# An Efficient SRAM-based Reconfigurable Architecture for Embedded Processors

Sajjad Tamimi, Zahra Ebrahimi, Behnam Khaleghi, Hossein Asadi, senior Member, IEEE

***Abstract*—Nowadays, embedded processors are widely used in wide range of domains from low-power to safety-critical applications. By providing prominent features such as variant peripheral support and flexibility to partial or major design modifications, *Field-Programmable Gate Arrays* (FPGAs) are commonly used to implement either an entire embedded system or a *Hardware Description Language* (HDL)-based processor, known as *soft-core* processor. FPGA-based designs, however, suffer from high power consumption, large die area, and low performance that hinders common use of soft-core processors in low-power embedded systems. In this paper, we present an efficient reconfigurable architecture to implement soft-core embedded processors in SRAM-based FPGAs by using characteristics such as low utilization and fragmented accessibility of comprising units. To this end, we integrate the low utilized functional units into efficiently designed *Look-Up Table* (LUT) based *Reconfigurable Units* (RUs). To further improve the efficiency of the proposed architecture, we used a set of efficient *Configurable Hard Logics* (CHLs) that implement frequent Boolean functions while the other functions will still be employed by LUTs. We have evaluated effectiveness of the proposed architecture by implementing the Berkeley RISC-V processor and running MiBench benchmarks. We have also examined the applicability of the proposed architecture on an alternative open-source processor (i.e., LEON2) and a *Digital Signal Processing* (DSP) core. Experimental results show that the proposed architecture as compared to the conventional LUT-based soft-core processors improves area footprint, static power, energy consumption, and total execution time by 30.7%, 32.5%, 36.9%, and 6.3%, respectively.**

## I. Introduction

Embedded systems are used in broad range of applications from indoor utensils to medical appliances and safety-critical applications. The hardware infrastructure of these systems typically consists of an embedded processor as the core and different peripherals specified for the target application [1], [2]. An embedded processor can be implemented as two commonly used platforms, i.e., either *hard-* or *soft-core*. Hard-core processors have received great attention due to the high abstraction level they provide to both developers and users [3], [4]. However, commercially off-the-shelf hard-core processors cannot fulfill design requirements when processor customizations such as *Instruction Set Architecture* (ISA) update is demanded by the customers. In addition, these processors typically provide specific and limited peripheral support that may not fit appropriately to a variety of embedded applications. Moreover, there is always likelihood of obsolescence of a hard-core processor which complicates the migration of the overlying embedded system to newer technologies and leads to expensive and even infeasible design update.

An alternative approach to implement embedded processors is using reconfigurable devices such as *Field-Programmable Gate Arrays* (FPGAs) which is referred to as *soft-core*. A typical soft-core processor, as illustrated in Fig. 1, can be reconfigured on an individual FPGA device during system runtime. The main privilege of soft-core over hard-core counterpart is its intrinsic flexibility to reconfigure itself with workload variations to provide higher level of parallelism which results in (a) shorter time-to-market, (b) inexpensive design update with upcoming technologies, (c) continuous adoption with advancements in SRAM-based FPGAs, and (d) flexibility to implement diverse range of applications. FPGAs, however, are conventionally equipped with abundant logic and routing resources to target general applications which results in high power consumption, large silicon die area, and low performance compared with *Application Specific Integrated Circuit* (ASIC) counterparts [5]. These intrinsic characteristics of FPGAs are in contrary with fundamental requisites of embedded systems [6].

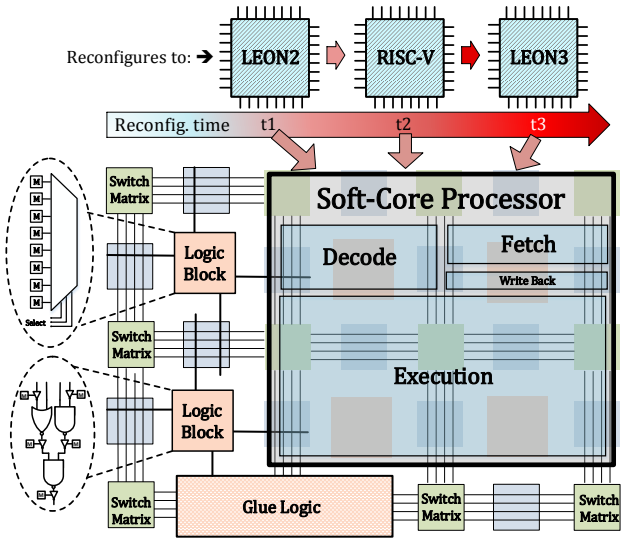


Fig. 1. Online reconfiguration to various soft-core processors

With the limitations of hard-core processors along with infeasibility of fabricating dedicated processors due to limitations in time and/or cost in one hand, and prominent features of FPGAs in the other hand, designers have ever-increasing tendency to use soft-core processors in embedded systems. Various open-source soft-core processors [7]–[12] have been widely used for academic and commercial purposes. Such soft-cores are developed based on full description of a processor from scratch using *Hardware Description Language* (HDL) [13], modification of pre-tested *Intellectual Property* (IP) cores (e.g., MicroBlaze and Nios II), or pre-characterized cores that are developed by higher abstraction level tools (e.g., SPREE [14] and Chisel [15]).

A major challenge of embedded soft-core processors is functional units with low utilization and/or *fragmented accesses*, e.g., floating point multiplier and divider. Such units are accessed sporadically for a short period of time and remain idle for significant amount of time. The large period of idle time can lead to significant static power dissipation which is a key design constraint in battery-powered mobile devices. In particular, with the downscaling of the transistor feature size and threshold voltage in smaller technology nodes, the role of static power has become more pronounced [16], leading to the era of *Dark Silicon* [17]. Furthermore, these low utilized units suffer from larger area (i.e., transistor count), which is another testimony for the ever-increasing static power in embedded systems.

To improve the power consumption of embedded processors, previous studies have suggested to replace the low utilized units with non-volatile reconfigurable units which can be reconfigured whenever an access to an unavailable unit is issued [18], [19]. Such a design suffers from area overhead caused by replacing the static CMOS units with *Look-Up Table* (LUT)-based equivalents. Several previous studies have also considered the use of reconfigurable architectures in conjunction with [20]–[23] or within the FPGA-based processors [18], [19], [24], [25] mainly to speed up the computations. Nevertheless, these architectures have two main drawbacks. First, the suggested reconfigurable architectures are designed *only* for a specific application domain and are not generic. Second, the suggested reconfigurable architectures which have been used within these platforms

are power inefficient. These shortcomings obstruct the use of such architectures in the majority of embedded systems.

In this paper, we propose an area and power efficient reconfigurable architecture for soft-core embedded processors by exploiting the low utilization and fragmented accesses of functional units. The proposed architecture is motivated by large area and high power consumption of the low-utilized functional units in SRAM-based soft-core processors. Such an architecture, however, necessitates a comprehensive profiling to determine the accessibility of processor functional units across a wide range of applications to (a) specify which functional units should be replaced with reconfigurable component and (b) determine how to efficiently integrate these units. For this purpose, we first profile a wide range of benchmarks to determine infrequently-used and fragmentally-accessed units that are the major contributor of the logic static power, and hence, are promising candidates to be merged in an individual *Reconfigurable Unit* (RU). Afterwards, to further improve the area and power efficiency of the proposed RU, we propose an algorithm to efficiently integrate these components as shared RUs. The proposed algorithm aims to minimize the number of configuration cells and logic resources in the RU by exploiting the similarities along with the non-uniform distribution of Boolean functions in the components. These homomorphic-structure functions are implemented as a set of small area-size *Configurable Hard Logics* (CHLs). To our knowledge, this is the first study that, in a fine-grained manner, improves the area and static power efficiency of FPGA-based soft-core processors using the concept of spatiality in component utilization. The proposed architecture can be employed in tandem with other methods that intend to architecturally improve the soft-core processors, e.g., multi-threading [26], power gating [27]–[29], or ISA customization [30], as they are orthogonal to our proposed architecture.

To evaluate the efficiency of the proposed architecture, we first execute MiBench benchmarks [31] on Berkeley open-source RISC-V processor [8] to obtain the access patterns and utilization rate of the RISC-V units. Afterwards, using an in-house script, we identified the candidate units to be integrated in an optimal number of RUs. Berkeley ABC [32] and Altera Quartus [33] are exploited for synthesizing the selected units and integrating into a shared RU. Thereafter, latest version of COFFE [34] is exploited to generate efficient transistor sizing used in HSPICE. Finally, we use Verilog-To-Routing (VTR) 7.0 [35] fed with the parameters obtained from HSPICE and Design Compiler to place and route the processors and obtain the design characteristics. Furthermore, we examine generality of the proposed architecture on an alternative open-source processor (LEON2), as well. Experimental results demonstrate that our proposed architecture improves the area, static power, energy, critical path delay, and total execution time of RISC-V by 30.7%, 32.5%, 36.9%, 9.2%, and 6.3%, respectively, compared to the conventional modular LUT-based soft-core processors. Examining the applicability of the proposed architecture on LEON2 revealed that the area, static power consumption, and energy of LEON2 is enhanced by 23.1%, 23.2%, and 23.2%, respectively.

**The novel contributions of this work are as follows:**

- We present a comprehensive design space exploration among prevalent infrastructures for datapath computational units of a processor to discriminate the proper infrastructure for embedded systems. This suggests integration of low utilized and fragmentally-accessed components into shared RUs.
- For the first time, we propose a metric to efficient integration of units in elaborately designed RUs based on component area, utilization ratio, and configuration time.
- By comprehensive characterization of most frequent and homomorphic-structure functions, we propose an area and power efficient synthesis algorithm which exploits efficient CHLs for implementing significant portion of functions in the proposed RUs. To the best of our knowledge, this is the first effort which investigates the applicability of CHLs in either soft or hard processors.
- We investigate the generality of the proposed architecture on an alternative embedded processor (LEON2) as well. In addition, to examine the insensitivity of shared RU, we use both academic and commercial synthesis tools.

The rest of this paper is as follows. The related work is reviewed in Section II. The motivation behind the proposed architecture is explained in Section III. We articulate the proposed architecture in Section IV. Details of the experimental setup and results are elaborated in Section V. Finally, Section VII concludes the paper.

## II. Related Work

Different approaches have been proposed to reduce the area and power of soft-core processors as well as to increase their performance, which can be classified in two categories. Approaches in the first category employ three techniques, (a) static or (b) dynamic power gating of components of execution stage [27]–[29], [36], and (c) dual threshold voltage to reduce power. Static power gating is applied offline at the configuration time. Therefore, the power efficiency of this technique is restricted to permanently-off components which depends on the target application. Dynamic power gating is applied online, i.e., during the application runtime, so it may have higher opportunity for power reduction. In practice, however, dynamic power gating encounters several shortcomings that hinders practicality of this technique. We will discuss advantages of the proposed architecture over dynamic power gating method in Section V.

As the second category, using reconfigurable devices to implement contemporary processors is proposed. Previous studies in this category can be classified as coarse-grained and fine-grained architectures. The former allows bit-level operations, while the latter enables relatively more complicated operations. In the coarse-grained architectures [20]–[23], the reconfigurable devices are used as *co-processor* to speed up computations and enhance the parallelism degree and energy efficiency by loop unrolling. The role of compiling algorithms, i.e., hardware-software partitioning, considering today's complex many-core architectures and multi-programming can adversely affect the efficiency of such architectures [37]. Thus, aforementioned architectures impose considerable area overhead of profiler and departure compilers from their straightforward flow to support branch prediction which may not always be correct in interactive and time-dependent embedded applications [23].

In fine-grained architectures, one or several types of instructions are selected to be executed on the reconfigurable platform. The main challenge in such architectures is choosing computation-intensive instructions. In this regard, CHIMAERA [24] has proposed integrating an array of *Reconfigurable Functional Units* (RFUs) into the pipeline of a superscalar processor which are capable of performing 9-input functions. The authors also proposed a C compiler that attempts to map the instructions into RFUs [24]. The OneChip architecture that integrates an RFU into the pipeline of a superscalar RISC processor has been proposed in [25]. The RFU is implemented in parallel with *Execute* and *Memory Access* stages and includes one or more FPGA devices and a controller. Thus, it can be internally pipelined and/or parallelized. In addition to significant area and power overheads imposed by multiple reconfiguration memories and additional circuitries, design complexity (e.g., modifying the compiler) is another drawback of the OneChip architecture. A heterogeneous architecture exploiting RFUs based on *Spin-Transfer Torque Random Access Memory* (STT-RAM) LUTs for the hard-core general-purpose

TABLE I
SUMMARY OF EXPLOITING RECONFIGURABLE FABRICS IN PROCESSORS

| | Granularity | Modification | Structure (Topology) | Improvement/Design Overhead |
|---|---|---|---|---|
| **MorphoSys [20]** | Coarse-grained | Not mentioned | 8x8 array of 32-bit ALU/multiplier, register file, modified TinyRISC processor, and memory unit | ✓ Performance<br>✕ No compiler support, no area and power evaluation |
| **ADRES [21]** | Coarse-grained | Developing specific compiler framework | VLIW processor with reconfigurable matrix and register file | ✓ Performance ✕ Large configuration overhead, limited benchmarks, no area and power evaluation |
| **PipeRench [22]** | Coarse-grained | Developing specific compiler framework | Array of *Processing Elements* (PEs) contains 8-bit ALU (LUT and carry chain) and register file | ✓ Reconfiguration time<br>✕ Performance (PEs communicate through global I/O), large bitstream, no area and power evaluation |
| **Warp [23]** | Coarse-grained | Not mentioned | ARM7 microprocessor with a FPGA contains 32 bit MAC, loop control hardware, and data address | ✓ Performance and energy due to loop unrolling<br>✕ 3-4X area overhead |
| **CHIMAERA [24]** | Fine-grained | Modified GCC compiler | Dynamically scheduled out of order superscalar processor contains 32 rows of 4-LUTs, carry chain, shadow register file, cache, and control unit | ✓ Performance<br>✕ No area and power evaluation |
| **OneChip [25]** | Fine-grained | Developing specific simulator framework | Superscalar pipeline,processor integrated,with RFU (parallel to EXE and MEM stages) | ✓ Performance<br>✕ No power evaluation |
| **[18], [19]** | Fine-grained | Reconfiguration and task migration algorithm | Replacement of ASIC-based low utilized functional units with STT-RAM 3-LUTs in IBM PowerPC | ✓ Performance and power<br>✕ Area overhead |
| **Proposed** | Fine-grained | No modification required | Integrating low utilized units in shared reconfigurable units in embedded processors (RISC-V and LEON2) | ✓ Area, power, performance, and energy<br>✕ Reconfiguration overhead (slight) |

processors has been proposed in [18]. The proposed RFUs attempt to mitigate unavailability of functional units using either static or dynamic adaptive algorithms for reconfiguration. In the static algorithm, during a learning phase, idle functional units are identified, and are reconfigured to most active units to provide higher availability. In the dynamic algorithm, the reconfiguration decision is made periodically while at the beginning of each interval, idle units are replaced with the active ones. Although 18% improvement in performance has been reported, considerable area and power overhead is expected due to larger area and higher power consumption of a reconfigurable functional unit compared with custom ASIC or standard cell based components. More importantly, substantial modifications in both compiler and processor architecture, especially bus lines and datapath, are required which are neglected in this study. A similar approach along with thermal-aware reconfiguration of functional units has been proposed in [19], wherein computations of thermally hotspot functional units are migrated to cooler units in order to balance the temperature distribution. Aforementioned drawbacks for [18] stand here, as well. The summary of previous studies is reported in Table I.

## III. MOTIVATION

The main prerequisite to integrate several functional units into a single shared RU is low utilization and fragmented access of each unit. Low utilization is a crucial constraint since implementing even few high utilized components as a single RU is inefficient as frequent access to such components leads to substantial execution time and power penalties. Fragmented accesses is another important criterion which facilitates integration of several functional components into a single RU. To comprehend this, assume two components $C_1$ and $C_2$ both having 50 percent utilizations. Accordingly, two access patterns to these components could be either as:

(a) $\{C_1, C_1, C_1, C_1 \rightarrow C_2, C_2, C_2, C_2 \rightarrow C_1, C_1, C_1, C_1 \rightarrow ... \rightarrow C_2, C_2, C_2, C_2\}$

(b) $\{C_1 \rightarrow C_2 \rightarrow C_1 \rightarrow C_2 \rightarrow ... \rightarrow C_1 \rightarrow C_2\}$

wherein an arrow ($\rightarrow$) denotes a need to reconfiguration. While both scenarios result in the same utilization rate, obviously, the former requires fewer number of reconfigurations.

### A. RISC-V Processor

We have examined these conditions by running MiBench benchmarks on RISC-V processor. RISC-V is a 64-bit in-order, open-source embedded processor suited for FPGA devices [38], [39] which consists of six pipeline stages, wherein its execution stage is equipped with both integer and floating point units, as summarized in Table II. Fig. 2 illustrates RISC-V components proportional to their area footprint on the silicon die. As shown in this figure, functional units occupy substantial portion (more than 44%) of total processor area (including caches). Area of caches and memory cells are calculated using CACTI (version 6.5) [40]. We modelled default RISC-V cache configuration in CACTI for both of I-cache and D-cache which both have 16 KB size with 16-bytes block size, 4-way associate, 32-bit input/output address and data bus, and tag size of 20 for I-cache and 22 for D-cache. Other parts of the processor such as fetch, decode,

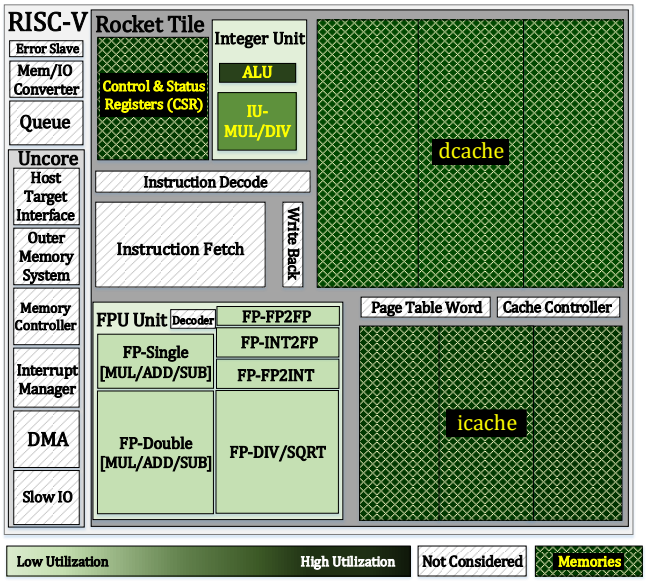


Fig. 2. RISC-V processor (area of each block is normalized to the chip area)

TABLE II
COMPONENTS UTILIZATION AND AREA COMPARISON IN RISC-V

| Computational Units in Processor Core | | Operation Descriptions | # 4-LUTs (Area ratio without Memories) | Utilization |
|---|---|---|---|---|
| Integer Unit | ALU | Integer Addition and Subtraction | 1150 (2.4%) | 44.8% |
| | IU-MUL/DIV | Integer Multiplication and Division | 4573 (9.5%) | 1.8% |
| Floating Point Unit (IEE754 2008 standard) | FP-Double [MUL/ADD/SUB] | Double Precision Multiplication, Addition, and Subtraction | 10918 (22.9%) | 0.16% |
| | FP-INT2FP | 32/64-bit Integer Numbers to Single/Double Precision Floating Point Format Conversion | 741 (1.5%) | 0.09% |
| | FP-FP2INT | Single/Double Precision Floating Point Numbers to 32/64-bit Integer Format Conversion | 2330 (4.3%) | 0.03% |
| | FP-FP2FP | NaN (not a number) values to an acceptable numbers Conversion | 2074 (19.4%) | 0.02% |
| | FP-Single [MUL/ADD/SUB] | Single Precision Multiplication, Addition, and Subtraction | 3152 (6.58%) | 0.01% |
| | FP-DIV/SQRT | Single/Double Precision Division and Square Root Operations | 9269 (4.9%) | 0.001% |
| Other Pipeline Stages | Fetch | Instruction Fetch Unit | 11140 (23.3%) | - |
| | Decode | Instruction Decode Unit | 1974 (4.1%) | - |
| | Write Back | Write Back Unit | 571 (1.2%) | - |

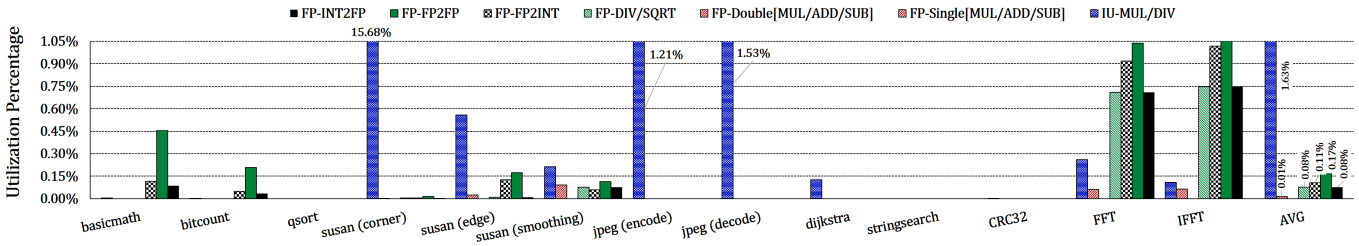


Fig. 3. Utilization rate of RISC-V components running MiBench benchmarks

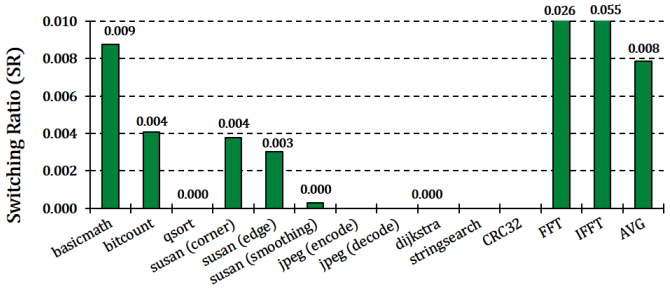


Fig. 4. Average inter-component switching ratio in the proposed RUs

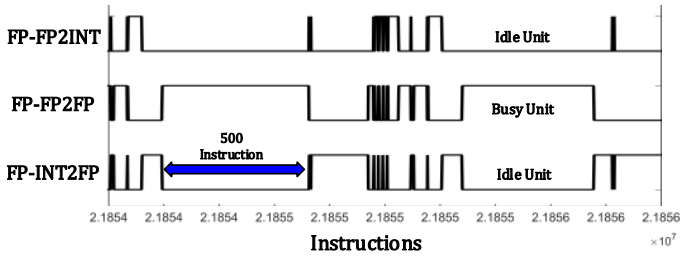


Fig. 5. Access pattern to low-utilized functional units in RISC-V processor

write-back stage, and peripheral units are illustrated by shaded blocks. Diamond blocks are memories such as registers and caches.

### B. Resource Utilization and Access Patterns

The utilization rate of RISC-V functional units over $3.4 \times 10^9$ instructions for MiBench benchmark is illustrated in Fig. 3. As shown in this figure, the highest accessed unit (i.e., IU-MUL/DIV) has been utilized only for 0.8% on average in all benchmarks. Therefore, the first condition, i.e., low utilization, is fulfilled. It should be noted that the *Arithmetic Logic Unit* (ALU) which has utilization rate greater than 43.3% is excluded from the results. The ALU is frequently used by other pipeline stages, e.g., instruction and decode stage.

Fragmented accesses to the low-utilized units in processor data path can alleviate the reconfiguration overhead and, subsequently, the execution time. To specifically clarify the importance of fragmented accessibility, we use the *Switching Rate* (SR) parameter (i.e., number of inter-unit switching to the total instructions) between components for each benchmark[1]. For example, if accesses to three units are $\{C_1, C_1 \rightarrow C_2 \rightarrow C_3, C_3, C_3 \rightarrow C_2\}$, then SR is $\frac{3}{7}$ where 3 is the number of inter-unit switching and 7 is the total number of instructions. The results reported in Fig. 4 show that, on average, SR is $4.9 \times 10^{-3}$ in MiBench benchmarks, which indicates one switching per 204 instructions (i.e., $\frac{1}{0.0049}$). This value would become one switching per 535 instructions if FFT benchmark is excluded.

Notice that some benchmarks utilize only a unique component during the whole execution time. For example, *jpeg* benchmark only utilizes MUL-DIV unit as it can be observed in Fig. 3. Thus, SR for such benchmarks is zero. This observation has further motivated us to propose one individual RU instead of several under-utilized units that may never be used in most applications. As an example, Fig. 5 illustrates the fragmented accessibility of three functional units within FPU running the FFT benchmark which has the highest SR. As shown in this figure, for a long sequence of instructions, one of the units in the FPU unit is busy while the other two components are idle. This motivates the concept of shared reconfigurable unit from two perspectives. First, since at most one of the internal units in the FPU unit is accessed each time, integrating these internal units is conceivable. Second, the large idle time of unused units imposes significant leakage power, particularly considering their large area.

[1]We take the average in the benchmarks having multiple subs.

## IV. Efficient SRAM-based Reconfigurable Architecture

Here, we first explore the design space for possible implementation of functional units inside a processor and discuss advantages and limitations of each design style. Afterwards, we elaborate our proposed architecture.

### A. Design-Space Exploration

Based on the underlying infrastructure, i.e., ASIC or *ReConfigurable* (RC), the implementation structures for functional units can be classified in four categories which are demonstrated in Fig. 6. We will compare these design concepts against the proposed architecture with respect to area and performance.

**ASIC/module and RC/module:** These architectural design styles (shown in Fig. 6.b and Fig. 6.d, respectively) are the conventional architectures for implementing hard-core and soft-core processors. One major drawback of these design approaches is that the access pattern to the target module has not been well-considered. As it will be discussed later, there are infrequently-accessed modules within embedded processors that dedicating separate units for each of them is inefficient in terms of area and power. It has been reported that area and performance gap of ASIC/module and RC/module implementations of total processor core are 17-27x and 18-26x, respectively. A detailed comparison of processor main building blocks with respect to area and delay has also been presented in [41].

**ASIC/inst:** Designing a specific module per each instruction, as shown in Fig. 6.a, can significantly speed up the logic operations by making them dedicated, hence, small and efficient. Nevertheless, its main drawbacks such as complicated control unit and large address buses cause significant area and power, which limits performance gain, and makes it impractical to implement all instructions.

**RC/all:** In this implementation, a single reconfigurable unit is dedicated to all instructions which is represented in Fig. 6.c. This design style incurs significant reconfiguration overhead since it requires to be reconfigured once a new instruction needs to be executed. Area (i.e., the number of LUTs) in such an architecture is determined by area of the largest module. Beside large execution time of applications, the potential area efficiency of such architecture will be shaded if the design comprises a set of small and frequently used modules and large but infrequently used ones.

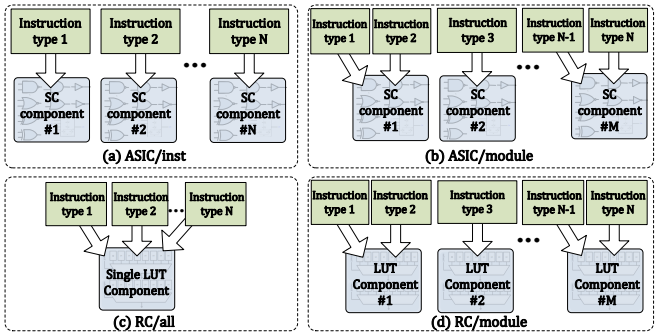


Fig. 6. Baseline architectures of implementing the functional units: (a) ASIC/inst (b) ASIC/module (c) RC/all (d) RC/module

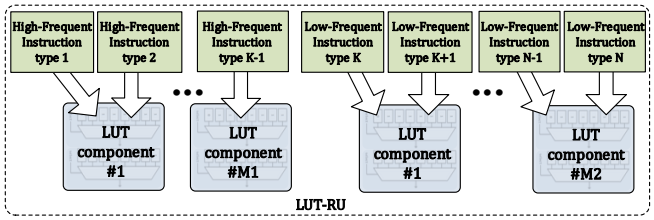


Fig. 7. The proposed LUT-RU architecture overview

### B. Proposed Architecture

The base of the proposed architecture is to integrate the low-utilized and fragmentally-accessed units into shared RUs to reduce logic resources and the number of configuration bits in the processor. Hence, this will improve area and power efficiency of the soft-core processor. To achieve an efficient architecture, we take the following steps: (a) finding the low-utilized candidate units to be integrated in RUs, (b) appropriate grouping of the candidate units into single or multiple RUs, and (c) optimizing the proposed architecture by exploiting the similarities in Boolean functions of associated units. We explain each step in the following.

*1) Candidate Units:* As discussed in Sec. III, low utilized and fragmentally-accessed units (i.e., small number of switching to/from a unit) is essential to migrate them to an RU. These parameters are determined by running the target application benchmarks and probing the trace file. For the RISC-V processor, as shown in Fig. 3, ALU and IU-MUL/DIV units have relatively high utilization (44.8% and 1.5%, respectively) compared to the average utilization rate of the floating-point units (0.24%). On the other hand, the area of the ALU is smaller than the FPU. Therefore, it is not effective to migrate the ALU unit into a reconfigurable unit. As presented in Table II, FPU unit occupies a significant fraction (more than 59.5%) of the processor area while its utilization is only about 0.24%. The same trace is used to obtain the total number of switching between units for each benchmark and estimate the reconfiguration overhead.

*2) Grouping into RUs:* The critical step in the proposed architecture is determining the number of RUs and assigning the candidate units to each unit. There is a trade-off between the number of RUs and total execution time. Smaller number of RU results in area-efficient architecture (i.e., more units are bounded in an RU) but increases the overall execution time due to the high number of reconfigurations in these units. We define the efficiency goal ($\lambda_{eff}$) as minimizing the $area \times execution\ time$ according to Equation 1.

$$\lambda_{eff} = Area \times T_{execution} \tag{1}$$

In this equation, $T_{execution}$ is composed of two parts, (a) required time to execute the instructions and (b) reconfiguration overhead (which can be the dominant part). Even in the case that reconfiguration is performed infrequently (e.g., once per 500 instructions), the reconfiguration time (which is typically more than $50us$) will dominate the overall execution time (can be estimated as $500 \times 5ns = 5us$ assuming a clock cyle of $5ns$). As a result, we can alter this formula to $\lambda_{eff} \propto Area \times T_{rcfg}$. In addition, the critical path delay of a circuit is not affected by the number of units inside an RU since each time only one component is implemented, and thereby, can take its original mapping. Assuming $N$ RUs, $\lambda_{eff}$ can be obtained from Equation 2. In this equation, area of each RU (in term of number of LUTs) is equal to the area of its largest candidate unit. Thus, integrating Components (denoted by $C_j$) with similar area/size into an individual RU is more prosperous.

$$\begin{aligned} \lambda_{eff} &\propto \sum_{i=1}^{N} Area(RU_i) \times T_{rcfg} \\ &= \sum_{i=1}^{N} max\{Area(C_j) | C_j \in RU_i\} \times T_{rcfg} \end{aligned} \tag{2}$$

The number of reconfigurations in a particular RU can be obtained from the original benchmark trace by discarding the non-existing units in that RU. For instance, if the original instruction sequence is $\{C_1, C_2, C_1, C_3, C_1, C_2, C_3, C_2\}$ and only $C_1$ and $C_2$ are integrated inside this RU, then the number of reconfigurations, i.e., $T_{rcfg}$, would be equal to three, (i.e., $\{C_1 \xrightarrow{1} C_2 \xrightarrow{2} C_1, \cancel{C_3}, C_1 \xrightarrow{3} C_2, \cancel{C_3}, C_2\}$). We carefully analyzed $\lambda_{eff}$ for possible architectural assortments of grouping six candidate components into two or three RUs which has 40 different cases as summarized in Fig. 8. All $\lambda_{eff}$ in this figure are normalized to the minimum case (i.e., the case with $RU1 = [1, 4]$, $RU2 = [2, 3]$, and $RU3 = [5, 6]$). For the sake of brevity, we have omitted the results of integrating candidate units into four and five RUs, since such classification does not gain much area-power improvement. Based on the results reported in Fig. 8, we propose three RUs based on the best classification, which is detailed in Table III. i.e., RU1=(FP Single[MUL/ADD/SUB], FP2INT), RU2=(FP Double[MUL/ADD/SUB], FP-DIV/SQRT), and RU3=(FP2FP, INT2FP).

Exploiting the concept of integrating several low-utilized and fragmentally-accessed units in one RU (LUT-RU architecture) improves several design parameters. First, the number of reconfigurable units in the pipeline stages will be reduced, and therefore the complexity of the control unit and address bus will be reduced. Second, number of write-back ports decreases, which consequently reduces the number of write-back buffers. It should be noted that the proposed architecture would not provide considerable advantage when (a) functional units are high utilized and have frequent access patterns that may cause considerable execution time overhead and (b) functional units differ significantly in size, e.g., merging two component with $1\times$ and $20\times$ LUTs may not be beneficial since the context switching overhead can shallow the small area gain.

### C. Optimizing RUs

By following the previous steps, the proposed LUT-RU architecture could be achieved. Here, we aim to further improve the efficiency of the proposed architecture by reducing the number of resources i.e., transistors and configuration cells, by using area and power efficient

TABLE III
GROUPING CANDIDATE UNITS INTO PROPOSED RUS

| RUs | Area (4-LUT) | Description |
|---|---|---|
| RU1 | 3152 | FP-Single[MUL-ADD-SUB] |
| | | FP-FP2INT |
| RU2 | 10918 | FP-Double[MUL-ADD-SUB] |
| | | FP-DIV/SQRT |
| RU3 | 2074 | FP-INT2FP |
| | | FP-FP2FP |

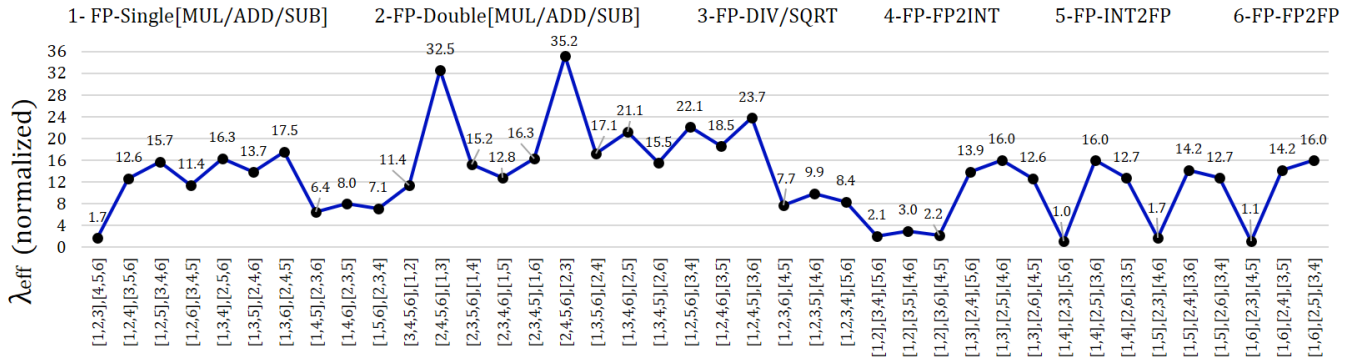


Fig. 8. Various architectural classification based on $\lambda_{eff}$

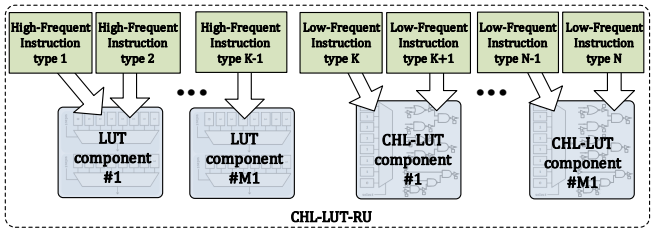


Fig. 9. The proposed CHL-LUT-RU architecture overview

CHLs instead of conventional LUTs. Previous studies have shown that 4-LUT based FPGA architecture has the minimum area among different LUTs [42]. Nonetheless, even in such a small-input LUT, the majority of LUT structure remains underutilized in a broad range of circuits [43]–[45]. For example, a considerable portion of functions do not necessarily have 4-inputs, so half of LUT configuration bits and multiplexer structure, or more, is wasted. Even if all inputs are used, a small set of specific functions are repeated significantly higher than others. Therefore, there is no need to allocate generous flexibility of 16 SRAM cells to implement them. The high number of SRAMs in LUT-based logic blocks can exacerbates the area and power consumptions in FPGA and soft-core processors. In recent studies, several fine-grained architectures have been proposed that use small reconfigurable logic cells solely or along with conventional LUTs [43]–[45]. CHLs are logic blocks that can implement the majority of functions with considerably smaller area and less delay as compared to their LUT counterparts. However, the main goal of these studies is enhancing of design constraints in FPGA platforms while their applicability in soft-core processors is not evaluated.

Accordingly, we have proposed an area-power optimization approach, i.e., CHL-LUT-RU, where for each RU, we synthesize each component (which is assigned to the target RU) into 4-input LUTs by the means of Berkeley ABC synthesis tool [32] to obtain 4-input functions and then extract its *Negation-Permutation-Negation* (NPN) class representation. To elaborate, $F = A\bar{B} + CD$ and $G = BC + AD$ are NPN-equivalent since each function can be obtained from the other by negating B and permuting A and C. Therefore, such functions can be implemented with the same homomorphic logic structure augmented with configurable inverters at its inputs and output. Our investigation, reported in Table IV, has revealed that a significant portion (66.8%) of functions in the proposed RUs have the same homomorphic structure based on the concept of NPN-class representation and can effectively be implemented with CHL1 and CHL2 which have been primarily introduced in [44]. It should be noted that although applicability of CHLs is motivated by function characterization in RISC-V processor, their effectiveness is also examined in LEON2 processor and large-scale commercial FPGA benchmarks [44], [45] wherein some of these benchmarks are soft-core processors and microprocessor by themselves. For instance, our investigation has revealed that CHL1 and CHL2 can implement 68% of functions in OR1200 in VTR benchmark suite and 59% of functions in TV80 benchmark in IWLS'05. It is also noteworthy that any generic function, even unidentified at design time, can be implement using pure CHLs. Albeit, in such non-LUT designs there will be an overhead in the number of used logic blocks (CHLs) due to elimination of 4-LUT and pure exploiting of CHLs which are less flexible than LUT in one-to-one implementation of some functions.

Fig. 10 illustrates the proposed *Configurable Logic Block* (CLB) wherein cluster size (N) is equal to 10 and CHL1:CHL2:LUT ratio is equal to 4:3:3, based on observations obtained from Table IV. Each proposed CLB reduces area by 28.6% compared to homogeneous LUT-based CLB. The proposed CHL-LUT-RU approach iterates over functions of components mapped to an RU and examines whether the function could be substituted with either one of CHLs. Otherwise, it will remain as a 4-LUT.

### D. Insensitivity of Proposed Architecture to Different Synthesis Tools

In addition to an academic synthesis tool (Berkeley ABC), we repeat the mapping flow with a commercial tool (Altera Quartus Integrated Synthesis (QIS) tool [33]) to investigate the sensitivity of the results to CAD flow. To this regard, QUIP targeting Startix IV device is exploited which directly synthesizes a VHDL, Verilog, or SystemVerilog circuit to hierarchical BLIF. Our investigation reveals that the coverage ratio of CHLs is almost the same obtained by an academic synthesis tool (Berkeley ABC). The comparison of the CHLs coverage ratio in both Berkeley ABC and Altera Quartus is detailed in Table IV. It is noteworthy that default synthesis constraint of ABC is area optimization while Quartus aims to reduce the delay which results in larger area (number of LUTs) and therefore power consumption [33]. To be consistent with other studies, we evaluate our proposed architecture using netlists of Berkeley ABC.

## V. Experimental Setup and Results

Here, we elaborate the implementation and evaluation flow of the proposed and baseline architectures. According to the overall evaluation flow illustrated in Fig. 11, we first examine the access pattern to each of the datapath components to identify low-utilized and

TABLE IV
Coverage Ratio of 4-Input NPN-Classes in Low-Utilized Component Functions in RISC-V Processor

| Cell | NPN-Class | Berkeley ABC | Altera Quartus |
|---|---|---|---|
| CHL1 | ABCD | 38.4% | 33.4 % |
| | AB+CD | | |
| | AB(C+D) | | |
| CHL2 | A(B+CD) | 28.4% | 21.1% |
| | A(B+C+D) | | |
| 4-LUT | Others | 33.2% | 45.5% |

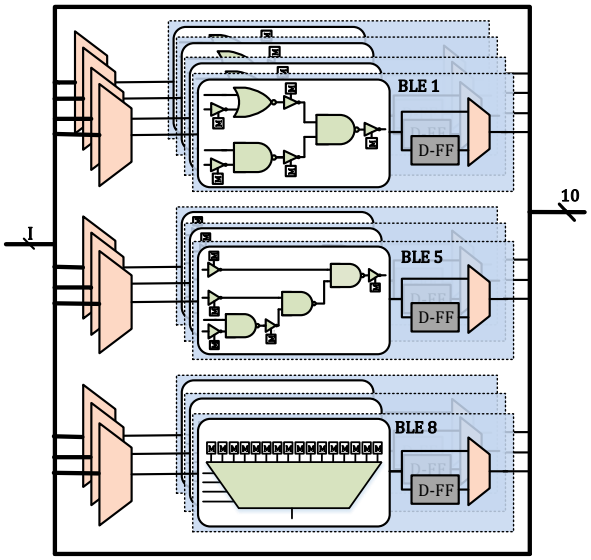


Fig. 10. Proposed Configurable Logic Block (CLB)

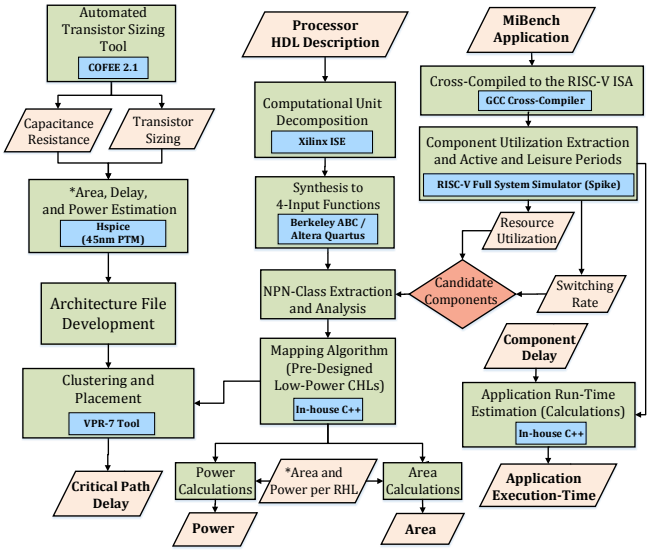


Fig. 11. Implementation and evaluation flow of the proposed architecture(s)

fragmentally-accessed ones. For this aim, each MiBench benchmark is cross-compiled to the RISC-V ISA using modified GCC [8]. Then, the utilization and access pattern to the functional units is obtained by the means of RISC-V ISA Simulator (Spike) [46]. Next, HDL description of each component is extracted from the processor HDL source code and then optimized and mapped to 4-input functions by the means of Berkeley ABC synthesis tool [32] and Altera Quartus [33]. The truth-table of each function is generated using an in-house C script and is fed to the Boolean Matcher [47] to extract the corresponding NPN-Class representation. We exploit COFFE [34] for efficient transistor sizing and area and delay estimation of LUT-based architectures (according to commercial architectures), while evaluation of the CHLs is performed using HSPICE circuit-level simulations with Nangate 45nm Open Cell Library [48]. The area and delay values of each cell are fed to the VPR [35] architecture file for circuit placement on FPGA-based platform. To obtain more accurate and reliable results, we run VPR simulations using different initial placements (i.e., different placement seeds) since the timing and area results are sensitive to initial placements. Lastly, area and critical path delay are extracted directly from VPR while power is estimated by post-processing VPR-generated reports such as number of logic blocks and routing switches (channel width). An in-house C script is used to estimate execution-time based on profiling the instruction execution flow reported by RISC-V ISA Simulator (Spike). In the C script, we have considered the switching time overhead which is needed whenever an RU should be reconfigured.

TABLE V
PARAMETERS IN THE PROPOSED AND BASELINE RC ARCHITECTURES

| Parameter | Definition | Value |
|---|---|---|
| N | Cluster (CLB) size | 10 |
| K | LUT size | 4 |
| I | Cluster inputs (from adjacent routing channels) | 22 |
| $F_{cin}$ | CB connectivity factor (determines no. of channel tracks connected to each CB) | 0.16 |
| $F_{cout}$ | Cluster output connectivity factor (determines no. of SBs a logic block output is connected to) | 0.1 |
| X | LUT input multiplexer size | 16 |
| $F_{fb}$ | Cluster feedback factor (determines no. of other logic blocks output in LUT input mux) | 5/16 |

TABLE VI
CONFIGURABLE HARD LOGICS AND LUT PARAMETERS

| Cell | Delay (ps) | Static Power (nW) | Config. Cell | Area (min width transistors) | Area ($\mu m^2$) |
|---|---|---|---|---|---|
| CHL1 | 67 | 180 | 5 | 72 | 8.8 |
| CHL2 | 69 | 180 | 5 | 72 | 8.8 |
| 4-LUT | 155 | 1388 | 16 | 267 | 32.5 |

### A. Preliminary

Table V demonstrates the architectural parameters and the description for the proposed and baseline architectures used in the VPR experiments. Similar to commercial FPGA devices [49], [50], we used the conventional island-style architecture exploited in VPR tool. To calculate the area, we first extracted total area from VPR post routing reports which uses minimum width transistor model [34] represented by Equation 3 in which $x$ is the ratio of transistor width to minimum width. For instance, a minimum-sized inverter area in the 45nm technology can be obtained by setting $x = \frac{90nm}{90nm} = 1$ for the NMOS, and $x = \frac{135nm}{90nm}$ for the PMOS transistor.

$$Min.\ width\ transistor(x) = 0.447 + 0.128x + 0.391\sqrt{x} \quad (3)$$

While Equation 3 can be directly used for comparison purpose, to obtain the geometric (i.e., layout) area, it is shown that multiplying it to $60F^2$, in which F is the technology size, gives an acceptable layout-level area [34], [35].

$$Area(x) = Min.\ width\ transistor(x) \times 60F^2 \quad (4)$$

### B. Area

According to Table III, low utilized components are integrated into 3 RUs with 3152, 10918, and 2074 LUTs. Therefore, the number of LUTs is decreased by 26.8% from 43743 (which is the total number of LUTs in all functional units in RC/module) to 32019 (number of LUTs in the proposed architecture). Taking into account that by reducing logic resources, routing resources are also reduced; hence, number of configuration bits in the proposed architecture is decreased by 30.9%, as shown in Fig. 12. Area of LUT-based architectures is first calculated with respect to the dimension-less area estimation in term of minimum width transistor count, and then the layout-level area is obtained using Equation 4. The layout area of ASIC-based structures is directly reported by Synopsys Design Compiler

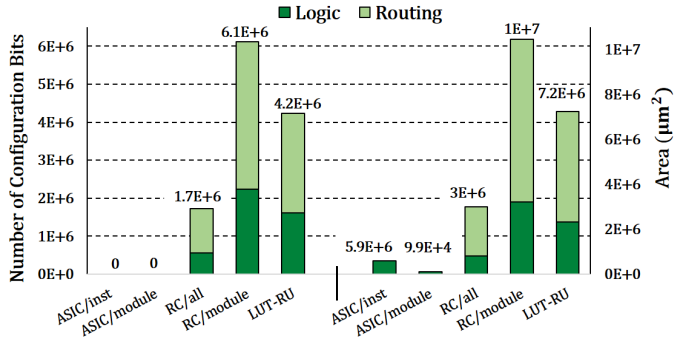


Fig. 12. Comparing the number of configuration bits and area of RISC-V components in the baseline and proposed architectures

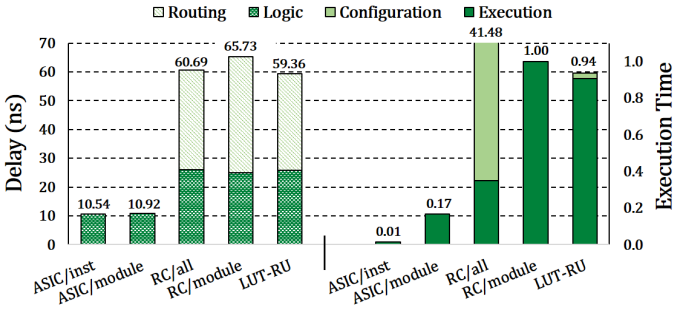


Fig. 13. Comparing the delay of functional units and execution time in different architectures in RISC-V processor

and then is multiplied to the number of instances. According to Fig. 12, the proposed LUT-RU architecture has reduced the area by 30.7%, compared with the RC/module baseline architecture. As expected, this improvement is lower than the improvement in the number of LUTs since some CHL cells have been substituted with LUTs.

### C. Critical Path Delay

The delay of the ASIC-based architectures is obtained from Synopsys Design Compiler. As shown in Fig. 13, the LUT-RU architecture does not affect the logic delay compared with the baseline RC/module since once a component, e.g., FP-FP2F, is configured in either RC/module or LUT-RU architectures, it would have identical delay in both architectures due to the same mapping and logic resource usages. Yet, the routing delay will be reduced since the channel width in LUT-RU architecture is considerably less than RC/module. Hence, the proposed LUT-RU architecture reduces critical path delay by 9.2% compared to the baseline RC/module architecture.

### D. Application Execution Time

To calculate the execution time, we multiplied execution time of each instruction to its corresponding component delay. In addition, for the proposed architectures, we added the reconfiguration time by calculating the write latency of the total number of configuration bits (sum of logic and routing) with the method proposed in [18][2]. Accordingly, we assume a 128-bit parallel data bus [18] and a write latency of 0.2ns for a single SRAM cell [51]. Thus, reconfiguration time of RU1, RU2, and RU3 in LUT-RU architecture will be $\frac{424619\times0.2ns}{128} = 0.66\,\mu s$, $\frac{1481035\times0.2ns}{128} = 0.23\,\mu s$, and $\frac{337531\times0.2ns}{128} = 0.52\,\mu s$, respectively, in which 424619, 1481035, and 337531 are the configuration bit count of the largest component in each RU. Due to the fact that the configurations time is prorated in overall execution time of the application, LUT-RU architecture outperforms the RC/module architecture by 6.3% improvement in execution time, as shown in Fig. 13.

[2]It is noteworthy configuring one component to another in [18] has been calculated based on configuring configuration cells of the smallest component which is not correct.

### E. Power and Energy Consumption

The total static power for each architecture is obtained by post-processing the VPR reports by summing the power of logic and routing resources in the target architecture. Notice that we consider the static power of both utilized and unutilized LUT/CHL cells in power and energy computation despite that the RUs do not use all cells. Fig. 14 compares the normalized static power consumption in different architectures. Compared with the baseline RC/module architecture, the proposed LUT-RU architecture reduces the power consumption by 32.5%. The static power of both logic and routing is also decreased since the LUT-RU architecture reduces both of the logic and routing resources. In addition, while the proposed architecture is most power-efficient in reconfigurable architectures and reduces the number of logic resources, however, the conventional ASIC implementation, i.e., ASIC/module, is still power-efficient than all other architectures.

The energy consumption of different architectures are obtained by Equation 5. Energy is estimated as summation of reconfiguration and operation energy while reconfiguration energy is zero in ASIC-based architectures. $Power_{total}$ stands for total power of RUs or total power of resources in baseline architectures and non-reconfiguring components (i.e., Integer Unit). For reconfiguration, a write energy of $3.5 \times 10^{-18} J$ for a single SRAM cell in 45nm technology is assumed [51]. Fig. 14 compares the normalized energy in the proposed and baseline architectures. As demonstrated in this figure, LUT-RU architecture improves the energy by 36.9% as compared to the baseline RC/module architecture.

$$E = (\sum_{i=1}^{N} \#SRAM(RU_i) \times \#Rcfg(RU_i) \times E_{SRAM}) + Power_{total} \times Execution\ time \quad (5)$$

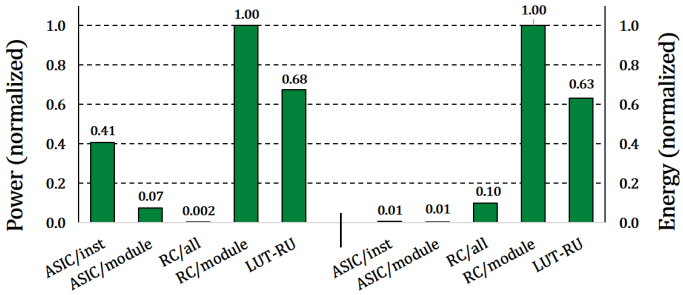


Fig. 14. Comparing the power and energy consumption of different architectures implementing the RISC-V processor

It is noteworthy that the proposed architecture advances power-gating approach in different aspects. First, the most challenging issue of power-gating is so-called inrush current, i.e., a substantial wake-up current occurs whenever a power-gated module turns abruptly on, which may cause violation scenarios such as instability of the registers content and functional errors resulting from dynamic voltage (IR) drop [52]. It also can increase the wake-up time and wake-up energy. Previous studies have reported inrush power contributes to 20% of the static power in FPGAs with 45nm technology size [53]. Second, power gating requires complicated controller and compiling techniques to (a) anticipate the idle intervals and evaluate whether the power gating is advantageous in spite of associated overheads, (b) predict data arrival in order to initiate the wake-up phase and assure that only a small fraction of resources is switched each time to preclude timing (and hereby, functional) errors, and (c) route signals

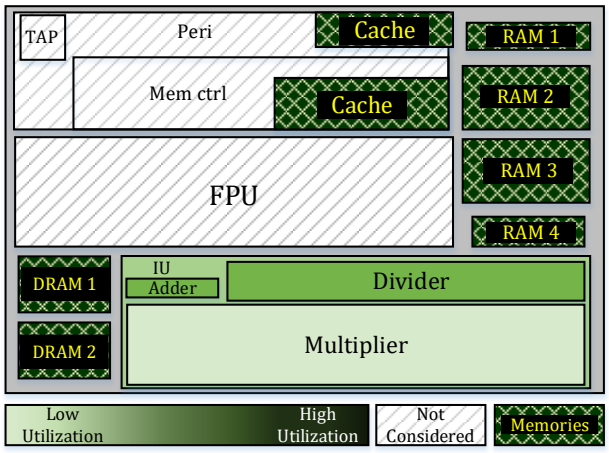


Fig. 15. LEON2 processor microarchitecture (the area of each block is normalized to the chip area.)

in such a way that they do not use the resources of power-gated regions. Another advantage of the proposed architecture with respect to power-gating is the improvement of reliability. When a soft-error flips a configuration cell in the proposed architecture, this cell will be automatically corrected in the next configurations (notice that the power-gating methods do no turn-off the SRAM cells). Last but not least, power-gating approach leaves the part of device unused, while the proposed architecture utilizes all the resources which shortens the wirelength and thereby, wire delays, consequently. Nevertheless, since FPGAs comprise abundant routing resources which mainly remain underutilized, power gating of the *routing resources* [54] can be employed in tandem with our proposed architecture to further increase the power efficiency.

### *F. Alternative Processor*

We have examined the applicability of the proposed architecture on an alternative open-source soft-core processor. We choose LEON2 which is SPARC-V8-compliant, as an alternative processor for several reasons. First, it is popular among a wide range of embedded processors and also compatible and extensible to be customized for diverse range of performance/cost trade-off, e.g., LEON2-FT. Second, the full HDL source code of total LEON2 (including FPU) is license free, while source code of FPU is not available in LEON3 and LEON4 which causes problem in running application with floating point instructions. Finally, in RISC-V case study, the integrated functional units all belong to FPU, but here in LEON2, as we will demonstrate in the following, all functional units are separate and thereby, integration of low-utilized units is more challenging.

Fig. 15 illustrates the synthesized floorplan of LEON2. As shown in this figure, LEON2 functional units include a 32-bit adder, 32-bit multiplier (that can be configured to 32x32, 32x16, 32x8, and 16x16), 32-bit divider, and an optional FPU. FPU as a co-processor is licensed and therefore, is not available in the processor source code. Therefore, we have omitted the FPU unit from our evaluations and focused on the integer functional units which proves the generality of the proposed architectures not just in FPU. Accordingly, in the rest of this section, the reported results for LEON2 processor are in the scope of *Integer Unit* (IU).

To identify the utilization and access pattern to the functional units, each MiBench benchmark is cross-compiled to the SPARC-V8 architecture using RTEMS [55]. Afterwards, Simics full system simulator [56] is used to obtain the utilization and access patterns of the functional units which are shown in Table VII and Fig. 16. Utilization ratio of the integer divider (DIV) is approximately $3.48 \times 10^{-6}\%$ while it occupies a significant area (79.1%) of total functional units. Therefore, it can be efficiently integrated with integer multiplier (MUL) into a shared RU. Hence, switching ratio of the proposed RU unit is $2.81 \times 10^{-6}$ on average, as demonstrated in Fig. 17. Subsequently, reconfiguration overhead of the proposed RU will be small and prorated in application execution time, leaving negligible impact on total execution time of applications. The rest of the evaluation flow for the proposed and baseline architectures in LEON2 processor is the same as RISC-V processor.

TABLE VII
UTILIZATION, AREA, AND CHLS COVERAGE RATIO IN LEON2 PROCESSOR FUNCTIONAL UNITS

| Functional Units | Utilization | Area (4-LUT) | | |
|---|---|---|---|---|
| | | RC/module | LUT-RU | CHL-LUT-RU |
| ADD | 15.42% | 176 | 176 4-LUT | 176 4-LUT |
| MUL | $3.4 \times 10^{-4}\%$ | 485 | 2498 4-LUT | 414 CHL1, 61 CHL2 , and 2023 4-LUT) |
| DIV | 9.49% | 2498 | | |

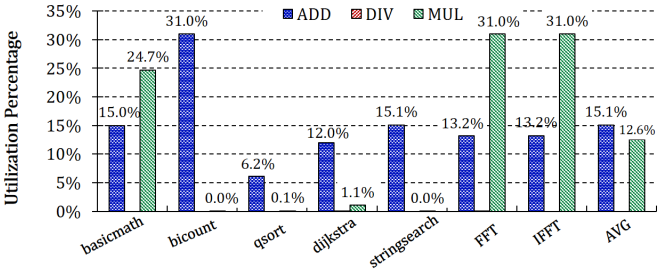


Fig. 16. Utilization ratio of LEON2 components in MiBench benchmarks

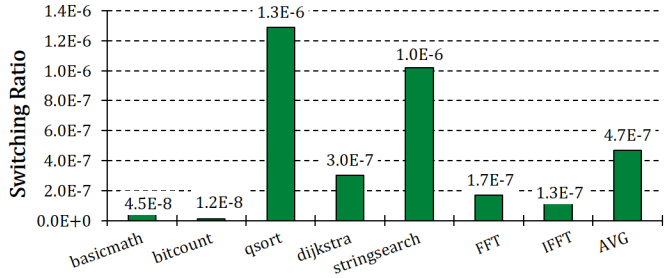


Fig. 17. Average inter-component switching ratio in LEON2 processor

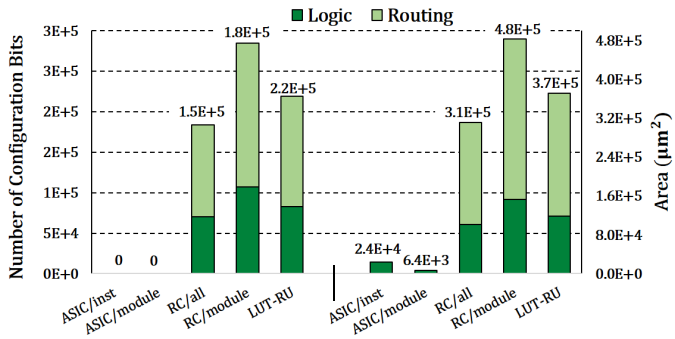


Fig. 18. Comparing the number of LUTs and area of LEON2 components in the baseline and proposed architectures

*1) Area:* Fig. 18 demonstrates the number of LUTs and configuration bits in the proposed and baseline architectures. As shown in this figures, the number of LUTs and configuration bits in LUT-RU has been reduced by 23.1%, compared to the baseline RC/module architecture. The total area for LUT-RU is also reduced by 23.1%. The detailed results for each architecture are shown in Fig. 18.

*2) Power and Energy Consumption:* As demonstrated in Fig. 19, static power of the LUT-RU architecture is reduced by 23.2% as compared to the RC/module architecture. In addition, compared to

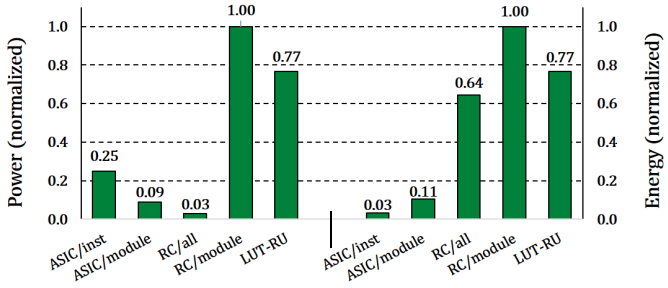


Fig. 19. Comparing the power and energy of different architectures implementing LEON2 processor

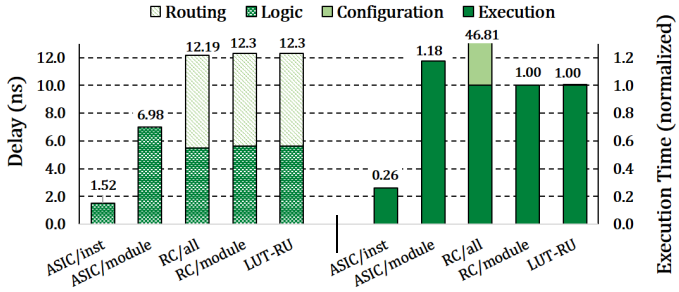


Fig. 20. Comparing the delay of functional units and execution time in different architectures implementing LEON2 processor

the RC/module architecture, LUT-RU architecture has reduced energy by 23.2%. These results are expectable since Divider is considerably smaller than Multiplier. Hence, the improvements are nearly equal between energy-power and between area-configuration bits.

*3) Critical Path Delay and Application Execution Time:* Fig. 20 indicates the critical path delay and total execution time of the baseline and proposed architectures. The results demonstrate that total execution time has only increased by $10^{-5}$% in LUT-RU compared with the RC/module architecture. This is due to negligible utilization ratio of the divider which reduces the reconfiguration time overhead.

It should be noted that the efficiency of the proposed architecture is irrelevant to the size of processor itself, as we have shown its effectiveness in the moderate-size RISC-V processor with $\sim 35,000$ LUTs, and a smaller one (LEON2) consists of $\sim 3,200$ LUTs. The key prerequisite for effectiveness of the proposed architecture is (a) low utilized units with fragmented access pattern, and (b) low utilized units have close area. Nevertheless, larger processors result in larger RUs and consequently, higher number of reconfiguration bits and therefore, reconfiguration time. The proposed architecture also improves the critical path delay which will directly enhance the execution time due to the shorter clock cycles which is valid in RISC-V case study.

### G. Evaluating CHL-LUT-RU

In the proposed CHL-LUT-RU architecture, each shared LUT that its function can be implemented with either CHL1 or CHL2, is simply substituted by an appropriate cell in the netlist BLIF file. The CHL-LUT-RU architecture has been evaluated after placement stage and the results are reported in terms of logic resources. Since the contribution of this work is focused on logic architecture, the impact of routing resources has not been considered. Nevertheless, a brief discussion about routing is represented in Section VI. We intend to consider proposing a customized (generic) routing architecture in the future track of this work. Table VI reports area, delay, and static power of 4-LUT and CHLs which are obtained using HSPICE simulations and with transistor sizing and sub-circuit models obtained from COFFE. The delays reported in Table VI are the average delay of all inputs

TABLE VIII
RESULTS FOR CHL-LUT-RU IN RISC-V AND LEON2 PROCESSORS NORMALIZED TO RC/MODULE (CONSIDERING LOGIC RESOURCES)

| Parameters/ Achitectures | RISC-V | | LEON2 | |
|---|---|---|---|---|
| | LUT-RU | CHL-LUT-RU | LUT-RU | CHL-LUT-RU |
| # of Config. Bits | 0.72 | 0.65 | 0.78 | 0.69 |
| Area | 0.72 | 0.59 | 0.78 | 0.66 |
| Critical Path | 1.04 | 0.93 | 0.93 | 0.89 |
| Execution Time | 1.06 | 0.97 | 0.94 | 0.89 |
| Energy | 0.71 | 0.48 | 0.78 | 0.55 |
| Static Power | 0.75 | 0.47 | 0.72 | 0.53 |

of target CHL or LUT. Static power of each cell is measured using HSPICE simulations with the same transistor technology files and temperature used in Nangate library (i.e., 25 °C).

The summary of results is reported in Table VIII. As demonstrated in this table, the proposed CHL-LUT-RU architecture has further reduced the number of SRAM configuration bits, area, static power, critical path delay, execution time, and energy compared to the proposed LUT-RU in the scope of logic resources. Note that reducing configuration bits can be translated as the circuit reliability due to the reduction of the number of susceptible bits to soft errors [57], [58]. In addition, while our proposed architectures, particularly CHL-LUT-RU, have significantly reduced the logic resources, the conventional ASIC implementation, i.e., ASIC/module, is still power-efficient than all other architectures. This is mainly due to the large number of configuration cells used in the other architectures while the ASIC implementation of a 4-input Boolean function consumes only 35nW on average, whereas this value is greater than 180nW for CHL cells.

## VI. DISCUSSION

Here, we discuss some points related to the proposed architecture.

*1) **Routing architecture**:* We have not investigated the routing architecture for CHL-LUT-RU due to its high complexity in both implementation and analysis. However, since the routing resources in a typical island style FPGA architecture are surrounded by logic clusters, reducing the number of required logic clusters by the factor of $S$ will reduce the number of routing resources (i.e., switch boxes) and thereby their area and power by the same factor. In addition, taking the routing delay into account, the critical path delay will not be increased since at each time only one of the functional units is mapped on the corresponding RU owning the same (or more in the case that is smaller than RU) routing and logic resources as in its baseline architecture. In particular, we expect that the critical path delay may also be improved since higher number of resources (more flexibility to place and route) will be allocated to specific components. Furthermore, in the proposed architecture, proximity of the logic clusters of each component may increase after the mapping due to the exclusive mapping of components and reduction of the array size. It can further improve the routing delay.

*2) **Optimal heterogeneous mapper**:* A heterogeneous architecture such as the proposed CHL-LUT-RU may need modifications in the original mapping algorithm that is used in a homogeneous, pure 4-LUT based architecture [43]; therefore, detailed investigation of the intra-cluster architecture (e.g., ratio of CHL to LUTs) and mapping to such an architecture is required. We will cover details of both the routing and logic architectures as an extension of this work.

*3) **Reconfiguration overhead**:* Integrating underutilized components into shared RUs can impose reconfiguration overhead which can be problematic in hard real-time designs. Although the impact of reconfiguration time on the total application runtime is negligible (about 2%), it can be eliminated using the concept of shadow SRAM cells [59]. Nevertheless, a trade-off between eliminating the reconfiguration delay and area-power overhead of shadow SRAMs should be taken into account.

TABLE IX
SUMMARY OF THE RESULTS FOR VARIOUS SYNTHESIS TOOLS AND PROCESSORS ACROSS ALL BENCHMARK (NORMALIZED TO RC/MODULE).

| Parameter | Area | | | Critical Path | | | Execution Time | | | Energy | | | Static Power | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Processor | RISC-V | | LEON2 | RISC-V | | LEON2 | RISC-V | | LEON2 | RISC-V | | LEON2 | RISC-V | | LEON2 |
| Synthesis Tool | Berkeley ABC | Altera Quartus | Berkeley ABC | Berkeley ABC | Altera Quartus | Berkeley ABC | Berkeley ABC | Altera Quartus | Berkeley ABC | Berkeley ABC | Altera Quartus | Berkeley ABC | Berkeley ABC | Altera Quartus | Berkeley ABC |
| ASIC/inst (logic) | 0.06 | 0.06 | 0.05 | 0.16 | 0.16 | 0.12 | 0.01 | 0.01 | 0.26 | 0.01 | $4.18 \times 10^{-3}$ | 0.3 | 0.41 | 0.26 | 0.25 |
| ASIC/module (logic) | 0.01 | 0.01 | 0.01 | $1.8 \times 10^{-3}$ | 0.16 | 0.57 | 0.17 | 0.16 | 1.18 | 0. 01 | 0.01 | 0.11 | 0.07 | 0.07 | 0.09 |
| RC/all | 0.29 | 0.24 | 0.64 | 0.93 | 1.02 | 0.99 | 41.48 | 29.57 | 46.81 | 0.1 | 0.08 | 0.64 | $4.2 \times 10^{-3}$ | $2.7 \times 10^{-3}$ | 0.03 |
| RC/module | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| LUT-RU | 0.69 | 0.71 | 0.77 | 0.91 | 0.97 | 1 | 0.94 | 0.99 | 1 | 0.63 | 0.69 | 0.77 | 0.68 | 0.69 | 0.77 |

*4) **Applicability with Non-Volatile Memories (NVM)**:* Although our proposed architecture is targeted for SRAM-based soft-core processors, it is definitely not limited to SRAMs and can be expanded to NVM platforms as well. However, despite promising characteristics of NVM such as negligible leakage power, they encounter with several unaddressed deficiencies such as write latency and energy and still are in prototyping phase. In addition, the proposed architecture may require large number of reconfigurations which is not consistent with currently-available NVMs.

*5) **Applicability in emerging applications**:* One of the promising application of the proposed architecture can be in implementing *Machine Learning* (ML) algorithms, e.g., *Neural Networks*. Despite the widespread research on efficient implementation of Neural Networks, only a limited range of objective functions, especially sum or mean squared error used in linear regression, has been implemented on the state-of-the-art FPGAs [60]. This is because, operations such as division and square root used in the gradient descent cycle of objective function either occur infrequently or consume significant amount of resources. The proposed architecture can be beneficial in these applications when for example, floating-point multiplier can be merged with divider since in ML algorithms, a stream of multiplication-additional (MAC) operations is followed by a division. Therefore, the context-switching inside an RU will be tolerable. As an interesting point, in the proposed architecture we have already merged these units within RISC-V processor. The proposed architecture is also applicable for component-based designs other than soft-core processors, such as DSP cores that are exploited in ML algorithms [60]. We have examined the applicability of the proposed architecture on a floating point DSP core from OpenCores [61] which its contributing components are listed in Table X. Depending on the distribution of the operations (tunable according to the target application), several components of the DSP core [60] can be merged into a shared RU. One appropriate integration to save considerable area and power can be achieved by merging the divider, multiplier, and subtractor units. Our experimental results for this DSP core reveal that area is saved by 50.2%, with 50.1% improvement in static power while keeping the latency intact.

Another use-case of the proposed method can be in FPGA-assisted Cloud computing, a.k.a, FPGA virtualization [62] which incorporates FPGAs in datacenters in order to improve the processing capacity and power consumption. A challenging task in such platforms is an efficient use of FPGA resources since an FPGA is expected to afford computation of different users (i.e., several tasks) at the same time. Since the proposed architecture integrates low-utilized functional units into a single RU, provided that we keep the FPGA size the same, further tasks can be implemented into the same FPGA device. This will reduce the cost-per-user or cost-per-task.

TABLE X
COMPONENTS AREA IN THE DSP CORE [61]

| Functional Unit | Adder | Subtractor | Round | Divider | Multiplier | Exception |
|---|---|---|---|---|---|---|
| # of 4-LUTs | 689 | 2667 | 434 | 3723 | 2688 | 1067 |

## VII. CONCLUSION

In this paper, we presented a reconfigurable architecture for embedded soft-core processors, which aims to improve area and power efficiency of FPGA-based soft-core processors while maintaining flexibility to customize and update the processor for designers. In the proposed architecture, we integrated low-utilized and fragmentally-accessed components into a single RU and used efficient configurable logics along with the mapping algorithm to implement the overlapping functions of functional components mapped to each RU. Our integration is based on carefully profiling the frequency and access pattern of instructions in MiBench applications, in order to well select the merged components. We have implemented RISC-V processor to examine the efficiency of the proposed architectures. Experimental results show that, as compared to conventional FPGA-based soft-core processors, the proposed architecture improve the area footprint, static power, energy consumption, critical path delay, and total execution time by 30.7%, 32.5%, 36.9%, 9.2%, and 6.3%, respectively.

## REFERENCES


[1] D. D. Gajski and F. Vahid, "Specification and design of embedded hardware-software systems," *Design & Test of Computers, IEEE*, vol. 12, no. 1, pp. 53–67, 1995.

[2] P. Marwedel, *Embedded system design: Embedded systems foundations of cyber-physical systems*. Springer Science & Business Media, 2010.

[3] M. Barr, *Programming embedded systems in C and C++*. " O'Reilly Media, Inc.", 1999.

[4] P. Marwedel and G. Goossens, *Code generation for embedded processors*. Springer Science & Business Media, 2013, vol. 317.

[5] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 2, pp. 203–215, 2007.

[6] M. Sarrafzadeh, F. Dabiri, R. Jafari, T. Massey, and A. Nahapetan, "Low power light-weight embedded systems," in *Low Power Electronics and Design. ISLPED'06. Proceedings of the 2006 International Symposium on*. IEEE, 2006, pp. 207–212.

[7] (Accessed March, 2016) Gaisler research website. [Online]. Available: www.gaisler.com/

[8] K. Asanović and D. A. Patterson, "Instruction sets should be free: The case for RISC-V," Technical report, University of California at Berkeley, http://www. eecs. berkeley. edu/Pubs/TechRpts/2014/EECS-2014-146. pdf, Tech. Rep., 2014.

[9] (Accessed March, 2016) Opensparc website. [Online]. Available: www.opensparc.org

[10] (Accessed March, 2016) Openrisc website. [Online]. Available: www.openrisc.io

[11] F. Plavec, B. Fort, Z. G. Vranesic, and S. D. Brown, "Experiences with soft-core processor design," in *null*. IEEE, 2005, p. 167b.

[12] L. Barthe, L. V. Cargnini, P. Benoit, and L. Torres, "The SecretBlaze: A configurable and cost-effective open-source soft-core processor," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), International Symposium on*. IEEE, 2011, pp. 310–313.

[13] J. G. Tong, I. D. Anderson, and M. A. Khalid, "Soft-core processors for embedded systems," in *Microelectronics. ICM'06. International Conference on*. IEEE, 2006, pp. 170–173.

[14] P. Yiannacouras, J. Rose, and J. G. Steffan, "The microarchitecture of fpga-based soft processors," in *Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*. ACM, 2005, pp. 202–212.

[15] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: constructing hardware in a scala embedded language," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 1216–1225.

[16] A. A. Bsoul and S. J. Wilton, "An FPGA architecture supporting dynamically controlled power gating," in *Field-Programmable Technology (FPT), 2010 International Conference on*. IEEE, 2010, pp. 1–8.

[17] H. Esmaeilzadeh, E. Blem, R. St Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *ACM SIGARCH Computer Architecture News*, vol. 39, no. 3. ACM, 2011, pp. 365–376.

[18] A. R. Ashammagari, H. Mahmoodi, and H. Homayoun, "Exploiting STT-NV technology for reconfigurable, high performance, low power, and low temperature functional unit design," in *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2014, p. 335.

[19] A. R. Ashammagari, H. Mahmoodi, T. Mohsenin, and H. Homayoun, "Reconfigurable STT-NV LUT-based functional units to improve performance in general-purpose processors," in *Proceedings of the 24th edition of the great lakes symposium on VLSI*. ACM, 2014, pp. 249–254.

[20] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and M. C. Eliseu Filho, "Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications," *Computers, IEEE Transactions on*, vol. 49, no. 5, pp. 465–481, 2000.

[21] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix," in *Field Programmable Logic and Application*. Springer, 2003, pp. 61–70.

[22] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. R. Taylor, "Piperench: A reconfigurable architecture and compiler," *Computer*, vol. 33, no. 4, pp. 70–77, 2000.

[23] R. Lysecky, G. Stitt, and F. Vahid, "Warp processors," in *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 11, no. 3. ACM, 2004, pp. 659–681.

[24] Z. A. Ye, A. Moshovos, S. Hauck, and P. Banerjee, *CHIMAERA: a high-performance architecture with a tightly-coupled reconfigurable functional unit*. ACM, 2000, vol. 28, no. 2.

[25] J. E. Carrillo and P. Chow, "The effect of reconfigurable units in superscalar processors," in *Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays*. ACM, 2001, pp. 141–150.

[26] M. Labrecque and J. G. Steffan, "Improving pipelined soft processors with multithreading," in *Field Programmable Logic and Applications, FPL. International Conference on*. IEEE, 2007, pp. 210–215.

[27] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose, "Microarchitectural techniques for power gating of execution units," in *Proceedings of the 2004 international symposium on Low power electronics and design*. ACM, 2004, pp. 32–37.

[28] S. Roy, N. Ranganathan, and S. Katkoori, "A framework for power-gating functional units in embedded microprocessors," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 17, no. 11, pp. 1640–1649, 2009.

[29] A. A. Bsoul, S. J. Wilton, K. H. Tsoi, and W. Luk, "An fpga architecture and cad flow supporting dynamically controlled power gating," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 1, pp. 178–191, 2016.

[30] P. Yiannacouras, J. G. Steffan, and J. Rose, "Exploration and customization of FPGA-based soft processors," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 2, pp. 266–277, 2007.

[31] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Workload Characterization, 2001. WWC-4. IEEE International Workshop on*. IEEE, 2001, pp. 3–14.

[32] BLSG, "ABC: A system for sequential synthesis and verification," *Berkeley Logic Synthesis and Verification Group*, 2011.

[33] J. Pistorius, M. Hutton, A. Mishchenko, and R. Brayton, "Benchmarking method and designs targeting logic synthesis for fpgas," in *Proc. IWLS*, vol. 7, 2007, pp. 230–237.

[34] S. Yazdanshenas and V. Betz, "Automatic circuit design and modelling for heterogeneous fpgas," in *Field-Programmable Technology (FPT), 2017 International Conference on*. IEEE, 2017, pp. 9–16.

[35] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, Liu *et al.*, "VTR 7.0: Next generation architecture and CAD system for FPGAs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 7, no. 2, p. 6, 2014.

[36] M. Hosseinabady and J. L. Nunez-Yanez, "Run-time power gating in hybrid arm-fpga devices," in *Field Programmable Logic and Applications (FPL), 24th International Conference on*. IEEE, 2014, pp. 1–6.

[37] J. Teich, "Hardware/software codesign: The past, the present, and predicting the future," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1411–1430, 2012.

[38] VectorBlox/risc-v. VectorBlox Computing Inc. [Online]. Available: https://github.com/VectorBlox/orca

[39] Y. Lee, A. Waterman, R. Avizienis, H. Cook, C. Sun, V. Stojanović, and K. Asanović, "A 45nm 1.3 ghz 16.7 double-precision gflops/w risc-v processor with vector accelerators," in *European Solid State Circuits Conference (ESSCIRC), 2014-40th*. IEEE, 2014, pp. 199–202.

[40] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," *HP Laboratories*, pp. 22–31, 2009.

[41] H. Wong, V. Betz, and J. Rose, "Comparing FPGA vs. custom CMOS and the impact on processor microarchitecture," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 2011, pp. 5–14.

[42] E. Ahmed and J. Rose, "The effect of lut and cluster size on deep-submicron fpga performance and density," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 3, pp. 288–298, 2004.

[43] I. Ahmadpour, B. Khaleghi, and H. Asadi, "An efficient reconfigurable architecture by characterizing most frequent logic functions," in *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*. IEEE, 2015, pp. 1–6.

[44] A. Ahari, B. Khaleghi, Z. Ebrahimi, H. Asadi, and M. B. Tahoori, "Towards dark silicon era in fpgas using complementary hard logic design," in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*. IEEE, 2014, pp. 1–6.

[45] Z. Ebrahimi, B. Khaleghi, and H. Asadi, "PEAF: A Power-Efficient Architecture for SRAM-Based FPGAs Using Reconfigurable Hard Logic Design in Dark Silicon Era," *Computers, IEEE Transactions on*, In press, 2017.

[46] (Accessed August, 2016) Behavioural simulation (spike). [Online]. Available: www.lowrisc.org/docs/untether-v0.2/spike

[47] D. Chai and A. Kuehlmann, "Building a better boolean matcher and symmetry detector," in *Proceedings of the conference on Design, automation and test in Europe: Proceedings*. European Design and Automation Association, 2006, pp. 1079–1084.

[48] (Accessed March, 2016) Nangate Open Cell Library. [Online]. Available: www.nangate.com/

[49] "Stratix-2 platform FPGA hand book," Altera, April 2011.

[50] "Virtex-4 platform FPGA user guide," Xilinx, December 2008.

[51] A. Chen, J. Hutchby, V. Zhirnov, and G. Bourianoff, *Emerging nano-electronic devices*. John Wiley & Sons, 2014.

[52] A. A. Bsoul and S. J. Wilton, "A configurable architecture to limit wakeup current in dynamically-controlled power-gated fpgas," in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*. ACM, 2012, pp. 245–254.

[53] S. Sharp. Power management solution guide - xilinx. [Online]. Available: https://www.xilinx.com/publications/archives/solution_guides/power_management.pdf

[54] Z. Seifoori, B. Khaleghi, and H. Asadi, "A power gating switch box architecture in routing network of sram-based fpgas in dark silicon era," in *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2017, pp. 1342–1347.

[55] (Accessed August, 2016) RTEMS Cross Compilation System (RCC). [Online]. Available: www.rtems.org

[56] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[57] S. Yazdanshenas, H. Asadi, and B. Khaleghi, "A scalable dependability scheme for routing fabric of sram-based reconfigurable devices," *IEEE Trans. VLSI Syst.*, vol. 23, no. 9, pp. 1868–1878, 2015.

[58] H. Asadi and M. B. Tahoori, "Analytical techniques for soft error rate modeling and mitigation of fpga-based designs," *IEEE Trans. VLSI Syst.*, vol. 15, no. 12, pp. 1320–1331, 2007.

[59] W. Zhang, N. K. Jha, and L. Shang, "Low-power 3D nano/CMOS hybrid dynamically reconfigurable architecture," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 6, no. 3, p. 10, 2010.

[60] D. Mahajan, J. Park, E. Amaro, H. Sharma, A. Yazdanbakhsh, J. K. Kim, and H. Esmaeilzadeh, "Tabla: A unified template-based framework for accelerating statistical machine learning," in *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 14–26.

[61] (Accessed November, 2017) Opencores. [Online]. Available: www.opencores.org
[62] S. Yazdanshenas and V. Betz, "Quantifying and mitigating the costs of fpga virtualization," in *Field Programmable Logic and Applications (FPL), 27th International Conference on*. IEEE, 2017, pp. 1–7.

**Sajjad Tamimi** has received the B.Sc. and M.Sc. degrees in computer engineering from *Iran University of Science and Technology* (IUST) and *Sharif University of Technology* (SUT), Tehran, Iran, in 2014 and 2016, respectively. He has been with the *Data Storage, Processing, and Networks* (DSN) Laboratory at the Department of Computer Engineering, SUT, as a research assistant for two years. His current research interests include reconfigurable computing, embedded system design, and computer architecture.

**Zahra Ebrahimi** has received the B.Sc. and M.Sc. degrees in computer engineering SUT, Tehran, Iran, in 2014 and 2016, respectively. She has been with the DSN Laboratory at the Department of Computer Engineering, SUT, as a research assistant for four years. Her current research interests include reconfigurable computing and computer-aided design.

**Behnam Khaleghi** has received his B.Sc. and M.Sc. degrees in computer engineering from SUT, Tehran, Iran, in 2013 and 2016, respectively. He is currently working as a research assistant in the DSN Laboratory at the Department of Computer Engineering, SUT. He spent the summer 2014 and 2015 as a research assistant at the Chair for Embedded Systems in the Karlsruhe Institute of Technology (KIT). His research interests include reconfigurable computing, CAD, and reliable system design. He has two Best Paper Nominations at the DAC'17 and DATE'17.

**Hossein Asadi** (M'08, SM'14) received the B.Sc. and M.Sc. degrees in computer engineering from the SUT, Tehran, Iran, in 2000 and 2002, respectively, and the Ph.D. degree in electrical and computer engineering from Northeastern University, Boston, MA, USA, in 2007.

He was with EMC Corporation, Hopkinton, MA, USA, as a Research Scientist and Senior Hardware Engineer, from 2006 to 2009. From 2002 to 2003, he was a member of the Dependable Systems Laboratory, SUT, where he researched hardware verification techniques. From 2001 to 2002, he was a member of the Sharif Rescue Robots Group. He has been with the Department of Computer Engineering, SUT, since 2009, where he is currently a tenured Associate Professor. He is the Founder and Director of the *Data Storage, Networks, and Processing* (DSN) Laboratory, Director of Sharif *High-Performance Computing* (HPC) Center, the Director of Sharif *Information and Coummnications Technology Center* (ICTC), and the President of Sharif ICT Innovation Center. He spent three months in the summer 2015 as a Visiting Professor at the School of Computer and Communication Sciences at the Ecole Poly-technique Federele de Lausanne (EPFL). He is also the co-founder of HPDS corp., designing and fabricating midrange and high-end data storage systems. He has authored and co-authored more than eighty technical papers in reputed journals and conference proceedings. His current research interests include data storage systems and networks, solid-state drives, operating system support for I/O and memory management, and reconfigurable and dependable computing.

Dr. Asadi was a recipient of the Technical Award for the Best Robot Design from the International RoboCup Rescue Competition, organized by AAAI and RoboCup, a recipient of Best Paper Award at the 15th CSI Internation Symposium on *Computer Architecture & Digital Systems* (CADS), the Distinguished Lecturer Award from SUT in 2010, the Distinguished Researcher Award and the Distinguished Research Institute Award from SUT in 2016, and the Distinguished Technology Award from SUT in 2017. He is also recipient of Extraordinary Ability in Science visa from US Citizenship and Immigration Services in 2008. He has also served as the publication chair of several national and international conferences including CNDS2013, AISP2013, and CSSE2013 during the past four years. Most recently, he has served as a Guest Editor of IEEE Transactions on Computers, an Associate Editor of Microelectronics Reliability, a Program Co-Chair of CADS2015, and the Program Chair of CSI National Computer Conference (CSICC2017).