2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) | 978-1-6654-3577-2/20/$31.00 ©2021 IEEE | DOI: 10.1109/IPDPSW52791.2021.00033

RV-CAP: Enabling Dynamic Partial Reconfiguration for FPGA-Based RISC-V System-on-Chip

Najdet Charaf, Ahmed Kamaleldin, Martin Thümmler, and Diana Göhringer
Chair of Adaptive Dynamic Systems, Technische Universität Dresden, Germany
E-mail: {najdet.charaf, ahmed.kamal, martin.thuemmler, diana.goehringer}@tu-dresden.de

***Abstract*—Partial reconfiguration is remaining the core technique to build adaptive systems-on-chip (SoCs) for modern FPGA architectures. Current support for reconfiguration management targets a few adaptive SoC architectures. Therefore, the adoption of a new instruction set architecture like RISC-V to manage the reconfiguration process requires the development of suitable reconfiguration management along with the needed hardware and software modules. In this work, we present a solution to enable partial reconfiguration for FPGA-based RISC-V SoC. The developed RV-CAP controller consists of hardware modules and a set of improved software drivers to ease the reconfiguration process from the RISC-V processor. Our dynamic partial reconfiguration controller achieves a high reconfiguration throughput of 398.1 MB/s with low resource utilization overhead. The proposed reconfiguration management is implemented and evaluated using Xilinx Kintex-7 FPGA with the ability to be portable for any other Xilinx FPGAs supporting partial reconfiguration. Our goal is to enable open-source soft-core RISC-V processors to manage and interact with reconfigurable hardware accelerators on FPGA-based SoCs.**

***Index Terms*—RISC-V, Field programmable gate arrays (FPGAs), Dynamic partial reconfiguration.**

I. Introduction

In the Internet-of-Things (IoT) era, edge and near-sensor computing imposing new impulses and motivation for novel computational concepts based on openness and flexibility as well as low latency and high performance [1]. Therefore, the focus is shifting to the development of agile hardware systems [2] that provide the required computing performance along with modularity, adaptability, and openness. Hence, field-programmable gate arrays (FPGAs) by their adaptive characteristics besides the modularity and open-source attribute of RISC-V (an instruction set architecture (ISA)) [3] draw the course for the new movement of realizing agile hardware systems.

Currently, several new platforms are available that combine a RISC-V processor (either soft- or hard-core) with an FPGA fabric on the same chip. From the industrial environment comes the Microchip Polarfire system-on-chip (SoC) [4], which tightly couples a quad-core RISC-V-based application-class processor supporting the Linux operating system with a low-power FPGA fabric on a single chip for IoT and industrial applications. The new platform can enable offloading of application tasks from the programmable software to the FPGA in an a priori defined hardware architecture during design time. Other interesting works in this area are Flexbex [5] and Arnold [6], they provide the extension of a RISC-V-based microcontroller unit (MCU) with an embedded FPGA (eFPGA) for embedded-class applications for ultra-low-power devices suitable for near-sensor computing (e.g., [7]). However, the above mentioned RISC-V based FPGA systems-on-chip (SoCs) lack the capability of dynamic partial reconfiguration (DPR) to exchange parts of the running applications on the FPGA fabric without halting the whole offloaded applications. Their application domains are limited to low-power applications and are not suitable for high workload dynamic applications due to the limited size of the eFPGA and high fabrication cost.

High-end FPGAs, such as Xilinx or Intel devices, are considered suitable platforms for high workload dynamic applications. Moreover, they provide the capability for DPR, which leads to the fragmentation of the FPGA fabric into multiple isolated partition regions large enough to host different hardware modules that can be exchanged during runtime without interfering or halting other active reconfigurable partitions as well as hardware modules on the static region [8]. They also have fast prototyping implementation flows for a wide range of applications and hosts different types of soft-core processors as CPUs along with hardware-accelerated modules.

In this work, we propose a novel DPR controller called RV-CAP for FPGA-based RISC-V SoCs. RV-CAP extends FPGA-based RISC-V SoCs to support DPR for their tightly coupled reconfigurable accelerators for more flexibility and dynamic workload. It consists of a hardware-implemented part for the DPR controller on the FPGA fabric and a set of software drivers and application programming interfaces (APIs) to manage the DPR process via a programmable software environment from the RISC-V processor. The controller is implemented and tested on a Xilinx Kintex-7 FPGA and can be ported to any Xilinx FPGA devices. The proposed solution features high reconfiguration throughput of 398.1 MB/s with low hardware resource cost on the FPGA. The foremost contributions of this work are summarized as follows:

- A high-throughput DPR controller for FPGA-based RISC-V SoCs that enables control of the reconfiguration process from the RISC-V processor.
- A set of APIs that provide the ability to manage the reconfiguration process from the software environment.

The rest of the paper is organized as follows: Section II covers the background and related work. Section III gives a detailed overview of the proposed DPR controller for FPGA-based RISC-V SoCs.The evaluation and experimental results are presented in Section IV. Finally, Section V summarizes this work and gives an outlook on future work.

978-1-6654-3577-2/21/$31.00 ©2021 IEEE
DOI 10.1109/IPDPSW52791.2021.00033

II. Background and Related Work

Current adaptive hardware systems proposed by academic or industry are usually based on interactions between low-level software and reconfigurable hardware modules in a tightly integrated co-design between software kernels and hardware-accelerated functions [9]. Adaptive hardware systems aim to modify computational resources in terms of quantities, types, and functionality so that different applications can be exchanged at runtime based on the application domain of the main system (e.g., cyber-physical systems, software-defined radio). In the last decade, there have been many attempts to realize adaptive hardware platforms capable of managing and controlling reconfigurable hardware modules within a software-programmable environment [10]. In this context, most contributions target FPGAs that, besides an on-chip CPU (either hard- or soft-core processor), have the capability of DPR technique on their programmable logic (PL) for a tight interaction between programmable software and reconfigurable hardware modules. DPR is realized by defining one or more reconfigurable partitions (RP) that host various reconfigurable hardware modules (RM) loaded onto the FPGA in the form of partial bitstreams (PB) that can be swapped at runtime. In addition, the static region of the system mainly hosts the soft-core processor, the reconfiguration controller, and other system peripherals used for external data acquisition.

The DPR management from the software side imposes several challenges related to the abstraction of the reconfiguration process from the CPU side, in addition to the reconfiguration efficiency in terms of reconfiguration time especially in cases of real-time applications [11]. Consequently, DPR management requires a custom hardware implementation for a DPR controller between the CPU and the dedicated programmable region. Besides, the management of the hardware-accelerated modules on the PL from the software running on the CPU. Therefore, a change in adaptive systems requirements (e.g. hard real-time scenarios), or adoption of a new instruction set (e.g. RISC-V) requires the development of new software drivers or overlays along with updates to the hardware implementation of the DPR controller for compatibility purposes.

In the last years, several works are proposed by literature for DPR controllers including custom DPR controllers or software abstraction layers for DPR managing. In [12], the authors proposed a high-speed DPR controller for Xilinx FPGA devices. The controller's system architecture is designed for loading partial bitstreams from an off-chip memory to the FPGA configuration memory at a data rate close to the physical data rate of Xilinx's internal access configuration port (ICAP) primitive. The proposed controller uses direct memory access (DMA) component for data transfer, freeing the adaptive SoC's CPU to execute other tasks. Similarly, the ZyCAP manager [13] features a high throughput DPR for Xilinx Zynq FPGAs, customized for ARM processors hosted on the PS side of the Zynq device. ZyCAP provides a set of high-level driver interfaces to manage the reconfiguration process from the PS side. However, ZyCAP is exclusively compatible with Xilinx Zynq FPGAs and its portability to other devices requires hardware and software modifications. Meanwhile, the proposed DPR manager by Carlo et al. [14] provides the portability to several Xilinx FPGA devices with an ICAP interface. Furthermore, it supports a safe DPR for real-time and mission-critical adaptive applications. As a consequence, a new set of features are required for the implementation of the DPR controller with customization on the operation modes. Therefore, the controller features different modes of operations depending on the application requirements and invokes a cyclic redundancy check and error correction on the loaded partial bitstream before transferring it to the configuration memory. Besides, the DPR process is software managed by a LEON3 soft-core processor.

In the same contest, for hard real-time adaptive applications, the RT-ICAP controller [15] is introduced as a time-predictable DPR controller aiming to reduce the worst-case execution time (WCET) to perform the configuration in a determined amount of time. The controller features the capability of partial bitstream compression before transferring it to the FPGA configuration memory to reduce its size and therefore reduce the reconfiguration time. However, extra on-chip memory is reserved on the FPGA fabric to store the compressed partial bitstream. The DPR process is software managed through a custom real-time soft-core processor. It is used for encryption and security applications. Therefore, AC_ICAP [16] is introduced as a light DPR controller that can operate autonomously or with a light soft-core processor (e.g., Microblaze). The controller is customized for partial reconfiguration of LUT resources only featuring low resource overhead and high reconfiguration throughput. However, it lacks the portability for the new generation of FPGA architectures. Accordingly, from the above-mentioned work, designing DPR controllers depends on one side the timing characteristics and reconfiguration sensitivity of the target applications as well as the architecture of the hardware/software co-design of the target adaptive platform.

Several DPR controllers are proposed to support the management of the reconfiguration process from operating systems running on application class processors (i.e., ARM processors) to enhance software productivity. Hence, suitable software drivers and interfaces between the CPU and the FPGA are required. Alkadi et al. [17] proposed a set of software drivers running on Linux for Xilinx Zynq devices to access the processor configuration access port (PCAP). Another novel approach is called pynqpartial [18]. This is introduced as a software-only implementation for managing DPR from the PynQ platform. Thus, a set of python packages are implemented on the ARM processor on the PS side using the existing PCAP interface to access the configuration memory. However, the pynqpartial shows a poor reconfiguration throughput. Meanwhile, the authors of [19] improved the reconfiguration throughput while maintaining a high level of abstraction for DPR management from a petalinux operating system targeting a Xilinx Zynq Ultrascale+ FPGA.

Accordingly, and to the best of our knowledge, our proposed DPR controller is the first developed solution for managing DPR for FPGA-based RISC-V SoCs. RV-CAP provides the hardware

requirements for managing DPR on the FPGA fabric, along with the necessary APIs to manage the reconfiguration process from the RISC-V processor. The proposed solution contributes to the current motivation for the use of open-source SoCs in adaptive hardware systems.

III. Reconfiguration Management in Detail

We have designed suitable reconfiguration management for FPGA-based RISC-V SoCs and a hardware/software solution for the reconfiguration process. RV-CAP targets high-end FPGA devices, in our case Xilinx FPGA devices, which have the DPR capability. The RV-CAP controller is responsible for low-level control of the DPR process over the FPGA fabric. In addition, we developed a set of APIs for managing the reconfiguration process from the RISC-V processor within a programmable software environment. The selected FPGA-based RISC-V SoC is equipped with a 64-bit RISC-V application class processor that is compatible with the RV64GC ISA. Therefore, the selected SoC is capable of running heavy workload applications with multiple coupled hardware accelerators for various hardware-accelerated tasks.

A. FPGA-Based RISC-V SoC

We use an open-source RISC-V SoC architecture built around a 64-bit, single issue, in-order RISC-V CPU CVA6 (formerly Ariane) [20]. Ariane core has support for hardware multiply/divide, atomic memory operations, and single and double floating-point operations. Therefore, Ariane complies with the RISC-V general-purpose extension (G) and supports variable compressed instructions length (C). Besides, a hardware implementation of a memory management unit to support an operating system [21]. Thus, Ariane has been chosen for its characteristic as an application class processor suitable for high workload SoC implementations.

Figure 1 shows a schematic overview of the target FPGA-based SoC. The SoC uses a bus-based architecture in which the Ariane processor, SoC peripherals, and hardware accelerators communicate via a 64-bit AXI-4 crossbar. The processor core is the master unit on the bus, while the remaining peripherals

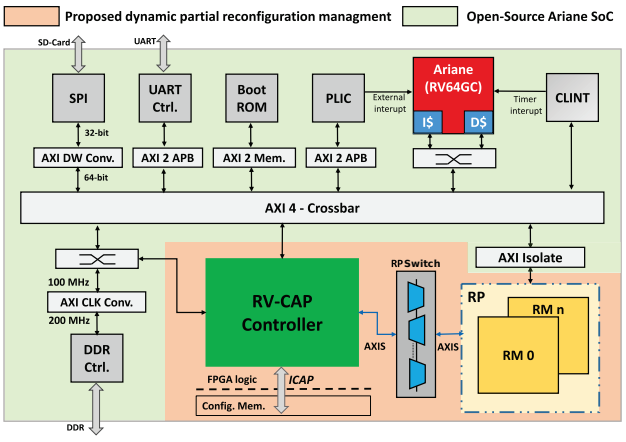

Fig. 1: A schematic overview of the target FPGA-based RISC-V SoC.

are slave memory-mapped components within the processor's address space ranges. In addition, on-chip boot memory is used to store application instructions for execution. The AXI-4 bus allows several hardware accelerators to be tightly coupled to the processor. One or more RPs can be created to host different RMs for hardware accelerators. The SoC uses a set of open-source, non-coherent on-chip communication components for data width converters, AXI isolation interfaces, and crossbars [22]. Therefore, the integration of the proposed RV-CAP controller requires the insertion of an additional crossbar component and AXI data width converter between the main AXI-4 bus and the controller for read/write (R/W) data directly from/to DDR memory. Furthermore, AXI isolator components are inserted between the RPs and the main AXI-4 bus for PR decoupling during the reconfiguration process to isolate the RPs from the overall SoC.

We have developed a set of software drivers to access the SoC I/O peripherals to load the partial bitstreams from an external SD card into the SoC's DDR memory. We also developed a set of utility modules to communicate with memory-mapped peripherals to read and write data across the processor's address space. To read and write logical blocks from the SD card, the serial-parallel interface (SPI) peripheral is used to communicate between the AXI-4 bus and the external SD card. A set of file I/O software functions based on the minimalist implementation of the file allocation table (FAT32) have been developed to support file reading, writing, and overwriting. A set of software timer modules is created to access the local interrupt controller (CLINT) of the SoC core and use it as a real-time counter to measure the reconfiguration time of different bitstream sizes.

For SoC programming, the RISC-V GNU compiler toolchain [23] is used to compile the application source codes (i.e., *C* codes) for the RV64GC architecture. Finally, the generated binary files are stored in the SoC boot memory on the FPGA.

B. RV-CAP Controller

The RV-CAP controller for the proposed reconfiguration management is shown in Figure 2. The approach presented here provides a high data throughput rate to the FPGA configuration memory via the ICAP primitive. Further, auxiliary functions related to RP decoupling and R/W data stream from/to RMs are supported. The controller has four interfaces to the RISC-V SoC. Two 64-bit AXI interfaces to the main AXI-4 crossbar to control the DMA controller and R/W control signals to the RP controlling interface, as shown in Fig. 2. Two other interfaces for transferring data from the external DDR either the partial bitstream to the ICAP primitive or the application data to the RM. In addition, the DMA controller interrupts are directly connected to the processor-level interrupt controller (PLIC) to support non-blocking mode during data transfer and free up the processor for other tasks. Our proposed controller supports two operation modes, reconfiguration mode, and acceleration mode and operates with a single clock source in a fully synchronized design. The clock frequency is set to 100 MHz due to the ICAP maximum frequency on FPGAs of 7 series.

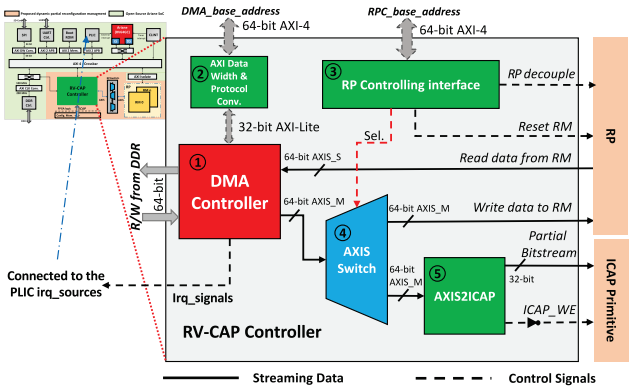

Fig. 2: Overview of RV-CAP controller system architecture

The RV-CAP internal architecture consists of ① a Xilinx DMA controller connected to the SoC DDR controller through an additional crossbar, as shown in Fig. 2. The DMA controller is a master component to the DDR controller. Besides, the DMA is configured to transfer a 64-bit data word from the SoC DDR memory. ② AXI data width and protocol converters are used to convert between the 64-bit AXI-4 standard of the main bus and the AXI-lite control interface of the Xilinx DMA. ③ An RP control interface is implemented to provide R/W control signals to the RMs including RP coupling/decoupling. ④ An AXI stream switch is inserted between the DMA and ICAP output ports to select whether the RV-CAP controller operates in reconfiguration mode or acceleration mode by connecting the DMA data stream interfaces to the RM or ICAP primitive.⑤ An AXI stream to ICAP converter is implemented to connect the AXI stream data to the ICAP data port. The AXIS2ICAP block shown in Fig.2 is responsible for converting a 64-bit data word fetched from the DDR memory into two 32-bit data words, which are written in order to the ICAP data port. Besides, the valid stream signal is inverted and connected to the ICAP data port. The R/W select input port is permanently set to zero.

The reconfiguration process takes place in three steps to successfully load a new partial bitstream with a new RM into the RP. All reconfiguration steps are managed by the RISC-V processor through a set of API functions. While loading a partial bitstream into the FPGA configuration memory, the DMA non-blocking mode is selected. As shown in listing 1, the first step initializes the RM by reading the *pbit_size* of the partial bitstream file stored on the external SD card and loading it to a defined destination address in the DDR memory. The first step is performed by the FAT32 I/O file system software modules. The second step is to decouple the RP interfaces by sending a decoupling signal to the RP control interface, followed by a *select_ICAP* signal to select the RV-CAP reconfiguration mode. Finally, the *reconfigure_RP* step is executed by starting the DMA read channel to read from the DDR, followed by selecting the interrupt configuration mode of the DMA. After the DMA completes transferring the entire bitstream payload, an interrupt signal is sent to the PLIC to indicate completion of the reconfiguration process, and the RM module is ready to communicate with the RISC-V processor.

LISTING 1: The RV-CAP reconfiguration process

```
1  // initialize the start address and load the
2  // partial bitstream from the SD-card to the DDR
3  // destination address
4  init_RModules(*reconfig_module, RM_number,
                 *pbit_fat_partition);
5  void init_reconfig_process() {
6    // decouple the RP
7    decouple_accel(1);
8    // configure the AXIS-Switch to forward the write
9    // stream data to ICAP primitive
10   select_ICAP(1);
11   // load the requested RM into the RP
12   reconfigure_RP(reconfig_module->start_address,
                    reconfig_module->pbit_size, mode);
13   decouple_accel(0);
14 }
15 // start the reconfiguration process
16 // set the DMA mode: non-blocking mode
17 void reconfigure_RP (*data, pbit_size, mode){
18   // starting DMA by set the CR register to 1
19   dma_start();
20   // set the DMA non-blocking mode through the
21   // irq. bit of the CR register
22   dma_config(mode);
23   // DMA start reading from the DDR by setting the
24   // source address of the partial bitstream on the
25   // DMA_SA register and the partial bitstream
26   // length on the DMA_Length register
27   dma_write_stream(*data, pbit_size);
28 }
```

C. Supporting DPR Vendor Controller

Normally, a master device (e.g. MicroBlaze) is used to transmit the partial bitstream from a storage device (e.g. DDR) to the ICAP. In this work we provide an alternative design which allows a RISV-V processor to take over the task of the master device to read and write the FPGA configuration memory through the Internal Configuration Access Port (ICAP). Xilinx offers an IP core called AXI_HWICAP. However, some modifications had to be made in order to integrate the IP core into the Ariane SoC. On the hardware side we add a data width converter (from 64-bit to 32-bit) as well as a protocol converter (from AXI4 to AXI4-Lite) to match the IP core AXI4-Lite slave specification. Furthermore, we re-sized the internal write FIFO of the HWICAP module to 1024 to improve the time transfer.

However, to allow the HWICAP to be controlled by RISC-V, we have modified the driver to enable partial reconfiguration through the ICAP. A section of the modified driver is shown in listing 2, which is an API of the reconfiguration procedure. The *reconfig_module* parameter refers to a unique input containing the bitstream name, the functionality of the RM, the start address corresponding to the start address where the bitstream is stored in the DDR, and the bitstream size. The reconfiguration procedure starts with two functions, the first decouples the reconfigurable partition from the static part and the second initializes the ICAP with the desired values and disables the global interrupt signal. The function *reconfigure_RP* is executed separately for each reconfigurable module. It loads the data of the partial bitstream from the DDR and sends them in 4-Byte words to the HWICAP. The filling and flushing of the internal write FIFO are repeated

LISTING 2: Reconfiguration process with AXI-HWICAP

```
// initialize the start address and load the
// partial bitstream from the SD-card to the DDR
// destination address
init_RModules(*reconfig_module, RM_number,
              *pbit_fat_partition);
void init_reconfig_process() {
  // decouple the RP
  decouple_accel(1);
  // init the HWICAP & disable the global interrupt
  init_icap();
  // load the requested RM into the RP
  reconfigure_RP(reconfig_module->start_address,
                 reconfig_module->pbit_size);
  decouple_accel(0);
}
//start the reconfiguration process
void reconfigure_RP (*data, pbit_size){
  // start transfering the bitstream
  while (pbit_size) {
    read_fifo_vac(); // read the write fifo vacancy
    do {
      write_into_fifo(ICAP_WF,*data++);
    } while (fifo_is_not_full)
    // transfer the data in to the ICAPE primitive
    write_to_icap();
    // wait until HWICAP is done with configuration
    icap_done();
  }
}
```

until the complete partial bitstream has been transferred. In the end, a terminal message informs that the reconfiguration was successful and the reconfiguration procedure ends by coupling the reconfiguration partition to the static part.

IV. Experimental Results and Evaluation

Our work aims for enabling fast partial reconfiguration on FPGA-based RISC-V systems. This section summarizes our results on the reconfiguration throughput and resource utilization of the different deployment options of the RV-CAP framework. First, we evaluate the reconfiguration speed and resource utilization of the RV-CAP and AXI HWICAP controller with Ariane SoC. We then compare them with state-of-the-art implementations.

A. Resource Evaluation

The RV-CAP is developed using Xilinx Vivado Design Suite HLx 2019.1. The Genesys2 board (Kintex7 XC7K325T) is used for our implementation. In the SoC design, all components run at 100 MHz. The driver APIs are implemented using C language and the RISC-V GNU compiler toolchain is used to generate executable code for the Ariane core. The RP size is defined to be 3200 LUTs, 6400 FFs, 20 DSP blocks, and 30 BRAMs, and is hosting the accelerators. The partial bitstream size is 650892 bytes.

Table I presents the details of the RV-CAP controller comparing to the AXI HWICAP controller. Both controllers essentially consist of two modules. One that contains the main controller, a soft DMA controller for RV-CAP, and a Xilinx AXI_HWICAP controller for HWICAP. The second module contains all AXI components (e.g. data width converter) and an additional PR

TABLE I: Resources utilization of the RV-CAP controller compared to AXI_HWICAP on Xilinx Kintex-7 FPGA.

DPR Controller	Modules	Resource Utilization			Throughput (MB/s)
		LUTs	FFs	BRAMs	
RV-CAP	RP cntrl. + AXI modules	420	909	0	398.1
	DMA Cntrl.	1897	3044	6	
AXI_HWICAP with RC64GC	AXI modules	909	964	0	8.23
	AXI_HWICAP	468	1236	2	

controller within RV-CAP. The maximum AXI burst size of the DMA controller is set to 16 and, as already mentioned in the implementation section, all components of the SoC communicate over a bus width of 64 bits except for the HWICAP and the control signals, which have a data width of 32 bits. As we can see from the table, the AXI_HWICAP occupies fewer resources than the RV-CAP, but in return, we get only a throughput of about 2% of the theoretical ICAP maximum throughput.

B. Reconfiguration Time

The reconfiguration time is measured by the CLINT component with a clock timer frequency of 5 MHz. The measurement starts with the beginning of the data transfer of the partial bitstream to ICAP and ends until the bitstream is completely transferred to the configuration memory. In the case of using the AXI-HWICAP controller, the reconfiguration overhead is measured as the time required from decoupling the RP till it is coupled again. The reconfiguration time T_r, including the control overhead, is 156.45 ms. This results in a reconfiguration throughput of 4.16 MB/s.

Software access is improved by unrolling the loop when writing to the HWICAP FIFO keyhole register. This is a valuable improvement since the Ariane core is not allowed to start speculative memory access to the non-cacheable memory address area of the HWICAP. This means that the Ariane pipeline must block after each loop iteration until the conditional jump is executed completely. The maximum reconfiguration throughput achieved is 8.23 MB/s using a 16-unrolled data transfer loop. The expected further increase in throughput for a higher loop unroll factor is less than 5%. In the case of using the RV-

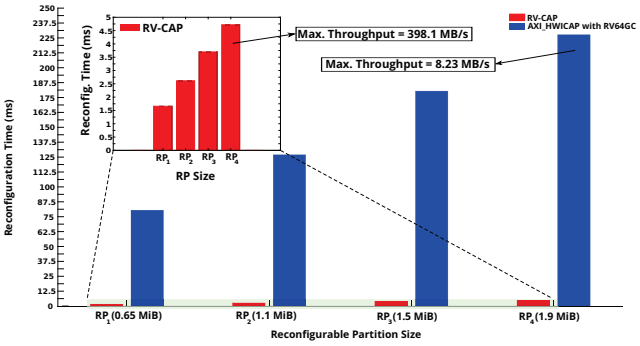

Fig. 3: Reconfiguration time with respect to different RP sizes.

TABLE II: Comparison of resource utilization and reconfiguration throughput of various state-of-the-art DPR controllers for different adaptive SoC architectures.

DPR Controller	SoC Processor [a]	Custom software Drivers	DPR Controller Resources Utilization			Throughput (MB/s)	Frequency (MHz)
			LUTs	FFs	BRAMs		
Vipin et. al. [12]	MicroBlaze	-	586	672	8	399.8	100
ZyCAP [13]	ARM	✓	620	806	0	382	100
Anderson et. al. [14]	LEON3	✓	588	278	1	395.4	100
AC_ICAP [16]	MicroBlaze	-	1286	1193	22	380.47	100
RT-ICAP [15]	Patmos	✓	289	105	0	382.2	100
PCAP [24]	ARM	-	0	0	0	128	100
Xilinx PRC [25]	ARM	-	1171	1203	0	396.5	100
Xilinx AXI_HWICAP [26]	ARM	-	538	688	0	14.3	100
Xilinx AXI_HWICAP (with RISC-V)	RV64GC	✓	1377	2200	2	8.23	100
RV-CAP	RV64GC	✓	2317	3953	6	398.1	100

[a] The used processor for managing DPR on the target SoC (either via soft- or hard-core processor)

CAP controller, the timing measurement was performed in a similar way as described in the previous case. However, there are two main differences: first, an additional timing overhead called decision time T_d occurs, which is the time for choosing between ICAP and accelerator, and second, the interrupt mode is used. The timing results for RV-CAP with the interrupt mode are T_d = 18 µs, T_r = 1651 µs. We measured with various sizes of partial bitstreams, the reconfiguration speed is calculated from the recorded time. The maximum reconfiguration throughput achieved is 398.1 MB/s. The results are shown in Figure 3.

C. Comparison with State-of-the-Art Implementations

In this section, the RV-CAP controller is compared with several state-of-the-art DPR controllers that have been proposed for various adaptive SoCs. The purpose of this comparison is to evaluate the specifications and performance of the proposed solution on a fair basis. As shown in Table II, several implementations are proposed for different adaptive SoCs with different processor architectures [12, 13] and specific application requirements [14, 15]. Several DPR controllers are available from vendors [23, 24, 22], but they are not adapted for integration with RISC-V processors. The comparison is also based on the hardware resource utilization of the DPR controller and the support of custom software drivers for managing reconfiguration by the SoC processor. The maximum achievable reconfiguration throughput is also compared between the proposed work and prior implementations.

As shown in Table II, the proposed RV-CAP controller achieves a reconfiguration throughput of 398.1 MB/s. It is considered that the maximum theoretical ICAP throughput for the R/W bitstream to the ICAP primitive is 400 MB/s [13]. Compared to [12], the APIs we developed lead to an execution overhead that reduces the maximum achievable throughput by 1.9 MB/s. However, the RV-CAP achieves better reconfiguration throughput compared to the proposed implementation in [13] for ARM-based adaptive SoC. Moreover, the RV-CAP reconfiguration throughput outperforms the Xilinx ICAP controller. On the other hand, the hardware resource utilization is higher compared to [12, 13, 15] because the DMA implementation used consumes large internal buffers. The DPR controller resource utilization figures of [12, 13, 15] do not include the sizes of the buffers and DMAs used within the DPR controller. Finally, few modern approaches [13, 14, 15] support custom software drivers for DPR management to control the reconfiguration process within a programmable software environment.

D. Case Study: Adaptive Image Processing Application

In this section, three basic image processing filters are used as reconfigurable hardware modules to evaluate the proposed RV-CAP controller. The three filters are Sobel, Median, and Gaussian filters processing an image size of 512x512 pixels a 8-bit to represent 256 gray values. The filters are developed using Xilinx Vivado high-level synthesis (HLS) tool with 64-bit AXI-stream interfaces. The three filters are generated and synthesized separately as three RMs that are hosted by a single RP. The AXI-stream interface connects between the image filter and the AXI-stream R/W interface of the DMA inside the RV-CAP controller (as shown previously in Fig. 2). The RMs are operating at a clock frequency of 100 MHz fully synchronized with the complete system. The image input is stored in the DDR memory to be

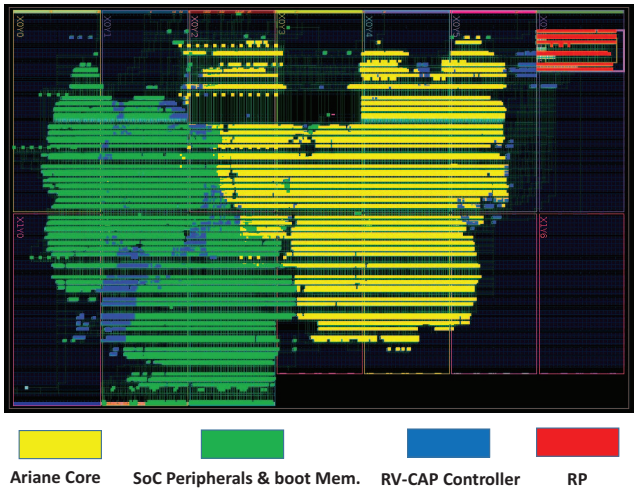

Fig. 4: An overview of the full SoC floorplan on a Kintex-7 FPGA.

TABLE III: Resources utilization of the full SoC with one RP on Xilinx Kintex-7 FPGA.

SoC Components		Resources Utilization			
		LUTs	FFs	BRAMs	DSPs
Full SoC		74393	64059	92	47
Ariane Core		39940	22500	36	27
Peripherals & Boot Mem.		28832	31404	20	0
RV-CAP controller		2421	3755	6	0
RP		3200	6400	30	20
RMs[a]	Gaussian	901 (28.15%)	773 (12.07%)	4 (13.33%)	0 (0.0%)
	Median	2325 (72.65%)	998 (15.59%)	2 (6.66%)	0 (0.0%)
	Sobel	1830 (57.18%)	3224 (50.37%)	2 (6.66%)	16 (0.8%)

[a] Percentage utilization of the RP.

TABLE IV: The image processing accelerators execution time at frequency = 100 MHz.

Accelerator	Decision Time (T_d) (µs)	Reconfiguration Time (T_r) (µs)	Compute Time (T_c) (µs)	Total (T_{ex}) (µs)
Gaussian	18	1651	606	2275
Median	18	1651	598	2267
Sobel	18	1651	588	2257

loaded by the RV-CAP controller (in accelerator mode) after the reconfiguration process. Fig. 4 shows the complete floorplan of the full RISC-V SoC with one RP and the full SoC resources utilization are listed in Table III.

It is shown that the RV-CAP controller consumes 3.25% of the total SoC resources in terms of LUT and FFs. Table IV shows the total execution time for the three image filter RMs. As the total execution time is computed as follow: $T_{ex} = T_d + T_r + T_c$. The decision time (T_d) is the measured time for the software to select the requested RM and start fetching the partial bitstream from the DDR. Reconfiguration time (T_r) is the RV-CAP controller time to transfer the requested bitstream from the DDR to the configuration memory. Compute time is the accelerator computation time to apply the filter on an image and write back the output to the memory. We expect in the case of a high-workload application that the reconfiguration time is negligible to the computation time.

V. Conclusion

In this work, a novel partial reconfiguration controller with high reconfiguration throughput is proposed to enable DPR on FPGA-based RISC-V SoCs. The RV-CAP supports the management of DPR within a programmable software environment through a set of developed software drivers running on the RISC-V processor. The high-speed DPR controller achieves a reconfiguration throughput of 398.1 MB/s. In addition, the RV-CAP controller supports the management of the RPs and the hosted RMs by providing controlled stream interfaces between the DDR memory and the corresponding RMs. Finally, the proposed implementation can be ported to all Xilinx FPGA devices that support DPR.

VI. Acknowledgment

This work is partially funded by the European Social Fund in Germany ESF and co-financed by tax funds based on the budget approved by the members of the Saxon State Parlament as part of the ReLearning project under grant agreement number 100382146 and partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 287022738 – TRR 196.

References

[1] R. Wittig, et al., "Modem Design in the Era of 5G and Beyond: The Need for a Formal Approach," in Proc. 27th International Conference on Telecommunications (ICT), Bali, Indonesia, 2020, pp. 1-5.

[2] A. Amid et al., "Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs," IEEE Micro, vol. 40, no. 4, pp. 10-21, July. 2020.

[3] A. Waterman et al., "The RISC-V Instruction Set Manual, Volume I: User-Level ISA, " EECS Department, University of California, Berkeley, May 2017. [Online]. Available: /https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

[4] Microchip, PolarFire SoC FPGAs (Architecture, Applications, Scurity Features, Design Environment, Design Hardware), 2020. [Online]. Available: /https://www.microsemi.com/document-portal/doc_download/1244582-polarfire-soc-brochure

[5] N. Dao, et al., "FlexBex: A RISC-V with a Reconfigurable Instruction Extension," in International Conference on Field-Programmable Technology (FPT), USA, 2020, pp.1-6.

[6] P. Schiavone, et al.,"Arnold: An eFPGA-Augmented RISC-V SoC for Flexible and Low-Power IoT End Nodes" in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 29, no. 04, pp. 677-690, 2021.

[7] D. Palossi, et al., "A 64-mW DNN-Based Visual Navigation Engine for Autonomous Nano-Drones," in IEEE Internet of Things Journal, vol. 6, no. 5, pp. 8357-8371, 2019.

[8] A. Vaishnav, et al., "FOS: A Modular FPGA Operating System for Dynamic Workloads," in ACM Trans. Reconfigurable Technol. Syst. 13, 4, Article 20, 2020, pp. 1-28.

[9] S. A. Fahmy, "Design Abstraction for Autonomous Adaptive Hardware Systems on FPGAs," in NASA/ESA Conference on Adaptive Hardware and Systems (AHS), Edinburgh, 2018, pp. 142-147.

[10] K. Vipin and S. A. Fahmy, "FPGA Dynamic and Partial Reconfiguration: A Survey of Architectures, Methods, and Applications," in ACM Computing Survey 51, 4, Article 72, 2018, pp. 1-39.

[11] L. Pezzarossa, et al. "Can real-time systems benefit from dynamic partial reconfiguration?," in IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC), Linkoping, 2017, pp. 1-6.

[12] K. Vipin and S. A. Fahmy, "A high speed open source controller for FPGA Partial Reconfiguration," in International Conference on Field-Programmable Technology, Seoul, 2012, pp. 61-66

[13] K. Vipin and S. A. Fahmy, "ZyCAP: Efficient Partial Reconfiguration Management on the Xilinx Zynq," in IEEE Embedded Systems Letters, vol. 6, no. 3, pp. 41-44, Sept. 2014

[14] S. Di Carlo, et. al. "A portable open-source controller for safe Dynamic Partial Reconfiguration on Xilinx FPGAs," in 25th International Conference on Field Programmable Logic and Applications (FPL), London, 2015, pp. 1-4

[15] L. A. Cardona and C. Ferrer, "AC_ICAP: a flexible high speed ICAP controller. Int. J. Reconfig. Comput., Article 9, 2015, pp. 1-15.

[16] L. Pezzarossa, et al., "A Controller for Dynamic Partial Reconfiguration in FPGA-Based Real-Time Systems," in IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC), Toronto, ON, 2017, pp. 92-100.

[17] M. A. Kadi, P. Rudolph, D. Goehringer and M. Huebner, "Dynamic and partial reconfiguration of Zynq 7000 under Linux," in International Conference on Reconfigurable Computing and FPGAs (ReConFig), Cancun, 2013, pp. 1-5.

[18] B. Janßen, et al., ”A dynamic partial reconfigurable overlay concept for PYNQ,” in 27th International Conference on Field Programmable Logic and Applications (FPL), Ghent, 2017, pp. 1-4.
[19] A. R. Bucknall, et al., “Build Automation and Runtime Abstraction for Partial Reconfiguration on Xilinx Zynq Ultrascale+,” in International Conference on Field-Programmable Technology (FPT), USA, 2020, pp.1-6.
[20] CVA6 RISC-V CPU [Online]. Available: https://github.com/openhwgroup/cva6
[21] 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,” in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 27, no. 11, pp. 2629-2640.
[22] A. Hurth, et al., “An Open-Source Platform for High-Performance Non-Coherent On-Chip Communication,”arXiv preprintarXiv:2009.05334, 2020.
[23] RISC-V GNU Compiler Toolchain [Online]. Available: https://github.com/riscv/riscv-gnu-toolchain
[24] C. Kohn “Partial Reconfiguration of a Hardware Accelerator on Zynq-7000 All Programmable SoC Devices XAPP1159”, Xilinx Inc. Application Notes, 2013.
[25] Xilinx Inc.“Partial Reconfiguration Controller PG193” v1.3, October 2018.
[26] Xilinx Inc. “AXI HWICAP PG134” v3.0, October 2016.