ICAP-I: A Reusable Interface for the Internal Reconfiguration of Xilinx FPGAs

Victor Lai and Oliver Diessel

School of Computer Science and Engineering,
The University of New South Wales, Sydney, NSW 2052 AUSTRALIA
{victorl, odiessel}@cse.unsw.edu.au

***Abstract*—Application circuits configured on Xilinx Virtex series FPGAs are able to reconfigure the FPGA at run time using the on-chip ICAP. Traditional methods of accessing the ICAP using OPB-based and PLB-based schemes are unnecessarily complex and rarely reused. In this study, a new interface for accessing the ICAP is introduced. The interface is easy to use, it can readily be integrated to different systems, it is customizable, and it is reusable. A demonstration is crafted to show the use of the new interface. Performance results for the new interface and two existing OPB-based and PLB-based methods are compared.**

I. Introduction

One of the most intriguing features of reconfigurable systems is their ability to dynamically change their structure and functionality in order to better suit the task at hand. For Xilinx Virtex series FPGAs, those changes to structure and functionality are made by loading configuration bitstream data through one of several configuration ports [11]. External configuration ports such as the SelectMAP and JTAG interfaces are typically driven by an external device such as a PC. In contrast, the internal configuration access port (ICAP) can be directly accessed by application circuits configured on the FPGA, allowing them to change their own structures and functionalities at run time. To achieve this, different circuits with different functionalities are loaded onto the FPGA when needed by those applications. This has the potential of allowing applications to have a smaller operational area on the FPGA and of consuming less power.

When using Xilinx's Early Access Partial Reconfiguration (EAPR) toolflow, internal reconfiguration of an FPGA involves generating alternative circuit configurations as a set of Partial Reconfigurable Modules (PRMs) that can be swapped into predefined Partial Reconfigurable Regions (PRRs) at runtime [12]. A storage device is needed to hold the configuration data (bitstreams) of the PRMs which can be read by the application when needed. Once read, the bitstreams are then forwarded and written into the ICAP, where those bitstreams update the circuits inside the PRRs, reflecting the loading of new functionalities onto the FPGA.

Traditionally, access to the ICAP is made possible by using the OPBHWICAP peripheral attached to the On-Chip Peripheral Bus (OPB, see [2]) or by using a Xilinx Intellectual Property Interface (IPIF) peripheral attached to the Processor Local Bus (PLB, see [1]), with the operations of the ICAP controlled by software running on a processor core on the FPGA. However, the use of the OPB and PLB can take up relatively large amounts of resources; the need to integrate several components makes usage relatively complex; and the developed ICAP peripherals, such as the OPBHWICAP, were designed to be used with particular bus systems in each case. These peripherals cannot therefore be reused with other bus types or systems that do not use buses at all.

TABLE I
Features Found in Previous Studies

Features	Designs				
	Sedcole [2]	Claus [1]	Cuoccio [3]	Bok [4]	ICAP-I
Storage device required	Yes	Yes	Yes	Yes	Yes
CPU core required	Yes	Yes	Yes	No	No
Bus system used	OPB	PLB	Any	None	Any
DMA to Storage Device	No	Yes	Yes	Yes	Yes
Readback supported	Yes; but slow	Yes	Yes	No	Yes

Table 1 summarizes the features of typical reconfigurable platforms in previous studies that have made use of or provided access to the ICAP. Sedcole et. al. [2] and Claus et. al. [1] both investigated feasible platforms for reconfiguration using the ICAP with OPB-based and PLB-based systems respectively. Cuoccio et. al. [3] tried to generalise the OPB-based and PLB-based platforms found in [1] and [2] by providing a toolflow that incorporated these systems. Bok et. al. [4], while attempting to discover an efficient way of providing error checking to FPGA configuration bitstreams during reconfiguration, used a much simpler controller that had limited access to the ICAP. The commonalities of these studies are: 1) their use of a storage device to provide bitstream data to the ICAP at run time; 2) the need for a controller to operate the ICAP on the FPGA; and 3) the ICAP and the storage device are both physically close to allow better reconfiguration performance. We feel that the platforms in Sedcole [2] and Claus [1] are unnecessarily complex to implement (i.e. they require too many components) simply for the sake of accessing the ICAP alone. In contrast, we feel that the controller described in Bok [4] is too simplistic and is subsequently not sufficiently general to provide all the features that a designer may require (e.g. readback from the ICAP).

In this study, a new interface for using the ICAP is introduced. The interface implements a wrapper that provides a new and easy-to-use interface for accessing the ICAP without needing a bus or processor core. However, the interface can still be integrated with OPB, PLB or other bus systems if necessary. The proposed ICAP interface (ICAP-I) is specifically designed to be reusable, simple to use, flexible, and conservative in its use of resources. Configuration bitstream data are pregenerated and stored in a storage device. When needed, those configuration bitstream data are read out, and written into the ICAP. Section 2 introduces the design of our proposed ICAP-I. Section 3 introduces a demo developed to show the ease of using ICAP-I and compares the performance of ICAP-I with previous methods. Section 4 concludes this study with a summary of the main contributions and directions for future work.

II. Design

This section introduces the new interface for accessing the ICAP. The structure of our interface is described and discussed in detail.

ICAP-I, illustrated in Fig. 1, is a set of VHDL modules, which describe logic blocks that can be instantiated and included in a system. These blocks provide a set of predefined, simple and efficient mechanisms for accessing the ICAP.

A. Storage Device Module

The Storage Device Module provides configuration data for the ICAP and stores configuration data that is read from the ICAP while not preventing the application(s) from using the Storage Device for other purposes.

The Storage Device Controller Module provides signals for operating a storage device. A Storage Device Controller Module can be written for any device that has the capabilities of reading or storing data. This module can be replaced when a different device is used. Examples of storage devices are BRAM, SD RAM, flash memory or even an ethernet port. A Storage Device driver module needs to support both device-generic operations (e.g. read or write) and device-specific operations (e.g. erase for flash memory). These operations are defined inside a VHDL package and can be imported by the application.

The Storage Device Scheduler mediates the data flow into and out of the Storage Device. An application may use a Bulk Transfer Unit (BTU), which is included for direct transfers between the ICAP and the storage device. This is done by simply specifying the amount of data that needs to be transferred and the start address of this data in the storage device. The Storage Device Scheduler allows the application to share access to the storage device while maintaining operations handled by the BTU. Within the current implementation, the scheme for accessing the storage device is to share this access equally between the BTU and the application. The Storage Device Scheduler is required to support device-generic operations to control the data flow in and out of the Storage Device Module. Device-specific operations are passed directly from the application to the Storage Device Controller Module.

The Storage Device Module can be used by instantiating it inside the application and is meant to operate together with an instance of the ICAP IF Module. Generic variables can be altered at the time of instantiation to modify, to keep or to leave out the various internal components of the Storage Device Module (e.g. the BTU can be left out entirely if it is not needed).

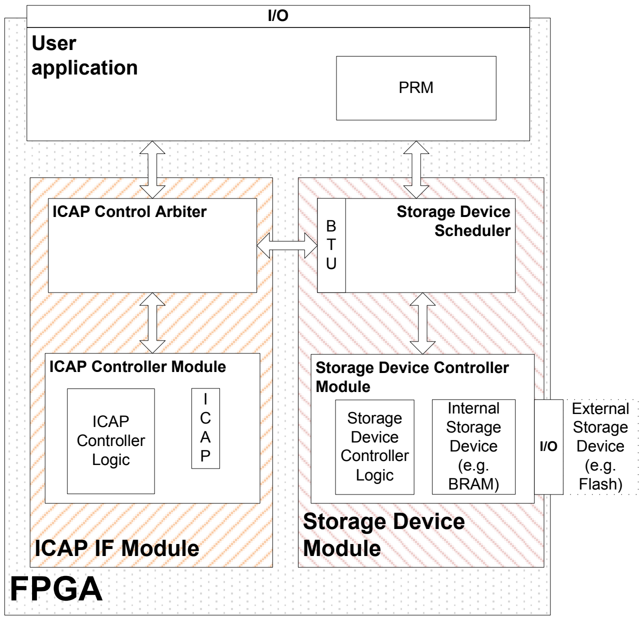

Fig. 1. ICAP-I

B. ICAP IF Module

The ICAP IF Module accepts and processes read and write request operations for the ICAP from the application and from the (BTU inside) the Storage Device Module.

The ICAP Controller Module provides signals for operating the ICAP. This module should handle lower level signaling for operations such as nop, abort, read and write. Read and write operations are considered to be device-generic operations while nop and abort are considered to be device-specific operations for the ICAP. These operations are declared as operands inside a VHDL package that can be included by the application and the Storage Device Module.

The ICAP Control Arbiter mediates requests from the application and the Storage Device Module for access to the ICAP. The ICAP Control Arbiter is required to be aware of device-generic operations to direct the flow of data between the ICAP, Storage Device and the application. In contrast, device-specific operands are passed directly to the ICAP Controller Module without interpreting them.

For internal reconfiguration of the FPGA, configuration bitstreams are loaded into the ICAP while the application is executing. These configuration bitstreams may be generated on the fly while the application is executing or are pregenerated prior to the execution of the application. The Storage Device provides ICAP-I a source for pre-generated bitstreams while the application provides the ICAP-I a source for bitstreams that are generated on the fly. In the case of compressed bitstreams, the application is required to read the compressed bitstreams from the Storage Device first, in order to decompress the data before sending it to the ICAP via the ICAP IF Module.

C. *Discussion*

The interface allows for any Storage Device with arbitrary data and address width to be connected to the embedding system by attaching to the Storage Device Scheduler. One Storage Device may be swapped for another without the need to respecify the rest of the design. However, the application will still need to be aware of changes in device-specific operations when the Storage Device changes.

Similarly, the current interface allows for any ICAP device with arbitrary data and address width. However, only 8-bit and 32-bit devices currently exist.

To gain access to the ICAP, the application is only required to instantiate the ICAP IF Module with the Storage Device Module being an optional component. At the time of instantiation, the application can change the properties of the internal components of the ICAP IF Module and the Storage Device Module using generic variables. The resources required for the needed components are then automatically generated by the synthesis tool. This allows the application designer to easily choose a bare minimum of the components and resources that are needed for a particular design.

The ICAP-I in the present study is capable of being connected to or of being used by any application or system. However, because ICAP-I is designed to be light weight, it can only be connected to a single application or system at a time. In cases where multiple applications or systems want to gain access to the ICAP, an additional arbiter (e.g. a bus system) will be needed at the application layer to arbitrate between multiple access requests by the different applications or systems.

A significant problem for the current ICAP-I design is its ability to transfer data in an efficient way. Currently, the implementation requires a minimum of two cycles for transfers between the ICAP and the Storage Device; at least one cycle for an operation (i.e. read or write) on the ICAP and at least one cycle for an operation on the Storage Device. The ICAP and the Storage Device must both finish their current operation before the next operation can begin. This is inefficient because the Storage Device and the ICAP are separate entities and they should be able to operate independently.

A solution to this problem is to pipeline (i.e to latch) the control and data bits that are in flight between the Storage Device Module and the ICAP IF Module. However, an additional resource hazard detection unit will need to be implemented within the Storage Device Scheduler to ensure that access to the Storage Device by the application does not begin until a previous (i.e. a pipelined) transfer of a read from the ICAP to the Storage Device has finished. The internal states of the Storage Device Scheduler will also need to be modified to allow simultaneous requests to both the ICAP IF Module and the Storage Device Controller Module.

III. Results

This section compares the performance of the current implementation of the ICAP-I to previous OPB and PLB-based ICAP implementations. Whilst we are able to report our findings for the ICAP-I, it has been difficult to locate papers that report a breakdown of the amount of resources for comparable systems. Thus we were unable to obtain essential information for comparing the ICAP in the OPB and PLB-based implementations directly. The results reported here are approximations obtained for the resources used by the components of these systems as sourced from [2, 6-10]. A simple demonstration, which shows the ease of using the ICAP-I for internal reconfiguration on a Virtex4 series FPGA, is also described.

A. *Demonstration*

Our demonstration is hosted on a Xilinx Virtex-4 FX LC Evaluation Kit from Avnet Memec [13]. The flash memory chip is used as an example of a storage device. The flash memory is preloaded with four different partial bitstreams for Partial Reconfiguration Region 1 (PRR1) using a design that is able to communicate to the PC via a parallel port. The FPGA is then configured with the design in Fig. 2. Buttons B0 to B3 each trigger the loading of one of the four different types of bitstreams. Switches S0 to S3 are used as a 4-bit input to the module currently loaded in. Patterns in the output, due to the effects of the 4-bit inputs and the different functionalities of the various partial bitstreams, can be observed on the outputs LED0 to LED3.

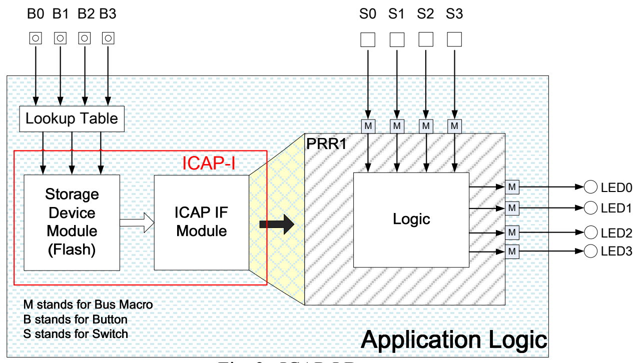

Fig. 2. ICAP-I Demo

B. *Performance*

For the present study, the ICAP-I was implemented on a Xilinx Virtex4 FX12 FPGA that has an ICAP with a data width of 32-bits. The results of an implementation to perform internal reconfiguration on an FPGA from the present study (ICAP-I) and the results from two previous implementations (PLB and OPB) [1] can be seen in Table 2.

The amount of resources required for each implementation that is reported below is approximated from the sum of the

(Xilinx's XST) synthesis results of different core components gathered from several sources. The OPB implementation, which makes use of the OPBHWICAP peripheral from Xilinx and a MicroBlaze softcore processor, uses at least 639 Flip-flops, 993 Look-up Tables (LUTs) and 1 BRAM. The PLB implementation required more resources, taking up an area equivalent to at least 2427 Flip-Flops, 2608 LUTs and 2 BRAMs, due primarily to the PowerPC core (as estimated using Xilinx's Floorplanner). In contract the ICAP-I as a whole uses a total of 208 Flip-Flops, 462 LUTs and 0 BRAMs. ICAP-I can therefore make do with considerably less resources than the previous PLB and OPB implementations. However, this is largely due to the reduction in resources previously taken up by the CPU cores.

The OPB implementation was able to achieve a throughput of 5MB/s read/write to the ICAP [1]. The PLB implementation achieved a much higher throughput of 95MB/s read/write to the ICAP. This was close to the theoretical maximum throughput (100MB/s) of the ICAP with 8-bits data width. Our interface system (excluding the Storage Device Controller Module) is currently capable of supporting up to 180MB/s throughput using a 32-bit interface. However the system is currently constrained by the performance of the Storage Device (flash memory) used, which only has a throughput of 29MB/s (using the AT49BV322A chip [10]). This means that our system as a whole currently only provides 29MB/s throughput to the ICAP. The storage device used can thus significantly impact the performance of ICAP-I. In future studies, we aim to improve the performance of the ICAP-I, by reducing the critical path delay and by pipelining its internal operations to provide better throughput. We expect to obtain further significant improvement by porting the system to another development board with a faster storage device. The maximum theoretical throughput with a 32-bit ICAP interface is 400MB/s.

TABLE 2
RESULTS FOR THE ICAP-I FROM THE PRESENT STUDY, PLB AND OPB IMPLEMENTATIONS FROM [1]

Components	OPB Implementation				
	Flip-Flops	LUTs	BRAMs	Frequency (MHz)	Throughput (MB/s)
OPB Interface[1]	266	273	1	250	5
MicroBlaze [6]	373	720	8	?	
	PLB Implementation				
PLB Interface[2]	699	880	2	241	95
PPC405 CPU Core[3]	1728	1728	?	?	
	ICAP-I				
ICAP-I[4]	177	303	0	90	180
SD Controller Module (Flash)	31	159	0		29

[1] Estimated from OPB arbiter with 2 Masters and a processor interface [7], OPBHWICAP peripheral with OPB IPIF [1] and minimum resources configuration for 1 OPB IPIF for the storage device [8] but controller is omitted (not reported in [1]).

[2] Estimated from PLB arbiter with 2 Masters [6], ICAP Controller with the PLB IPIF [1] and minimum resources configuration for 1 PLB IPIF for the storage device [10] but controller is omitted (not reported in [1]).

[3] Estimated from the CLBs omitted on a Virtex4 FX12 FPGA.

[4] Excludes the SD Controller Module.

IV. CONCLUSIONS

Previous methods for accessing the ICAP in order to perform internal reconfiguration on a Xilinx FPGA have been complex, one-off designs which required the integration of many components.

The study described in this paper introduces a new interface for the ICAP that is lightweight, is easy to use, can be easily integrated into any application or system, and can be customized for the needs of specific applications such that the resources used are minimized. The VHDL specification of ICAP-I takes advantage of Modular I/O to facilitate module reusability. The VHDL specification also takes advantage of Conditional Code Generation to allow users to easily choose the essential components from the ICAP-I for their designs. ICAP-I uses fewer resources than previous OPB and PLB implementations. Although ICAP-I beats existing OPB and PLB implementations on throughput to the ICAP, the end-to-end throughput including access to the storage device is still not as high as it could be due to limitations of the storage device tested, the need to improve the critical path of the design, and the need to pipeline accesses.

Future work will aim to demonstrate the interface with a variety of storage devices and to make ICAP-I publicly available.

REFERENCES

[1] C. Claus, F. H. Muller, J. Zeppenfeld & W. Stechele. A new framework to accelerate Virtex-II Pro dynamic partial self-reconfiguration. Parallel and Distributed Processing Symposium. March 2007, pp. 1 - 7.

[2] P. Sedcole, B. Blodget, T. Becker, J. Anderson & P. Lysaght. Modular dynamic reconfiguration in Virtex FPGAs. *Computers and Digital Techniques, IEE Proceedings*. May 2006, pp. 157 – 164.

[3] A. Cuoccio, P. R. Grassi, V. Rana, M. D. Santambrogio & D. Sciuto. A Generation Flow for Self-Reconfiguration Controllers Customization. *Electronic Design, Test and Applications*. Jan 2008, pp. 279 – 284.

[4] K. Bok , R. Chaves, G. Kuzmanov, L. Sousa & A. Genderen. Dynamic FPGA Reconfigurations with Run-Time Region Delimitation. *Proceedings of the 18th Annual Workshop on Circuits, Systems and Signal Processing (ProRISC)*. 2007, pp. 201 – 207.

[5] P. Huerta, J. Castillo, J. I. Martines & V.Lopez. A MicroBlaze Based Multiprocessor SoC. *WSEAS Transactions on Circuits and Systems*. 2005.

[6] Xilinx Inc., Processor Local Bus (PLB) v3.4 Data Sheet DS400. April 2009.

[7] Xilinx Inc., PLB Arbiter v1.02e Data Sheet DS469. September, 2005.

[8] Xilinx Inc., OPB IPIF v3.01c Data Sheet DS414. December, 2005.

[9] Xilinx Inc., PLB IPIF v2.02a Data Sheet DS448. April, 2005.

[10] ATMEL Corporation, AT49BV322A-70TI - 32-megabit (2M x 16/4M x 8) 3-volt Only Flash Memory. 2004.

[11] Xilinx Inc., Virtex-4 FPGA Configuration User Guide v1.11 UG071. June, 2009.

[12] Xilinx Inc., Early Access Partial Reconfiguration User Guide for ISE 8.1.01i v1.1 UG208. March, 2006.

[13] Avnet Memec.Virtex-4 Fx12 LC Development Board User's Guide v1.1. May, 2006.