

A High Speed Open Source Controller for FPGA Partial Reconfiguration

Kizheppatt Vipin, Suhaib A. Fahmy

*School of Computer Engineering
Nanyang Technological University, Singapore
{vipin2, sfahmy}@ntu.edu.sg*

Abstract—Partial Reconfiguration (PR) is an advanced technique, which improves the flexibility of FPGAs by allowing portions of a design to be reconfigured at runtime by overwriting parts of the configuration memory. PR is an important enabler for implementing adaptive systems. However, the design of such systems can be challenging, and this is especially true of the configuration controller. The generally supported methods and IP have low throughput, resulting in long configuration time that precludes PR from systems where this operation needs to be fast. In this paper, we present a high-speed configuration controller that provides several features useful in adaptive systems. The design has been released for use by the wider research community.

I. INTRODUCTION

The functionality of an FPGA is determined by the contents of its configuration memory. Changing the contents of the configuration memory allows a new circuit to be implemented; this is known as reconfiguration. Usually, the contents of the whole configuration memory are modified; selectively modifying only portions of the configuration memory is known as partial reconfiguration (PR).

PR has several advantages, such as more fine-grained flexibility and the ability to time-multiplexing more functions on smaller FPGAs, resulting in a reduction in power consumption and cost. When designing a PR system, the user decides on which parts of the design can be reconfigured at runtime; these are called reconfigurable modules (RMs). RMs are then implemented in user-defined areas on the FPGA, known as reconfigurable regions (RRs).

PR is gaining increased importance due to its suitability for implementing adaptive systems. These are systems that can modify their operation at run time, based on internal or external constraints. For example, a cognitive radio can use spectrum sensing to locate a suitable transmission before switching to receive mode [1]. The advantage of using PR here is that multiple mutually exclusive functions can be swapped in and out on demand, saving area and power.

One key consideration when implementing PR-based systems is reconfiguration time. This is the time it takes to configure the system from one operating mode to another, and includes reconfiguring one or more RRs. A lengthy reconfiguration time is not suitable for several classes of applications.

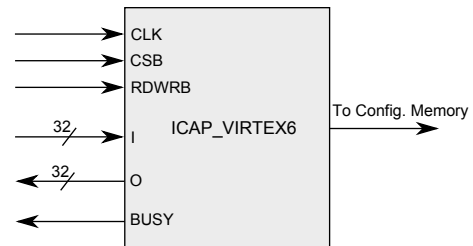


Fig. 1. Xilinx ICAP Primitive.

Presently, reconfiguration of average-sized regions takes a time in the order of milliseconds. Minimising reconfiguration time is essential to further adoption of PR in a wider range of applications.

The fundamental limiting factor that impacts reconfiguration time is the speed of writing data to the configuration memory. We refer to this data as a bitstream, and the data required to configure a RR is called a partial bitstream. The hard-macro in Xilinx FPGAs that serves the purpose of writing to the configuration memory is the Internal Configuration Access Port (ICAP) as depicted in Fig.1. Maximising ICAP throughput has a significant effect on minimising configuration time. Vendor-provided IP cores to control the ICAP generally have very low throughput. There is also minimal support for building a processor-free PR system. The result is that real-world reconfiguration time is generally high, and coupled with the system overhead of implementing PR, this puts many designers off using it in their systems.

In this paper, we present an ICAP controller that consumes minimal area, while maximising throughput. The controller implements additional features useful in adaptive system implementation. We also show how the Dynamic Reconfiguration Port (DRP) feature in recent Xilinx FPGAs enables reconfiguration throughput to be further improved. We are releasing this design publicly to assist those interested in implementing PR-based adaptive systems [2].

The rest of this paper is organised as follows: Section II discusses related work, Section III introduces some important background on PR, Section IV presents the system architecture and the functions of each module, Section V presents

the implementation results and a comparison with previous implementations, and Section VI concludes the paper.

II. PREVIOUS WORK

Reconfiguration time in PR systems depends upon a number of factors. System-level design decisions, such as the number of regions to use and how modules are allocated to regions, can make a modest impact. Efficient partitioning [3] has been shown to have some impact, while bitstream compression [4] has also been suggested. Considered floorplanning of PR regions has also been shown to have an impact [5]. However, the fundamental limit is governed the speed at which bitstreams can be written to the configuration memory.

Traditionally, the reconfiguration operation is controlled by a processor, through a vendor-provided ICAP controller such as the OPBHWICAP or XPSHWICAP, connected as a slave device to the processor bus [6], [7]. Using these vendor-provided controllers gives low throughput in the region of 4.66-10.1 MBytes/sec [8], [9]. This should not be the case, as the ICAP hard macro itself supports speeds of up to 400 MBytes/sec.

In [10], the authors proposed to connect the ICAP controller to the fast simplex link (FSL) bus of the processor. The drawback is that the processor becomes consumed with the task of requesting configuration data from the external memory and sending it over the FSL bus. The resulting throughput of under 30 MBytes/sec remains well below the theoretical limit of the ICAP.

Using DMA to transferring partial bitstreams from external memory has previously been shown to be effective in increasing throughput [11]. Elsewhere, some have tried to achieve better performance by over-clocking the ICAP primitive [12]. Since the maximum frequency at which the controller can operate depends upon manufacturing variability and specific placement and routing, this would need to be determined on a device-by-device basis, which is cumbersome.

Other work on optimised ICAP controllers has often made unrealistic assumptions, such as the complete configuration bitstream being stored in FPGA Block RAM [9]. This is not practical, as FPGAs have limited memory that is often insufficient for even a small number of bitstreams, and these memories are often required for system implementation.

In [13], the authors review a significant number of ICAP controllers and evaluate their performance. They also develop a cost model for determining the performance of controllers. They consider a real system architecture, where bitstreams are initially fetched from external memory into FPGA local memory, and later used for configuration.

In this paper, we present an ICAP controller for PR systems that enables the loading of bitstreams from external memory at speeds very close to the theoretical limit of the ICAP primitive, while consuming minimal area. We further enhance the controller's capabilities with features that assist in the implementation of adaptive systems that use PR. We compare our work with previous implementations, and show that it is both faster, and more compact.

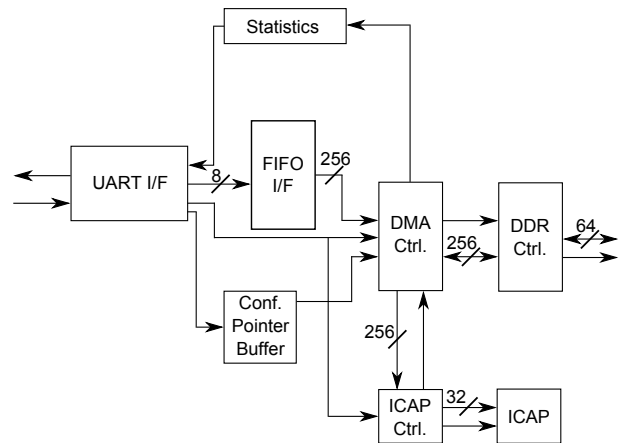


Fig. 2. System architecture.

III. PARTIAL RECONFIGURATION OVERVIEW

Xilinx supports PR through their software package, PlanAhead. First, the whole design is partitioned into *static logic* and a number of *reconfigurable modules* (RMs). The functionality of the static logic does not change during runtime. Reconfiguration control must be implemented in the static logic. The netlist for each RM is generated using Xilinx XST or any third-party synthesis tools. During design time, each RM is assigned to a reconfigurable region (RR). RRs should be rectangular in shape and are composed of different FPGA primitives such as configurable logic blocks (CLBs), Block RAMs etc. The smallest addressable segment of the FPGA configuration memory is known as a frame. Partial bitstreams corresponding to every required configuration for each RR are generated using PlanAhead and the associated tool-chain. The size of the bitstream is proportional to the size of the RR in frames, and this directly determines reconfiguration time for the region.

The ICAP is a hard macro present in the FPGA, which is used to access the configuration memory from within. The ICAP present in Virtex-6 devices, ICAP_VIRTEX 6, is shown in Fig. 1. The ICAP data interface can be set to one of three data widths: 8, 16, or 32 bits. *CSB* is the active-low interface select signal, *RDWRB* is the *read/write* select signal. *BUSY* is valid only for read operations and remains low for write operations. The maximum recommended frequency of operation for the ICAP is 100 MHz.

IV. SYSTEM ARCHITECTURE

The overall system architecture for our controller is shown in Fig. 2. The design we present here functions within an adaptive system, for which we define two phases. The first is the preparation phase, in which all the required partial bitstreams are fetched from outside the system, and stored in the on-board memory of the adaptive system. These bitstreams may originate from a file system on some non-volatile storage, or alternatively be sent from a host PC. In the latter case, we can use a simple interface such as UART, as this preparation phase is not time-critical. The bitstreams are read in over

this interface, then pushed through DMA into the external memory, while storing their location labels in a Configuration Pointer Buffer for later use. Once preparation is complete, the system enters the runtime phase, where the adaptive system can now autonomously load partial bitstreams and reconfigure regions. This involves the transfer of partial bitstreams from the on-board memory to the ICAP controller using the DMA controller.

A. DDR Memory Controller

The DDR controller controls the external memory based on the commands from the DMA controller. This is a DDR3 controller, which controls a 64-bit wide external memory. This core is generated using Xilinx's memory interface generator (MIG) wizard [14]. The core's read and write data ports are 256 bits wide and run at 200 MHz.

B. DMA Controller

The DMA controller is an important component in this system, and is largely responsible for the high speed. This block performs a DMA write operation to store partial bitstreams in the external memory during the preparation phase, as well as reading the bitstreams from memory when reconfiguring regions in the runtime phase. Before storing a partial bitstream to memory, the DMA controller is armed with the DMA transfer length in bytes and the starting memory location at which to store the bitstream. This information is provided to the controller using two internal registers, which can be set by the external host. During the write operation, the DMA controller keeps track of the number of bytes present in the FIFO, and whenever sufficient data is present for a write operation (here, 64 bytes), it is transferred to the memory controller. During read operations, the DMA controller instructs the memory controller to generate memory read sequences, until the specified number of bytes are read. To achieve high throughput, the controller issues back-to-back read commands. For read operations, the address and data length can be obtained either from a host system, or from the configuration pointer buffer. The DMA controller and the memory controller coordinate their operations using a producer-consumer model handshake.

C. FIFO Interface

Partial bitstreams are temporarily stored in a FIFO before being loaded into the DDR memory from the host system. This interface serves two purposes: the host interface can be changed without affecting the remainder of the system, and data can be packed, allowing the host interface width to be different from the data width of the system memory interface. At present, UART data comes in one byte at a time and the system memory controller write width is 32 bytes; this difference is managed by the FIFO interface.

D. Configuration Pointer Buffer

This buffer stores the size of the partial bitstreams as well as their starting location in the external memory. These

parameters are stored automatically when the DMA controller transfers partial bitstreams to the external memory in the preparation phase. Each set of parameters has a reference number, which is their ordinal number of transfer from the host system. The advantage of using this buffer is that when the system is in the runtime phase, an adaptation controller can load partial bitstreams using just their pointer label, rather than having to be aware of any further bitstream details. The DMA engine configurations as well as programming sequences are done automatically by the state machine controlling this buffer.

E. Statistics

The statistics block contains two hardware counters for system performance monitoring. The first counter measures the overall performance by measuring the number of clock cycles required for reconfiguration to complete after the command is issued. The second counter measures the performance of the ICAP controller.

The values present in these registers can be accessed from a host system through the UART interface, or used within the rest of the adaptive system. The presence of hardware counters provides precise system performance measures in comparison to inaccurate software counters. that can be utilised by the adaptive control for efficient system management.

Another parameter monitored by this block is the number of times each partial bitstream is used for reconfiguration. In an adaptive system scenario, this information gives an overview of the conditions in which the system is operating, since different partial bitstreams are used depending upon system conditions. This information can be later used for further PR design optimisation such as region partitioning [3] and improved performance through configuration prefetching [15].

F. UART

The system can interface with an external host using a serial interface. This RS-232 interface can be used to transfer partial bitstreams into external memory and for issuing commands from a host PC. A simple serial interface has several advantages including the lack of a special driver being needed for communication. In a completely self-contained system, there is no need for this interface, as the bitstreams can be fetched internally from non-volatile memory during the preparation phase. The commands available for host system control are listed in Table I.

G. ICAP Controller

The detailed architecture of the ICAP controller is shown in Fig. 3. It consists of an asynchronous FIFO, clock manager (MMCM), ICAP control state machine (ICAP SM) and the ICAP hard-macro. The asynchronous FIFO is used to temporarily store the partial bitstreams from external memory, before sending to the ICAP macro. The system is able to achieve high throughput due to the presence of the asynchronous FIFO that has different read and write clock frequencies. The depth of this FIFO can be configured to achieve better performance. Operation of read and write ports is managed using the *fifo_full*

TABLE I
SUPPORTED HOST COMMANDS.

Command	Action
SRST	Soft Reset: Reset all logic except the memory controller
SLEN	Set byte transfer length for DMA
SADR	Set the starting address for DMA
CMOD	Command mode: Disable DMA controller
DMOD	Data mode: Enable DMA controller
PICP	Start reconfiguration
CINT	Reconfiguration using the pointers from config. buffer
RST1	Read statistics register 1
RST2	Read statistics register 2
NCON	Number of times the specified partial bitstream is used
SCFQ	Set ICAP clock frequency

and *fifo_empty* signals, synchronised with the write and read clock domains, respectively. In our design, the write clock frequency is equal to the DDR controller clock frequency and the write data width is 256 bits. Whenever valid data emerges from the DDR memory, it is stored in this FIFO. The read clock frequency of the FIFO is equal to the ICAP controller clock frequency, and the read width is set to the maximum allowable 32 bits for performance.

An MMCM (mixed-mode clock manager) is used to derive the required ICAP clock frequency from the on-board clock source. The MMCM output clock frequency is set to 100 MHz, which is the maximum frequency recommended by Xilinx.

During PR, we are only interested in writing into the configuration memory, so the *read/write* port of ICAP is permanently grounded. As we only support writing to the ICAP, the controller is more compact than many existing designs. The write operation to the ICAP is managed by a state machine that continuously senses the *fifo_empty* signal of the asynchronous FIFO. Whenever the *fifo_empty* signal is de-asserted, it indicates valid data is available in the FIFO. The state machine asserts the *read_enable* signal of the FIFO and after one clock cycle, asserts the *icap_enable* signal of the ICAP. When the *empty* signal becomes high, the *read_enable* and *icap_enable* signals are de-asserted. In order to minimise resource utilisation, no counters are implemented in the ICAP controller to track the number of bytes written to the configuration memory. The DMA controller must ensure that all required configuration bytes are read from external memory.

The *fifo_full* signal is used by the DMA controller. This signal is asserted when half the FIFO is filled. Whenever the DMA controller senses this signal is high, it stops issuing memory read commands. This signal ensures that no buffer overflow occurs.

H. Using the Dynamic Reconfiguration Port (DRP)

Researchers have tried over-clocking the ICAP to achieve higher performance. According to Xilinx, the maximum clock frequency at which ICAP should operate is 100 MHz. It has been reported that the ICAP can run at up to 550 MHz [12]. The drawback is that this maximum clock frequency depends upon the device speed grade, manufacturing variability, and detailed custom placement and routing. A clock frequency

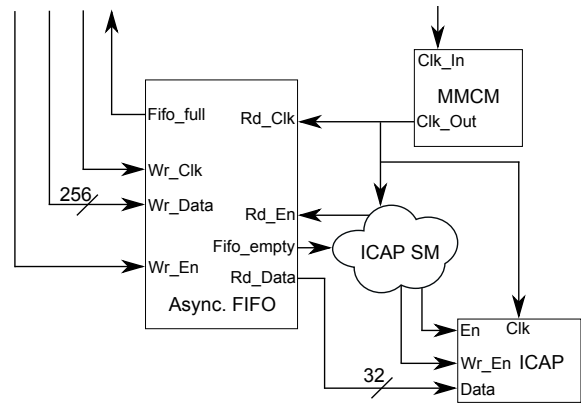


Fig. 3. ICAP controller architecture.

TABLE II
RESOURCE UTILISATION.

Module	Registers	LUTs	BRAMs	Max. Freq.
ICAP Controller	74	38	8	516
DMA Controller	598	548	0	265

which is suitable for one device may not be suitable for another device, even with the same speed grade, and Xilinx does not guarantee proper ICAP operation above 100 MHz. Clock generating circuits cannot be modified using PR, since Xilinx requires that all the clock modifying components such as phase locked loops (PLLs) and digital clock managers (DCMs) reside in the static region. In order to overcome this issue, we make use of a feature available in the Virtex-6 MMCM known as the dynamic reconfiguration port (DRP).

The DRP makes it possible to configure the output frequency of the MMCM at runtime. This is achieved by modifying the internal registers of the MMCM using the DRP. The system starts operation at 100 MHz. Subsequently, the operating frequency is increased, until the reconfiguration process fails. The operating frequency is set to be below the failing frequency.

Presently the clock frequency needs to be manually tuned by issuing commands from the host system. In future designs, we will propose ways to automate this process.

V. RESULTS

The ICAP controller design was synthesised using Xilinx ISE 13.3, and was implemented using Xilinx PlanAhead 13.3. In addition to the system discussed in Section IV, two partially reconfigurable modules were also implemented in order to test the performance and validate functionality as shown in Fig. 4. The modules are assigned to two separate regions, one large and the other small. The system was hardware validated by testing it on a Xilinx ML605 evaluation board, which contains a Virtex 6 XC6VLX240T FPGA. The resource utilisation for the two major modules in the design is given in Table II. The ICAP controller is able to achieve a maximum frequency of 516 MHz when implemented as a standalone module and the complete system is able to run at 200 MHz.

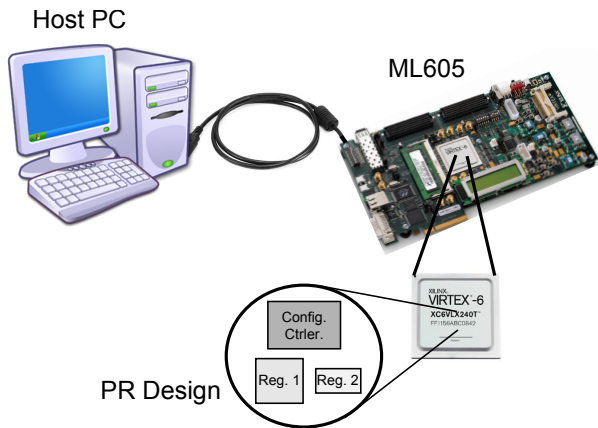


Fig. 4. Controller Performance Test Setup.

Table III shows the performance of the ICAP controller as well as overall system performance. These values are calculated with the help of the statistics counters present in the system. The theoretical maximum performance is 400 MB/s, based on the 100 MHz clock and 32-bit ICAP width. Total performance is slightly less than this due to the initial memory access latency and because DDR read operations are 64-byte aligned. As the bitstream size increases, the initial latency becomes negligible compared to the total reconfiguration time, and hence the throughput increases. From Table III it can be seen that the controller takes about 633 microseconds to configure a 400 CLB region (253096 bytes), an improvement of 44 times over the Xilinx XPS_HWICAP, which would require 27.8 milliseconds. Our controller would take a few milliseconds for a near complete FPGA reconfiguration rather than hundreds of milliseconds.

The maximum throughput and resource utilisation of some other ICAP controller implementations are shown in Table IV. Our implementation performs better than all these implementations, and is also highly compact.

In addition, we tried to improve the ICAP performance by overclocking it using the DRP feature of the MMCM. The ICAP controller was able to successfully reconfigure the system up to 210 MHz. Presently we need to manually verify that there are no configuration errors by checking the functionality of the reconfigured modules. More thorough checking would require ICAP read capability, which we will investigate in future work. The overall system performance for different clock frequencies is given in Fig. 6. Above 210 MHz,

TABLE III
BITSTREAM SIZE AND SYSTEM PERFORMANCE.

Bitstream Size (Bytes)	Recon. Time (us)	ICAP Throughput (MB/S)	Total Throughput (MB/S)
5000	12.86	395.50	388.60
12568	31.75	399.94	395.84
63456	159.00	399.79	399.23
126912	317.50	400.00	399.70
253096	633.00	399.96	399.80

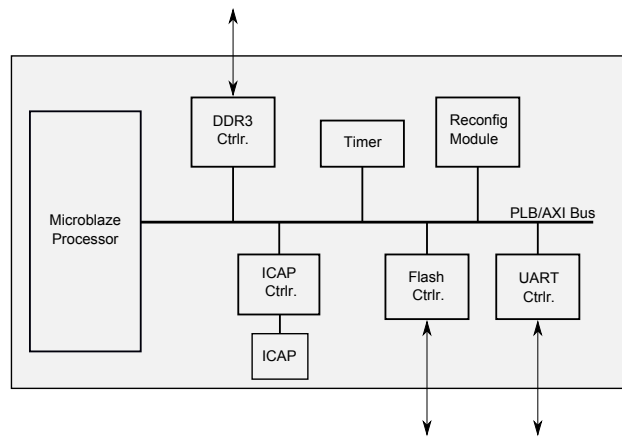


Fig. 5. Processor based PR System.

TABLE IV
PERFORMANCE COMPARISON OF ICAP CONTROLLER IMPLEMENTATIONS

Implementation	Throughput (MB/S)	Registers	LUTS	BRAMs
[Liu et al. 2009] [9]	235.20	1083	918	2
[Claus et al. 2008] [16]	295.40	NA	NA	NA
[Manet et al. 2008] [17]	353.20	NA	NA	NA
[Liu et al. 2009] [9]	371.40	963	469	32
[Liu et al. 2009] [11]	392.74	367	336	0
Xilinx (PLB) [7]	8.48	746	799	1
Xilinx (AXI) [18]	9.10	477	502	1
Proposed (with DMA)	399.80	672	586	8

no reconfiguration occurs, and above 300 MHz, initiating a reconfiguration freezes the whole FPGA. At 210 MHz, the overall throughput is 838.55 MB/S, which is more than double the throughput at 100 MHz, resulting in a corresponding decrease in reconfiguration time.

In order to compare the performance of the widely used Xilinx ICAP controllers, a typical processor-based PR system was also implemented as shown in Fig. 5. This system consists of a MicroBlaze soft processor, a DDR3 memory controller, the ICAP controller, a timer, Xilinx flash controller, UART controller, and a reconfigurable module. All the peripheral devices were initially connected to a 64-bit wide PLBv46 bus. The partial bitstreams can be stored either in DDR3 memory or in the flash memory. Partial bitstreams are transferred to the DDR3 memory using the UART interface and written to the flash memory using a host flash memory writer. The timer peripheral is used to determine the time required for reconfiguration. The system runs at 100 MHz with the instruction as well as data memory implemented in internal BRAMs. Software for performing the PR operation was written in the C language using the Xilinx Software Development Kit (SDK), and the hardware platform was implemented using Xilinx Embedded Design Kit (EDK) 13.3 and PlanAhead 13.3. The low-level routines for controlling the ICAP controller as well as flash memory are taken from Xilinx standard libraries.

Reconfiguration commands are issued from the host system using the UART interface. If the partial bitstreams are stored in DDR3 memory, they are transferred using the UART interface

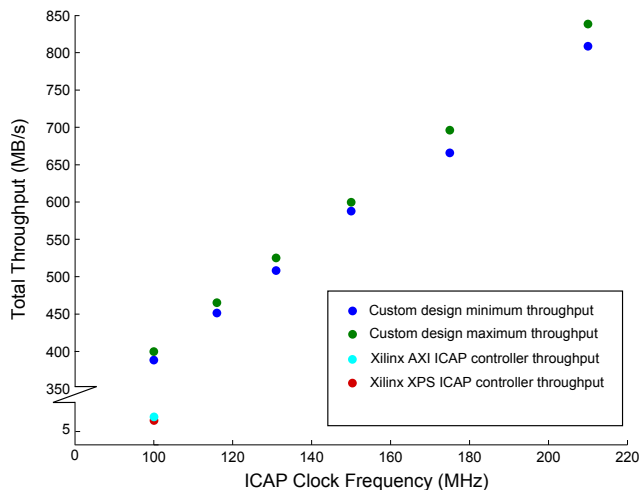


Fig. 6. Frequency vs Total Throughput.

by calling a routine. When the processor receives a reconfiguration command, it resets the performance measurement timer and invokes appropriate routines to transfer the partial bitstream to the ICAP controller depending upon its storage location. Once the reconfiguration operation is completed, the timer is halted and the value stored in it is read. The timer reports the total number of clock cycles required for the operation, and from this the throughput can be determined. For the PLB system, Xilinx's XPS_HWICAP [7] was used as the ICAP controller. When the bitstreams are stored in flash memory, the reconfiguration throughput is only 0.47 MB/s and when stored in the DDR3 memory, the throughput is 8.4 MB/s. These values prove that present processor-based ICAP controllers are unsuitable for time-critical reconfiguration scenarios.

The same experiment was repeated using the latest AXI-bus based design. In this system, the DDR3 controller is connected to an AXI4 bus and other peripherals to AXI4-lite bus. The ICAP controller used in this experiment is the AXI_HWICAP [18]. When using the AXI-bus, system performance is slightly improved. The reconfiguration throughput while using the flash is 0.49 MB/s, and using the DDR3 memory to store the bitstreams gives 9.1 MB/s. These values are still well below what is possible, as we have shown with our design.

VI. CONCLUSION AND FUTURE WORK

In this paper we presented the design of a high speed reconfiguration controller, which minimises reconfiguration time and provides additional features for adaptive systems implementation. The design achieves near to the theoretical maximum performance of the Xilinx ICAP primitive. The result is that reconfiguration time is reduced by an order of magnitude compared to the standard vendor-advised approaches. Furthermore, by using the DRP feature of the clock generator, the design is able to achieve even higher performance, with the throughput being adjustable.

In the future, we will investigate how maximum throughput can be determined automatically. For the design presented in this paper, we use a write-optimised data path that consumes

minimal resources. Tuning throughput would require the ability to read from the ICAP to verify valid reconfiguration. We also plan to extend the application of this platform for processor based designs.

Finally, we are releasing this design in the public domain, so it can be used by researchers intending to incorporate PR into their systems, in the hope that it might spur further adoption of this key capability of FPGAs.

REFERENCES

- [1] J. Lotze, S. Fahmy, J. Noguera, B. Ozgul, L. Doyle, and R. Esser, "Development framework for implementing FPGA-based cognitive network nodes," in *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM)*, 2009.
- [2] ICAP controller source code. [Online]. Available: <http://www.github.com/archntu/prcontrol.git>
- [3] K. Vipin and S. A. Fahmy, "Efficient region allocation for adaptive partial reconfiguration," in *Proceedings of the International Conference on Field Programmable Technology (FPT)*, 2011.
- [4] Z. Li and S. Hauck, "Configuration compression for virtex FPGAs," in *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2001.
- [5] K. Vipin and S. A. Fahmy, "Architecture-aware reconfiguration-centric floorplanning for partial reconfiguration," in *Reconfigurable Computing: Architectures, Tools and Applications – Proceedings of the International Symposium on Applied Reconfigurable Computing (ARC)*, 2012, pp. 13–25.
- [6] *DS280: OPB HWICAP*, Xilinx Inc., July 2006.
- [7] *DS586: XPS HWICAP*, Xilinx Inc., July 2010.
- [8] C. Claus, F. H. Muller, J. Zeppenfeld, and W. Stechele, "A new framework to accelerate Virtex-II Pro dynamic partial selfreconfiguration," in *Proceedings of IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, 2007.
- [9] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time partial reconfiguration speed investigation and architectural design space exploration," in *Proceedings of International Conference on Field Programmable Logic and Applications (FPL)*, 2009.
- [10] M. Hubner, D. Gohringer, J. Noguera, and J. Becker, "Fast dynamic and partial reconfiguration data path with low hardware overhead on Xilinx FPGAs," in *Proceedings of IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*, 2010.
- [11] S. Liu, R. N. Pittman, and A. Forin, "Minimizing partial reconfiguration overhead with fully streaming DMA engines and intelligent ICAP controller," Microsoft Research, Tech. Rep. MSR-TR-2009-150, Sept. 2009.
- [12] S. G. Hansen, D. Koch, and J. Torresen, "High speed partial run-time reconfiguration using enhanced ICAP hard macro," in *Proceedings of IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*, 2011.
- [13] K. Papadimitriou, A. Dollas, and S. Hauck, "Performance of partial reconfiguration in FPGA systems: A survey and cost model," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 4, no. 4, pp. 36:1–36:24, Dec. 2011.
- [14] *UG86: Xilinx Memory Interface Generator (MIG) User Guide*, Xilinx Inc., Sept. 2010.
- [15] S. Hauck, "Configuration prefetch for single context reconfigurable coprocessors," in *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 1998.
- [16] C. Claus, B. Zhang, W. Stechele, L. Braun, M. Hubner, and J. Becker, "A multi-platform controller allowing for maximum dynamic partial reconfiguration throughput," in *Proceedings of the IEEE International Conference on Field Programmable Logic and Applications (FPL)*, 2008.
- [17] P. Manet, D. Maufroid, L. Tosi, G. Gailliard, O. Mulertt, M. D. Ciano, J. D. Legat, D. Aulagnier, C. Gamrat, R. Liberati, V. L. Barba, P. Cuvelier, B. Rousseau, and P. Gelineau, "An evaluation of dynamic partial reconfiguration for signal and image processing in professional electronics applications," *EURASIP Journal on Embedded Systems*, vol. 2008, pp. 1–11, 2008.
- [18] *DS817: AXI HWICAP*, Xilinx Inc., June 2011.