ZyCAP: Efficient Partial Reconfiguration Management on the Xilinx Zynq

Kizheppatt Vipin, *Student Member, IEEE*, and Suhaib A. Fahmy, *Senior Member, IEEE*

***Abstract*—New hybrid FPGA platforms that couple processors with a reconfigurable fabric, such as the Xilinx Zynq, offer an alternative view of reconfigurable computing where software applications leverage hardware resources through the use of often reconfigured accelerators. For this to be feasible, reconfiguration overheads must be reduced so that the processor is not burdened with managing the process. We discuss partial reconfiguration (PR) on these architectures, and present an open source controller, ZyCAP, that overcomes the limitations of existing methods, offering more effective use of hardware resources in such architectures. ZyCAP combines high-throughput configuration with a high-level software interface that frees the processor from detailed PR management, making PR on the Zynq easy and efficient.**

***Index Terms*—Accelerator architectures, field-programmable gate arrays (FPGAs), reconfigurable computing.**

I. Introduction

HYBRID field-programmable gate arrays (FPGAs) integrate capable embedded processor cores with a reprogrammable fabric, bringing together software and custom hardware in a manner that makes reconfigurable computing attractive beyond its traditional support base. A software-centric view, but one in which complex computation can be offloaded to custom hardware accelerators, has long interested the reconfigurable computing community, and hence such coupling has been explored in the past. The advent of devices such as the Zynq from Xilinx [1], integrating ARM processors with a reconfigurable fabric, suggests that this view will begin to dominate.

In this letter, we explore how partial reconfiguration (PR) can be exploited efficiently on such architectures. Traditional approaches have often assumed a dedicated processor for managing the PR process, yet it is expected that managing PR in such systems will now be just one of the embedded processor's many tasks, and hence this must be done in a way that does not impact overall system performance. Fig. 1 shows a simplified block diagram of the Xilinx Zynq architecture. The programmable logic (PL) is attached to the processing system (PS) through multiple ARM AMBA AXI ports, offering high bandwidth coupling between them: two 32-bit AXI master (GP master) interfaces; two 32-bit AXI slave (GP slave) interfaces, and four 64-bit high-performance AXI slave (HP) interfaces. The processor configuration access port (PCAP) interface enables full and partial reconfiguration of the PL from the PS.

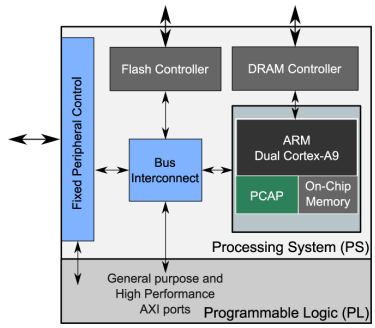

Fig. 1. Zynq Architecture showing PS, PL and the interconnects.

Unlike previous FPGAs with hard processors requiring significant infrastructure in the programmable logic, the Zynq PS is fully capable, and the PL is viewed as an auxiliary resource that can be used to improve application performance. We now consider software applications making use of the programmable fabric to implement accelerators, rather than the PL implementing a self-contained system with some processors for support.

When we factor in that the available PL area is somewhat restricted, we envisage the use of the PL to implement a variety of small accelerators, with the processor loading them dynamically as needed. PR becomes essential as it enables such a time-multiplexed use of the PL, increasing effective logic capacity, while also contributing to improved power consumption and reduced configuration overhead, when compared to static approaches.

Within this new paradigm, it is essential that the overhead of managing PR does not compromise the other tasks being undertaken by the processor; inefficient PR management can significantly diminish any acceleration benefits. Currently supported methods of PR management on the Zynq fail to address this issue. Our previous work demonstrated that custom reconfiguration controllers can considerably reduce reconfiguration overhead in processor-less systems [2]. In this letter, we present a new open-source reconfiguration controller and associated driver that significantly improve reconfiguration throughput on the Zynq while freeing the processor to work on other tasks. Aside from improving performance, it also simplifies the management of PR processes through a high-level driver interface.

II. Partial Reconfiguration

Partial reconfiguration (PR) design involves defining regions on the FPGA, called partially reconfigurable regions (PRRs),

Manuscript received February 04, 2014; accepted March 26, 2014. Date of publication March 28, 2014; date of current version August 26, 2014. This manuscript was recommended for publication by Z. Shao.

The authors are with the School of Computer Engineering, Nanyang Technological University, Singapore 639798 (e-mail: vipin2@ntu.edu.sg; sfahmy@ntu.edu.sg).

Color versions of one or more of the figures in this letter are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/LES.2014.2314390

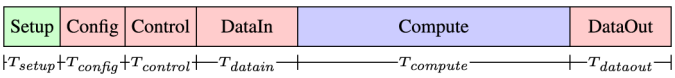

Fig. 2. Task profile for implementing hardware acceleration [4].

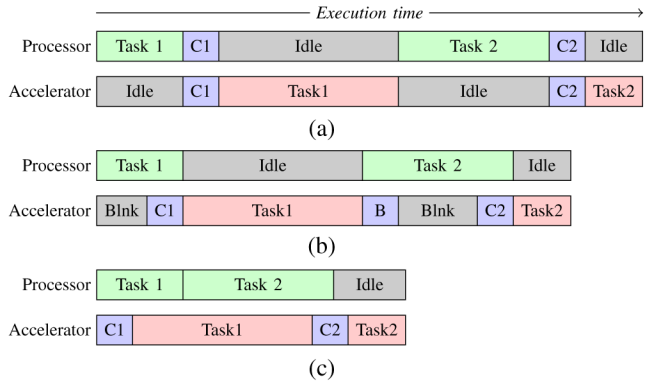

Fig. 3. Effect of overlapping hardware and software execution. (a) Processing and reconfiguration happening sequentially. (b) Reconfiguration in parallel with processing for dependent tasks. C1 and C2 represent accelerator reconfigurations and B represents blanking the PRR. (c) Software and accelerators running independent tasks with minimal software management overhead.

which can house different modules and be dynamically changed at runtime by loading a partial bitstream [3]. Though Zynq supports full reconfiguration of the PL from the PS, PR still offers several advantages. Firstly, during a full reconfiguration, the entire PL is out of use. Secondly, in a system with multiple independent accelerators, a full reconfiguration forces reconfiguration of all of them even if not required. This is especially troublesome if the software drivers must reinitialize them each time a configuration is completed. Finally, in cases where the PL is being used to connect external peripherals, such as sensors or actuators, a reconfiguration breaks this link. PR overcomes all these limitations, and is hence a key enabler for this paradigm of reconfigurable computing.

To understand the impact of PR on system performance, consider the typical profile for an accelerator task as depicted in Fig. 2. The system configures the accelerator on the fabric, sends input data, triggers execution, then reads back the output after execution. T_{setup} is the time taken to decide whether a reconfiguration is required, T_{config} is the reconfiguration time, T_{control} is the time taken to trigger the accelerator, T_{datain} is the time to send data to the accelerator, T_{compute} is the accelerator execution time and T_{dataout} is the time for the results to be read back. This profile shows that efficient management functions are paramount in maximising the benefits offered by acceleration. T_{datain} and T_{dataout} depend upon the system architecture and how data movement is managed. T_{control} is usually negligible, involving register configurations. A PR system should minimize T_{setup}, while also maximizing reconfiguration throughput to minimize T_{config}. If the processor is used to manage all the reconfiguration steps, then it is not available for other tasks. This is especially true when the number of accelerator tasks and frequency of reconfiguration increases [5].

Now let us consider the impact of PR on system power consumption. Since the size of partial bitstreams can be significantly smaller compared to a full bitstream, the power consumed for reconfiguration can be lower for PR, due to shorter reconfiguration time.

PR has also been used to save power by blanking unused PRRs with blank bitstreams. For PR blanking of a region to provide an energy saving, it is required that [6]:

$$T_{\mathrm{reconfig}} > P_{\mathrm{pr}} \times S_{\mathrm{bit}}/P_{\mathrm{ext}} \times t_{\mathrm{inactive}}$$

where P_{pr} is the power consumption for PR, S_{bit} is the bitstream size, P_{ext} is the power consumed to transfer the partial bitstream from external memory to the FPGA and t_{inactive} is the time for which a PRR remains inactive. Hence, maximizing reconfiguration throughput (T_{reconfig}) increases potential power savings.

The desire is that the processor handles only high-level reconfiguration management while the lower level mechanics are managed separately. The advantage of this approach is that execution of tasks on the processor and reconfiguration of the PL can be overlapped. Fig. 3 shows the profile for an application comprising two software and two hardware tasks executed alternately. In Fig. 3(a), the processor manages configuration, and so must wait for this to complete before executing its software tasks. Fig. 3(b), shows how the overall execution time is reduced when the processor is only tasked with initiating the reconfiguration. The reconfigurable region can be blanked when no accelerator is used to reduce power consumption without compromising system performance. In Fig. 3(c) we show the potential gains for independent tasks; now that the processor is freed from low-level configuration management, it can continue with other tasks (subject to dependencies).

III. Partial Reconfiguration Management on the Zynq

In this section we discuss existing Zynq PR schemes, and introduce a custom PR controller that overcomes the limitations of these methods. The PL can be reconfigured from the PS or from within the PL itself. The PS uses the device configuration interface (DevC), which has a dedicated DMA controller to transfer bitstreams from external memory to the processor configuration access port (PCAP) for reconfiguration. The Zynq also has an internal configuration access port (ICAP) primitive in the PL, as found in other Xilinx FPGAs. The ICAP has a 32-bit, 100 MHz streaming interface, providing up to 400 MB/s reconfiguration throughput.

A. State of the Art

Officially Xilinx supports two schemes for PR on the Zynq, one through the PCAP and the other through the ICAP. By specifying the starting location and size, the library function `XDcfgTransferBitfile()` can be used to transfer PR bitstreams from external memory (DRAM) to the PCAP. The main advantage of this scheme is that it does not require any PL resources and gives a moderate reconfiguration throughput of 128 MB/s. The main drawback is that it blocks the processor during reconfiguration, precluding overlapped reconfiguration as discussed in Section II.

Xilinx also provides an IP core (AXI_HWICAP) and library function (`XHwIcapDeviceWrite()`) to enable PR using the ICAP. The AXI-Lite interface of the core is used to connect it to the PS through a GP port. Since this method is not DMA

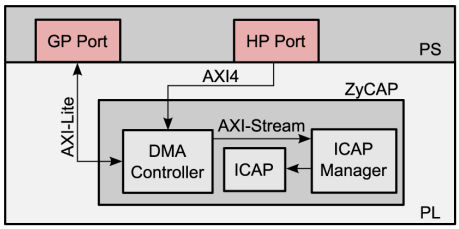

Fig. 4. ZyCAP showing interface connections.

based, throughput is only 19 MB/s. This approach also blocks the processor, and is hence inferior to the PCAP approach.

We have modified the ICAP approach by interfacing the hard DMA controller in the PS with the AXI_HWICAP IP and writing a custom driver function. An interrupt from the DMA controller is used to indicate completion of reconfiguration. The achievable throughput in such a case is 67 MB/s, which is significantly slower than through the PCAP. Since the AXI_HWICAP IP has a single AXI-Lite interface, it is not possible to connect it to the HP port for better performance. However, this scheme has an advantage that reconfiguration can be overlapped with processing.

B. ZyCAP PR Management

To achieve maximum performance, we have developed a custom controller, called ZyCAP, and an associated driver, to verify whether such a solution can improve on PCAP performance, while reducing processor PR management overhead. Previous experiments with traditional FPGAs showed that a custom solution can provide near theoretical peak reconfiguration throughput [2]. But such custom controllers were designed for processor-less systems, and hence did not provide a software-centric view or run-time reconfiguration management, making them difficult to port to the Zynq.

ZyCAP has two interfaces, an AXI-Lite interface connected to the PS through a GP port and an AXI4 interface connected to an HP port as shown in Fig. 4. Since it adheres to Xilinx's *pcore* specification, ZyCAP can be used like other IP cores in Xilinx XPS. Internally, ZyCAP instantiates a soft DMA controller, an ICAP manager and the ICAP primitive. The DMA controller is configured with the starting address and size of the PR bitstream through the AXI-Lite interface and bitstreams are transferred from external memory (DRAM) to the controller at high speed through the HP port using the burst-capable AXI4 interface. The ICAP manager converts the streaming data received from the DMA controller to the required format for the ICAP primitive. ZyCAP raises an interrupt once the bitstream has been fully transferred to the ICAP.

ZyCAP achieves a reconfiguration throughput of 382 MB/s (95.5% of the theoretical maximum), improving over AXI_HW ICAP, DMA based AXI_HW ICAP, and PCAP by $20\times$, $5.7\times$, and $2.98\times$, respectively. The deviation from theoretical maximum is due to the software overhead for DMA controller configuration, DRAM access latency and interrupt synchronization. A comparison of different PR methods in terms of resource utilization and reconfiguration throughput is shown in Table I.

TABLE I
COMPARISON OF RESOURCE UTILIZATION FOR DIFFERENT PR METHODS ON THE ZYNQ.

Method	Resource Utilisation			Throughput
	FFs	LUTs	BRAMs	(MBytes/sec)
PCAP	0	0	0	128
Xilinx ICAP (non-DMA)	443	296	0	19
Xilinx ICAP (with DMA)	443	296	0	67
ZyCAP	806	620	0	382

C. Run-Time PR Management

Along with high reconfiguration throughput, lean run-time reconfiguration management is also required for better system performance. The ZyCAP software driver implements management functions such as transfer of bitstreams from nonvolatile memory to the DRAM, memory management for partial bitstreams, bitstream caching, ZyCAP hardware management and interrupt synchronization. The driver provides an API through which high-level software applications can manage PR.

The driver is initialized with the `Init_Zycap()` call, which allocates buffers in DRAM for storing bitstreams, configures the *DevC* interface, and configures the interrupt controller. The number of bitstreams buffered in DRAM is configurable and defaults to five. A reconfiguration is initialized using the `Config_PR_Bitstream()` call, by specifying only the bitstream name. Unlike existing vendor APIs, the software designer does not need to know where the bitstream is stored or what the bitstream size is.

The driver internally manages partial bitstream information such as the bitstream name, size and DRAM location. When a configuration command is received, it first checks if the bitstream is cached in DRAM, and if so configures the ZyCAP soft DMA controller with the bitstream location and size to trigger reconfiguration. If it is not cached, it is transferred from nonvolatile memory (SD card) to a buffer in the DRAM and the corresponding data structure is created. If all DRAM bitstream slots are full, the *least recently used* (LRU) bitstream is replaced. The driver also enables precaching of bitstreams in the DRAM using the `PrefetchPRBitstream()`API.

The driver supports deferred interrupt synchronization, which enables nonblocking processor operation during reconfiguration. By setting the `intr_sync` argument in `Config_PR_Bitstream()`, the function returns immediately after configuring the DMA controller. The interrupt corresponding to the reconfiguration can be synchronized later using the `Sync_Zycap()` call before accessing the reconfigured peripheral. In this way the processor is free to execute other software tasks while reconfiguration is in progress. If `intr_sync` is set to zero, the driver operates in blocking mode and returns only after reconfiguration.

IV. CASE STUDY

To analyze the effect of different PR schemes on overall system performance, we consider a case study from [4]. The experiment involves image edge detection after a low-pass filter is applied. Each image is processed twice. First, through a median filter followed by Sobel edge detection, then a smoothing filter followed by Sobel. The modules used for the experiments are reconfigured sequentially in a single PRR. An image is first

TABLE II
COMPUTATION TIMING PARAMETERS

Parameter	Desig.	Value (Seconds)
Decision time	T_{setup}	0
Reconfiguration time	T_{config}	0.970/T
Transfer of control time	$T_{control}$	1.12×10^{-6}
Data send time	T_{datain}	$(B/400.5) \times 10^{-6}$
Compute time	$T_{compute}$	0
Data receive time	$T_{dataout}$	$(B/134.2) \times 10^{-6}$

transferred from external memory to a processing core and the processed image is streamed back to the memory via DMA. After each algorithm, the output is analyzed by the processor for quality checks.

For our experiments, we use the *ZedBoard* [7]. The PRR size is 2300 CLBs, 60 DSP blocks and 50 BRAMs, large enough to accommodate the largest module (smoothing filter). The partial bitstream size is 1,018,080 Bytes while a full Zynq bitstream would be 4,045,564 Bytes. A soft DMA controller is used to transfer data between the external memory and the processing core through an HP port and a hardware timer is interfaced for accurate performance measurement. All PL components run at 100 MHz. The hardware and software for this evaluation are developed using Xilinx EDK and PlanAhead 14.6.

DMA transfers between the external memory and the PRR are measured at 382 MB/s. Throughput between the processor and the external memory is 128 MB/s. The latency for accessing a peripheral from the processor is 140 ns. To configure the DMA controller and manage data movement, 8 registers are configured by the processor, consuming 1.12us. These map to execution time parameters as shown in Table II, for processing B Bytes of data at a reconfiguration speed of T MB/s.

Since this application uses a single PRR and follows a predefined reconfiguration sequence, no decision time is required ($T_{setup} = 0$). Reconfiguration time depends upon the reconfiguration scheme used, while $T_{control}$ corresponds to DMA controller configuration. $T_{compute} = 0$ since the cores operate in streaming mode. Each iteration requires two configurations and two sets of DMA operations. For schemes that do not support overlapped reconfiguration, the processor can only execute its quality checks after configuring the hardware for next iteration. For overlapped schemes, the processor can do this while the hardware is being reconfigured.

Fig. 5 shows the effect of the different reconfiguration schemes on system throughput. As image size increases, parallel hardware and software execution (solid lines) has a clear benefit. In these cases, when the software execution time is smaller than the reconfiguration time, the PCAP method has a significant advantage over the DMA based AXI_HW_ICAP due to its higher throughput. However, as the data size increases (above 512×512 pixels), overlapped reconfiguration becomes more important, and DMA based AXI_HW_ICAP outperforms PCAP since software execution time is now comparable to reconfiguration time. For large frame sizes, the performance of the DMA based methods converges since the reconfiguration time begins to diminish with regard to software execution time. The same is true for blocking nonDMA based methods, but

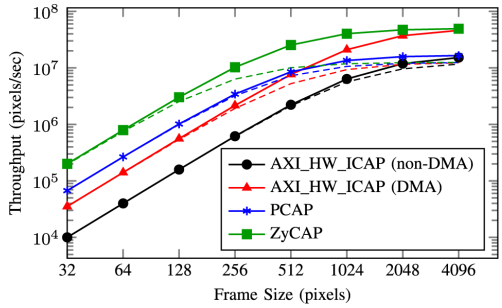

Fig. 5. Comparison of total number of pixels processed for different PR schemes. Solid lines represent hardware-software coexecution and dotted lines represent sequential hardware and software execution.

they saturate at a lower overall throughput. At an image size of 512×512, ZyCAP increases application throughput by $11.35\times$, $3.28\times$, and $2.96\times$, over AXI_HW_ICAP, DMA based AXI_HW_ICAP, and PCAP, respectively.

V. CONCLUSION AND FUTURE WORK

We have discussed the role of partial reconfiguration in hybrid FPGA platforms such as the Xilinx Zynq. We presented ZyCAP, a controller that significantly improves reconfiguration throughput in Zynq system over standard methods, while allowing overlapped execution, resulting in improved overall system performance. ZyCap also plays a significant role in automating PR development on hybrid FPGAs [8].

ZyCAP is has been implemented for the Standalone operating system, and we aim to add support for Linux. ZyCAP hardware can be similarly used with soft processors like MicroBlaze, but driver software modifications are required for interrupt management. This would make ZyCAP portable across all Xilinx PR capable FPGAs. We are releasing this design in the public domain to help encourage adoption of PR on hybrid FPGA platforms [9].

REFERENCES

[1] UG585: Zynq-7000 All Programmable SoC Technical Reference Manual Xilinx Inc., Mar. 2013.
[2] K. Vipin and S. Fahmy, "A high speed open source controller for FPGA partial reconfiguration," in *Proc. Int. Conf. Field Programmable Technol. (FPT)*, 2012, pp. 61–66.
[3] UG702: Partial Reconfiguration User Guide Xilinx Inc., Oct. 2010.
[4] E. El-Araby, I. Gonzalez, and T. El-Ghazawi, "Exploiting partial run-time reconfiguration for high-performancee reconfigurable computing," *ACM Trans. Reconfigurable Technol. Syst. (TRETS)*, vol. 1, no. 4, pp. 21:1–21:23, Jan. 2009.
[5] M. Gokhale, P. Graham, E. Johnson, N. Rollins, and M. Wirthlin, "Dynamic reconfiguration for management of radiation-induced faults in FPGAs," in *Proc. Int. Parallel Distrib. Process. Symp.*, 2004, p. 145.
[6] S. Liu, R. N. Pittman, A. Forin, and J. Gaudiot, "On energy efficiency of reconfigurable systems with run-time partial reconfiguration," in *Proc. IEEE Int. Conf. Appl.-specific Syst. Arch. Process. (ASAP)*, 2010, pp. 265–272.
[7] ZedBoard: Hardware User's Guide Jan. 2013.
[8] K. Vipin and S. A. Fahmy, "Automated partial reconfiguration design for adaptive systems with CoPR for Zynq," in *Proc. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, 2014.
[9] Zycap [Online]. Available: https://github.com/archntu/zycap