARM Cortex-M4 processor and a RISC-V Ibex processor. The performance and energy consumption are evaluated at the kernel and application levels. Our results confirm that VWR2A is faster and more energy-efficient than the two considered general-purpose processors also for control-intensive code.

Index Terms—programmable cores, CGRA, reconfigurable architecture, accelerators, low-power, embedded systems

1 Introduction

By 2030, 24 billion IoT devices are expected to be used worldwide [2]. From industry to healthcare, many domains could benefit from their promise of better efficiency and lower costs. However, existing platforms have not yet achieved long-lasting battery operation and often require a daily charge. Shifting the computational load from the cloud towards the edge improves the energy efficiency of embedded devices by reducing communication energy but increases their performance requirements.

The recent trend in research is towards heterogeneous platforms containing multi-core architectures with heterogeneous processors and hardware accelerators. These systems can adapt their performance and power consumption based on their current operating state (e.g., their workload and battery load), enabling high-performance and energy-efficient execution. In this context, hardware accelerators have become a standard in state-of-the-art platforms [3], [4], [5], because of their better efficiency at executing repetitive operations compared to a general-purpose processor (GPP).

The flexibility-performance trade-off of hardware accelerator design has led to two categories of accelerators:

a) domain-specific instruction-set processors (DSIPs) or programmable cores (e.g., coarse-grained reconfigurable arrays (CGRAs) [4])
b) application-specific integrated circuits (ASICs) or custom accelerators (e.g., FFT [3], CNN [6])

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 [3], which can execute FFTs of different sizes much more efficiently than the platform ARM Cortex-M4 core. However, it is possible to build programmable architectures tailored specifically for algorithms from a given application domain (i.e., DSIP) without becoming GPPs. VWR2A [1] is an example of such an architecture. It is optimized for the biomedical domain and can outperform custom accelerators when complete applications are considered because programmable architectures can execute a broader range of tasks. The authors of [1] focused on evaluating the performance and energy trade-offs for data-intensive kernels, like most of the literature, but not for control-intensive ones. Being able to execute control-intensive code efficiently would strengthen even

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.

- *Benoît W. Denkinger, Miguel Peón-Quirós, and David Atienza are with the EPFL, Lausanne, Switzerland.*
- *Mario Konijnenburg is with imec, Netherlands.*
- *Francky Catthoor is with imec, and KU Leuven, Belgium.*

more the advantage of a DSIP design compared to custom accelerators and GPPs because it broadens its applicability, reducing the proportion of code executed on GPPs.

Control-intensive kernels contain a significant number of operations related to control (e.g., loop, if-else structure) compared to arithmetic operations performed on data. One example of such code is analyzed in Figure 2, and more examples are shown in Figure 3. Traditionally, GPPs execute such tasks because no hardware accelerator is primarily designed for them, limiting the performance improvement of hardware accelerators at the application level [7].

Some techniques, such as zero-overhead loops [8] or branch prediction [9], can improve the performance of processors but they do not always translate into energy savings [10] due to their significant impact on circuit area. For control-intensive kernels, workload parallelization at the data level is often not possible, and other mechanisms, such as instruction level parallelism (ILP), are required to improve the execution of such code. Very-long instruction word (VLIW) processors can exploit code ILP and increase instructions-per-cycle (IPC) with a higher energy efficiency compared to superscalar general-purpose processors because part of the work is relayed to the compiler rather than to runtime logic.

In [11], the authors combined a VLIW processor with a CGRA to optimize performance on both data- and control-intensive code. However, such a design is not the most optimal in terms of area as it contains a VLIW processor and a CGRA. Therefore, replacing these two DSIPs with a single one would be more area- and cost-effective and save energy because of the reduced wiring. In [1], VWR2A demonstrated an improvement factor of $23\times$ and $66\times$ in performance and energy, respectively, compared to the ultra low-power Samsung reconfigurable processor (ULP-SRP) [11].[1]

This paper explores the performance and energy efficiency of accelerators based on domain-specific programmable cores for control-intensive code on energy-constrained embedded systems. Even though they are not optimized for such code, their flexibility allows them to execute those kernels. Based on an existing architecture that has proven its efficiency on data-intensive kernels [1], we designed a new version optimized for control-intensive code. We evaluate its performance and energy consumption on control-intensive code compared to two general-purpose baseline processors: an ARM Cortex-M4 [12] and a RISC-V based lowRISC Ibex [13] (formerly known as the Zero-riscy [14]). The Cortex-M4 is a middle-class processor with simple branch prediction (branch forwarding) and DSP-extensions, while the Ibex is optimized for control-dominated code [14].

The main contributions of this paper are:

- An evaluation of control-intensive kernels and applications starting from a very recently introduced reconfigurable accelerator architecture originally proposed in the literature [1] for the execution of data-intensive applications and its optimization for the execution of control-intensive code.
- An evaluation of its performance and energy consumption in control-intensive kernels and applications. Particularly the execution of queues, which are common structures used in many algorithms and usually not executed by an accelerator.
- A comparison with two general-purpose processors, one of which is optimized to execute control-intensive code.

The rest of this paper is organized as follows. First, we present the existing solutions and discuss their limitations in Section 2. Then, we show the details of a programmable architecture and its extensions in Section 3 and evaluate its power consumption on control-intensive kernels in Section 4. The experimental setup to evaluate the architecture is described in Section 5, and the results are presented and analyzed in Section 6. Finally, we draw the main conclusions of our study in Section 7.

2 Related work

Hardware accelerators are primarily designed for data-intensive code as it usually represents most of the execution time. However, once this code is accelerated, control-intensive code or control flow (e.g., branches, conditions) becomes predominant, limiting the impact of the accelerator [7], especially at the application level. Reconfigurable architectures are usually preferred as their flexibility can offer good performance for multiple kernels. In particular, CGRAs are often used in low-power platforms because of their proven performance and energy efficiency [1], [4], [15]. However, the traditional modulo scheduling (software pipelining) [16] for kernel mapping fails to convert nested loops and complex if-else structures to static code for CGRAs. Some software pipelining solutions or mapping techniques have been proposed to solve this problem [17], [18], but this work focuses on solutions at the architecture level.

At the system level, different solutions have been proposed. For example, the MorphoSys [19] platform combines a RISC processor and a CGRA. Despite the good performance of the CGRA on data-intensive kernels, the control code (inherent to any application) is left to the RISC processor, limiting the performance gain at the application level. The authors of [7] proposed the ADRES framework to solve this problem. ADRES combines a VLIW and a CGRA to efficiently execute control-intensive and data-intensive code, respectively. While the authors of ADRES improved the overall performance, they did not focus on energy consumption.

In [1], the authors proposed a Very-Wide-Register Reconfigurable-Array (VWR2A) architecture. VWR2A is a CGRA-like architecture augmented with low-power memory structures, made of a scratchpad memory (SPM) and very-wide registers (VWRs), and integrating VLIW concepts, such as specialized slots (e.g., a load-and-store unit (LSU)). Such architecture has shown better energy efficiency on data-intensive code than ULP-SRP [11], an instantiation of the ADRES framework. The authors of [15] proposed an Integrated Programmable Array (IPA): a 4x4 reconfigurable array of reconfigurable cells (RCs). Some RCs have an LSU

1. A portion of the significant difference in energy can be explained by noting that ULP-SRP has been simulated with a post-layout netlist, while VWR2A was simulated with a post-synthesis netlist.

connected to a shared multi-bank tightly coupled data memory (TCDM) through a logarithmic interconnect. The RCs are augmented with jump and conditional jump instructions to execute nested loop structures. While IPA and VWR2A demonstrated good energy efficiency for data-intensive kernels with nested loops (e.g., FIR, FFT, convolution), their efficiency on complex conditional structures (e.g., if-else) has not been studied.

SNAFU [20], an ultra-low-power CGRA generation framework, proposes an architecture with heterogeneous processing elements that are specialized for specific tasks, similar to the specialized slots of VWR2A. SNAFU introduces the spatial vector-dataflow execution model to reduce energy consumption. This model is conceived to map the complete data flow graph (DFG) of an innermost kernel to the complete accelerator array, configuring each RC to perform one single operation. This model, conceptually closer to that of a super-systolic array, contrasts with the programming model of VWR2A, in which each RC has a small instruction memory to implement a small program.

VWR2A and SNAFU present two additional significant differences: first, SNAFU uses a traditional memory hierarchy with multiple master ports connected to the platform's bus and main memory. On the contrary, VWR2A has a wide memory hierarchy that offers a high bandwidth and low energy solution compared to SNAFU. SNAFU implements scratchpad memories to limit costly accesses to the platform's main memory (in latency and energy), particularly between reconfigurations of the array. The latter is due to the second main difference: VWR2A has a specialized slot, namely a loop controller unit (LCU), enabling the mapping of multi-level nested loops. As this allows VWR2A to cover longer fragments of code independently, the GPPs present in the system can sleep for longer periods or perform other unrelated tasks. In comparison, SNAFU is limited to the innermost loop, relying on the GPP and possibly multiple reconfigurations for the outer loops within one kernel. Finally, similar to the IPA, SNAFU has demonstrated chiefly its efficiency on data-intensive kernels (e.g., FFTs, DWT) but not for control-intensive code.

Other proposals offer architecture solutions, but they target high performance rather than energy efficiency. For example, Elastic CGRAs [21] introduce elasticity in the processing elements (PEs)' interconnection network to enable efficient use of PEs with operations that have varying latency (e.g., memory access vs. ALU operation). However, for low-power CGRAs, the environment is typically more constrained, and the latency of each operation is known at compilation time, allowing efficient scheduling.

For conditionals, partial and full predication techniques have been proposed for CGRAs [22], [23] However, while such techniques improve performance, they are usually worse in energy [10], [14]. Moreover, these methods have been applied in the context of modulo scheduling, therefore limited to the condition(s) present in the innermost loop of a kernel.

This paper focuses on optimizing and evaluating the extensions introduced explicitly in the VWR2A architecture [1] to improve the execution of control-intensive kernels and applications and to evaluate its resulting higher efficiency compared to central processing unit (CPU) for this type of code.Therefore, we compare it against two baseline GPPs. The first one is an ARM Cortex-M4 processor [12], the original host processor of the system-on-chip (SoC) where VWR2A was included. However, this middle-class processor might not be the most energy-efficient for control-dominated code. Therefore, for generality, we also compare the architecture with a lowRISC Ibex processor [13] optimized for control-intensive kernels [13], [14]. The Ibex implements the RISC-V RVC32IM instruction set architecture (ISA) with a two-stage pipeline.

Reconfigurable architectures often lack compiler support as they require significant work to integrate the architecture features properly. This area is under active research [15], [24], [25], but we do not explore it yet in this paper and instead rely on manual mapping of kernels on the architecture.

3 Domain-specific instruction set processor architecture description

Figure 1a shows the VWR2A implementation and its integration inside a SoC as proposed by the authors of [1]. It features a 4x2 reconfigurable array extended with specialized slots: LCU, load-and-store unit (LSU), and multiplexer controller unit (MXCU). These slots and the 4 PEs are called a *column*, and they share a program counter (PC). They are all programmed in parallel at the beginning of a kernel execution by loading a maximum of 64 instruction words from the context memory to their internal instruction memory. Each of the two columns can run independently or in synchronization. The PEs are interconnected to their close neighbors. Data exchange between the specialized slots and-or the PEs is done through the single-port scalar register file (SRF) present in each column.

The synchronizer (see Figure 1a) receives the acceleration requests and the direct memory access (DMA) transfer requests from the host processor. It manages the scheduling of the kernel requests and notifies the host processor once an acceleration is finished through one of the processor interrupt lines. It also forwards the data transfer requests to the DMA to move data between the SoC static random access memory (SRAM) and the VWR2A SPM.

The data memory block in Figure 1a instantiates low-power structures such as an SPM and VWRs [26], [27]. Conceptually, the SPM can be considered as an L2 or background memory and the VWRs as L1 or foreground memories. The SPM is shared between the columns and has a 4096 bits width (128 words of 32 bits). Each column has three VWRs with a dual interface: 4096 bits on the SPM side and 128 bits on the PEs' side (i.e., one 32-bit word per PE). This enables the transfer of a wide line (i.e., 4096 bits) between the SPM and the VWRs in one cycle, whereas the PEs can consume data in 32-bit words. These transfers are controlled by the LSU with an optimized ISA for this task. The VWRs are divided into four slices to enable concurrent access from the PEs of one column. The details of the SPM, the VWRs, and the LSU are not discussed in this paper as they are less relevant for control-intensive code. However, their low-power features and single-cycle transfer play an essential role in the overall performance and energy efficiency of the architecture.

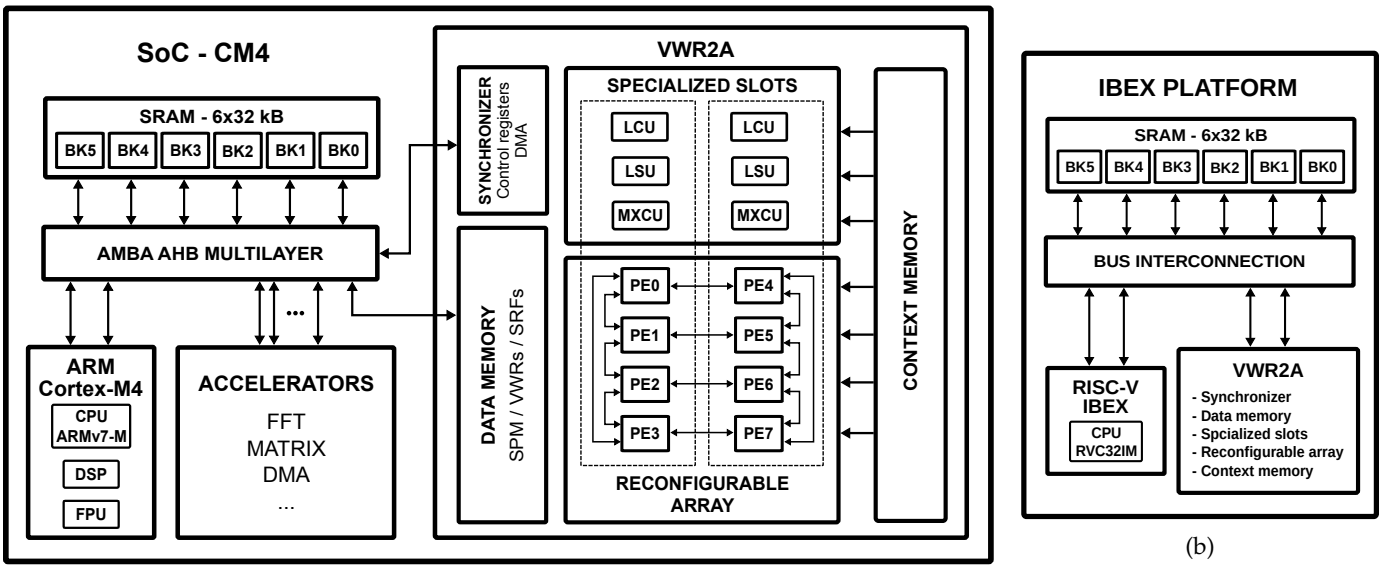


Fig. 1: (a) Original VWR2A architecture implementation and integration inside a SoC as presented in [1]. (b) Ibex platform augmented with VWR2A used for comparison.

TABLE 1: PE instruction word format and size.

Field	muxAsel	muxBsel	muxFsel	aluOp	rfWe	rfSel
Bits	16:13	12:9	8:5	4:2	1	0
Instruction word width						17 bits

The specialized slots LCU and MXCU were included in the VWR2A architecture specifically to tackle control-intensive code. These slots enable separating the control instructions from the main stream of instructions, which exposes more parallelism to the platform in general and the PEs in particular.

3.1 Processing Elements

Although the PEs are not shown in the specialized slots block in Figure 1, they are specialized for data processing by design. As all the extra computations not directly related to data are not executed by the PEs anymore (e.g., loop counter increment, jump), their design can be optimized. For example, the PEs' instructions do not have an immediate field but hardcoded values for two multiplexer inputs: 0 and 1. This reduces the instruction width to 17 bits. Table 1 shows the format of this instruction: muxA/B/Fsel select the two ALU's input operands and the status flags, respectively; aluOp, the executed operation (e.g., addition, shift); and rfWe is the ALU output write enable bit to the register file address given by rfSel. Moreover, as the PEs focus on data-related computations, a small register file of two entries per PE is enough. The datapath width is 32 bits to stay generic and compatible with standard processors.

The performance of the PEs is not crucial for control-intensive code. For example, the PEs implement a one-cycle signed or fixed-point multiplier; however, control code usually consists of simpler operations, such as addition and subtraction. Two important operations for control-intensive code are operand selection based on the sign and the zero flags. These operations select one of the two arithmetic logic unit (ALU) input operands as output based on the value of the sign flag —movs.sf instruction— or the zero flag —movs.zf instruction. Similar to the PEs' interconnection for data, the PEs can select their internal flags or those from their neighboring PEs. In both cases, the flag value refers to the previous cycle operation result.

3.2 Specialized slots for control-intensive code

The specialized slots are optimized for a subset of tasks and can run in parallel to take advantage of the ILP inherent to any code, particularly control code. Each slot is optimized for its specific set of tasks with a custom ISA, datapath, and register file. This allows performant and energy-efficient execution of different types of kernels. While the slots are designed for certain tasks, they can also execute unrelated instructions. For example, the LSU usually takes care of incrementing an address pointer. However, if the LSU is already executing another instruction at a certain cycle, the address pointer incrementation can be offloaded to another slot, as long as the ISA of that slot supports it. Such a scenario uses the SRF to share data between the slots and improves performance by increasing ILP. Here we describe each specialized slot important for the execution of control-intensive code, its tasks, and the related optimizations.

3.2.1 Loop Controller Unit

Compared to a common CGRA using modulo scheduling and hardware kernel execution control —with a prologue, steady-state, and epilogue scheme [4]— the LCU is programmable. Its primary task is to handle loop control (e.g., counter increments, branches) and conditions (e.g., if-else blocks). Therefore, it can execute jump, conditional jump, signed addition/subtraction, logical bit operations (and, or, xor), and left/right logical and arithmetic shift.

The branch-if-greater-or-equal operation implements pre-decrementation of the counter by one, enabling a single-cycle counter update and branch. Compared to the original VWR2A design [1], we updated the LCU with one additional jump operation that can generate its destination addresses at runtime. This instruction reduces the code size of long if-then-else structures, which are required to access a specific slice and data of a VWR, and efficiently maps irregular memory access patterns. The jump and branch instructions update the PC register shared by all the units in a column (i.e., the four PEs and the specialized slots).

The LCU instructions have 20 bits, with a 6-bit immediate field. The destination address is stored in the immediate field for conditional jump instructions. This design is sufficient for simple conditions that are data-dependent. However, complex conditions may require multiple data-dependent comparisons. To accommodate for such a scenario, the LCU can execute a conditional jump based on the PEs' condition flags (i.e., equal and greater-or-equal flags). The flags of the four PEs are OR-ed to produce a single flag for both conditions that are connected to the LCU. This allows efficient execution of data-dependent conditions and possibly multiple comparisons in one cycle. Without this simple feature, some kernels can still be executed but have to use the shared SRF to transmit data between the PEs —where the data are compared— and the LCU —where the branch decision is taken. This results in penalty cycles that would increase, for example, the execution time of the *Median* and *Delineation* kernels (see Section 5) by $6\,\%$ and $28\,\%$, respectively. For the morphological filtering (MF) kernels (see Section 5), it is even more critical as they could not be mapped on VWR2A without this feature. In this case, the additional instructions required to use the shared SRF do not fit in the internal instruction memory of the LCU.

In theory, the LCU can implement any nested loop depth; however, the instantiation used for our experiments is limited by its internal data register file and instruction memory size of 4 and 64 words, respectively. If bigger loop depths are present in the target application domain, then two columns can be used, or the SRF can be used; alternatively, the template can be re-instantiated to a version with more data registers in the LCU.

The LCU's datapath width is fixed to 8 bits: 1 sign bit + 7 bits (-128:127). Most of the time, the size of a loop is limited to a maximum of 128 iterations. This is due to the VWRs size (i.e., 128 words) and the fact that each iteration usually consumes at least one data element. Therefore, after 128 iterations, the loop is exited to load the next batch of data using the LSU. The LCU's datapath width can accommodate bigger loop sizes by using negative values or implementing nested-loop structures.

3.2.2 MultipleXer Controller Unit

The MXCU computes the addresses of the VWRs' words passed to the PEs. While many data accesses with very regular patterns are usually seen for data-intensive code, the inverse is often true for control-intensive code: few data accesses with irregular patterns. In both scenarios, the number of instructions of the innermost loop mapped on the architecture is critical for high performance, and the MXCU plays a crucial role.

The MXCU has an instruction width of 27 bits and an eight entries register file. The instructions do not have an immediate field, but the ALU's input multiplexers have entries hardcoded to 0, 1, and 2. These values are enough to create most of the access patterns. The ALU can execute addition/subtraction, logical bit operations (and, or, xor), and left/right logical bit shift. The datapath width of 5 bits is fixed by the address range of the VWRs: $2^5 = 32$ (i.e., the VWRs word width, 128, divided by four slices, one per PE).

4 Illustrative kernel mapping analysis

The specialized slots are the main reason for the architecture performance and energy efficiency. They remove the latency of control-related instructions from the PEs, increasing the ILP of the architecture. Their design relies on the fact that many kernels can be divided into four —usually independent— tasks: execution control, data loading and storing, data address update, and data computation. These tasks are mapped on the LCU, the LSU, the MXCU, and the PEs, respectively, maximizing the architecture ILP.

To illustrate the use of the architecture, we analyze the mapping of an ascending sorting algorithm that is part of the median kernel discussed in Section 5. This example highlights the key features of the architecture and helps to understand the performance results of Section 6. The detailed analysis at the instruction level is done in Appendix A for conciseness.

Figure 2 shows the C code of the ascending sorting algorithm and its high-level mapping on the architecture. The LCU manages the 2-level nested loop. The counting direction is reversed to take advantage of the single-cycle counter decrement and branch (see 3.2.1). Moreover, the size of the loops is divided by two because the sorting is parallelized over the PEs of one column. This allows using all the four available PEs, therefore improving performance. The MXCU generates the addresses for accessing the i-element and the j-element of the data array. In this particular example, the MXCU only needs to increase the address by one for every inner loop iteration. The LSU loads the input data array to the foreground memory (i.e., the VWRs). Depending on the size of the array, the LSU might be called more than once.

Figure 2 details how the code is adapted for the architecture. The PEs assigned to each line of code are given on the right side of the block. For example, PEs 0 and 2 start by loading the data[i] value (minVal) in one of their internal registers, while PEs 1 and 3 store its index (minIdx). The array is split in two: PEs 0 and 1 sort the lower part of the array, and PEs 2 and 3 the upper part. The performance improvement of the parallelization offsets the overhead required at the end to recover the complete output. This is not always the case and should be evaluated for every kernel.

Table 2 compares the outer and inner loop size of the ascending sorting algorithm for different architectures: a Cortex-M4 processor (CM4), an Ibex processor, and VWR2A. For the inner loop, the two cases for a *true* and a *false* if-condition are evaluated. The outer and inner loops are highlighted in the PEs block of Figure 2. While the outer loop is two instructions longer for VWR2A than for the

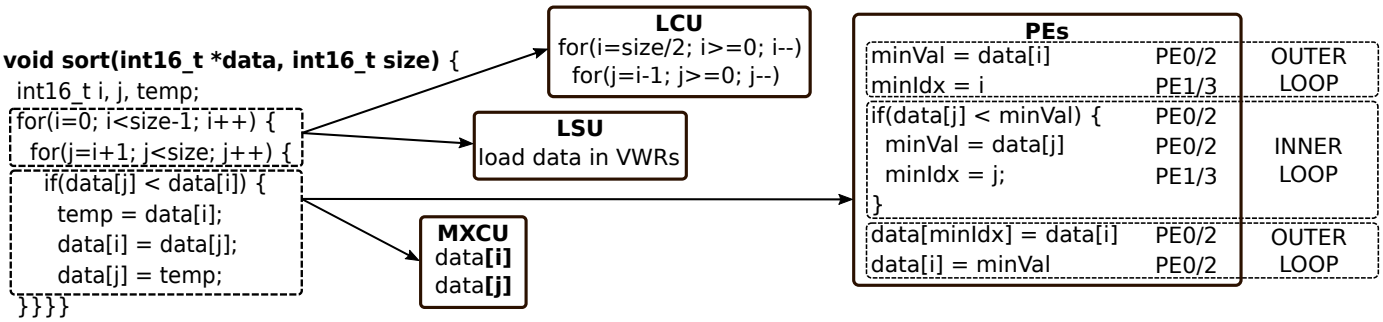


Fig. 2: Sorting algorithm C code (left) and its high-level mapping on VWR2A (right).

TABLE 2: Sorting algorithm outer- and inner-loop length (in cycles) comparison between ARM Cortex-M4 (CM4), Ibex, and VWR2A architectures.

		CM4	Ibex	VWR2A
outer loop	#instructions	6	6	8
	#iterations	31	31	15
inner loop	#instructions true	10	9	2
	#instructions false	8	7	2
	#iterations[2]	30	30	15

TABLE 3: Main features comparison between the ARM Cortex-M4 and the RISC-V Ibex processors.

	ARM Cortex-M4	RISC-V Ibex
ISA	ARMv7-M	RVC32IM
pipeline	3 stages	2 stages
32-bit integer multiplier	1 cycle	3-4 cycles
32-bit integer divider	2-12 cycles	37 cycles
branch prediction	target forwarding	pipeline stall
FPU	hardware	NA (software)
Extension(s)	SIMD, 1 cycle MAC	NA

Cortex-M4 and the Ibex processors, the inner loop is much faster. In terms of performance, the latter is usually the most important. The performance improvement of VWR2A comes from the instruction parallelization over the specialized slots (ILP) as depicted in Figure 2. Moreover, as long as there are no data dependencies over the iterations, the PEs can execute the condition independently using the mov.sf instruction (see Section 3.1).

5 Experimental Setup

5.1 Ultra-low power embedded platform

The VWR2A was initially integrated into an ultra-low power SoC (Figure 1a) intended for biomedical applications [3] (referred to as the ARM Cortex-M4 SoC in this paper). The main features of that SoC are an ARM Cortex-M4 processor and 6 SRAM banks of 32 KiB (192 KiB in total) that can be accessed in parallel and individually power gated. The same setup is used in this paper. The platform has multiple custom accelerators (e.g., FFT, Matrix); however, such fixed-function accelerators do not have the flexibility to execute control-intensive code.

In order to better demonstrate the efficiency of a programmable accelerator architecture, we also implemented a platform featuring an Ibex core, 192 KiB of SRAM divided into six banks, and VWR2A (Figure 1b) for comparison with the ARM Cortex-M4 SoC. The performance of the different implementations can be directly compared as the platforms are simulated at the cycle-accurate RTL level. However, the Ibex platform is simplified (i.e., not an actual SoC), and the energy numbers presented in Section 6 are lower-bound values. Therefore, a direct comparison between the ARM Cortex-M4 SoC and the Ibex platform is not completely fair. The main difference between the Ibex platform and the ARM Cortex-M4 SoC is the consumption of the bus interconnection. While the Ibex platform has a custom 3-masters and 7-slaves interconnection bus, the ARM Cortex-M4 SoC has an 19-masters and 14-slaves AMBA AHB multilayer interconnection bus.

Table 3 compares the main features of the Cortex-M4 and the Ibex processors. These are two representative cases of state-of-the-art programmable processors that target low-power devices, with the Ibex targeting specifically control-intensive workloads. As long as the code is dominated by control and the Ibex core can execute as fast as the Cortex-M4 processor, it will be more energy efficient because of its simpler architecture (even if the overhead in power mentioned previously is removed).

5.2 Performance and energy evaluation methodology

The ARM Cortex-M4 SoC and the Ibex platform, both including VWR2A, have been synthesized with the TSMC 40 nm LP CMOS technology at 80 MHz (the original frequency of the ARM SoC). We ran post-synthesis simulations to get cycle-accurate execution time and measure the cells switching activity used for power estimation with Synopsys PrimePower tool [28].

5.3 Representative set of software benchmarks for the biomedical target domain

In this Section, we discuss the link between the characteristics of the control-intensive benchmarks analyzed in Section 6 and the architectural features of VWR2A. The software benchmarks are divided into standalone kernels and applications. Standalone kernels are extracted from existing biomedical applications to evaluate the architectures at the kernel level. The impact at the application level is evaluated on three biosignal applications. Only the processing steps are considered, not the acquisition phase. The standalone

2. The number of iterations is reduced by 1 for every outer loop iteration.

```
void rsp_delineation(int16_t *data) {
00:  static int16_t first_win=1, lastValleyIndexRSP=0;
01:  static int16_t lastSigDiff=1, lastVal=0;
02:  static int16_t sig_diff[DATA_LENGTH];
03:
04:  if (first_win==1) {
05:    sig_diff[0]=0;
06:  } else {
07:    sig_diff[0]=data[0]-lastVal;
08:  }
09:
10:  if(sig_diff[0]==0) {
11:    sig_diff[0]=lastSigDiff;
12:  }
13:
14:  for(int16_t i=1; i<DATA_LENGTH; i++) {
15:    sig_diff[i] = data[i]-data[i-1];
16:  }
17:
18:  for(int16_t i=1; i<DATA_LENGTH; i++) {
19:    if(sig_diff[i]>0) {
20:      sig_diff[i] = 1;
21:    } else if (sig_diff[i]<0) {
22:      sig_diff[i] = -1;
23:    } else {
24:      sig_diff[i] = sig_diff[i-1];
25:    }
26:  }
27:
28:  first_win=0; lastSigDiff=sig_diff[DATA_LENGTH-1];
29:  last_val=data[DATA_LENGTH-1];
30:
 ...
}
```

(a)

```
void mf_lp_filter(int16_t *data) {
00:  static int16_t B[FILT_WIN*FILT_WIN] = {
01:    0, 0, 1, 5, 1,
02:    1, 0, 0, 1, 5,
03:    5, 1, 0, 0, 1,
04:    1, 5, 1, 0, 0,
05:    0, 1, 5, 1, 0
06:  };
07:  static int16_t original[HF_FILT_250HZ_WIN];
08:  static int16_t dilation[HF_FILT_250HZ_WIN];
09:  static int16_t erosion[HF_FILT_250HZ_WIN];
 ...    ...
17:  for(int16_t i=0; i<DATA_LENGTH; i++) {
18:    int16_t* Bi = B + FILT_WIN*i;
19:    original[i] = data[i];
20:    dilation[i] = original[0] + Bi[0];
21:    erosion[i]  = original[0] - Bi[0];
22:    for (int16_t j=1; j<FILT_WIN; j++) {
23:      if (original[j] + Bi[j] > dilation[i]) {
24:        dilation[i] = original[j] + Bi[j];
25:      }
26:      if (original[j] - Bi[j] < erosion[i]) {
27:        erosion[i] = original[j] - Bi[j];
28:      }
29:    }
 ...    ...
40:  }
}
```

(b)

```
void mf_dilation_queue(int16_t *data) {
00:  static int16_t q_val[WIN_SIZE], q_pos[WIN_SIZE];
01:  static int16_t front=0, back=0;
02:
03:  for (int16_t i=0; i<DATA_LENGTH; i++) {
04:    while (front != back && data[i] > q_val[back]) {
05:      back = (back-1)&(WIN_SIZE-1);
06:    }
07:    back = (back+1)&(WIN_SIZE-1);
08:    q_pos[back] = i;
09:    q_val[back] = data[i];
10:    if (WIN_SIZE <=
11:          ((i - q_pos[(front+1)&(WIN_SIZE-1)])&(WIN_SIZE-1))) {
12:      front = (front+1)&(WIN_SIZE-1);
13:    }
14:    input = q_val[(front+1)&(WIN_SIZE-1)];
15:  }
}
```

(c)

Fig. 3: Examples of control-intensive code in C language. (a) Respiration delineation (partial code). (b) Morphological filtering low-pass filter (partial code). (c) Morphological filtering dilation queue.

kernels are generic and can be included in applications of various domains, not just the biomedical one. For the evaluation of the standalone kernels on VWR2A, the overhead time to transfer the data to its internal SPM (before kernel execution) and back to the SoC SRAM (after the execution) is accounted for, as well as the reconfiguration time (i.e., loading the corresponding kernel instructions from the context memory to the PEs and the specialized slots).

Control-intensive kernels spend most of their time executing control-related instructions. Table 4 shows this by comparing control- and data-intensive kernels. The proportion of arithmetic and logic operations (ALU column in Table 4) is not enough for distinguishing control-intensive from data-intensive code. However, the analysis of the assembly code produced for each kernel shows that most of these operations are devoted to controlling tasks (e.g., counter incrementation, conditions) for the former and direct computation on data to produce an output (e.g., input data multiplication, addition) for the latter. Load and store instructions hint at the type of code executed, and data-intensive kernels usually have a significant proportion of them. However, some control-intensive kernels, such as the median, can also have a high proportion of them. In the end, the best indicator is the number of branch instructions executed (e.g., branch-if-equal, jump, branch-if-not-equal). Table 4 shows a clear difference between control-intensive and data-intensive kernels. For our analysis, we define control-intensive code as having more than $25\,\%$ of branch instructions at execution time.

Three examples of control-intensive code in C language are shown in Figure 3. The respiration signal delineation (Figure 3a) and the morphological low-pass filter (Figure 3b) examples only show part of the code for conciseness. The FILT_WIN and WIN_SIZE variables of the morphological low-pass filter and dilation queue examples (Figures 3b, and 3c) are the sizes of the structuring elements: 5 and 75 elements, respectively.

TABLE 4: Control-intensive and data-intensive kernels' instructions profiling using the RVC32IM ISA.

Kernel		INSTRUCTION		
		ALU	**Load/Store**	**Branch**
Control-intensive	median	$29\,\%$	$30\,\%$	$41\,\%$
	delineation	$54\,\%$	$19\,\%$	$27\,\%$
Data-intensive	FFT radix-2	$48\,\%$	$42\,\%$	$10\,\%$
	FIR Filter	$51\,\%$	$31\,\%$	$18\,\%$

Similarly to the sorting algorithm code analyzed in Section 4, these examples can be divided into four tasks: execution control, data loading and storing, data address update, and data computation. Each of these tasks corresponds to one specialized slot of the VWR2A architecture, which improves the ILP. Moreover, the PEs enable parallelization at the data level when possible. For example, lines 14 to 15 of the respiration delineation code (Figure 3a), the input signal derivative computation, take two clock cycles as instructions can be parallelized over the slots. Moreover, in this particular case, the derivative computation can be parallelized by unrolling the loop over the 8 PEs as this kernel uses both columns.

5.3.1 Standalone kernels

The first kernel, *Median*, takes an input array of up to 28 values, sorts them in ascending order, and obtains the middle value (i.e., the median). As discussed in Section 4, this kernel is dominated by control instructions to reorder the array. The second kernel is *Delineation* (Figure 3a), which is a typical step of many biomedical applications that extracts the fiducial points of a signal (e.g., ECG, EEG). These points are later used to extract features. Delineation is a highly irregular code. It includes complex nested if-else structures

TABLE 5: Comparison of the units (PEs and specialized slots) local program memory static and dynamic usage for the data-intensive kernels presented in [1] and the control-intensive kernels presented in this paper.

Data-intensive	**LCU**	**LSU**	**MXCU**	**PEs**
avg instructions[3]	14	19	14	17
avg static active instructions[4]	39.7 %	49.4 %	37.9 %	45.2 %
avg dynamic active instructions	37.4 %	6.1 %	84.6 %	88.9 %
Control-intensive	**LCU**	**LSU**	**MXCU**	**PEs**
avg instructions	26	10	17	16
avg static active instructions	57.1 %	24.3 %	39.8 %	41.1 %
avg dynamic active instructions	60.6 %	2.0 %	67.9 %	70.9 %

that are not predictable because of their data dependency. Therefore, the workload cannot be parallelized, and the data are processed one by one. The *Median* and *Delineation* kernels are extracted from a cognitive workload estimation application [29], from the respiration (RSP) signal features extraction and delineation steps, respectively.

Morphological filters (MF) are used in many signal processing applications. In the biomedical domain, morphological filtering is used for two purposes: biosignal baseline removal and filtering (e.g., low-pass filter). Biosignals often contain low-frequency harmonics known as the baseline. Morphological filtering can be used to remove this baseline by applying sequentially an erosion, a dilation (Figure 3c), and again an erosion filter. The output is the baseline and can be subtracted from the original input. The implementation uses three queues (one per morphological filter), which require many control instructions. The data-dependent nature of a queue does not allow efficient parallelization at the data level. Therefore the input data are processed sample by sample. The analyzed VWR2A architecture can manage the three queues on its own, thanks to its flexibility. This data-structure based functionality is quite unusual for hardware accelerators but is used in many applications, which creates huge opportunities for programmable architectures such as VWR2A.

The second example code using MF is a low-pass filter that uses closing and opening morphological operators to produce a low-pass filter (Figure 3b). Due to the small size of the structure element (i.e., the weights matrix), the implementation does not use queues and can be parallelized at the data level with a small computation penalty to recover the correct output.

Table 5 shows the static usage of the units' local program memory and the dynamic profiling of their active processing time for data- and control-intensive code. The original VWR2A architecture presented in [1] focused on data-intensive code. In contrast, this paper focuses on control-intensive code, which stresses the processing elements (i.e., the PEs and specialized slots) differently. In particular, the LCU has a significantly higher active time because control-intensive kernels are characterized by a higher number of branch operations (see Table 4). Although the MXCU plays an important role for control-intensive code, it is not as active as for data-intensive code because fewer data are accessed.

TABLE 6: VWR2A components average power consumption and breakdown of kernels that use two columns.

VWR2A Component	Power (µW) avg	%	σ
Context memory (5.4 KiB)	64	2.7 %	±1.3 %
Synchronizer	45	1.9 %	±14.2 %
LCUs	46	2.0 %	±34.0 %
LSUs	84	3.6 %	±21.7 %
MXCUs	55	2.3 %	±25.0 %
PEs	407	17.4 %	±28.2 %
ALUs	(182)	(7.8 %)	(±34.8 %)
instruction memories	(123)	(5.2 %)	(±30.7 %)
PEs interconnection	(69)	(3.0 %)	(±34.7 %)
register files	(33)	(1.4 %)	(±34.1 %)
Data memory (35 KiB)	1644	70.1 %	±21.5 %
SPM (32 KiB)	(636)	(27.1 %)	(±8.8 %)
VWRs (3 KiB)	(1000)	(42.7 %)	(±31.5 %)
SRFs (64 B)	(8)	(0.3 %)	(±106.8 %)
Total	2344	100 %	±20.6 %

5.3.2 Biosignal applications

The first application is a heartbeat classifier that acquires a 3-leads ECG signal and classifies the beats into normal and abnormal to detect arrhythmia [30]. The beat classification is performed on a single lead. If a heartbeat is classified as abnormal, the two additional leads are also processed. Therefore, the application processes either a 1-lead or a 3-leads ECG signal, referred to as *1-lead* and *1+2-leads*, respectively, in Section 6. Depending on the health condition of the monitored patient, the second scenario (*1+2-leads*) is activated more or less often. This application uses the MF baseline removal and the MF low-pass filter kernels to condition the ECG signals before executing the rest of the application (classifier, features extraction, ...). The results are shown for the processing time of one window of 768 samples (3 seconds of ECG signal).

The second application processes the complete set of 12-leads ECG signals as required for medical standards (e.g., devices in hospitals). This application applies the MF baseline removal and low-pass filtering kernels before combining the leads and extracting the fiducial points. The results are also shown for the processing time of one window of 768 samples.

The last application, which contains the *Median* and the *Delineation* kernels, is a multi-signal cognitive workload estimation application [29]. For this paper, we focused on analyzing the performance and energy consumption of these two control-intensive kernels at the application level. The performance of the related CGRA architecture on the data-intensive kernels has already been presented by the authors of [1]. The results correspond to the processing of a window of 512 samples (102 seconds of respiration signal).

6 Experimental results

6.1 Architecture power analysis

We first evaluate the power breakdown of the architecture, notably the power consumption of the specialized slots

3. Static average of instructions that are not a nop.

4. Average percentage of instructions that are not nops based on each kernel length (not the total program memory size of 64 instructions).

optimized for control-intensive code in order to assess their low-power nature and understand better the architecture's performance and energy efficiency. Table 6 shows the average and standard deviation of the power breakdown of the VWR2A components when executing the standalone kernels that use two columns of the architecture (i.e., *Median 2x*, *Delineation*, *MF – Baseline removal 2-leads*, and *MF – Low-pass filter 2-leads*). The PEs and the data memory consume $87.5\,\%$ of the total power on average, while the specialized slots (i.e., LCUs, LSUs, and MXCUs) account only for $7.9\,\%$. The energy overhead of the specialized slots is minimal, therefore justifying their use as they drastically increase the performance by improving ILP. Without them, the PEs would have to handle also their instruction flow, increasing the overall number of cycles to execute a kernel.

The PEs power consumption is divided between the ALUs, the internal instruction memories —64 words of 18 bits per PE—, the reconfigurable array interconnection, and the internal register files —2 words of 32 bits per PE. The datapaths (i.e., ALUs, interconnection, and register files) account for $69.8\,\%$ of the PEs power consumption. This shows that most of the PEs' power is consumed for actual data computation (i.e., the datapaths).

The context memory, which holds the configuration words for the kernels, is implemented using standard-cell flip-flops. These flip-flops are active only during the configuration phase of the columns and clock-gated during the execution of a kernel. The latter takes at least a few thousand cycles, while the configuration phase takes a maximum of 64 cycles. Therefore, the context memory power consumption is very small.

The data memory alone consumes $70.1\,\%$ of the total power on average. While the SPM uses memory macros, the VWRs are implemented with standard cell libraries' latches. This partially explains why the VWRs represent $42.7\,\%$ of the total power consumption while the SPM accounts only for $27.1\,\%$. A custom design where larger cells and a pragma-guided regular floorplan is imposed, as proposed in [26], [27], [31], would undoubtedly reduce their power consumption further. The power consumption of the SRFs is almost negligible because of their relative small size compared to the SPM and VWRs.

The average power of each component as shown in Table 6 presents a marked variability, which is due to the specific instructions executed by each kernel on each slot. For example, the duty-cycle of the LCU is $64\,\%$ for the *Median 2x* kernel, whereas it reaches $76\,\%$ for the *MF – Baseline removal 2-leads* kernel. This difference translates into a higher relative average power for the LCU when executing the latter. Nevertheless, the absolute average power of the specialized slots is so small that the variation on the total system power is very small.

The SRFs have the highest variability, $\pm 106.8\,\%$, because some kernels, like the *Delineation*, rely mostly on the SRF of one column only. However, the power consumption of the SRFs has almost no impact at the system level because it represents only $0.3\,\%$ of the total power on average. The $\pm 20.6\,\%$ variation at the architecture level is mainly dependant on the variability of the VWRs power consumption. Some kernels, like the *Median 2x*, do not access the VWRs so much, while others heavily use them, like the *MF – Baseline removal 2-leads* that stores not only the input data but the queues as well.

6.2 Standalone kernels results

Table 7 shows the performance and energy consumption for the four evaluated architectures on seven computational kernels present in biomedical applications. VWR2A exhibits similar performance on both the Cortex-M4 SoC and the Ibex platform as their memory and bus architecture are similar (only the processors differ).

6.2.1 Median

The mapping of the *Median* kernel uses only one column of the reconfigurable array. Two cases are evaluated: 1 median executed in one column (the other one is idle) and two medians computed in parallel over two columns. The latter is the best case for VWR2A as the execution time is almost similar to computing one median, improving its energy efficiency compared to the processors.

The good performance of the VWR2A architecture is mainly due to the improvements in the sorting algorithm discussed in Section 4. Table 2 shows that the outer and inner loops of the sorting algorithm have two times fewer iterations than the Cortex-M4 and the Ibex implementations, entirely due to the architecture innovations of VWR2A (i.e., for the same algorithm). Moreover, the inner loop size is only two instructions, while the Cortex-M4 and the Ibex use at least 8 and 7 instructions, respectively. This should give approximately a $16\times$ performance improvement for VWR2A. However, the overhead of acceleration request from the host processor and the data transfer limit the overall performance. Moreover, the parallelization of the sorting algorithm on VWR2A requires a recovery step (executed on VWR2A) to get the correct output, reducing the performance improvement to a factor $8.5\times$ and $9.6\times$ compared to the Cortex-M4 and the Ibex, respectively, for the *Median 2x* case.

6.2.2 Delineation

This kernel uses the two columns of the accelerator. The first step of the kernel consists of the input signal derivative computation and extraction of local minimums and maximums. This step is easily parallelizable at the algorithmic and instruction level, which translates to a $19\times$ performance improvement compared to the Cortex-M4. The second step consists of analyzing the local minimums and maximums and selecting the valid ones. This is done through a three-level nested if-else structure that selects the next valid point based on data-dependent conditions, which limits parallelization at the algorithmic level. However, this step can take advantage of the ILP of the architecture and achieves an $8\times$ performance improvement compared to the Cortex-M4. This translates to an overall $11.8\times$ performance improvement at the kernel level compared to the Cortex-M4 (Table 7). For the Ibex platform, the results are very similar.

6.2.3 MF - Baseline removal

The morphological filter for baseline removal is the kernel in which VWR2A obtains the lowest speed-up with respect to the two CPUs. The reason is that the implementation

TABLE 7: Standalone kernels performance and energy comparison for the ARM Cortex-M4 SoC (**CM4**), the Ibex platform (**Ibex**), the ARM Cortex-M4 SoC including VWR2A (**CM4+VWR2A**), and the Ibex including VWR2A (**Ibex+VWR2A**).

Cycles	**CM4**	**Ibex**	**CM4 + VWR2A**	**Ibex + VWR2A**	**CM4 vs. CM4+VWR2A**	**Ibex vs. Ibex+VWR2A**	**CM4 vs. Ibex**
Median 1x	5015	6101	1098	1087	$4.6\times$	$5.6\times$	$0.8\times$
Median 2x	10 007	12 169	1183	1264	$8.5\times$	$9.6\times$	$0.8\times$
Delineation	32 113	31 820	2723	2713	$11.8\times$	$11.7\times$	$1.0\times$
MF – Baseline removal 1-lead	98 228	103 288	30 664	30 448	$3.2\times$	$3.4\times$	$1.0\times$
MF – Baseline removal 2-leads	205 768	208 611	31 577	31 212	$6.5\times$	$6.7\times$	$1.0\times$
MF – Low-pass filter 1-lead	96 784	152 333	8642	8523	$11.2\times$	$17.9\times$	$0.6\times$
MF – Low-pass filter 2-leads	191 515	304 649	10 121	9783	$18.9\times$	$31.1\times$	$0.6\times$
Average speed-up					$\mathbf{9.2\times}$	$\mathbf{12.3\times}$	$\mathbf{0.8\times}$
Energy (µJ)							
Median 1x	8.23×10^{-2}	7.72×10^{-2}	3.05×10^{-2}	2.61×10^{-2}	$-62.9\,\%$	$-66.2\,\%$	$-6.2\,\%$
Median 2x	1.70×10^{-1}	1.51×10^{-1}	3.97×10^{-2}	3.74×10^{-2}	$-76.7\,\%$	$-75.3\,\%$	$-11.2\,\%$
Delineation	5.74×10^{-1}	4.69×10^{-1}	1.30×10^{-1}	9.03×10^{-2}	$-77.3\,\%$	$-80.7\,\%$	$-18.3\,\%$
MF – Baseline removal 1-lead	1.72×10^{0}	1.52×10^{0}	8.79×10^{-1}	6.85×10^{-1}	$-48.9\,\%$	$-54.9\,\%$	$-11.6\,\%$
MF – Baseline removal 2-leads	3.70×10^{0}	3.08×10^{0}	1.35×10^{0}	9.79×10^{-1}	$-63.5\,\%$	$-68.2\,\%$	$-18.9\,\%$
MF – Low-pass filter 1-lead	1.70×10^{0}	1.63×10^{0}	3.10×10^{-1}	2.24×10^{-1}	$-81.8\,\%$	$-86.3\,\%$	$-4.1\,\%$
MF – Low-pass filter 2-leads	3.36×10^{0}	3.27×10^{0}	5.28×10^{-1}	3.57×10^{-1}	$-84.3\,\%$	$-89.1\,\%$	$-2.7\,\%$
Average energy saving					$\mathbf{-70.8\,\%}$	$\mathbf{-74.4\,\%}$	$\mathbf{-10.4\,\%}$

of the queues is only parallelized at the instruction level as the algorithm cannot be parallelized at the data level without significant overhead. The mapping is limited to one column so two executions can run in parallel, improving performance. The two scenarios, 1-lead and 2-leads, are reported in Table 7, using one, respectively two, columns. As this kernel is the worst in terms of speed-up it also gives the lowest energy savings for VWR2A compared to the ARM Cortex-M4 SoC and the Ibex platform: $48.9\,\%$ and $54.9\,\%$, respectively, for the 1-lead case, and $63.5\,\%$ and $68.2\,\%$, respectively, for the 2-leads case.

6.2.4 MF - Low-pass filter

This kernel also uses a single column, and the results for the cases of 1-lead, using one column, and 2-leads, using two columns, are reported (Table 7). This kernel is the best in terms of performance and energy savings for VWR2A because it contains more computation on data than the other kernels, although it is still dominated by control instructions. Moreover, because of the small size of the morphological structuring elements, an implementation using queues is not the most efficient. Therefore, a straightforward implementation, that can be parallelized with a small overhead on VWR2A, is used. This translates into energy savings up to $84.3\,\%$ and $89.1\,\%$ for the 2-leads case compared to the ARM Cortex-M4 SoC and the Ibex platform, respectively.

6.3 Standalone kernels analysis

VWR2A is the fastest implementation for all the kernels. This corroborates that an adequately designed reconfigurable architecture can also accelerate control-intensive kernels, in this case when optimized for the biomedical domain. In terms of energy, although VWR2A has higher power than the two considered processors, its speed-up translates to significant savings in energy consumption.

As we study only the kernels themselves here, we can analyze the performance and energy consumption of the VWR2A architecture in isolation and compare it directly with the execution on the two CPUs. Compared to the Cortex-M4 processor (not the SoC, only the CPU), VWR2A alone uses $5.8\times$ more power but is $9.2\times$ faster on average, hence more energy efficient. Moreover, VWR2A uses its internal memory structure to access the data (i.e., SPM and VWRs), which is more energy-efficient than the SoC AMBA AHB interconnection. This leads to a $70.8\,\%$ energy saving on average at the SoC level, which is combined with the higher performance leading to an energy-delay product (EDP) improvement of $43.8\times$.

Compared to the Ibex core (only the CPU), VWR2A alone requires $8.7\times$ more power but is $12.3\times$ faster, leading to an average energy saving of $74.4\,\%$ at the platform level (Ibex+VWR2A vs. Ibex). This translates to an EDP improvement of $56.6\times$ for the Ibex+VWR2A platform compared to the Ibex platform. Interestingly, although the Ibex processor is optimized for control-intensive code and has better energy efficiency than the ARM Cortex-M4 SoC, it is on average $1.2\times$ slower, translating to an EDP decrease of $1.1\times$ for our biosignal processing target domain.

6.4 Biosignal applications results

The performance and energy consumption of the three evaluated designs are shown in Table 8. The code executed on VWR2A was limited to the control-intensive kernels presented in Table 7 to evaluate the impact of control code acceleration at the application level. However, it has been substantiated in [1] that VWR2A can also execute most of the data-intensive kernels of the evaluated applications. Therefore, the numbers reported in Table 8 are the minimum energy savings that can be achieved using VWR2A.

6.4.1 Heartbeat classifier

Two scenarios have been considered: 1-lead and 1+2-leads. The second one is the optimal case for VWR2A as it can execute the two leads in parallel. For the first case (1-lead), only half of the VWR2A architecture is used (i.e., one column), reducing its energy efficiency. The execution is dominated

TABLE 8: Biosignal applications performance and energy comparison for the ARM Cortex-M4 SoC (**CM4**), the Ibex platform (**Ibex**), the ARM Cortex-M4 SoC including VWR2A (**CM4+VWR2A**), and the Ibex including VWR2A (**Ibex+VWR2A**). VWR2A is limited to execute only control-dominated kernels.

Cycles	**CM4**	**Ibex**	**CM4 + VWR2A**	**Ibex + VWR2A**	**CM4 vs. CM4+VWR2A**	**Ibex vs. Ibex+VWR2A**	**CM4 vs. Ibex**
Heartbeat classifier 1-lead (ECG)	463 025	672 639	208 525	289 881	2.2 ×	2.3 ×	0.7 ×
Heartbeat classifier 1+2-leads (ECG)	1 278 191	1 773 784	465 366	579 513	2.7 ×	3.1 ×	0.7 ×
Medical standard 12-leads (ECG)	4 261 659	10 132 440	894 567	1 003 344	4.8 ×	10.1 ×	0.4 ×
Cognitive workload est. 1-lead (RSP)	153 115	645 677	123 122	616 570	1.2 ×	1.0 ×	0.2 ×
Average speed-up					**2.7 ×**	**4.1 ×**	0.5 ×
Energy (µJ)							
Heartbeat classifier 1-lead (ECG)	4.4	7.4	3.9	3.7	−11.6 %	−50.5 %	+70.1 %
Heartbeat classifier 1+2-leads (ECG)	10.9	20.4	8.6	8.6	−20.7 %	−58.0 %	+87.9 %
Medical standard 12-leads (ECG)	40.3	78.4	25.1	22.2	−37.7 %	−71.6 %	+94.6 %
Cognitive workload est. 1-lead (RSP)	2.3	8.4	1.9	8.1	−17.7 %	−3.7 %	+262.1 %
Average energy saving					**−21.9 %**	**−46.0 %**	**+128.7 %**

by the morphological filtering —baseline removal and low-pass filter— of the leads, which takes more than 65 % and 75 % of the total processing time, for the 1-lead and 2-leads cases, respectively (for both the ARM Cortex-M4 SoC and the Ibex platform). At the application level, gains are limited compared to those at the kernel level (see Table 7). The reason is that the lead window size is bigger than the one used for the standalone kernels and the scratchpad memory of VWR2A is not large enough to hold all the data simultaneously, requiring some backup to the platform SRAM. This increases the bus interconnection energy consumption and limits the gain of VWR2A compared to both processors. Moreover, the acceleration is limited to the control-intensive kernels of the application while the processors execute the remaining part, limiting the overall savings of VWR2A.

6.4.2 Medical standard 12-leads

Like the previous application, executing the morphological filter for 12 leads requires some adaptation as the VWR2A internal memory is not large enough to store all the values (input data and kernel parameters) at once. Therefore, additional data transfers are needed to back up these values in the ARM Cortex-M4 SoC and the Ibex platform SRAM. This induces a penalty in performance and energy but using VWR2A is still more energy-efficient. This application is even the best regarding energy savings: VWR2A consumes 37.7 % and 71.6 % less energy than the ARM Cortex-M4 SoC and the Ibex platform, respectively. This is due to the even number of leads, which maximizes the VWR2A utilization and, therefore, its energy efficiency.

6.4.3 Cognitive workload estimation

This application is the worst in terms of overall performance and energy savings, particularly compared to the Ibex platform, because it is dominated by data-intensive kernels (as discussed previously) and given the focus of this paper, here we limited our study to accelerate only the control-intensive kernels (*Median* and *Delineation kernels*). These kernels account for less than 5 % of the total processing time on the Ibex platform, limiting the impact of VWR2A. For the ARM Cortex-M4 SoC, these kernels represent 21 % of the processing time, which translates to higher savings for VWR2A. For such application, having a reconfigurable accelerator is the best as it can accelerate both data-intensive and control-intensive kernels. For example, in [1], the authors reported energy savings of 66.3 % compared to the ARM Cortex-M4 SoC at the application level with all the possible kernels accelerated on the VWR2A. Nevertheless, VWR2A still exhibits 17.7 % and 3.7 % lower energy consumption than the ARM Cortex-M4 SoC and the Ibex platform, respectively.

6.5 Biomedical applications analysis

Compared to the results of the standalone kernels, the savings at the application level are more limited. When complete applications are considered, there are inevitable data-intensive steps that significantly worsen the performance and energy efficiency, particularly for the Ibex core, which is optimized for low-power execution of control code. This is particularly true for the *Cognitive workload estimation* application that is dominated by data-intensive kernels, such as an FFT and a FIR filter. At the kernel level, the Ibex core is slower but more energy efficient than the ARM Cortex-M4, but at the application level, it is both slower and less energy efficient with an EDP 3.7 × lower.

Similar to the case of standalone kernels, VWR2A is the best implementation in terms of performance and energy. VWR2A is 2.7 × and 4.1 × faster than the ARM Cortex-M4 SoC and the Ibex platform, respectively, on average. In terms of energy, VWR2A consumes 21.9 % and 46.0 % less energy on average than the ARM Cortex-M4 SoC and the Ibex platform, respectively. The performance and energy combined give VWR2A an improvement in EDP of 3.8 × and 12.2 × compared to the ARM Cortex-M4 SoC and the Ibex platform, respectively.

These results show the importance of evaluating architectures at the application level and the advantages of the flexibility of a reconfigurable accelerator, allowing both data-intensive [1] and control-intensive code to be executed more efficiently, in terms of performance and energy, than GPPs.

7 Conclusion

This paper has evaluated the performance and saving in energy consumption of a recent programmable core accelerator proposed by the authors of [1] for embedded systems with two general-purpose baseline processors for control-intensive kernels and applications. This programmable core, called VWR2A [1], is a CGRA-like architecture augmented with specialized slots. These slots leverage the ILP inherent to any kernel and remove the latency of the control instructions execution of the PEs in the reconfigurable array. This situation significantly improves the architecture performance on control-dominated kernels. Moreover, the specialized slots can be optimized for a specific set of tasks, which minimizes their power consumption.

The results of this work have shown that the programmable core is consistently faster and more energy-efficient than two representative processors of state-of-the-art programmable processors targeting low-power devices —an ARM Cortex-M4 processor and a RISC-V lowRISC Ibex processor— for kernels and applications. Thus, we have proven that the flexibility offered by programmable accelerators enables efficient execution of not only data-intensive kernels, but also of control-intensive ones. A key direction to increase energy efficiency in future heterogeneous embedded systems is to increase the code coverage of these programmable accelerators.

Acknowledgments

The authors of this paper would like to acknowledge the work of Emilio Domínguez Sánchez on improving the algorithmic implementation of the Morphological Filtering kernels for the general-purpose processors, which also improved the mapping on VWR2A.

References

[1] B. W. Denkinger, M. Peón-Quirós, M. Konijnenburg, D. Atienza, and F. Catthoor, "VWR2A: A Very-Wide-Register Reconfigurable-Array Architecture for low-power embedded devices," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 895–900.

[2] Transforma Insights. Accessed on: Apr. 4, 2022. [Online]. Available: https://transformainsights.com/news/iot-market-24-billion-usd15-trillion-revenue-2030

[3] S. Song, M. Konijnenburg, R. van Wegberg, J. Xu, H. Ha, W. Sijbers, S. Stanzione, D. Biswas, A. Breeschoten, P. Vis, C. van Liempd, C. van Hoof, and N. van Helleputte, "A 769 μW battery-powered single-chip SoC with BLE for multi-modal vital sign monitoring health patches," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 6, pp. 1506–1517, 2019.

[4] L. Duch, S. Basu, R. Braojos, G. Ansaloni, L. Pozzi, and D. Atienza, "Heal-wear: An ultra-low power heterogeneous system for bio-signal analysis," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2448–2461, 2017.

[5] E. Flamand, D. Rossi, F. Conti, I. Loi, A. Pullini, F. Rotenberg, and L. Benini, "GAP-8: A RISC-V SoC for AI at the Edge of the IoT," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2018, pp. 1–4.

[6] 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 (TCAD)*, vol. 37, no. 11, pp. 2940–2951, 2018.

[7] B. Mei, A. Lambrechts, J.-Y. Mignolet, D. Verkest, and R. Lauwereins, "Architecture exploration for a reconfigurable architecture template," *IEEE Design Test of Computers*, vol. 22, no. 2, pp. 90–101, 2005.

[8] G.-R. Uh, Y. Wang, D. Whalley, S. Jinturkar, C. Burns, and V. Cao, "Techniques for effectively exploiting a zero overhead loop buffer," in *Compiler Construction*, D. A. Watt, Ed. Springer Berlin Heidelberg, 2000, pp. 157–172.

[9] J. Lee and A. Smith, "Branch prediction strategies and branch target buffer design," *Computer*, vol. 17, no. 01, pp. 6–22, Jan 1984.

[10] O. Azizi, A. Mahesri, B. C. Lee, S. J. Patel, and M. Horowitz, "Energy-performance tradeoffs in processor architecture and circuit design: A marginal cost analysis," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. ACM, 2010, p. 26–36.

[11] C. Kim, M. Chung, Y. Cho, M. Konijnenburg, S. Ryu, and J. Kim, "ULP-SRP: Ultra Low-Power Samsung Reconfigurable Processor for Biomedical Applications," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 7, no. 22, pp. 1–15, 2014.

[12] ARM Cortex-M4 – Reference manual. Accessed on: Apr. 4, 2022. [Online]. Available: https://www.arm.com/products/silicon-ip-cpu/cortex-m/cortex-m4

[13] lowRISC – Ibex. Accessed on: Apr. 4, 2022. [Online]. Available: https://lowrisc.org/our-work/

[14] P. Davide Schiavone, F. Conti, D. Rossi, M. Gautschi, A. Pullini, E. Flamand, and L. Benini, "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications," in *27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2017, pp. 1–8.

[15] S. Das, K. J. M. Martin, D. Rossi, P. Coussy, and L. Benini, "An energy-efficient integrated programmable array accelerator and compilation flow for near-sensor ultralow power processing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 6, pp. 1095–1108, 2019.

[16] B. R. Rau, "Iterative modulo scheduling: An algorithm for software pipelining loops," in *Proceedings of the 27th Annual International Symposium on Microarchitecture*, ser. MICRO 27. ACM, 1994, p. 63–74.

[17] M. Karunaratne, C. Tan, A. Kulkarni, T. Mitra, and L.-S. Peh, "Dnestmap: Mapping deeply-nested loops on ultra-low power CGRAs," in *Design Automation Conference (DAC)*. ACM, 2018. [Online]. Available: https://doi.org/10.1145/3195970.3196027

[18] S. Yin, P. Zhou, L. Liu, and S. Wei, "Acceleration of nested conditionals on CGRAs via trigger scheme," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2015, pp. 597–604.

[19] H. Singh, M.-H. Lee, G. Lu, F. Kurdahi, N. Bagherzadeh, and E. Chaves Filho, "MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE Transactions on Computers*, vol. 49, no. 5, pp. 465–481, 2000.

[20] G. Gobieski, A. O. Atli, K. Mai, B. Lucia, and N. Beckmann, "Snafu: An ultra-low-power, energy-minimal CGRA-generation framework and architecture," in *ACM/IEEE International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1027–1040.

[21] Y. Huang, P. Ienne, O. Temam, Y. Chen, and C. Wu, "Elastic CGRAs," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2013, pp. 171–180.

[22] K. Han, J. Ahn, and K. Choi, "Power-efficient predication techniques for acceleration of control flow execution on CGRA," *ACM Transactions on Architecture and Code Optimization*, vol. 10, no. 2, May 2013.

[23] K. Han, J. K. Paek, and K. Choi, "Acceleration of control flow on CGRA using advanced predicated execution," in *2010 International Conference on Field-Programmable Technology*, 2010, pp. 429–432.

[24] T. K. Bandara, D. Wijerathne, T. Mitra, and L.-S. Peh, "REVAMP: A systematic framework for heterogeneous CGRA realization," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2022, pp. 918—-932.

[25] C. Tirelli, L. Ferretti, and L. Pozzi, "SAT-MapIt: A SAT-based Modulo Scheduling Mapper for Coarse Grain Reconfigurable Architectures," in *To appear in Design, Automation and Test in Europe Conference Exhibition (DATE)*, 2023.

[26] P. Raghavan, A. Lambrechts, M. Jayapala, F. Catthoor, D. Verkest, and H. Corporaal, "Very wide register: An asymmetric register file organization for low power embedded processors," in *Design, Automation and Test in Europe Conference Exhibition (DATE)*, 2007, pp. 1–6.

[27] F. Catthoor *et al.*, *Ultra-Low Energy Domain-Specific Instruction-Set*

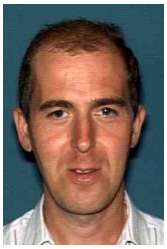
Processors, 1st ed. Springer Netherlands, 2010, ch. An asymmetrical register file: the VWR, pp. 199–222.
[28] Synopsys - PrimePower Q-2019.12. Accessed on: Apr. 4, 2022. [Online]. Available: https://www.synopsys.com/implementation-and-signoff/signoff/primepower.html
[29] F. Dell'Agnola, U. Pale, R. Marino, A. Arza, and D. Atienza, "Mbiotracker: Multimodal self-aware bio-monitoring wearable system for online workload detection," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 15, no. 5, pp. 994–1007, 2021.
[30] R. Braojos, G. Ansaloni, and D. Atienza, "A methodology for embedded classification of heartbeats using random projections," in *Design, Automation and Test in Europe Conference Exhibition (DATE)*, 2013, pp. 899–904.
[31] A. Teman, D. Rossi, P. Meinerzhagen, L. Benini, and A. Burg, "Controlled placement of standard cell memory arrays for high density and low power in 28nm fd-soi," in *The 20th Asia and South Pacific Design Automation Conference*, 2015, pp. 81–86.

Benoît W. Denkinger received his M.Sc. in robotics and autonomous systems from the Institute of Electrical and Micro Engineering, EPFL. He is currently pursuing a Ph.D. degree with the Embedded Systems Laboratory (ESL) at EPFL, Switzerland. His research interests include low-power architectures for biomedical applications and artificial intelligence (AI)-enabled Internet-of-Things (IoT) devices.

Miguel Peón-Quirós received a Ph.D. on Computer Architecture from UCM, Spain, in 2015. He collaborated as a Marie Curie scholar with imec (Leuven, Belgium) and as postdoctoral researcher with IMDEA Networks (Madrid, Spain) and the Embedded Systems Laboratory (ESL) at EPFL (Lausanne, Switzerland). He has participated in several H2020 and industrial projects and is currently part of EcoCloud, EPFL. His research focuses on energy-efficient computing.

Mario Konijnenburg (M'08) received an M.S. degree in electrical engineering from Delft University of Technology in The Netherlands in 1993. A Ph.D. degree was received from Delft University of Technology in 1999 on Automatic Test Pattern Generation for Sequential Circuits. He joined Philips Research / NXP Semiconductors and worked on methodologies to improve testability of designs. Currently, he is chip architect and R&D manager of the IC-design group at imec in Eindhoven, The Netherlands, targeting chip research for bio-medical applications covering SoC design, sensors, stimulators, power, and (RF) communication.

David Atienza (M'05-SM'13-F'16) is a Professor of electrical and computer engineering, Heads the Embedded Systems Laboratory (ESL), and is the Scientific Director of the EcoCloud Center for Sustainable Computing at the École Polytechnique Fédérale de Lausanne (EPFL), Switzerland. He received his Ph.D. in computer science and engineering from UCM, Spain, and imec, Belgium, in 2005. His research interests include system-level design methodologies for high-performance multi-processor system-on-chip (MPSoC) and low-power Internet-of-Things (IoT) systems, including new 2-D/3-D thermal-aware design for MPSoCs and many-core servers, and edge AI architectures for wearable systems and smart consumer devices. He is a co-author of more than 400 papers in peer-reviewed international journals and conferences, one book, and 14 patents in these fields. Dr. Atienza is an IEEE Fellow and an ACM Fellow.



Francky Catthoor received a Ph.D. in EE from the Katholieke Univ. Leuven, Belgium in 1987. Between 1987 and 2000, he has headed several research domains in the area of synthesis techniques and architectural methodologies. Since 2000 he is strongly involved in other activities at imec including co-exploration of applications, computer architecture, deep submicron technology aspects, biomedical systems and IoT sensor nodes, and photo-voltaic modules combined with renewable energy systems, all at imec Leuven, Belgium. Currently, he is an imec senior fellow. He is also part-time full professor at the EE department of the KULeuven.