2023 IEEE 16th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC) | 979-8-3503-9361-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/MCSoC60832.2023.00013

A remote partial-reconfigurable SoC with a RISC-V soft processor targeting low-end FPGAs

Yuji Yamada
School of Computing
Tokyo Institute of Technology
Tokyo, Japan
yamada@arch.cs.titech.ac.jp

Nesrine Berjab
School of Computing
Tokyo Institute of Technology
Tokyo, Japan
berjab@c.titech.ac.jp

Tomohiro Yoneda
School of Computing
Tokyo Institute of Technology
Tokyo, Japan
yoneda@arch.cs.titech.ac.jp

Kenji Kise
School of Computing
Tokyo Institute of Technology
Tokyo, Japan
kise@c.titech.ac.jp

***Abstract*—FPGAs are versatile devices that offer high performance and programmability, making them attractive solutions for diverse applications ranging from edge IoT systems to data center accelerators and beyond. Additionally, partial reconfiguration (PR) is a powerful technique in FPGA design that enables dynamic reconfiguration of specific regions.**

This paper proposes an FPGA-based SoC that incorporates remote PR and data management capabilities. Moreover, the proposed SoC includes a RISC-V soft processor and leverages TCP communication for remote management and control of the PR process. Its design is intended to be suitable for low-end FPGAs. Besides, since the proposed SoC uses an AXI-Stream interface for communication with the user circuit, the user can easily design and run various circuits using PR. We implement the proposed SoC on the Arty A7-35T FPGA board and verify its functionality using a convolutional neural network (CNN) accelerator operating on the MNIST dataset and a soft processor as an application circuit. The stability and correct operation of TCP/IP communication are confirmed even when the application circuit is frequently reconfigured. We measure the resource utilization and elapsed time needed to configure the application circuit in the proposed SoC. The feasibility evaluation results indicate that the proposed SoC is suitable for real-world use. The number of LUTs, FFs, and BRAMs consumed by the proposed PR controller are reduced to 55%, 24%, and 14%, respectively, compared to previous work, while the throughput is higher than that of a typical JTAG-based configuration.

***Index Terms*—FPGA, partial reconfiguration, TCP/IP, RISC-V, remote partial-reconfigurable SoC**

I. Introduction

The widespread use of IoT (Internet of Things) devices at the edge of the network has led to an explosion in the volume of data generated by various sensors and devices. The volume of the generated data can overwhelm network bandwidth and lead to latency issues if all data is sent to the cloud for processing. Edge computing has emerged as an efficient solution to move data processing and analysis closer to the edge of the network.

The FPGA technology has been proven to be an efficient solution for edge computing due to its unique characteristics and capabilities of configuration and customization to the specific hardware functions. This flexibility allows developers to tailor the FPGA to the specific requirements of edge computing by optimizing its performance and resource utilization.

The configuration of a remote FPGA from a local host can significantly improve development efficiency, as it enables easier updates and debugging. There are several ways to load the configuration bitstream file into the FPGA. JTAG is the typical interface for configuration where a direct cable connection between the programming host (usually a PC) and an FPGA is used. However, the remote FPGA configuration using long JTAG cables is not feasible. Therefore, a developer can configure a remote FPGA via an additional network-enabled controller, such as a PC, where the controller and FPGA are connected with a JTAG cable.

On the other hand, the use of an additional remote controller will increase the costs and complexities. This problem can be mitigated by implementing the remote controller within the programmable logic of the FPGA itself and utilizing the partial reconfiguration (PR) techniques, which allows specific regions of the FPGA to be reconfigured on-the-fly without affecting the entire device.

In PR systems, the programmable logic is divided into two areas: static logic and reconfigurable partition. In this paper, we call them *static area* and *dynamic area*. The static area has a consistent function through PR. It is used for critical functions such as controllers of network communication and PR. This allows the FPGA to be a network-enabled device without an additional external controller. The dynamic area can be reconfigured by PR. A bitstream file to configure a dynamic area of PR is called a *partial bitstream file*. A hardware module mapped to a dynamic area is called a *reconfigurable module*.

The purpose of this work is to realize a remote partial-reconfigurable and practical SoC using an FPGA vendor-independent soft processor, targeting low-end FPGAs and allocating small hardware resources for the static area. We select a RISC-V soft processor as an FPGA vendor-independent soft processor. Our target device is Arty A7-35T, a low-end FPGA board with AMD XC7A35T FPGA. To realize a practical SoC, we select a reliable network protocol of the TCP communication for the transfer of application data and a partial bitstream file, and the sufficient network throughput for such data transfer over 3MB/s is achieved.

The main contributions of this paper are summarized as follows:

- We propose a remote partial-reconfigurable and practical SoC targeting low-end FPGAs and allocating small hardware resources for the static area. The SoC receives

2771-3075/23/$31.00 ©2023 IEEE
DOI 10.1109/MCSoC60832.2023.00013

a partial bitstream file over the TCP communication. It also receives application data to be input into a reconfigurable module and returns its output to the remote host. The proposed SoC uses an AXI-Stream interface for communication to realize a flexible SoC platform.

- We implement the proposed SoC on an Arty A7-35T FPGA board and verify its functionality using two use cases. The first one uses a convolutional neural network (CNN) accelerator of MNIST [1], and the other uses a soft processor of typical five-stage pipelining as reconfigurable modules. We confirm that the TCP/IP communication of the SoC is stable where PR is performed frequently.
- To show the effectiveness of the proposal, we measure the FPGA resource utilization and network performance of the SoC. The results show that the proposal is practical and suits real-world use. The number of LUTs, FFs, and BRAMs consumed by the static logic is reduced to 55%, 24%, and 14%, respectively, compared to REoN [2], achieving a higher network throughput than a JTAG-based scheme.

II. Technical Background and Related Works

A. Partial bitstream file and ICAP

A partial bitstream file is generated with the source code of a reconfigurable module and the files to specify the floor plan of a static area using EDA tools. The partial bitstream file of AMD FPGAs consists of a header, a sync word, and an instruction sequence. The header is comprised of metadata such as the model number of a target FPGA and is mainly used by vendor tools. The sync word is a 32-bit magic number of 0xAA995566, following the instruction sequence for configuration. The instruction sequence contains the addresses of the configuration memory of the FPGA and the values to be written there.

Our target FPGA has a design element called Internal Configuration Access Port (ICAP) [3]. PR is done by inputting a partial bitstream file into ICAP, which has a 32-bit input port and operates at up to 100 MHz, resulting in a PR throughput of 400 MB/s at the maximum.

ICAP is used on the proposal by instantiating it in our Verilog HDL code.

B. Building RISC-V SoC with LiteX

An open-source framework called LiteX [4] supports efficient design and implementation of SoCs. It can generate Verilog HDL codes of a SoC with processors and API functions used in software executed on the processor. It supports various FPGA boards, and the generated SoC can be operated on a selected board.

LiteX offers many processors with different instruction set architectures (ISA) and microarchitectures. Since RISC-V is an open ISA and independent of FPGA vendors, various RISC-V processors [5]–[7] are available.

We select RISC-V and use the LiteX framework to generate a part of the proposed SoC.

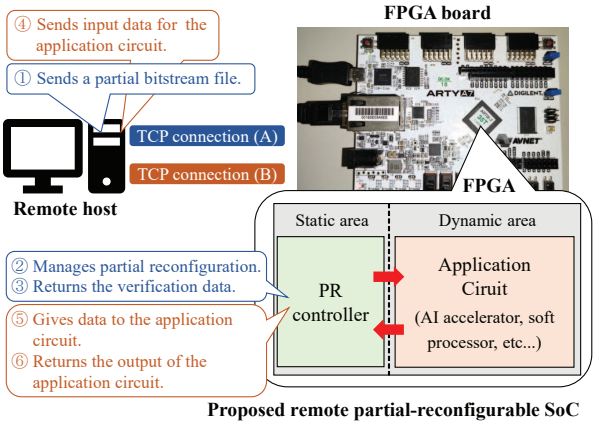


Fig. 1: The overview of our remote partial-reconfigurable SoC.

C. Related works

Some works [2], [8] exploit the strength of remote PR for network processing applications. To minimize hardware resource utilization, Mishra et al. [2] designed REoN, a protocol to enable reliable and efficient software-defined PR over a network. This protocol enables remote PR, allowing users to update and modify their functionality over a network. The proposed framework has been tested with a high-end FPGA, and REoN might not be suitable for certain low-end FPGAs (i.e. XC7A35T) with limited hardware resources. REoN can indeed be resource-intensive, particularly in terms of on-chip memory.

Some works [9], [10] propose the complete hardware implementations of the TCP/IP network stack. Implementing a hardware-based TCP module can consume a significant amount of FPGA resources for TCP packet processing, buffering, and flow control. Such a complete hardware approach is not suitable for the remote SoC targetting low-end FPGAs.

Chihara and Yamawaki [11] use Zynq FPGA for remote PR, where an onboard Arm processor is used as a PR controller. This method has the disadvantage that some FPGAs without a hard processor, including our target device of AMD XC7A35T, are not supported for this approach.

III. The Proposed FPGA-based SoC

The overview of the proposed SoC is shown in Fig. 1. The programmable logic of an FPGA has static area and dynamic area. The static area is used as the partial reconfiguration and data management controller named *PR controller*, which controls network, data transfer, and PR. The dynamic area is used for a reconfigurable module. The circuit realized as a reconfigurable module, such as AI accelerators and customized processors, is called *application circuit* in this paper.

The remote host and the PR controller establish two TCP connections named (A) and (B), which are used to transfer a partial bitstream file and application data, respectively.

We define the step-by-step usage of the SoC as follows.

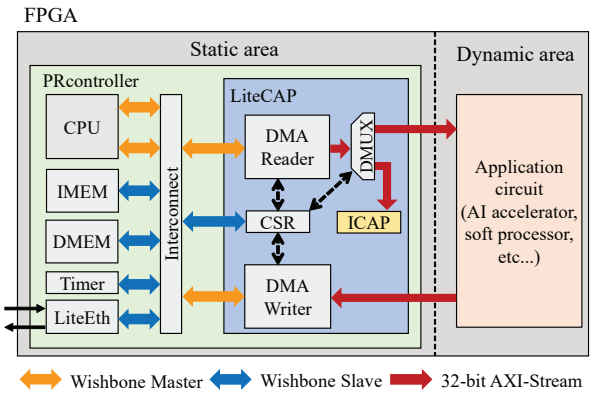


Fig. 2: The organization of a remote partial-reconfigurable SoC with a RISC-V soft processor targeting low-end FPGAs.

① The remote host sends a partial bitstream file to the SoC over TCP.
② The PR controller receives the segmented partial bitstream file from the remote host and performs PR for each received TCP segment.
③ When PR is complete, the controller returns the parity of the partial bitstream file, which is calculated during PR, to the remote host.
④ The remote host sends application data to the application circuit over another TCP connection.
⑤ The controller takes the segmented application data from the remote host and inputs it to the application circuit for processing.
⑥ When the application circuit finishes the process, it returns the application output to the remote host.

The first three steps use TCP connection (A) to transfer a partial bitstream file, while the latter steps use TCP connection (B) to transfer application input and output data. These two connections cannot be used at the same time.

How the above steps are defined simplifies the hardware organization of the SoC to be designed. Specifically, in step ②, by performing partial reconfiguration for each received TCP segment, the PR controller doesn't require large buffers to hold the entire bitstream file at once. Step ⑤ is also conducted in the same way. This can lead to lower block RAM consumption as the system handles smaller segments (Segments can be either bitstreams or application data) at a time.

A. Hardware design and implementation

Our designed hardware organization of the proposed SoC is shown in Fig. 2. The PR controller handles network communication, data transfer, and PR. To reduce the FPGA resource utilization, protocol stacks are not implemented as hardware. Instead, the protocol stack is implemented as software, and a soft processor in the SoC executes the software.

We use VexRiscv [5], which is a pipelined scalar processor of RISC-V achieving high performance with relatively low usage of hardware resources, as CPU on the upper left of Fig. 2. We do not use a virtual memory system that translates physical addresses to virtual addresses to reduce the hardware resources.

IMEM and DMEM are an instruction memory and a data memory, respectively. They are implemented using BRAMs. Although the LiteX SoC can run the operating system, we select not to use the operating system because we tries to reduce the hardware resource for these memory. We code some part of the software in C for this SoC.

Timer is a module with a 32-bit counter whose value is increased by 1 every clock cycle. It is used by the CPU to calculate the processing time.

LiteEth is a module to send and receive Ethernet frames via the Ethernet PHY outside the FPGA. MTU (Maximum Transmission Unit) for communication is set to the default value of 1,530 bytes. This module has multiple slots to store frames for each send/receive buffer, and the number of slots is set to the default value of 2 from the trade-off between performance and hardware resources. These buffers are realized using BRAM.

Interconnect is used as the wishbone bus for data transfer between the modules described so far.

LiteCAP is the original module for data transmission. The details of this module are described in the next subsection.

B. LiteCAP, our original hardware module

LiteCAP is designed for data transmission from DMEM to ICAP and between an application circuit and DMEM. It is designed by referring to RV-CAP [12] and adding unique functions while deleting unnecessary functions.

LiteCAP has five elements: (1) DMA Reader, (2) DMA Writer, (3) CSR (Configuration and Status Register), (4) DMUX (de-multiplexor), and (5) ICAP.

DMA Reader sends the data read from DMEM to DMUX with 32-bit AXI-Stream. This is controlled by CSR (dashed arrow). DMA Reader also calculates the parity of a partial bitstream file to verify its transmission and writes the result to CSR.

DMUX has a de-multiplexer that selects and sends data from DMA Reader to ICAP or an application circuit. The ICAP-side output has a combinational circuit for bit exchange required by ICAP. This module is controlled by CSR.

ICAP configures dynamic area using a partial bitstream file given by DMUX.

The input data to be processed by the application circuit is received from the DMUX, and the output data of the application circuit is transferred to DMA Writer.

DMA Writer writes the output of the application circuit to the DMEM. Since the output may be unstable during PR, this module has a mechanism to avoid its effect. This module is controlled by CSR.

In this way, we aim to reduce the hardware resource by implementing essential modules, increasing the performance using DMA transfers.

C. Software design and implementation

The network protocol is handled by lwIP [13], an open-source lightweight TCP/IP protocol suite written in C language. It operates in two distinct modes: OS mode and bare metal mode. We select the latter to reduce the FPGA resource usage.

To reduce the size of the entire software, we turn off the UDP and DHCP features of lwIP. We reduce the number of frame buffers in DMEM and the state machines for network connection. The standard I/O functions, such as printf, are not used, and the library is not linked to the software.

We write the device driver of approximately 200 lines of code to port lwIP to the SoC. We use the API functions of LiteX to implement the following features:

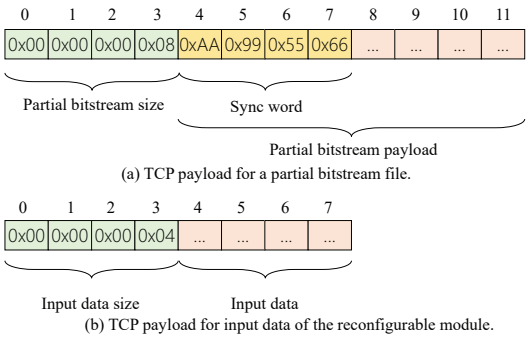
- Initialization: The SoC's MAC address, IP address, and the Maximum Transmission Unit (MTU) used by lwIP are initialized to the standard value of 1,500 bytes.
- Processing the received frames: The device driver copies the received frame stored in the LiteEth buffer to DMEM and calls a function provided by lwIP to process it. This feature is activated upon the reception of a frame.
- Transmission frame from DMEM to LiteEth: The device driver copies the transmission frame from DMEM to LiteEth. An API function controlling LiteEth is called to initiate the transmission.

We also write the software for the application layer using approximately 200 lines of code in C language. This software is called the *PR software*. The PR software configures LiteCAP to select the data path and transfers the data stored in DMEM to LiteCAP by launching DMA Reader according to the size of the data received with TCP. It also transfers the output data of the application circuit with DMA Writer. CPU busy-waits until the transfer is completed.

We design a simple application layer protocol above TCP for this, as shown in Fig. 3. The TCP payload first stores a 4-byte value that indicates the number of bytes of data to be transferred on the connection. Next, in the PR case (a), the sync word and data of a partial bitstream file follow. If it is not the PR case (b), the input data for the application circuit follows. It is checked by the software for the application layer, and the result is used to let DMUX select an appropriate path until the connection is closed.

In the case of (a), the software returns the parity of the partial bitstream file to the remote host as a TCP payload by calling lwIP functions after completing the DMA reading. In the case of (b), on the other hand, the software works to return the output data from the application circuit. The software waits until DMA Writer completes writing the output data. After that, the software returns the data to the remote host as a TCP payload.

We implement the above functions without the interrupt. We set the TCP MSS (Maximum Segment Size) to the standard value of 1,460 bytes, and the window size to twice the MSS.



(a) TCP payload for a partial bitstream file.

(b) TCP payload for input data of the reconfigurable module.

Fig. 3: The format of the simple application layer protocol. The TCP payload sending an 8-byte partial bitstream file (a) and sending 4-byte input data (b).

IV. Verification and evaluation

A. Verification and evaluation environment

For verification and evaluation, we use Arty A7-35T FPGA board, which is a low-end board with an AMD XC7A35T FPGA and capable of communicating over 100 Mbps Ethernet. The part of LiteEth that connects to its Ethernet PHY runs at 25 MHz. Other circuits of the proposed SoC run at 100 MHz.

For implementation, we use Vivado v2022.2, lwIP 2.1.3, riscv64-unknown-elf-gcc 12.2.0, and LiteX, whose commit date is March 8, 2023. Vivado default optimization schemes are used.

The VexRiscv CPU core is configured for RV32I with a single-cycle shifter, pipeline register bypass, and static branch predictor. Any instruction cache, data cache, multiplier, and divider are not used.

BRAM on an FPGA is used to store the entire software. The executable file of our software is 30,790 bytes, and the size of IMEM (instruction memory) is set to the same bytes.

The size of DMEM (data memory) can be estimated from the sum of the sizes of the `.data` and `.bss` sections of the software. Our software requires 8,088 bytes for these sections. By adding the stack and heap size, the DMEM size is set to 10,240 Bytes. Thus, the total on-chip memory for the software is about 40 KB.

Fig. 4 shows the area configuration, where dynamic area is shown by a purple bold square, and the rest part is used as static area. We use two application circuits to verify and evaluate the proposed SoC: A CNN accelerator and a RISC-V soft processor.

B. A CNN accelerator used as an application circuit

The first application circuit is a CNN accelerator described here.

Fig. 5 shows the network structure of the CNN accelerator for the MNIST dataset. This accelerator infers the handwritten digits of an 8-bit black and white image of 28×28 pixels. The input and output ports of this accelerator are 8-bit wide.

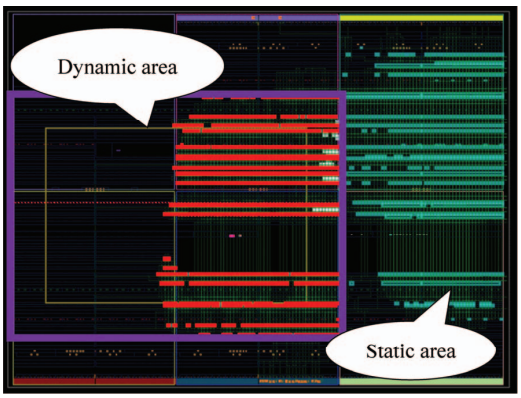


Fig. 4: The dynamic area (purple bold square) assigned to FPGA and the placement and routing result of a CNN accelerator is shown as red elements.

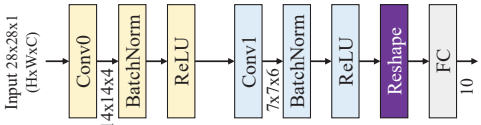


Fig. 5: The network structure of the CNN accelerator for the MNIST dataset.

Both weights and activation functions are quantized with a 4-bit width. This module is implemented with an NN accelerator framework called FINN [14].

After the placement and routing of this module, the partial bitstream file is generated. The result of configuring dynamic area with this file is shown as the red elements in Fig. 4. This module uses 3,581 LUTs and 3,449 FFs. Despite this utilization, there is still plenty of available resources within the low-end FPGA.

C. A soft processor used as an application circuit

The other application circuit is a RISC-V soft processor of an in-order scalar execution model, which is developed in Verilog HDL.

The block diagram of this processor is shown in Fig. 6. It has five pipeline stages (IF, ID, EX, MA, and WB) targetting RV32I instruction set except for fence, fence.i, ecall, ebreak, and CSR instructions.

The components of the yellow color are memory-related, the blue and red colors indicate combinational circuits, and the green color indicates pipeline registers.

When the processor core detects valid data input from the AXI-Stream interface, it releases the reset signal, and the processor begins operation. The instruction memory (imem) and data memory (dmem) are realized using BRAM, each with a capacity of 256 bytes. A program that prints "Hello, world!" is embedded in this instruction memory, and the result is transferred to the AXI-Stream interface. This module uses 1,557 LUTs, 1,498 FFs, and 2 BRAMs.

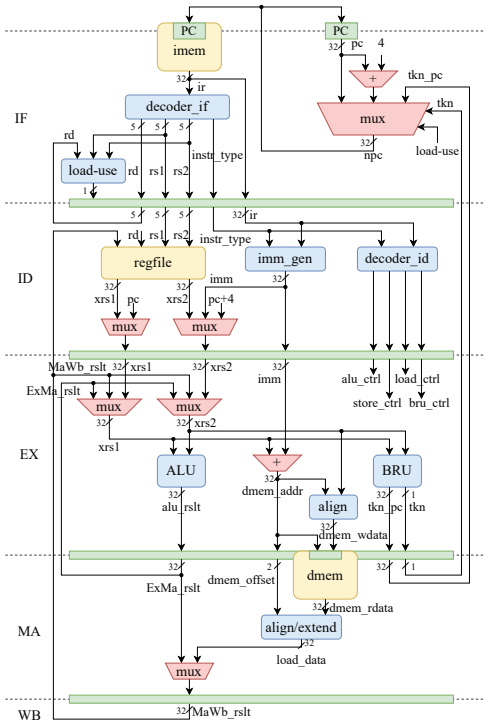


Fig. 6: The block diagram of a soft processor used as an application circuit.

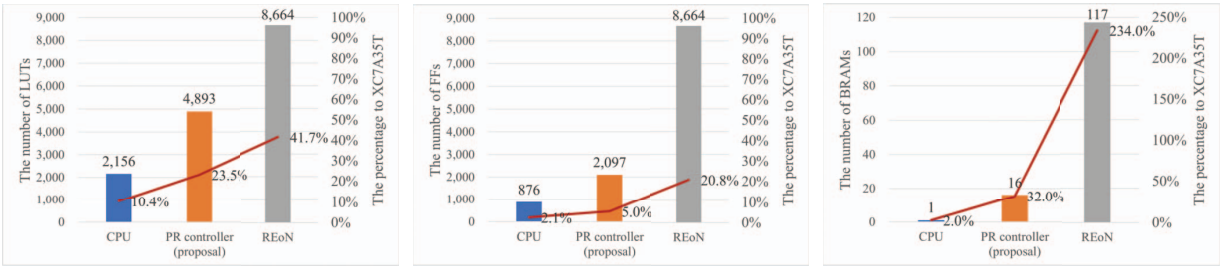
D. Verification result of the SoC

To verify that the proposed SoC works correctly in a realistic and noisy environment, we connect the proposed SoC and a PC to the LAN of our laboratory using an Ethernet switch, and we execute the remote partial reconfiguration repeatedly.

We continuously send the partial bitstream file of the CNN accelerator from the PC to the SoC. In this verification, 100 different MNIST images are used, and each image is sent to the SoC 100 times. We confirm that for each image, the SoC returns the correct inference result of the CNN accelerator, where the correctness of the inference result is confirmed by comparing the obtained results from FPGA and the results from Verilog HDL simulations. From the results of 21 hours of continuous operation of the SoC, we confirm that the SoC correctly performs 54,300 partial reconfigurations without any errors.

We test the partial reconfigurations with the bitstream file of the second application circuit of the soft processor and sending input data from a PC to the SoC repeatedly. We confirm that the SoC outputs the correct data of "Hello, world!" without any errors.

Through these verifications, we confirm that the proposed SoC works correctly and stably in a real-world environment.



(a) LUT Utilization. **(b)** FF Utilization. **(c)** BRAM Utilization.

Fig. 7: The amount of hardware used by CPU of the proposed SoC, the whole static area, and REoN.

E. Evaluation results: hardware resources

We measure the hardware resources of CPU, PR controller, and REoN [2], and the obtain data is shown in Fig. 7. The (a), (b), and (c) in this figure show the number of LUTs, FFs, and BRAMs, respectively.

This figure also shows the percentage of hardware resources to XC7A35T FPGA as red lines. The resource utilization for the PR controller, represented by the orange bars, is calculated by subtracting that of dynamic area from the entire SoC reported by Vivado.

The PR controller uses 4,893 LUTs, 2,097 FFs, and 16 BRAMs. It includes the CPU using 2,156 LUTs, 876 FFs, and 1 BRAM, less than half of the controller.

The resource utilization of REoN shown as gray bars is calculated according to formula (1). n is the resource utilization. p is the percentage of the resource utilization in Table I of [2]. N is the overall resources of the XC7VX690T FPGA [15] on NetFPGA SUME FPGA board used for their evaluations.

$$n = \frac{p}{100} \times N \tag{1}$$

The calculated resource utilizations of REoN are 8,664 LUTs, 8,664 FFs, and 117 BRAMs. REoN using a hardware-offloaded protocol stack consumes more BRAMs than the total number of BRAMs available in XC7A35T. This indicates that REoN is not suitable for certain low-end FPGAs with limited hardware resources.

The number of LUTs, FFs, and BRAMs consumed by the PR controller is reduced to 55%, 24%, and 14%, respectively, compared to those of REoN.

F. Evaluation results: processing time and throughput

To show the effectiveness of the proposal, we compare the processing time and throughput of the proposed PR method with the conventional JTAG-based PR of Vivado.

We use the following environment to measure the elapsed time for communication and PR. The WSL2 Ubuntu 22.04.2 virtual machine on a Windows 10 computer is used as the PC. This computer has an Intel Core i7-11700K CPU and 64 GB of main memory. The PC motherboard and the FPGA board are directly connected with a LAN cable.

We implement a Python script to send a partial bitstream file and measure the processing time. The measurement time is from the establishment of the TCP connection to the receipt of the parity. For JTAG-based PR, we connect the PC and FPGA board with a USB 2.0 cable. The elapsed time for `program_hw_devices` command of Vivado is measured by using `clock` command of Vivado.

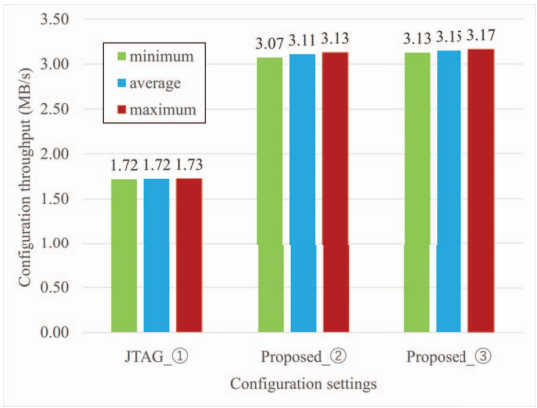


Fig. 8: The partial reconfiguration throughput in three settings.

We use three types of partial bitstream files to measure the throughput. (1) 2,017,663 bytes file of the accelerator for MNIST, including headers. (2) 2,017,492 bytes file of the accelerator starting with the sync word. (3) 10 Mbytes file of the accelerator made by adding about 8 Mbyte of zero padding. Since the size of the bitstream file for the CNN accelerator and the soft processor is the same, we use the files for the CNN accelerator here.

Fig. 8 shows the evaluation results of the partial reconfiguration throughput, which is calculated from the size of the bitstream file and the processing time. JTAG_① indicates the throughput of the setting of JTAG-based PR with the file (1). Proposed_② and Proposed_③ indicate the throughput of the proposal with the files (2) and (3), respectively. We measure the processing time ten times in each setting. The green, blue, and red bars indicate the minimum, average, and maximum throughput of the ten trials.

The evaluation results indicate that the proposal's throughput exceeds 3 MB/s, surpassing the throughput of the JTAG-based PR, which operates at 1.72 MB/s.

The discrepancy in throughput between REoN utilizing 10 Gbps Ethernet (63.1 MB/s) and the proposal using 100 Mbps Ethernet (3 MB/s) can be contextualized by the specific needs and applications for which each solution is designed. Even with significantly lower throughput, the proposal achieving 3 MB/s might be better suited for certain applications where high-speed communication is not the primary concern. For instance, some systems or applications, such as embedded devices, and IoT sensors, do not necessarily demand high-speed communication but instead focus on consistent, reliable data transmission within their specific constraints.

V. Conclusion

We propose a remote partial-reconfigurable SoC targeting low-end FPGAs. We implement the proposed SoC on an Arty A7-35T FPGA board for operational verification and evaluation.

From the verification, we confirm that the SoC operates correctly in a real-world environment. From the evaluation, we show that the resource utilization of the proposed PR controller is small and suitable for integration into low-end FPGAs. The remote partial reconfiguration of the proposed SoC is faster than JTAG and practical to complete in a realistic time. The number of LUTs, FFs, and BRAMs consumed by the proposed PR controller is reduced to 55%, 24%, and 14%, respectively, compared to REoN [2], while the throughput is higher than that of a typical JTAG-based configuration.

VI. Acknowledgement

This paper is based on results obtained from a project, JPNP23015, subsidized by the New Energy and Industrial Technology Development Organization (NEDO).

References

[1] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[2] V. Mishra, Q. Chen, and G. Zervas, "REoN: A protocol for reliable software-defined FPGA partial reconfiguration over network," in *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2016, pp. 1–7.

[3] Xilinx, "Vivado Design Suite 7 Series FPGA and Zynq-7000 SoC Libraries Guide (UG953)," Version: 2022.2 English, Accessed: 9. Aug. 2023. [Online]. Available: https://docs.xilinx.com/r/2022.2-English/ug953-vivado-7series-libraries

[4] F. Kermarrec, S. Bourdeauducq, J. L. Lann, and H. Badier, "LiteX: an open-source SoC builder and library based on Migen Python DSL," *CoRR*, vol. abs/2005.02506, 2020. [Online]. Available: https://arxiv.org/abs/2005.02506

[5] "VexRiscv," Accessed: 23. Jun. 2023. [Online]. Available: https://github.com/SpinalHDL/VexRiscv

[6] F. Zaruba and L. Benini, "The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology," Jul. 2019. [Online]. Available: https://github.com/openhwgroup/cva6

[7] "Ibex RISC-V Core," Accessed: 16. Aug. 2023. [Online]. Available: https://github.com/lowRISC/ibex

[8] B. Ringlein, F. Abel, A. Ditter, B. Weiss, C. Hagleitner, and D. Fey, "System Architecture for Network-Attached FPGAs in the Cloud using Partial Reconfiguration," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, 2019, pp. 293–300.

[9] M. Ruiz, D. Sidler, G. Sutter, G. Alonso, and S. López-Buedo, "Limago: An FPGA-Based Open-Source 100 GbE TCP/IP Stack," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, 2019, pp. 286–292.

[10] D. Sidler, G. Alonso, M. Blott, K. Karras, K. Vissers, and R. Carley, "Scalable 10Gbps TCP/IP Stack Architecture for Reconfigurable Hardware," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, 2015, pp. 36–43.

[11] Y. Chihara and A. Yamawaki, "An Investigation of Mobile Device Supporting Online Self-dynamic Partial Reconfiguration on a Testbed System." The Institute of Industrial Applications Engineers, 2021.

[12] N. Charaf, A. Kamaleldin, M. Thümmler, and D. Göhringer, "RV-CAP: Enabling Dynamic Partial Reconfiguration for FPGA-Based RISC-V System-on-Chip," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2021, pp. 172–179.

[13] "lwIP - A Lightweight TCPIP stack," Accessed: 22. Jun. 2023. [Online]. Available: https://git.savannah.nongnu.org/cgit/lwip.git/

[14] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. ACM, 2017, pp. 65–74.

[15] Xilinx, "7 Series Product Tables and Product Selection Guide (XMP101)," Version: 1.8 English, Accessed: 9. Aug. 2023. [Online]. Available: https://docs.xilinx.com/v/u/en-US/7-series-product-selection-guide