SLMLET: A RISC-V Processor SoC with Tightly-Coupled Area-Efficient eFPGA Blocks

Takuya Kojima*, Yosuke Yanai†, Hayate Okuhara‡, Hideharu Amano†, Morihiro Kuga§, and Masahiro Iida§
*Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, Japan
†Department of Information and Computer Science, Keio University, Yokohama, Japan
‡Department of Electrical and Computer Engineering, National University of Singapore, Singapore
§Department of Computer Science and Electrical Engineering, Kumamoto University, Kumamoto, Japan

***Abstract*—SLMLET is a low-power System-on-a-Chip (SoC), which is a promising device for edge computing. It consists of a RISC-V core and area-saving embedded Field-programmable gate arrays (eFPGA) blocks called Scalable Logic Module (SLM). This paper presents a fabricated SLMLET chip and a developed HW/SW co-design flow. The experimental results demonstrate that the RISC-V core can operate at up to 300MHz, and a hardware design on the SLM blocks can operate at up to 100MHz. By offloading compute-intensive processes to dedicated circuits implemented in the SLM blocks, it is possible to achieve up to 77 % energy reduction compared to software processing on the RISC-V core and other commercial microcontrollers.**

I. Introduction

With the advancement of IoT (Internet of Things), there is a growing need for edge devices, which are traditionally only equipped with simple microcontrollers, to perform certain calculations such as cryptography and anonymization as well as their basic functions. Equipping hardware accelerators satisfies the requirements for performance and power consumption; however, the flexibility of IoT devices is limited because the applications capable of performance enhancement are restricted, leading to a lack of versatility across various purposes.

SoC type FPGA chips that integrate CPU and FPGA into a single chip are advantageous to such IoT devices, as they can provide predictable computation performance, specialized hardware for target workloads, and high energy efficiency [1]. Major FPGA vendors have commercialized excellent SoC-type FPGA chips like Zynq [2]. However, they are overkill for simple IoT applications and sometimes consume too much energy.

This paper proposes an energy-efficient scalable SoC-type FPGA called SLMLET. The design strategy of SLMLET is as follows: 1) It uses an original eFPGA called SLM [3] that requires smaller configuration data than that for conventional look-up table-based FPGAs, as in [4]–[6]. 2) Multiple configuration data sets corresponding to multiple applications are stored in their local memory in the compressed form and exchanged application-by-application. This function contributes to the efficient use of a limited area of the chip. 3) SLMLET has a HyperBus interface to connect to HyperRAM, a low-power DRAM, and other SLMLT chips to extend the performance with parallel processing. SLMLET also supports real hardware-context migration; that is, it is possible to extract internal registers from one SLM, transfer them to another chip's SLM, and restart processing. In this way, SLMLET can easily configure multi-chip systems. All of the above functions are controlled by a RISC-V processor core tightly connected with SLM-based eFPGA cores.

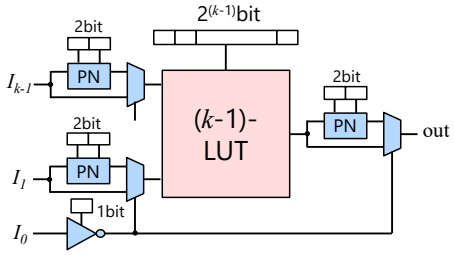


Fig. 1: SLM logic cell structure

When it comes to IoT application development with the FPGAs, efficient software/hardware co-design is essential in order to adapt to rapidly changing cyber-physical systems. This paper also proposes a software development kit (SDK) and software library for the SLMLET SoC built with RISC-V ecosystems. The SDK provides application developers with an API similar to CUDA and OpenCL, which are successful programming models for accelerators. Thereby, the developers easily write a program utilizing hardware on the FPGA core.

As a case study, we implement a couple of workloads on the SLMLET SoC and several conventional microcontrollers widely used in the IoT domain. The experiments demonstrate that the SLMLET outperforms the conventional SoCs for applications well-suited to the FPGA computation in terms of energy efficiency and performance.

II. Background and Related Work

A. SLM-based FPGA architecture

FPGAs comprise programmable logic cells and interconnection elements as their essential elements, arranged in a regular array structure [7]. In addition, modern FPGAs also have digital signal processing (DSP) blocks and memory blocks to increase their computational capability and storage capacity for intermediate data.

The logic cells in the FPGA are typically implemented with look-up tables (LUTs) and flip-flops (FFs). A k-input LUT can be programmed as a k-input truth table for a logic function. With a larger input size k, a single LUT can express more complex logic. It consequently reduces the total number of logic cells in the FPGA. However, it also exponentially increases the configuration memory size since it needs 2^k bits.

To address the issue, SLM has been proposed [3]. Its logic cell is optimized based on the insight that similar logic functions frequently appear in typical hardware designs. Figure

1 shows the structure of the SLM logic cell. Instead of k-LUT, the logic cell consists of a $(k-1)$-LUT, multiple programmable NANDs, and an inverter. The targeted k-input logic function is decomposed into two partial functions by applying Shannon expansion. The configuration memory of the LUT is filled with the truth table of one of the partial functions. Then, FPGA CAD tries to find an interconversion to make the other partial function NPN equivalent to the first one. The programmable NANDs around the LUT realize the interconversion. Although every function is amenable to such conversion, a report in [3] shows that conversion is possible for 90% of the functions in typical benchmarks.

An IP generator tool is available to customize an SLM-based FPGA architecture [3]. Similar to existing eFPGA frameworks like [8], users can change generated IP parameters, such as the number of basic logic elements (BLEs) in a logic block (LB) and the number of tiles in the FPGA. In addition, there are SLM-specific parameters, such as the number of inputs of the SLM and the number of programmable NANDs.

B. Prior work on eFPGA SoC

As introduced in Sec. I, several prior works have proposed SoCs with embedded FPGA (eFPGA) cores and processor cores. Arnold [5] has a RISC-V core and around 6K 4-input LUTs in the eFPGA. It is fabricated as a 3x3 mm^2 die with a 22 nm FDSOI process. Moreover, body biassing, which is a leakage power reduction technique, is available to save unnecessary power consumption when sleeping. The manually designed hard macro contributes to the high LUT density and high performance.

On the contrary, Renzini, *et al.* [6] have presented a soft macro-based eFPGA SoC with a RISC-V core. For power control applications, the SoC is implemented with a 90 nm process for power ICs. Due to the matured process and the LUT-based soft macro, it has a limited programmable logic, only about 50 6-LUTs.

The latest work, CIFER [4], has many cores of two types of RISC-V processors and an eFPGA with over 6K 6-LUTs. One type of the RISC-V core is a Linux-capable 64-bit processor and is responsible for system control. The SoC integrates four of such cores. The other type is a 32-bit processor for computational tasks. Those cores form a cluster of six, and the SoC comprises three such clusters, meaning a total of 18 cores are packed into the SoC. Despite the soft macro-based eFPGA, it archives a comparable LUT density to the hard macro-based eFPGA in [5] thanks to the 12 nm FinFET process. In addition, its operating frequency exceeds 1GHz. Nonetheless, when considering the affordable products for edge devices, the question remains whether such a rich SoC is suitable for edge devices.

III. Proposed SLMLET SoC

A. System Overview

This section describes the proposed SLMLET SoC. Figure 2a explains the system overview of SLMLET.

B. RISC-V Processor Core

The RISC-V processor is a 3-stage pipeline implementation of the RV32I instruction set. The core has dedicated memory blocks, 64KB each, for instruction and data. When the system boots, the program binary is loaded into the instruction memory and the data memory via the Serial Peripheral Interface (SPI). The SLMLET can connected to up to four SPI slaves, including the boot loader.

TABLE I: SLM block IP specification

Parameter	Value
BLE structure	5-input SLM
# of BLEs per LB	4
# of LB tiles	224
# of DSP tiles	8
Total SLM cells	896
Total FFs	1,024 bits
Switch topology	Wilton-type

C. SLM-based eFPGA Blocks

Two FPGA blocks composed of the SLM logic cells are integrated into the SLMLET. The current SoC design employs an identical IP for both blocks. Table I summarizes the specification of the SLM block IP. In addition to the SLM logic cells, the block has DSP blocks to implement unsigned 8x8-bit multipliers.

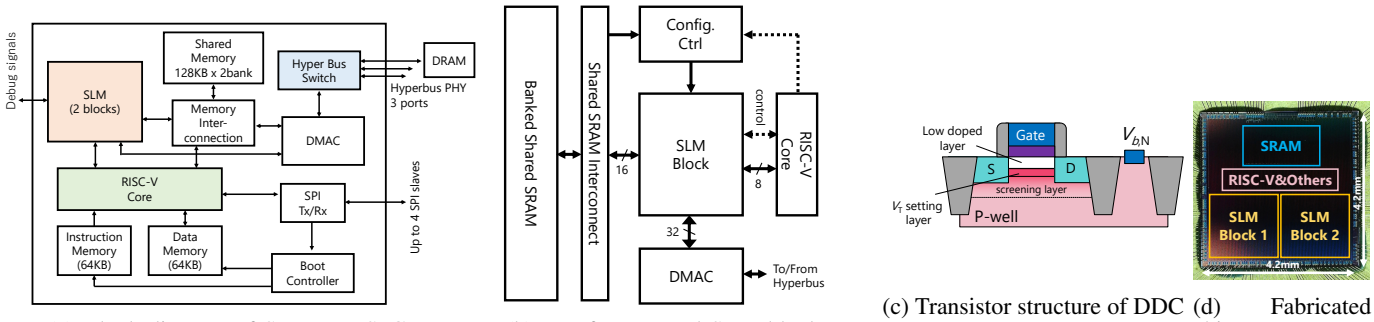
IO blocks are located on all four sides of the SLM block. The IO blocks on the left side of one SLM block are connected to those on the right side of the other SLM block. Therefore, it is possible to use both blocks as a single large block. The remaining IO blocks are connected to other modules in the SoC, as shown in Fig. 2b. The connection between the SLM block and the RISC-V core gives a memory-mapped interface and is utilized for control and status data transfer. Two banks of shared memory are also connected to the SLM block through the interconnect. While a bank of memory is connected to an SLM block, other modules, including the opposite side of the SLM block, cannot access the bank. In addition, Direct Memory Access (DMA) controllers offer the capability of streaming data transfer between the SLM block and the HyperBus, as explained later.

Each SLM block has its configuration controller so that both blocks can be reconfigured in parallel. The configuration data (i.e., bitstream of the FPGA) sets are stored in the shared memory and transferred to an SLM block. This action is triggered by RISC-V software, while the configuration itself is controlled by the dedicated controller.

To save memory usage and reduce configuration data transfer from the external memory to the shared memory, the configuration controller supports on-the-fly decompression of the bitstream data compressed with a simple compression method called Tag Less Compression (TLC) [9]. The total configuration data is already reduced to about 2/3 of the common FPGAs with 5-LUTs [10]. Furthermore, the TLC can compress the data with a 10% to 300% reduction depending on the appearance of continuous '0's.

D. Peripheral Interfaces

Assuming that the SLMLET is used as an edge device, the SPI is sometimes insufficient to handle data to/from the network. In addition, data-intensive applications, which require larger memory space than on-chip memory capacity, are also expected to be emerging IoT applications. Therefore, the SLMLET has three channels of HyperBus interfaces. HyperBus is developed by Cypress Semiconductor and is in accordance with JEDEC eXpanded Serial Peripheral Interface (xSPI)



(a) Block diagram of SLMLET SoC (b) Interface around SLM blocks (c) Transistor structure of DDC process (d) Fabricated chip

Fig. 2: An overview of the SLMLET SoC fabricated with the DDC 55nm process

standard. Despite the reduced number of pins, it achieves high-speed data transfer, and commercially available HyperRAM and HyperFlash can be connected to the interface. Notably, HyperRAM offers lower power consumption compared to conventional DDR memory, making it suitable for edge devices. Additionally, the HyperBus ports can be connected to other SLMLET chips to extend the performance with parallel processing.

E. Fabricated Chip

The SLMLET SoC was fabricated with the USCJ DDC 55nm process [11]. Its transistor structure is illustrated in Fig. 2c. The P-well is formed with three different layers. The low doped layer under the gate oxide contributes to small threshold voltage variation by reducing the random dopant fluctuation. The Vt setting layer enables multiple threshold voltages. These features are important for SoC designs composed of blocks with different operating requirements. The screening layer is a highly doped layer that terminates the depletion layer. Thanks to its high body coefficient of 240mV/V, the body bias control, which changes the bias voltage applied to the well contact in the figure, can adjust the threshold voltage and reduce leakage power. It is possible to balance the trade-off between leakage power and computational performance.

The RTL design of the SLMLET was synthesized with a low-voltage threshold (LVT) standard cell library using Synopsys Design Compiler. The layout was obtained with Cadence Innovus. The photo of the fabricated chip is shown in Fig. 2d. The chip size is 4.2 mm square. The two rectangular at the bottom are the SLM blocks.

The LB tile is laid out in a 106.80 um square area, and each tile is placed in a 16.20 um pitch. As Table I explains, the LB tile has four 5-input SLM logic cells. Therefore, the density of the SLM logic cells is about 264 SLM/mm^2. In the best-case scenario, the 5-input SLM is equivalent to a 6-input LUT. Considering the difference in the fabrication process, the density is comparable to 1541 LUT/mm^2 in the state-of-the-art implementations with 12 nm FinFET [4].

IV. HW/SW Co-design Flow and Library

This section explains the proposed HW/SW co-design flow for SLMLET applications and the software library. Figure 3 describes the flow of generating the executable binary. The flow is divided into two parts: the hardware development flow for the SLM blocks using FPGA CAD [3] and the software compilation flow based on an existing RISC-V toolchain. In

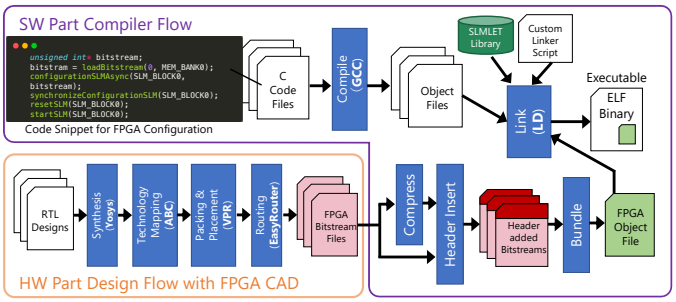


Fig. 3: Hardware (SLM) and software (RISC-V) co-design flow

the hardware development flow, widely used open-source EDA tools are employed. The given RTL design is synthesized with Yosys, technology-mapped with ABC, and clustered and placed with VPR. Lastly, routing is performed with EasyRouter, which is developed for SLM-based eFPGA [12]. The generated bitstream contains the configuration data for a single SLM block and is a fixed length of 14616 bytes if uncompressed. The bitstream file is converted to a binary format, adding a header to tell the SLMLET library whether the bitstream is compressed or not and the length of the bitstream. Then, multiple bitstreams (if present) are bundled into a single object file to enable a user program to switch the configuration of the SLM blocks at runtime.

Currently, C language is the only supported language for the software part on the RISC-V core. The source code is compiled with GCC, and the SLMLET library and the bundled bitstream file are linked to generate the executable binary. Porting and enabling lightweight languages widely used in embedded systems, such as MicroPython and mruby, is crucial to boost IoT application development. Furthermore, supporting modern languages like Rust, which excel in memory safety and concurrency, is also important to exploit heterogeneous parallel processing platforms comprising FPGAs and processors. Such language support is a future work of this research.

A. SMLET Library

The developed library includes software primitives, such as standard I/O for printf and scanf, SPI communication, and HyperRAM data management. In addition, it provides an API to control the SLM blocks and data transfer between the RISC-V core and the SLM blocks without the need for detailed knowledge of the SLMLET SoC. Primary API functions are `loadBitstream` to load bitstream data onto the shared memory, `configurationSLM` to reconfigure the

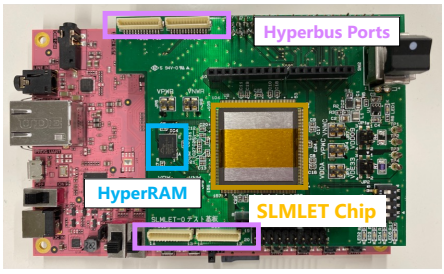


Fig. 4: Evaluation board for SLMLET with PYNQ-Z2

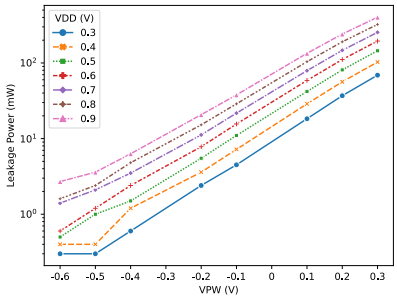


Fig. 5: Leakage power consumption with body bias control

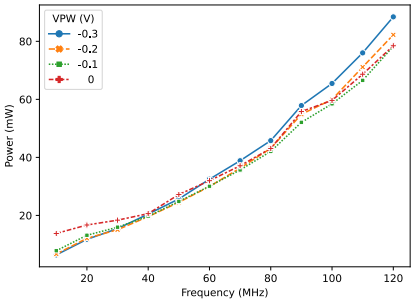


Fig. 6: Power consumption of SLMLET

SLM block, `resetSLM` to reset the SLM block, `startSLM` to activate the SLM block, and `writeSLM` and `readSLM` to access the memory-mapped area of the SLM block.

In addition to `configurationSLM`, a non-blocking function `configurationSLMAsync` is also available. In the case of the non-blocking function, the processor core can continue other tasks during the reconfiguration.

Data transfer through the interface directly connected to the RISC-V core requires different controls depending on the data type. For example, when transferring 32-bit `int` data, a total of four data transfers are required because of the 8-bit interface. The `writeSLM` and `readSLM` functions are, therefore, type-generic macros, and the compiler determines the number of data transfers according to the type of the trasferred data variable.

V. Experiments and Results

In this section, we evaluate the operating frequency, power consumption, and the effect of body bias control through experiments using the developed SoC. Additionally, we compare performance and energy efficiency against commercially available devices used as edge devices, employing several applications implemented with the proposed development flow.

A. Evaluation board and environment

We developed an evaluation board shown in Fig. 4. Its pin header is designated to connect TUL PYNQ-Z2 board featuring Xilinx Zynq-7000. To test the SLMLET SoC efficiently, we developed a Python library, *PySLMLET*, which is a driver software to control the SLMLET SoC and run on the PYNQ-Z2 board. In addition, dedicated hardware was implemented in the Programmable Logic (PL) part of the PYNQ-Z2 board to communicate the SLMLET SoC. The developed ecosystem can interface with SCPI/VISA-compatible stabilized DC power supplies, allowing for fully automated tests under various conditions.

B. Body bias control for leakage power reduction

First, we measured the leakage power consumption of the SLMLET SoC while changing the supply voltage (VDD) and body bias voltages. The VDD was set from 0.3 V to 0.9 V in 0.1 V increments, and the body bias voltage to p-well of nMOS transistors, VPW, was set from -0.6 V to +0.3 V in 0.1 V increments. The same level of body bias voltage was applied to n-well of pMOS transistors. The measured leakage power consumption is shown in Fig. 5.

Depending on the body bias voltage and supply voltage, the leakage power consumption varied from 0.3mW to 402mW.

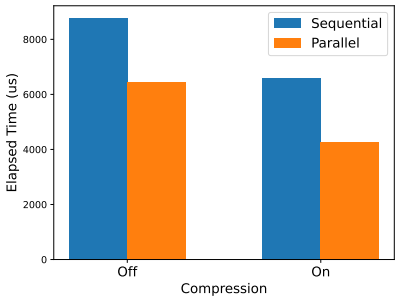


Fig. 7: Elapsed time to reconfigure both blocks (50MHz)

In the case of the standard voltage condition (VDD=0.9V, VPW=0.0V), the leakage power consumption is 72.9mw. Hence, 99.6% of the leakage power can be saved when sleeping the SLMLET SoC. For extended battery life, such a feature is indispensable for edge devices, which are often in a sleep state. Although the computation performance can be boosted by applying forward body bias, the evaluation results suggest that the overhead is not justified. Thus, only reverse body bias is considered in the following experiments.

C. Functional verification of RISC-V core

The RISC-V core is responsible for the whole system control of the SLMLET SoC. Thus, the function of the RISC-V must be verified. For this purpose, we used an official test suite [13], which is a set of tests to verify whether RISC-V ISA is correctly implemented. The test results show that instructions except for half-length memory access instructions (*lh,lhu,sh*) are correctly implemented. That defect is attributed to an incorrect RTL design and has already been fixed.

Then, we study the operating conditions of the RISC-V core while changing the operating frequency up to 120MHz. The plot shows the cases using VDD voltages that minimize the power consumption at each frequency and body bias voltage. The result indicates a strategic shift around 40MHz. For lower frequencies than 40MHz, leak power dominates, making it more power-efficient to increase VDD and apply a strong reverse bias. On the contrary, for frequencies above 40MHz, dynamic power is the primary power consumption, suggesting that a weaker reverse bias (e.g., -0.1V) and a lower VDD should be used.

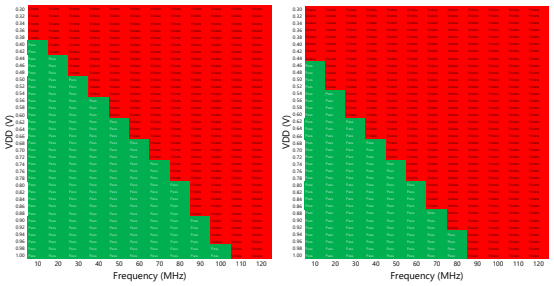
D. Reconfiguration Time

Next, we evaluated the time required to reconfigure the SLM blocks. To measure the time, we employ a bitstream, which configures four memory-mapped 32-bit registers in an

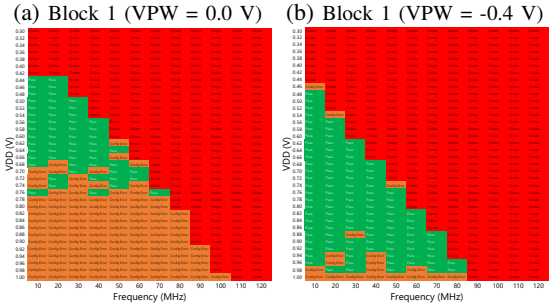
TABLE II: Comparison of the evaluation results when each operates at the maximum frequency

App.		SLMLET(soft)	SLMLET	ESP32[a]	RP2040[b]	GW1NR-9
sram_memcpy	f_{max}	300 MHz	100 MHz	n/a	n/a	n/a
	Cycle	**28,679**	32,994	n/a	n/a	n/a
	Power	428.6 mW	268.5 mW	n/a	n/a	n/a
	Latency	95.60 us	329.9 us	n/a	n/a	n/a
	Energy	40.98 uJ	88.60 uJ	n/a	n/a	n/a
CRC32	f_{max}	300 MHz	70 MHz	240 MHz	200 MHz	50 MHz
	Cycle	14,117	**2,151**	n/a	n/a	n/a
	Power	393.3 mW	170.0 mW	312.5 mW	154.8 mW	281.9 mW (comp), 268.3 mW (trans)
	Latency	47.05 us	**30.73** us	77.58 us	66.59 us	885.7 us
	Energy	18.50 uJ	**5.22** uJ	24.24 uJ	10.31 uJ	335.0 uJ
AES128	f_{max}	300 MHz	28 MHz	240 MHz	200 MHz	34 MHz
	Cycle	1,865	**976**	n/a	n/a	n/a
	Power	426.8 mW	158.9 mW	306.0 mW	158.9 mW	618.5 mW (comp), 278.7 mW (trans)
	Latency	**6.217** us	34.86 us	7.814 us	14.00 us	1333 us
	Energy	2.653 uJ	6.518 uJ	2.391 uJ	**2.223** uJ	237.9 uJ

[a] The maximum frequency configurable by the user is 240MHz
[b] We tested up to 200MHz



(a) Block 1 (VPW = 0.0 V) (b) Block 1 (VPW = -0.4 V)



(c) Block 2 (VPW = 0.0 V) (d) Block 2 (VPW = -0.4 V)

Fig. 8: Operating range of SLM blocks

SLM block. Figure 7 shows the elapsed time to reconfigure both blocks with and without the TLC when the operating frequency is 50MHz. The measured time includes the time to load the bitstream to the shared SRAM and the time to write the bitstream to the SLM block by the configuration controller.

The LB utilization is 29.3%, and the compressed bitstream size is 49.8% of the original size. As explained in Section IV-A, the written bitstream size to the SLM block is fixed regardless of the compression, but the time to load the bitstream can be reduced. Therefore, the compressed bitstream reduces the bitstream load time by 24.8% and 33.9% for the sequential and parallel reconfiguration, respectively. Consequently, combining the compression and parallel reconfiguration reduces 51.5% of the time to reconfigure both blocks when compared to the uncompressed sequential reconfiguration.

E. Performance of SLM blocks

Figure 8 shows shmoo plots of the operating range of the SLM blocks with the same bitstream used in the reconstruction time evaluation. The red area indicates the failure of the operation, and the green area indicates the success. Block 1 operates as expected, and the operating frequency is 100MHz

TABLE III: FPGA resource usage comparison

		SLMLET	GW1NR-9
sram_memcpy	Logic	378 (42.2%)	n/a
	FF	164 (18.3%)	n/a
	LB	100 (45.3) %	n/a
CRC32	Logic	377 (42.1%)	1981 (22.9%)
	FF	131 (14.6%)	911 (13.6%)
	LB/CLS	114 (50.8%)	1422 (32.9%)
	BSRAM	n/a	3 (12.0%)
AES128	Logic	784 (87.5%), 844 (94.2%)	3427 (39.7%)
	FF	293 (32.7%), 139 (15.5%)	1337 (20.0%)
	LB/CLS	211 (94.2%), 220 (98.2%)	2472 (57.2%)
	BSRAM	n/a	2 (8.00%)

with zero bias voltage. If -0.4 V of the reverse bias voltage is applied, the operating frequency decreases to 80MHz while the leakage power also decreases, as shown in Fig. 5.

On the other hand, block 2 does not operate as expected when a high supply voltage is applied, even at 10MHz. Further analysis revealed that the configuration controller does not correctly reconfigure the block. Therefore, we conducted experiments under different voltage conditions during reconfiguration and FPGA operation. The orange area in Fig. 8c and Fig. 8d indicates that the circuit built in the SLM block 2 correctly operates, but the configuration controller does not work as expected. These results indicate the operating range of both blocks is almost the same. The issue of the configuration controller is still under investigation.

F. Case study as an edge device

Lastly, we evaluated the SLMLET SoC using the following applications: 1) sram_memcpy: 1KB data copy in the shared SRAM, 2) CRC32: CRC32 calculation from 1KB binary data (polynomial: 0x04C11DB7), and 3) AES128: 128-bit AES encryption.

Two versions of each application were implemented: a hardware version utilizing the SLM blocks and a software version that runs only on the RISC-V core. The software implementations of CRC32 and AES128 are based on MiBench. Regarding the AES128, the hardware version uses two SLM blocks because a single block is not large enough to implement the entire application.

Except for sram_memcpy, we also compared the SLMLET with two commercially available microcontrollers: Espressif Systems ESP32 and RP2040 (Raspberry Pi Pico). Both of them are fabricated with a 40nm process and have a dual-core processor. Moreover, we compared the SLMLET SoC with a discrete FPGA, Gowin GW1NR-9, which is mounted on the

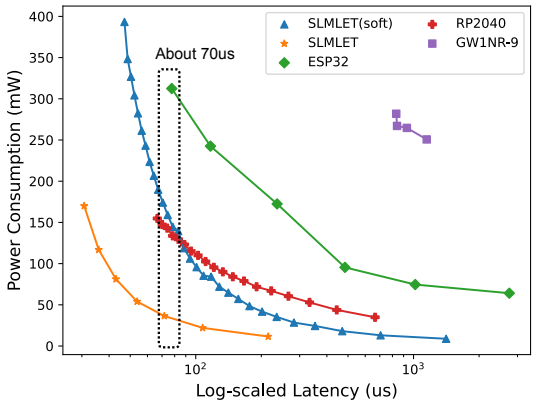


Fig. 9: Power consumption comparison for CRC32

Tang Nano 9K board. Due to the discrete FPGA, GW1NR-9 does not have a tightly connected processor, unlike the SLMLET SoC. Thus, data transfer between a host processor and GW1NR-9 is performed via UART with an available maximum baud rate of 921600 bps. The UART module is also made with its reconfigurable logic using a UART IP.

The results are summarized in Table II. The software implementations of SLMLET achieve the maximum operating frequency of 300MHz, regardless of the application, when the supply voltage is 1.0 V. Even at the standard voltage of 0.9V, the RISC-V core operates at 280MHz. Because of the 16-bit width channel between the shared SRAM and the SLM blocks, the hardware version of sram_memcpy cannot outperform the software version. Due to the limited operating frequency of the SLM block, it takes three times longer to execute the hardware version than the software version. However, the RISC-V core can execute other tasks while the SLM block is copying data.

Regarding AES128, the critical path across two SLM blocks results in a low operating frequency of 28MHz. Despite the half of the cycle count, the latency of the hardware version is longer than the software version because of the low operating frequency. Likewise, the other microcontrollers also outperform the hardware version in terms of latency and energy consumption. However, the hardware version is faster than the software version when the operating frequency is around 30MHz. Due to the limited logic resources, the used hardware version still has the potential for further acceleration.

The hardware version of CRC32 is the most successful among the three applications. The hardware version reduces the cycle count by 85% compared to the software version. Although the maximum operating frequency of the SLM block is 70MHz, the latency of the hardware version is shorter than that of the software version. As a result, the hardware version reduces energy consumption by 70%, 21%, and 50% compared to the software version of SLMLET, ESP32, and RP2040, respectively.

Figure 9 shows the power consumption of each device when executing CRC32. The hardware version always lower power consumption than the other devices. When comparing the power consumption with about 70us of latency, the hardware version, the software version, ESP32, and RP2040 consume 36mW, 159mW, 312mW, and 133mW, respectively. The lowest power consumption of the hardware version is achieved at 30MHz with a supply voltage of 0.64V and a reverse bias voltage of -0.3V. The power reduction compared to the software version is 77%.

Finally, we compared the SLMLET SoC with GW1NR-9. Table III gives a detailed comparison of the FPGA resource usage between SLMLET and GW1NR-9. The power consumption of GW1NR-9 is very different between the computation and data transfer. That is why the two power consumptions are reported in Table II. Due to the logic consumption of the UART, it needs more resources than the SLMLET. Furthermore, the execution time of GW1NR-9 is dominated by the data transfer through UART, even though the processed 1KB data transfer time is excluded from the measured time for CRC32. The dynamic power consumption is not negligible, even in the standby state, and it consumes about 250mW. As a result, the energy consumption of GW1NR-9 is one to two orders of magnitude larger than the SLMLT SoC.

GW1NR-9 is based on 5-LUT FPGA and fabricated with a 55nm process, which is the same feature size as the SLMLET prototype chip. Thus, the achieved operating frequency is similar to the hardware version of the SLMLET. However, the SLMLET has a margin to apply a reverse bias of -0.3V at 50MHz for CRC32, reducing the leakage power to 5mW.

VI. Conclusion

This paper presents the SLMLET SoC, which integrates a RISC-V processor core and SLM-based eFPGA blocks, and the developed HW/SW co-design flow. The experimental results demonstrate that the SLMLET can reduce energy consumption by 77%, offloading compute intensive tasks to application-specific hardware on the SLM blocks. Moreover, the tightly coupled RISC-V core and SLM blocks expand the range of applications that benefit from the hardware acceleration. For future work, we will investigate methods to determine the optimal voltage from FPGA CAD timing reports and other information to improve energy efficiency further.

Acknowledgment

This work was supported by VDEC in collaboration with Cadence Design Systems, Inc. Ths work was also supported by JST CREST JPMJCR19K1 and JST PRESTO JPMJPR22P5.

References

[1] S. Biookaghazadeh, M. Zhao, and F. Ren, "Are FPGAs Suitable for Edge Computing?" in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.
[2] Xilinx, "Zynq 7000 data sheet:overview," https://docs.xilinx.com/v/u/en-US/ds190-Zynq-7000-Overview.
[3] M.Kuga, et.al., "An eFPGA Generation Suite with Customizable Architecture and IDE," *IEICE Trans. on Fund. of Elect. , Comm. and Comput. Sciences*, vol. 106, no. 3, pp. 560–574, 2023.
[4] Chang, Ting-Jung, et.al., "CIFER: A 12nm, 16mm 2, 22-Core SoC with a 1541 LUT6/mm 2 1.92 MOPS/LUT, Fully Synthesizable, CacheCoherent, Embedded FPGA," in *2023 IEEE CICC*. IEEE, 2023, pp. 1–2.
[5] e. S.P.Davice, "Arnold: An eFPGA-augmented RISC-V SoC for flexible and low-power IoT end nodes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 4, pp. 677–690, 2021.
[6] Renzini, Francesco, et.al., "A fully programmable eFPGA-augmented SoC for smart power applications," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 67, no. 2, pp. 489–501, 2019.
[7] H. Amano, *Principles and structures of FPGAs*. Springer, 2018.
[8] X. Tang, E. Giacomin, B. Chauviere, A. Alacchi, and P.-E. Gaillardon, "OpenFPGA: An open-source framework for agile prototyping customizable FPGAs," *IEEE Micro*, vol. 40, no. 4, pp. 41–48, 2020.
[9] T. Takagi, et.al., "Tag-less compression for FPGA configuration data," in Proc. of SASIMI 2022, 2022, pp. 81–82.
[10] M.Amagasaki, et.al, "Slm: A scalable logic module architecture with less configuration memory," *IEICE Trans. on Fund. of Elect. , Comm. and Comput. Sciences*, vol. 99, no. 12, pp. 2500–2506, 2016.
[11] K. Fujita, Y. Torii, M. Hori, J. Oh, L. Shifren, P. Ranade, M. Nakagawa, K. Okabe, T. Miyake, K. Ohkoshi *et al.*, "Advanced channel engineering achieving aggressive reduction of V T variation for ultra-low-power applications," in *2011 International Electron Devices Meeting*. IEEE, 2011, pp. 32–3.
[12] Q. Zhao, M. Amagasaki, M. Iida, M. Kuga, and T. Sueyoshi, "An automatic FPGA design and implementation framework," in *23rd FPL*. IEEE, 2013, pp. 1–4.
[13] RISC-V Software, "riscv-mini," https://github.com/riscv-software-src/riscv-tests.