Exploring Dynamic Partial Reconfiguration in a Tightly-coupled Coprocessor Attached to a RISC-V Soft-processor on a FPGA

Jairo Walber Abdala Castro and Aurelio Morales-Villanueva
College of Electrical and Electronics Engineering
Universidad Nacional de Ingeniería
Lima, Perú
E-mail: jabdalac@uni.pe, amorales@uni.edu.pe

***Abstract*—Dynamic reconfigurable processors take advantage of the flexibility of general-purpose processors architectures to execute instruction programs, and the flexibility of reconfigurable logic, implementing specialized hardware designs with better performance and energy-efficiency. During runtime, the reconfigurable logic is modified to fit calculations of the application. Prior works propose new reconfigurable architectures that lack of a software ecosystem or architectures based in intellectual property instruction sets. In the present work, partial dynamic reconfiguration is implemented to a tightly coupled coprocessor attached to a RISC-V soft-core that executes Linux. The complete system is tested on the Nexys 4-DDR development board with the AES and DES encryption algorithms. The results show a speedup of up to 249.91 times faster execution and an average reconfiguration time of 151 ms.**

***Index Terms*—field programmable gate array, RISC-V, Rocket-chip, partial reconfiguration, tightly-coupled**

I. Introduction and motivation

Algorithms for multimedia, signal processing and artificial intelligence applications are increasing in complexity, demanding computing intensive tasks that require a greater calculous capacity in processors, which is a problem for embedded systems with energy constraints. For this reason, dedicated hardware units are added in systems on chips (SoCs) to accelerate the execution time of special algorithms, getting a better performance than the equivalent software program, but lacking of the flexibility of a general-purpose processing (GPP) architectures when executing different operations.

Reconfigurable computing positions in the middle of application-specific integrated circuits (ASICs) and GPPs, by having a better flexibility than hardwired integrated circuits and getting a better performance than general-purpose processors in algorithms that benefit from parallel execution. The combination of a GPP architecture and a reconfigurable array takes advantage of the flexibility of instruction programs executed in the core of the processor, and an adaptable hardware unit that increases performance. These systems are known as dynamically reconfigurable processors.

The technology that implements the reconfigurable arrays are mainly explored in two options: Field programmable gate arrays (FPGAs) and coarse-grained reconfigurable arrays (CGRAs) [1]. CGRAs include complex programmable functional units that allow a more efficient reconfiguration time and less space occupation when implementing a complex accelerator hardware design. But currently, CGRAs are not a mature technology and is restricted to non-commercial production. On the other hand, FPGAs are the most common silicon devices for reconfigurable computing, and there are plenty of tools and sophisticated chips commercialized in large-scale by companies that have an active research in the area.

FPGAs are built around static random access memory (SRAM) cells, where most SRAM-based FPGAs support partial reconfiguration, a technology that enables resource routing changes in a partially reconfigurable region (PRR), while keeping the static region unaltered. The PRR can be modified during runtime without stopping the execution in the static region, a technique also known as dynamic partial reconfiguration (DPR). Then, the DPR operation available in commercial FPGAs can be used to test a complete dynamically reconfigurable processor architecture.

Dynamically reconfigurable processors are available in some commercial SoCs [2], [3]; however, these processors have been more explored in research papers [1], [4], [5]. The design of dynamically reconfigurable processors can also be adapted from commercial architectures. For example, Rossi et al. [6] proposes a heterogeneous system that implements reconfigurable arrays of different granularities, controlled by a hardcore ARM processor. Also, in [7], the authors propose a reconfigurable instruction set based on ARM that uses an interface connected to the core to decode and buffer data and instructions executed by a reconfigurable coprocessor; where the complete system is implemented in a FPGA.

In this work, we present for the first time, to the best of our knowledge, a tightly-coupled reconfigurable coprocessor attached to a Linux-capable RISC-V processor core implemented in a FPGA. The communication between the coprocessor and the core uses an available interface included in the open-source project Rocket-Chip, a configurable SoC generator design written in Chisel with a customized RISC-V core.

978-1-6654-1221-6/21/$31.00

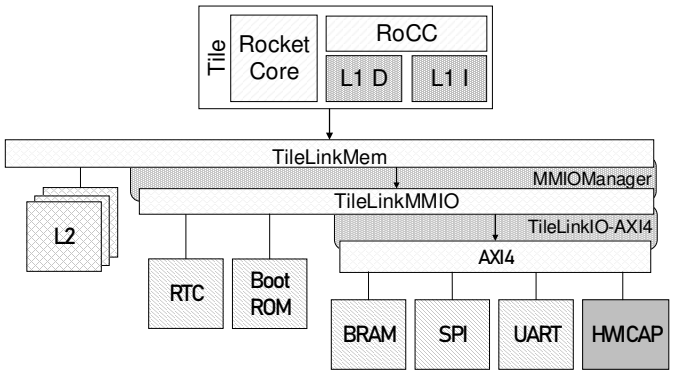


Fig. 1. Interface connections in a lowRISC SoC project for Xilinx FPGAs.

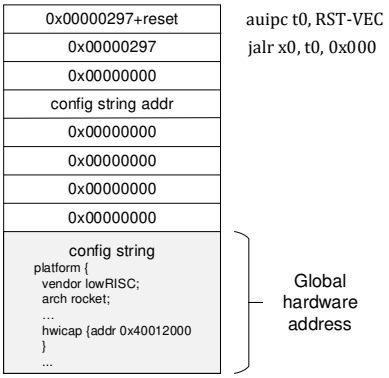


Fig. 2. Contents of the boot ROM.

II. Reconfigurable controller

The present work uses the project lowRISC SoC 0.5 version, an adaptation of 64-bit Rocket-Chip generated processors to get a fully functional open-source SoC capable to execute Linux. The resource usage of the project fits into the Xilinx Artix-7 model FPGA included in the low-end Nexys 4-DDR development board.

A. Internal Access Port of the FPGA

Xilinx FPGAs incorporate the Internal Configuration Access Port (ICAP) for projects that use DPR, where the Xilinx IP core AXI HWICAP is a controller that enables bidirectional data transfers to the ICAP using the standard Advanced Extensible Interface Lite 4 (AXI4-Lite) for an on-chip communication with an embedded processor.

Fig. 1 shows the interface connections in a typical lowRISC SoC project for Xilinx FPGAs, with the AXI HWICAP added to the system. The Rocket tile groups a RISC-V core, a RoCC (Rocket Custom Coprocessor) unit and L1 data and instruction caches; this tile uses the TileLink interface for cache-coherent memory access, which is adapted to TileLinkMMIO for communication with memory-mapped IO devices and then, to AXI4, in order to connect devices compatible with this interface. To attach the AXI HWICAP, the IP core is configured, generated and connected as a slave in the interface, by pairing the AXI4-Lite signals to the master. The base address and mask address of the IP Core are also configured for memory mapped access in AXI4-Lite.

B. Software support for AXI HWICAP

Access to the partial reconfiguration controller is managed on Linux, so a device structure with a description of the available components in the system has to be passed during early initialization. Currently, RISC-V ISA specification implements the device tree standard format to pass the hardware information, however, the Rocket-Chip generator used in the lowRISC SoC 0.5v, implements an older proposed format for RISC-V architecture called configuration string, which is backwards compatible with the device tree and allows customizable information formats.

To pass the AXI HWICAP address information at boot, the base address is specified in a Scala string that represents a simplified device tree, and Chisel compilation writes the content from the Scala string to a boot memory called BootROM. Fig. 2 shows the instructions and the information filled in the Boot ROM, included the AXI HWICAP address description format, where the instructions form the first-stage bootloader, a program that gives the control to the boot program written in BRAM resources of the FPGA.

In Linux, communication in the user space with external hardware requires drivers, so to transfer the bitstream information to the AXI HWICAP, a compiled driver must be added to the Linux executable image. Xilinx provides a software layer for bare-metal applications that makes use of the AXI HWICAP, and a character device driver, for applications executed on Linux, but oriented to the IP cores OPB HWICAP and XPS HWICAP, two older versions of the connected controller in the system. The bare-metal and the character driver include some architecture depended libraries that are mainly focused to be executed on the IP Microblaze soft-core processor, so a customized version of the available driver has to be prepared to be used in the RISC-V soft-core processor, with some modifications in configuration registers and the internal FIFOs manipulation for the AXI HWICAP. Fig. 3 depicts a simplified flowchart of the writing bitstream

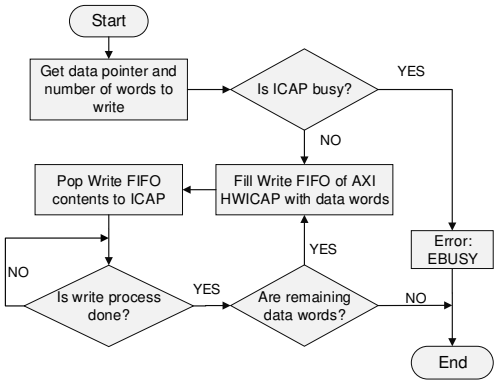


Fig. 3. Simplified flowchart of bitstream configuration using the AXI HWICAP.

process included in the customized AXI HWICAP driver, where the process is executed when a write system-call to the driver is used in the user space.

III. Tightly-coupled coprocessor

The Rocket-Chip generator includes an interface called RoCC directly attached to each RISC-V core in a Rocket tile. The RoCC interface uses the standard RISC-V instruction format with a customized opcode for coprocessor communications. When a custom instruction is detected, the coprocessor receives the information of the source registers, performs the operations and returns the result to the destination register if the register was specified. Signals in the RoCC interface do not only connect the coprocessor with the core, but also, communication is extended to the L1 data cache, the page table walker (PTW), the floating-point unit (FPU), the control and status registers (CSRs) and the arbitrated uncached TileLink (aUTL) as shown in Fig. 4.

In order to enable customized instructions to control and transfer data to the coprocessor, the RISC-V toolchain already supports custom instructions at compilation time, but the Linux kernel source code needs a modification in order to avoid detecting illegal instructions when executing customized instruction programs. The modification in the Linux kernel initializes the custom instructions support by writing in the mstatus CSR in every context switch, which is not secure in a multithreading level, because the coprocessor lacks of a save/restore system and access to the coprocessor can result in process overlaps, but allows programs in the Linux user space to execute customized instructions.

The RoCC unit represents the reconfigurable logic of the full system on the FPGA. An asynchronous behavior between the static region and the reconfigurable region can result in a cross-domain crossing issue; however, the RoCC interface already uses a handshaking ready-valid protocol, that adds a clock cycle latency. During the reconfiguration time, the signals that connect the reconfigurable region to the static region may take unpredictable values, so valid signals of the handshaking protocol can be asserted during the process and initiate a data transfer with the connected elements. In order to avoid an unexpected behavior, the RoCC unit needs another decoupling section prior to being connected to the processor. Fig. 5 shows the decoupling logic implemented in the static region, as an intermediate stage for the reconfigurable coprocessor communication. All the valid signals have a low logic level until the corresponding RoCC instruction is detected, enabling the direct pass of the RoCC signals to the core; when the busy signal is set to low to indicate the end of calculations, the RoCC signals are decoupled again.

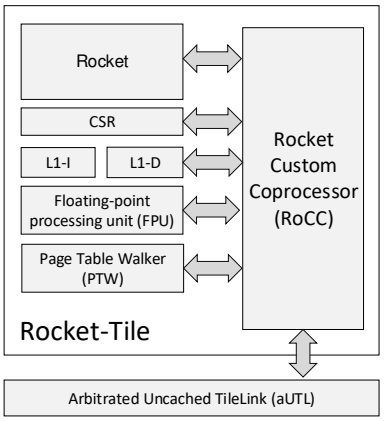


Fig. 4. RoCC Interface connections.

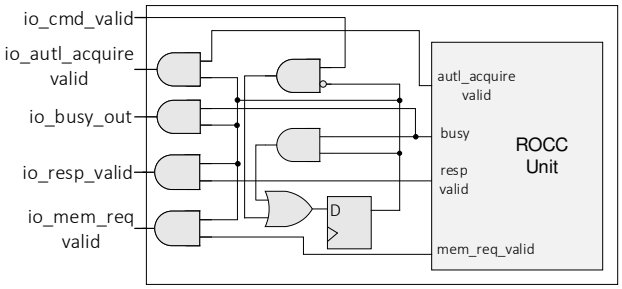


Fig. 5. Decoupling system of the RoCC unit.

IV. Reconfigurable accelerators

The present work implements the 128-bit Advanced Encryption Standard (AES) and the 64-bit Data Encryption Standard (DES) algorithms in the PRR using the RoCC interface, where both algorithms are implemented for data encryption and decryption in different designs. The accelerators use the group of signals of the RoCC interface that connects the coprocessor to the RISC-V core and the L1-Data cache, in order to read data directly from data cache.

Enabling DPR in the system requires to floorplan the PRR in the chip using the Xilinx Vivado tool, specify the position and shape of the PRR, and get the resources available in the region. The chosen PRR has a simple rectangular shape that represents 9.4% of the available LUTs, and the region only includes CLBs resources, without any other type of resources

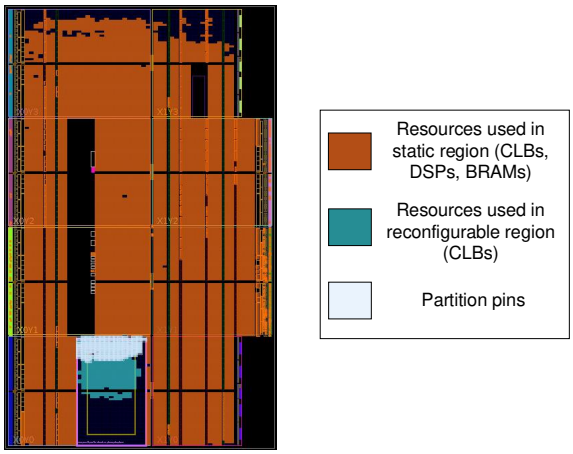


Fig. 6. FPGA floorplanning of the entire system with the AES encryption accelerator in the PRR.

TABLE I
RESOURCE UTILIZATION OF ACCELERATORS IN THE FPGA

Accelerators	Slice LUTs			Slice Registers		
	#	PRR	Total	#	PRR	Total
DES Encryption	1574	26.23%	2.48%	196	1.63%	0.15%
DES Decryption	1765	29.41%	2.78%	198	1.65%	0.15%
AES Encryption	1649	27.48%	2.60%	196	1.63%	0.15%
AES Decryption	1769	29.48%	2.79%	679	5.66%	0.53%

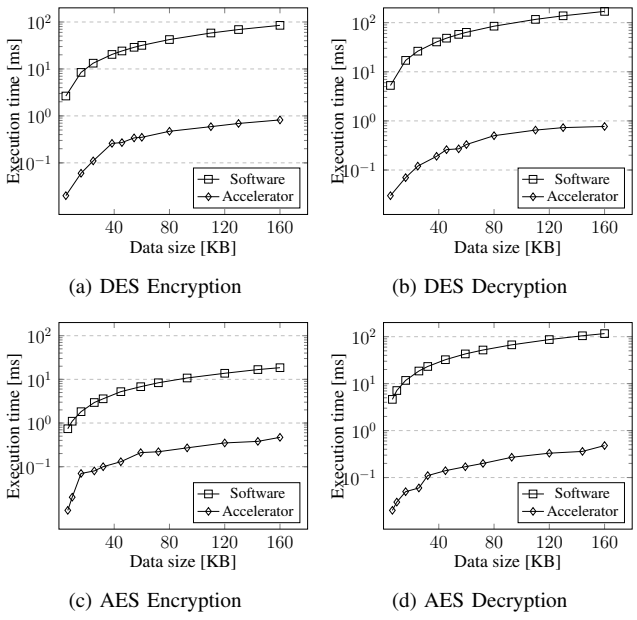
such as BRAMs or DSPs because the selected accelerators are only implementable with CLBs.

Table I shows the resource utilization of each accelerator after the implementation process. The biggest implemented design takes 2.79% of the available slice LUTs in the Xilinx XC7A100T-1CSG324C FPGA and 29.48% of the available slice LUTs in the PRR. Partial bitstreams of each accelerator are generated with Vivado tool 2015.4, using the SelectMAPx32 configuration mode. Fig. 6 depicts the floorplaning of the entire system implemented in the selected Xilinx FPGA with the AES encryption accelerator.

V. EXPERIMENTAL RESULTS

The complete system was tested on the Nexys 4-DDR development board that includes the Xilinx XC7A100T-1CSG324C Artix-7 FPGA and 128 MiB DDR2 RAM. The RISC-V core obtained from the Rocket-Chip was customized in lowRISC SoC to work with 25MHz and included 16 KiB of L1 data cache and 16 KiB of L1 instruction cache. The tests were performed using the Linux kernel version 4.6.2.

Fig. 7 shows the execution time for each encryption/decryption algorithm measured inside the test programs (using the *syscall()* library function), comparing the results obtained from the software executed in the core and the results obtained from the attached accelerators. All the encryption and decryption algorithms execute in less time when using



(a) DES Encryption (b) DES Decryption

(c) AES Encryption (d) AES Decryption

Fig. 7. Performance comparison of the accelerators and software equivalent programs.

TABLE II
ACCELERATORS SPEEDUP

Accelerators	Average speedup
DES Encryption	102.56
DES Decryption	200.27
AES Encryption	41.67
AES Decryption	249.91

the accelerators. Table II includes the calculated average speedups for each algorithm executed using the accelerators in comparison to the equivalent software execution.

Accelerator switching process is manually managed in the Linux user space. The average reconfiguration time that takes the AXI HWICAP to download the partial bitstream contents of each accelerator in the PRR is 151ms.

VI. CONCLUSIONS AND FUTURE WORK

This work implements, to the best of our knowledge, the first physical implementation of a reconfigurable coprocessor attached to a RISC-V soft-core processor, improving the performance of different encryption/decryption algorithms with less FPGA resources, in comparison to a full static design. The design keeps the software flexibility of programs executed on Linux and the software ecosystem available for RISC-V processors. The speedup of the coprocessors and the reconfiguration times enable the acceleration of some programs that use two or more of the presented algorithms, switching the RoCC configuration during execution.

Future works will extend and improve the software and hardware support (BRAMs and DSPs) to manage the coprocessor switching process, in order to reduce the reconfiguration penalty time for programs that require more than one coprocessor, using techniques such as configuration prefetching and reuse [8].

REFERENCES

[1] Y. Lu, L. Liu, J. Zhu, S. Yin, and S. Wei, "Architecture, challenges and applications of dynamic reconfigurable computing," J. Semicond., vol. 41, no. 2, p. 021401, 2020.

[2] T. Sato, H. Watanabe, and K. Shiba, "Implementation of dynamically reconfigurable processor DAPDNA-2," in 2005 IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test, 2005. (VLSI-TSA-DAT), 2005.

[3] T. Fujii et al., "New generation dynamically reconfigurable processor technology for accelerating embedded AI applications," in 2018 IEEE Symposium on VLSI Circuits, 2018.

[4] H. Amano, "A survey on dynamically reconfigurable processors," IEICE trans. commun., vol. E89-B, no. 12, pp. 3179–3187, 2006.

[5] L. Yan, B. Wu, Y. Wen, S. Zhang, and T. Chen, "A reconfigurable processor architecture combining multi-core and reconfigurable processing unit," in 2010 10th IEEE International Conference on Computer and Information Technology, 2010.

[6] D. Rossi, F. Campi, S. Spolzino, S. Pucillo, and R. Guerrieri, "A heterogeneous digital signal processor for dynamically reconfigurable computing," IEEE J. Solid-State Circuits, vol. 45, no. 8, pp. 1615–1626, 2010.

[7] J. Yin, Z. Xu, X. Fang, and X. Zhou, "The design of reconfigurable instruction set processor based on ARM architecture," in Communications in Computer and Information Science, Singapore: Springer Singapore, 2018, pp. 66–78.

[8] A. Morales-Villanueva, R. Kumar, and A. Gordon-Ross, "Configuration prefetching and reuse for preemptive hardware multitasking on partially reconfigurable FPGAs," Proc. 2016 Design, Automation and Test in Europe Conference & Exhibition (DATE), pp. 1505-1508, 2016.