A Novel Framework for Effective Preemptive Hardware Multitasking on FPGAs

Krzysztof JOZWIK\textsuperscript{a,} Nonmember, Hiroyuki TOMIYAMA\textsuperscript{††}, Shinya HONDA\textsuperscript{†}, and Hiroaki TAKADA\textsuperscript{†}, Members

**SUMMARY**

Modern FPGAs (Field Programmable Gate Arrays), such as Xilinx Virtex-4, have the capability of changing their contents dynamically and partially, allowing implementation of such concepts as a HW (hardware) task. Similarly to its software counterpart, the HW task shares time-multiplexed resources with other HW tasks. To support preemptive multitasking in such systems, additional context saving and restoring mechanisms must be built practically from scratch. This paper presents an efficient method for hardware task preemption which is suitable for tasks containing both Flip-Flops and memory elements. Our solution consists of an offline tool for analyzing and manipulating bitstreams, used at the design time, as well as an embedded system framework. The framework contains a DMA-based (Direct Memory Access), instruction-driven reconfiguration/readback controller and a developed lightweight bus facilitating management of HW tasks. The whole system has been implemented on top of the Xilinx Virtex-4 FPGA and showed promising results for a variety of HW tasks.

**key words:** dynamic partial reconfiguration, hardware multitasking

1. Introduction

In real-time applications where high responsiveness is demanded from a system, executing tasks till their completion, in a sequential manner, is often not acceptable. If the time in which resources are allocated to a task is too long, then other tasks may not be able to complete its job before a deadline. In these systems, an OS (Operating System) contains mechanisms to save and restore the context of the task, required for its preemption. By means of these mechanisms, the OS can suspend execution of the task at any time and then start it from the point where it was previously stopped. While these mechanisms have been known in software for a long time, their corresponding functionality for the HW task [1]/thread [2] is a new topic and a matter of debate.

This paper makes the following contributions to research on HW Preemptive Multitasking. It presents a solution to preemption of HW tasks, based on bitstream readback, which is suitable to the majority of existing PR (Partially Reconfigurable) FPGA architectures. The solution has a small impact on the original HW task’s characteristics, such as circuit area and clock frequency. Moreover, it supports HW tasks consisting of both FF (Flip-Flops) and memory elements.

In this work, we also show a complete design environment which consists of a PR design flow back-end tool and an embedded system framework. The framework consists of an efficient reconfiguration/readback controller, a HW task control bus, and associated preemption management logic which all help in reducing preemption time and CPU utilization.

Finally, it is the first work presenting a system developed on top of the Virtex-4 FPGA which allows much finer reconfiguration granularity and higher reconfiguration speeds than when compared to its predecessors.

The rest of this work is organized in the following way. Section 2, having given background knowledge on HW task preemption, presents details on its previous implementations and compares them with this work. Section 3 shows details of our approach to HW task preemption. Section 4 presents the experimental results. Finally, in Sect. 5, the research work is concluded and future plans revealed.

2. Related Research

2.1 Basic Concepts in HW Task Preemption

Existing PR FPGAs, such as Xilinx Spartan-6, Xilinx Virtex series, new Xilinx 7-series, and new Stratix-V from Altera, store configuration of the currently programmed digital circuitry in a SRAM-based (Static Random Access Memory) CM (Configuration Memory). This memory is programmed by means of a configuration bitstream which can be further divided into smaller sections called configuration frames. These configuration frames are the smallest addressable units of the CM, and we can also think of them as units of configuration. In all Virtex architectures, the CM is arranged in vertical frame regions which are one bit wide and are configured by the configuration frames. In Virtex architectures preceding Virtex-4, the regions stretch from the bottom to the top of the FPGA, which results in long configuration frames. On the other hand, in Virtex-4 and newer FPGAs, these regions do not span height of the device, which makes the configuration frames shorter and allows reconfiguration at a finer granularity.

While the state of a software task is usually stored on the stack or TCB (Task Control Block), the state of a HW task is distributed over the logic elements such as FFs and memory elements located on the FPGA. There are
two known approaches to access the state of the HW task, namely, CPA (Configuration Port Access) and TSAS (Task Specific Access Structures).

In the CPA methods [3], [4], all state-related frames which correspond to a given HW task are read from the FPGA’s CM through its configuration port. Later on, they are filtered in order to retrieve the state of the HW task. In order to restore the state, it has to be injected into these frames. Then, FPGA must be configured with them and reset in the HW task asserted. Assertion of the reset is required to restore the state of FFs. It should be noted that positions, within the state-related frames, at which state has to be injected do not have to be the same as those from which state was filtered out. Those manipulations in the frames are time-consuming and involve a considerable level of overhead because the task’s state constitutes only a part of those frames. Moreover, they additionally require reasonable knowledge about the structure of the bitstream.

On the other hand, the TSAS methods use an extra interface, implemented as a part of the task, to access the state information directly [5], [6]. In these method, only the state-related data is read, which leads to high efficiency of accessing the data. Nevertheless, the additional access structures significantly increase logic area and decrease clock frequency of the digital circuit implementing the HW task. As the circuit area is increased, the resulting configuration bitstream may be bigger and the reconfiguration time longer.

From the point of view of the CPA-based methods, smaller frame regions in Virtex-4 and newer architectures will often result in more efficient state access. Additionally, in these architectures, state related to FFs is packed into single frames whereas, in previous architectures, it is scattered onto multiple frames. Another difference is bandwidth of a configuration port. For Virtex-4 and its successors, it is eight times higher than for the older devices from the same series. Because all these differences have an impact on the HW task preemption, all previous approaches have to be re-considered.

2.2 Previous Works

Work in [3] presents a CPA-based method for HW task pre-emption. It targets Xilinx Virtex-I architecture and is suitable for HW tasks which only contain FF elements. In that approach, a CPU (Central Processing Unit) reads state-related frames through a configuration port and filters out the contained state. The retrieved state is stored in a data base. Whenever state restore is requested, the CPU injects the state to the state-related frames and configures FPGA with them. Each of the FFs which constitutes the state of the task has a separate entry in the data base. Each entry contains information about the current state of the FF and its address indicating position on the FPGA. If the state of the FFs is contained within a small number of frames, then amount of data needed to store this state in the data base may be bigger than the total size of the containing frames. Since the state is stored in the database, process of filtering it from, and injecting it to, the state-related frames is complex and cannot be easily accelerated by moving it to hardware.

Works in [5], [6] present different TSAS-based methods to state access i.e. memory-mapped, scan chain, and shadow scan chain based methods. Although they do not depend on a specific architecture, they often lead to an unacceptable increase in circuit’s area and decrease in its clock frequency. Moreover, they present a solution for HW tasks which only contain FFs.

In contrast to [3] which stores the state of separate FFs in the database, in the proposed approach, whole state-related frames are stored in a bitstream repository. Details of the HW task’s state model used in this approach can be found in Sect. 3.1. The advantages of the model will be visible especially in the newer Virtex architectures, due to lower data redundancy in state readback. Moreover, this work uses a different method to filter and inject the state of a HW task. It could be more easily implemented in hardware, thereby practically eliminating one of the most time-consuming factors in the task preemption. It is described in Sect. 3.3. Lastly, we also show a method to perform the preemption of all kinds of FFs that can be found in HW tasks i.e. FFs with different configurations of set and reset signals. As opposed to [5], [6], the presented approach practically does not affect characteristics of a digital circuit implementing a HW task. Contrary to all previous works on the task preemption, this work also shows how to preempt HW tasks which contain memory elements.

As mentioned in Sect. 2.1, newer Virtex architectures offer increased bandwidth of a configuration port. For this reason, an efficient mechanism to transfer configuration and state data between the configuration port and an external storage memory is required. To this end, we have developed a high-speed reconfiguration/readback controller, called ICAP-DMA, which performs burst data transfers, thereby improving overall preemption performance.

3. Our Approach to Hardware Task Context Switching

3.1 HW Task State Model

In our approach, we use a model of the HW task state (Fig. 1) in which whole frames, containing state of FFs and memory elements, are stored in a HW task repository. In the model, a mapping between the state-related frames stored in the bitstream and those in the FPGA’s CM is established and used afterwards to save and restore the state of the task. The state-related frames are read from the CM, then pre-processed if needed, and finally stored in the bitstream, as described by State Descriptors. State Pre-processing denotes state filtering and injection. Frames are stored in the bitstream in an already pre-processed form. In case of Xilinx FPGAs, this pre-processing is necessary for frames containing the state of FFs, but it is not needed for frames containing the state of memory elements.

Although implementation of the model depends on the specific architecture, it is applicable to a wider range of
FPGA architectures whose CM is divided into frame regions and configuration port is capable of bitstream readback. All Xilinx FPGAs fulfill these two conditions required by the model. Altera devices are currently not supported because they do not allow readback for security reasons.

3.2 Configuration of Xilinx FPGAs

In Xilinx FPGAs, configuration control logic consists of a packet processor, a set of registers, and global signals controlled by these registers. The packet processor [7] is responsible for control of the data flow from the configuration interface to the set of registers whereas, the registers themselves control all remaining aspects of reconfiguration. Spartan-6 and all Virtex family devices can be reconfigured dynamically and partially by means of three different interfaces. One of them is ICAP (Internal Configuration Access Port) which is a part of the FPGA fabric. It is accessible by user logic, once appropriate configuration, containing its primitive, has been loaded onto FPGA during power-up. In Virtex-4 and its successors, ICAP interface has two 32-bit data ports. One is for incoming data, used to program the CM or change state of the internal registers. The other one is for outgoing data, used to read contents of the CM or the current state of registers. In Spartan-6 devices, the port data-width is 16 bits whereas, in Virtex-I and Virtex-II it is eight bits. All writes to and reads from the FPGA’s CM are pipelined through a one frame long buffer. Therefore, in case of readback, at the time of reading a frame from ICAP, the frame that succeeds it is actually read from the CM and shifted to the buffer. For Spartan-6 devices, the maximum ICAP’s clock frequency is 20 MHz. For Virtex-I and Virtex-II it is 50 MHz. In Virtex-4 and its successors, ICAP can operate at up to 100 MHz and its highest theoretical throughput is 400 MB/s. In practice, it can be only achieved for reconfiguration as in case of readback the packet processor may require additional wait cycles which decrease the overall throughput [7].

3.3 Hardware Task Preemption in Xilinx FPGAs

This section shows a new and efficient method to manipulate the state of a HW task within the state-related frames. Thanks to this method, state filtering and injection can be more easily implemented in hardware. Previous works did not adequately address the issues related to preemption of FFs and memory elements which are present in Xilinx FPGAs. Solving them is indispensable to make the CPA method working on these devices. For this reason, this section also describes these issues and shows how to deal with them.

In all PR Xilinx FPGAs, storage components physically implementing FF elements have two programmable states (Fig. 2). One of them, \texttt{INIT0/INIT1}, is taken on during FPGA power-up. The other one, \texttt{SRLOW/SRHIGH}, is taken on whenever the user logic set/reset signal is asserted. Both states are defined by two different bits in the configuration bitstream. In the readback bitstream, state of a captured FF is stored on the power-up reset state position, while the user reset state bit is not changed. In order to restore the state of the FF, the captured state from the readback bitstream must be extracted and placed onto the user reset state position. By configuring the FPGA with the resulting bitstream and asserting the user reset signal, state of the FF can be restored. If we configure the FPGA leaving the captured state on the power-up state position, the power-up state of the FF will be changed. However, this state will not be ever used during a life-cycle of a HW task. Therefore, it will not affect correctness of the approach. Similarly, modification of bits corresponding to FFs which are unused by a
given HW task will not affect its operation. Both techniques can be used to optimize the state filtering and injection algorithm, making it more regular and easily implementable in hardware.

In some designs, due to optimizations during the logic synthesis, logic connected to the FF’s set/reset pin (Fig. 2) can be placed on the FF’s data input pin, thereby making it impossible to restore its state afterwards. To prevent it, we can guide the synthesis process with constraints which force set/reset signals to be synthesized as a part of the FFs’ reset/set signal paths. However, if user logic implements some function e.g. transition to another state in FSM (Finite State Machine) which utilizes this path and we modify FF’s preset condition, then the user logic may work incorrectly. Therefore, after resetting the FF state to the intended value, we have to restore its initial preset condition.

Another problem arises in Virtex-2, Virtex-4, and Virtex-5 devices in which the storage components natively support an FDRS primitive (D-type FF with set and reset). In these devices, the FDRS is physically implemented with both SR and REV pins [8] connected to the user logic (Fig. 2). According to physically implemented FF’s truth-table, the REV pin inverts the reset/set condition (the one that applies when a signal at the SR pin is asserted) and the reset condition predominates over the set condition. While resetting the FF to zero will still be possible, even if a signal at the REV pin is asserted, trying to set it to a logical one will result in resetting it to a logical zero. In order to overcome this issue, we have to disconnect the REV pin of all FDRSes, restore their state by pulsing one on the SR pin, and then restore the original connection, as shown in Fig. 2.

As mentioned in the documentation of Virtex-2 and Virtex-4 [7],[9], active readback of LUT-RAMs (Lookup Table-based Random Access Memory) and BRAMs’ (Block Random Access Memory) contents may lead to their corruption. It comes from the fact that both user logic and FPGA configuration logic access them through the same interface, which may result in data conflicts. We may circumvent this obstacle firstly, by halting user logic access to these elements while active readback is in progress and secondly, by constraining RP (Reconfigurable Partition) [10]. It must completely fall into the configuration frame region and its right-most column of BRAM resources must be left unused, so that the first frame of neighboring BRAM column from the static part of the system is not loaded into the configuration pipeline (see Sect. 3.2). This way, no memory resources of the static part of the design will be affected by active readback of the RP’s memory frames.

### 3.4 BitFormatter Tool and Resulting Preemption Procedure

Here, we propose a design flow backend tool, called BitFormatter, whose role is to analyze the configuration bitstream representing a given HW task and to extract all the state-related information, creating a sort of extension which facilitates the process of its context save and restore.

This extension follows the state model described in Sect. 3.1. While the tool and the bitstream will depend on a given Xilinx FPGA architecture, the structure of the extension will remain unchanged.

As its inputs, the tool takes an original configuration bitstream and a logic allocation file generated by Xilinx partition-based PR design flow [10] or EAPR (Early Access PR) design flow [11]. As an output, a DBF file (developed DPRS Bitstream Format) is generated.

The DBF file can be divided into five sections, as shown in Fig. 3. Main Header corresponds to the header of the original configuration file. SDDT (State Data Descriptor Table) Header contains information about the SDDT itself, whereas each SDD (State Data Descriptor) is composed of three-word elements. The first one is the offset of the first frame which contains the state information, calculated versus beginning of the configuration bitstream. Knowing the base address of the configuration bitstream, we can calculate the address of the state-related frame within the HW task repository. The second word in the SDD indicates the number of words in the state frames. Finally, the third word in the SDD is the start address of the first frame in the CM. Primary Configuration Bitstream section represents the original configuration bitstream that may be slightly modified for the purpose of the dynamic disconnection of signals, described in Sect. 3.3. Secondary Configuration Bitstream is an additional bitstream generated by the BitFormatter for the purpose of restoring the HW task’s configuration.

In the HW task’s state capture process, frames pointed by the SDDs are read from the FPGA’s CM and written at indicated offsets in the Primary Configuration Bitstream. They cannot be used directly and have to be preprocessed before being used afterwards in restoring the state, as described in Sect. 3.3. This pre-processing i.e. state filtering and injection can be either done in hardware, while reading the data from FPGA’s configuration port, or in software, once the readback process is completed. The HW task restore process is done by configuring FPGA with the aforementioned modified bitstream and asserting the reset signal for one clock cycle, which restores the HW task to the state from which it has been suspended. Later on, by loading the Secondary Configuration Bitstream, the FFs’ preset settings are restored to their initial values.

Dynamic disconnection of FDRS type FFs, mentioned in Sect. 3.3, is handled in two steps. First step is done offline by the BitFormatter tool which preformats the Primary Con-
configuration Bitstream, disconnecting the appropriate signals. Second step is done at run-time, by reconfiguring FPGA with the Primary and Secondary Configuration Bitstreams.

Although currently only synchronous circuits with one clock domain are supported in the HW tasks, support for both multiple clock domain locally synchronous circuits and asynchronous logic is left as a future topic of research.

3.5 Embedded System Architecture

3.5.1 Framework Overview

Together with the design flow back-end tool presented in Sect. 3.4, the framework in Fig. 4 constitutes our design environment for FPGA-based, preemptive HW multitasking embedded systems. In this framework, there are three interconnected components which aim at improving the preemption performance and decreasing the CPU utilization during the preemption. These are ICAP-DMA, HTIC (Hardware Task Interface Control) bus and HW Task Wrapper.

ICAP-DMA is a reconfiguration/readback controller whose two main functions are: transfer of HW tasks’ configuration bitstreams between an external memory and ICAP, and control over the physical aspects of the preemption process. The controller’s architecture allows for burst transfers of both configuration and readback data, which results in significantly improved preemption times. Moreover, it allows for buffering of all requests related to preemption, thereby reducing CPU utilization. Management of the physical aspects of the preemption is done via the HTIC bus which connects ICAP-DMA with the HW Task Interface logic inside the HW Task Wrapper.

There have been already works which used burst transfer to speed up reconfiguration [12], [13]. When compared to them, this work proposes a novel architecture of the controller which not only accelerates reconfigurations but also readbacks and additionally handles the physical aspects of the reconfiguration and preemption. These physical aspects are described in Sect. 3.5.2.

Selection of PLB (Processor Local Bus) as the main communication structure between the HW tasks and the processor was dictated by performance metrics as well as level of support from the current Xilinx embedded development tools. We implemented the framework on top of the Virtex-4 FPGA for which only IBM CoreConnect buses are supported. Among these buses, PLB offers the highest throughputs. Nevertheless, the developed framework could be also utilized with other interconnects, after some additions. These could be limited to developing a new interfacing logic for a HW task, taking advantage of the already available HTIC bus slave module.

3.5.2 Hardware Task Interface Control Bus Structure

The HTIC bus (Fig. 5) is used to control the state of signal ports at the physical interface between the static part of a design and the reconfigurable region. Specifically, reconfigurable region’s output pins must be isolated from the static part during reconfiguration as spurious signals may appear on them. Moreover, HTIC handles clock gating and disabling memory write accesses in the HW task. It has a simple, one master (ICAP DMA) - multiple slaves (HW Task Wrappers) topology and is a write only bus. Every HTIC bus slave interface has a unique ID assigned during system design and used later on when being addressed by ICAP DMA.

Every transaction starts with the master placing ID of the intended RP on the Addr bus, appropriate values of Clk.Out En/Dis, and then asserting Req. Clk.Out En/Dis, Rst inform the slave logic what necessary actions, such as clock suspension/restart and outputs enable/disable, have to be performed.

3.5.3 Reconfiguration/Readback Controller

The developed peripheral, called ICAP-DMA (shown in Fig. 6), uses separate interfaces for data transfers and its control. The PLB master interface is used for DMA-based data transfers between ICAP and external bitstream storage. The DCR (Device Control Register) bus interface off-loads the data interface and is used to control the peripheral. Besides a set of control registers, there are also a few buffers accessible from the control interface. Readback Sequence Buffer is used to store the ICAP’s packet processor-specific instructions (see Sect. 3.2) needed either to retrieve a frame from the CM or to read a value of one of the ICAP’s internal...
When creating the HW task interfacing logic, we utilized a PLB interfacing module generated by Xilinx EDK (Em-
bedded Development Kit) tool’s IP Creation Wizard. A few modifications are needed in the IP in order to support run-
time pluggable HW modules i.e. HW tasks. One of them is inclusion of decoupling logic at the outputs of the user module symbolizing a HW task’s container. Moreover, HW task’s clock and memory write disabling circuitry as well as mentioned before HTIC bus slave interface must be added to the IP. After such modifications, implemented HW tasks can be plugged in as the user logic part of the standard PLB interfacing module.

Since all the tasks will utilize the same PLB inter-
face, such an organization, where only the part of the IP
which varies is treated as a HW task, results in smaller con-iguration bitstreams. Because of the simplified interface
provided by the IP, amount of routing crossing the static-
reconfigurable region interface is reduced, making the tim-
ing closure easier. Since Virtex FPGAs are rich in global buffer resources, these were used as a clock suspension logic. First of all, they do not introduce additional clock skew, as in case of simple clock gating, and there are enough of them to be used in systems with multiple RPs. For the time being, in the developed framework, all tasks utilize the slave PLB interface.

4. Experimental Results

4.1 Testing Environment and Its Limitations

As a testing platform for our framework, we used a Xil-
inx ML410 board hosting a Virtex-4 FX60 speed grade -11 FPGA which was programmed with our system shown in
Fig. 7. The system has been implemented using Xilinx EDK and PlanAhead v12.4 tools. Partial bitstreams representing HW tasks, as well as FPGA initialization file, were copied onto the Compact Flash card and used afterwards by the sys-

All measurements related to preemption times have been performed using the external DDR2 memory serving as bitstream storage. In the presented system, both the CPU and the peripherals, connected to it through the PLB bus, were running at 100 MHz. The CPU data and instructions as well as the SDDTs were placed in a cacheable memory region. The reconfiguration controller was connected to the DDR2 memory controller, Xilinx MPMC (Multi-Port Memory Controller) IP core, through a 64-bit PLB bus.

Unfortunately, the PLB interface of the memory con-
troller could not provide transfer rates of 400 MB/s, needed
to stream the reconfiguration data to ICAP at full speed.
Measurements of transfer rates with a timer peripheral, shown in Fig. 7, were confirmed with an extensive analysis by ChipScope Pro v12.4, Xilinx tool for real-time debugging and verification of FPGA. Consequently, bitstream readback rate was bounded by a value of about 240 MB/s whereas, reconfiguration varied from 274–327 MB/s.

### 4.2 Hardware Task Performance Measurements

Table 2 shows the synthesis results of HW tasks used in tests. CLB-FSM represents a HW task with a few FSMs and user-accessible control and data registers. DES56 is a cryptographic IP taken from www.opencores.org with additional registers to control its operation. LUT-RAM Copy represents a HW task with FSMs in the control path and arithmetic circuits as well as LUT-RAMs in the datapath.

Values in the brackets denote performance and area penalty due to applied synthesis constraints and implemented memory access disable signal.

| Parameter | Task A | Task B | Task C |
|-----------|--------|--------|--------|
| Slices    | 152 (2.7%) | 537 (−3.1%) | 1185 (−2.1%) |
| FFs       | 172 (0%) | 578 (0%) | 475 (−0.2%) |
| LUTs (4−in) | 187 (−0.5%) | 660 (−3.4%) | 1560 (−2.9%) |
| CLK Max. Freq [MHz] | 199.05 (−3.6%) | 231.1 (0%) | 199.05 (−3.6%) |

Task A - CLB-FSM, Task B - DES56, Task C - LUT-RAM Copy

* % - relative performance/area penalty due to applied synthesis constraints and implemented memory access disable signal

### 4.3 HW Task Preemption Times

Table 3 shows RP (Reconfigurable Partition) area’s utilization metrics (column two to four) depending on its size and type of the HW task, as indicated in column one. Both RPs are fully contained in the frame regions and aligned with the frame boundary i.e. their configuration frames are used merely to configure the RPs, not the static area of the design that surrounds them. This is a necessary condition to perform preemption of memory-based HW tasks, as mentioned in Sect. 3.3. It also results in a smaller bitstream, for a given circuit size. Columns five and six show state size of the state frames referenced by the SDD entries and size of the bitstreams generated by the BitFormatter tool, respectively.

| HW Task (RP ID. ) | Impl. Results (RP Utiliz. [%]) | State Save (size [KB]) | State Restore (Prim. + Sec. BitStrm. [KB]) |
|-------------------|--------------------------------|-------------------------|----------------------------------------|
| Task A (RP01)     | 3.42 3.03 2.8                 | 14 (2.24)               | 302.38+7.02                            |
| Task B (RP01)     | 11.48 10.73 9.41              | 21 (3.36)               | 302.38+10.49                           |
| Task C (RP01)     | 35.99 25.37 7.73              | 47 (14.09)              | 302.38+21.9                            |
| Task A (RP02)     | 8.2 7.27 6.72                 | 16 (2.56)               | 115.2+8.01                             |
| Task B (RP02)     | 27.55 25.47 22.58             | 17 (2.72)               | 115.2+8.51                             |
| Task C (RP02)     | 77.59 60.9 18.55              | 20 (6.41)               | 115.2+10.49                            |

Task A - CLB-FSM, Task B - DES56, Task C - LUT-RAM Copy

RP01 size: 3072 slices, 6144 LUTs, 6144 FFs (three frame regions)

RP02 size: 1280 slices, 2560 LUTs, 2560 FFs (one frame region)
buffered in it without disturbing the bitstream transfers.

Because of characteristic of the configuration logic in Virtex-4, one dummy frame and one dummy word have to be read at each readback of the single frame associated with the FFs-related descriptor, resulting in less than 50% efficiency. ICAP-DMA automatically discards the dummy data and sends only the valid frames to memory. Consequently, results in Fig. 8 already take into account readback of only the valid data.

The HW task’s state restore consists of two components: Primary Configuration Bitstream Processing and Secondary Configuration Bitstream Processing, as shown in Fig. 9 and described in details in Sect. 3.4. Configuring FPGA with the Primary Configuration Bitstream takes up significantly more time than that with the second bitstream. In general, reconfiguration time is directly proportional to size of the bitstream, excluding other factors such as deviations in transfer speeds from memory to ICAP. It is worth noting that size of the Primary Configuration Bitstream and the configuration time for the HW tasks used in our tests vary only with the size of the RP. This is because, in our tests, we used uncompressed bitstreams which are generated by default by Xilinx design tools.

Total Preemption time for a given HW Task could be described by a formula:

\[ T_{total} = T_{save} + T_{restore} \]

where \( T_{save} \) denotes the state save time, and \( T_{restore} \) the state restore time.

Figure 10 shows all the components of the task preemption and their relative impact on its total time. The major components are: Primary Configuration Bitstream Processing, constituting up to 71.33% in case of CLB-FSM task in RP#1, and State Pre-processing which takes up as much as 49.75% in case of LUT-RAM Copy task in RP#2.

4.4 Comparison with Previous Works

With regard to work presented in [3], only the reconfiguration and readback speeds can be compared quantitatively. In our approach, these are about eight times higher than those presented in the previous work, excluding the latency incurred by the external memory controller. One of the reasons for that is the different FPGA architecture whose configuration port allows higher throughputs. Another reason is the developed reconfiguration/readback controller which is capable of burst data transfers. State filtering and injection times were not presented in [3]. However, as mentioned before, the method of storing HW Task’s state in our system leads to more regular and faster processing and it is much more feasible to move it to hardware.

The memory-mapped and shadow scan-chain TSAS methods from [6] still outperform our approach in terms of state save time for FF elements. For the similar DES56 IP core operating at the same clock frequency, our approach is still about 40 times slower. If the state pre-processing was implemented in hardware, the corresponding State Pre-processing component (see Fig. 8) would be practically eliminated and this difference reduced to about 2.3x. Moreover, speed results obtained in our approach do not depend on the HW task clock frequency, as in case of the TSAS methods. Some complex HW tasks may not meet timing constraints imposed by the communication structure used to access state. In such a case, they have to work at lower clock frequency, thereby increasing the state access time in the TSAS methods. Furthermore, their reconfiguration time can be longer, due to increased circuit’s size, thereby increasing the total preemption time.

In the proposed approach, resource utilization of the DES56 IP core, shown in Sect. 4.2, is improved and clock frequency is not changed. On the other hand, the TSAS methods presented in [6], for the same kind of core, increase
amount of FFs by 5–112%, amount of LUTs by 28–96% and
decrease clock frequency by 2–17%. Since target FPGA ar-
chitecture used in our experiments and the one used in [6]
differ, we cannot compare these results directly. Neverthe-
less we can clearly see that, when compared to those TSAS
methods, impact of our approach on the circuit’s character-
istics is negligible.

Regretfully, we could not compare results of preemp-
tion of the HW Tasks with memory elements since these
were not presented in any previous work.

Considering the possibility of further improvements,
the obtained timing results and the advantages of used
method versus previous CPA method [3], related to context-
switch efficiency, and TSAS-based approaches [5], [6], re-
lated to HW task performance, prove to be very attractive.

5. Conclusions

In this document we presented a novel and holistic solu-
tion to efficient HW task preemption for Xilinx FPGAs.
Firstly, an in-depth analysis of issues related to preemption
of FF-based and memory-based HW tasks has been con-
ducted. Secondly, system implemented on top of the Virtex-
4 FPGA, serving as a proof of the concept, has been shown.
Experimental results prove efficiency of our approach.

Acknowledgments

We would like to thank prof. Masato Edahiro for his invalu-
able comments and help related to this paper.

This work was in part supported by KAKENHI 22700050.

References

[1] B.H. Walder and M. Platzer, “A runtime environment for reconfig-
urable hardware operating systems,” FPL, 2004.
[2] D. Andrews et al., “Programming models for hybrid FPGA-CPU
computational components: A missing link,” IEEE Micro, vol.24,
2004.
[3] H. Kalte and M. Porrmann, “Context saving and restoring for multi-
tasking in reconfigurable systems,” FPL, 2005.
[4] L. Levinson et al., “Preemptive multitasking on FPGAs,” FCCM,
2000.
[5] S. Jovanovic et al., “Hardware preemptive multitasking mechanism
based on scan-path register structure for FPGA-based reconfigurable
systems,” AHS, 2007.
[6] D. Koch et al., “Efficient hardware checkpointing: Concepts, over-
head analysis, and implementation,” FPGA, 2007.
[7] Xilinx, Inc., UG071 (v1.10): Virtex-4 FPGA Configuration User
Guide, 2008.
[8] Xilinx, Inc., UG070 (v2.5): Virtex-4 FPGA User Guide, 2008.
[9] Xilinx, Inc., XAPP988 (v1.0): Correcting Single-Event Upsets in
Virtex-4 Platform FPGA Configuration Memory, 2008.
[10] Xilinx, Inc., UG702 (v12.3): Partial Reconfiguration User Guide,
2010.
[11] Xilinx, Inc., UG208 (v1.2): Early Access Partial Reconfiguration
User Guide - For ISE 9.2.04i, 2008.
[12] C. Claus et al., “A multi-platform controller allowing for maximum
dynamic partial reconfiguration throughput,” FPL, 2008.

Krzysztof Jozwik received the B.Sc. degree in electrical and computer engineering from
Warsaw University of Technology, Poland, in 2007 and M.Sc. degree in information science from
Nagoya University, Japan, in 2010, where he is currently pursuing the Ph.D. degree in dyna-
merically reconfigurable embedded systems.

Hiroyuki Tomiyama received his Ph.D. de-
gree in computer science from Kyushu Univer-
sity in 1999. He is at present a full professor
with the College of Science and Engineering,
Ritsumeikan University.

Shinya Honda received his Ph.D. degree in
the Department of Electronic and Informa-
tion Engineering, Toyohashi University of Tech-
nology in 2005. In 2006, he joined the Cen-
ter for Embedded Computing Systems, Nagoya
University, as an assistant professor, where he is
now an associate professor.

Hiroaki Takada is a Professor at the Depart-
ment of Information Engineering, the Gradu-
atent School of Information Science, Nagoya Uni-
versity. He is also the Executive Director of
the Center for Embedded Computing Systems
(NCES). He received his Ph.D. degree in In-
formation Science from University of Tokyo in 1996.

[13] J. Delorme et al., “New opbhwicap interface for realtime partial re-
configuration of FPGA,” ReConFig, 2009.
[14] R. Nair et al., “A symbol based algorithm for hardware implementa-
tion of cyclic redundancy check (CRC),” VHDL International Users’
Forum, 1997.