# Dynamic FPGA Reconfigurations with Run-Time Region Delimitation

Kees van der Bok [1],    Ricardo Chaves[2,1],

Georgi Kuzmanov[1],    Leonel Sousa[2],    Arjan van Genderen[1]

[1]Computer Engineering Lab, EEMCS, TUDelft    [2] Instituto Superior Técnico/INESC-ID

http://ce.et.tudelft.nl/    http://sips.inesc-id.pt

c.vanderbok@student.tudelft.nl    ricardo.chaves@inesc-id.pt

*Abstract*— **This paper presents the implementation of a dynamic FPGA partial reconfiguration system, with on-the-fly writing region delimitation. Considering that in the reconfiguration bitstreams can become corrupt or can be obtained from a non secure source, delimiting the region where they can be uploaded becomes necessary. This novel concept of run-time region delimitation will allow for a much safer, while looser, usage of IPCores and co-processors. In the same manner as a user uploads software applications to a specific region in the memory, the work presented in this paper, will allow for hardware cores to be uploaded to specific regions of dynamic hardware space, safely. The computational structures presented in this paper, used for the dynamic partial reconfiguration with region delimitations, have been evaluated for a Virtex II Pro 30. Implementation results suggest that the dynamic reconfiguration system with region delimitation can be implemented with a device occupation of less than 5% without performance degradation.**

*Keywords*— **Dynamic reconfigurations, run-time region delimitation.**

## I. Introduction

Recent FGPA allow for partial reconfiguration of one of its regions, while the rest remain active and working; this is called active dynamic reconfiguration. This functionality allows for a very wide range of computational units to be used on a single device. Whenever a given computational core is required, it just has to be downloaded from a storage device and uploaded to the FPGA device [1].

In this paper we propose a solution for secure on-chip partial reconfiguration. The solution is composed of two components namely a unit responsible for the partial reconfiguration process and another one capable of checking the region of the FGPA being reconfigured. The unit controlling the reconfiguration process is referred to as the Partial Reconfiguration Manager (PRMU). This unit retrieves the reconfiguration bitstream from a storage device and send it to the FPGA reconfiguration interface, the Internal Configuration Access Port (ICAP). The reconfiguration data can be stored either on-chip or off-chip. The ICAP is the interface that enables on-chip access to the re-

configuration logic, and is supported by most of the Xilinx VIRTEX FPGAs.

Considering that in the reconfiguration bitstreams can become corrupt or can be obtained from a non secure source, delimiting the region where they can be uploaded becomes necessary. Moreover, critical failures can occur to the static part of a given architecture on the FPGA, when the dynamic reconfiguration is performed without region control. This paper also proposes a novel design capable of interpreting the reconfiguration bitstream and identify the FPGA regions being targeted for reconfiguration. In case an invalid region is target to be reconfigured, an abort signal is generated in order to halt the invalid reconfiguration process. This concept of run-time region delimitation will allow for a much safer, while looser, usage of IPCores and co-processors. In the same manner as a user uploads software applications to a specific region in the memory, the work presented in this paper, will allow for hardware cores to be uploaded to specific regions of dynamic hardware space, with high performance and safely. Implementation results for the VIRTEX technology on a VIRTEX II Pro 30 FPGA suggest:

- An occupation of less than 5% of the available reconfigurable hardware resources.
- Reconfiguration throughputs of 400 Mbit/s, limited by the ICAP interface.
- Secure loading of cores into specific regions of the FPGA.

The paper is organized as follows: Section II presents an overview on partial reconfiguration of FPGA devices and reconfiguration bitstream characteristics. In Section III the structures proposed for the partial reconfiguration with region delimitation are described. Implementation results and a comparison with related art is presented in Section IV. Section V concludes this paper with some final remarks.

## II. Partial Reconfiguration Background

The current generation of reconfigurable devices has

the capability to reconfigure part of its available resources while, at the same time, the remaining resources of the device continue to be active and performing computation. This operation is called dynamic partial reconfigurability. In this section the partial reconfiguration of the Xilinx FPGA devices are described.

## A. Module Based Partial Reconfiguration

The method to reconfigure a core or region on the FPGA is designated by Module Based Partial Reconfiguration. The original intention of this methodology is to enable the well known principle of functional decomposition. Each module can be designed, verified and tested separately and merged afterwards.

Although the module based design methodology was traditionally intended for the described purpose, it has turned out to be applicable to re-configurable designs as well see [2], [3]. A module based partial reconfigurable design is composed of modules which are either static i.e. not reconfigurable or dynamic i.e. reconfigurable. Dynamic modules can be replaced by other modules by means of partial reconfiguration. So module based partial reconfiguration enables swapping of hardware on a module basses, which is in fact hardware task swapping.

Each module of the modular design is statically assigned to a specific area of the FPGA. In the design flow this phase is referred to as initial budgeting. A design can contain multiple reconfigurable and non-reconfigurable modules. The most simple reconfigurable design is composed of one static and one dynamic module. The units discussed in this paper, capable of performing secure partial reconfiguration, need to be instantiated as a static module. The proposed solution supports multiple reconfigurable modules.

Several interfaces are made available to reconfigure the FPGA device, namely: JTAG, serial slave and select map, and ICAP. The ICAP is the only interface that allows on-chip reconfiguration, i.e. reconfiguration controlled from within the chip, and is thus the chosen reconfiguration interface.

## B. Internal Configuration Access Port

The ICAP interface is a subset of the selectmap interface, selectmap enables complete off-chip reconfigurations. Because the ICAP does not allow a complete reconfiguration it does not need some of the pins the selectmap interface is equipped with. Except for this pins both interfaces are identical. Both selectmap and ICAP allow reconfiguration data to be written or read from the reconfiguration logic. The ICAP interface ports are described in table I.

TABLE I
ICAP INTERFACE PORTS

| Name | Direction | Description |
|------|-----------|-------------|
| CLK | Input | Clock signal |
| CE | Input | Config. Enable |
| Write | Input | Write read signal |
| BUSY | Output | Busy, buffer full |
| O[7:0] | Output | Config. data output (read) |
| I[7:0] | Input | Reconfig. data input (write) |

The ICAP interface is based on a handshake protocol. The BUSY pin is asserted when the buffer is full, in this case the data on the databus is ignored. The CE pin is used to pause the reconfiguration process. Supplied data is only accepted when the CE signal is asserted and the BUSY signal is not. There is an alternative way of controlling the ICAP, namely by keeping the CE asserted and using the clock pin to perform a write or read action. This method of control is based on the fact that a write or read action is performed on the falling edge of the clock signal if and only if the CE pin is asserted.

The ICAP can be instantiated as a primitive in a HDL based design. Furthermore, Xilinx offers a core designated as HW_ICAP. This HW_ICAP is an Onchip Peripheral Bus (OPB) device, allowing the ICAP to be controlled by means of a micro-processor. The proposed design does not use this OPB device, since it lacks in performance.

There is a marginal difference between the VIRTEX II Pro and VIRTEX-4 ICAP. The former features 8-bit data ports operating at 50 MHz, while the later has 32-bit data ports operating at 100 MHz. This implies a reconfiguration-rate of 400 Mbit/s for a Virtex II Pro and 3.2 Gbit/s for the Virtex 4 technology [4].

In order to abort the current reconfiguration process, a control signal sequence can be applied to the ICAP interface. This abort sequence is initiated by toggling the WRITE signal while the CE signal is asserted.

## C. Reconfiguration Logic

The reconfiguration logic is the entity performing the actual reconfiguration. The ICAP interface is only the interface to this entity. The Virtex reconfiguration logic was designed so that an external source may have complete control over the reconfiguration functions by accessing and loading addressed internal reconfiguration registers. These registers are addressed and loaded with the instruction/packets contained in the reconfiguration bitstream. In Table II, the most significant internal reconfiguration registers, used in the device reconfiguration, are presented.

TABLE II

VIRTEX II PRO INTERNAL RECONFIGURATION REGISTERS.

| Register Name | Description |
|---|---|
| CMD | Command Register |
| FLR | Frame Length Register |
| FAR | Frame Address Register |
| FDRI | Frame Data Input Register |
| CRC | Cyclic Redundancy Check |
| MFWR | Multiple frame Write Register |

These registers provide the following information to reconfigure the FPGA.

Frame Address Register (FAR): The FAR has the frame address for the next reconfiguration data input write cycle. This register indicates which frame, and where in the frame, the 32-bit data reconfiguration data is to be written. The register is automatically incremented after every written word. The size of a frame is specified in the Frame Length Register.

Frame Length Register (FLR): The FLR is used to indicate the frame size to the internal reconfiguration logic. The value loaded into this register is the number of actual reconfiguration bits that get loaded into the reconfiguration memory frames.

Frame Data Register Input (FDRI): The FDRI forms a pipeline input stage to store the reconfiguration Data Frames in the reconfiguration memory. Starting with the frame address specified in the FAR, the FDRI writes its contents to the reconfiguration memory frames. The FDRI automatically increments the frame address.

Multiple Frame Write Register (MFWR): The Multiple Frame Write Register is used when the bitstream compression option is enabled. When more than one frame has identical data, the Data Frame can be loaded once into the Multiple Frame Write Register then copied into multiple memory address locations.

CRC Register: The CRC is loaded with the Cyclic Redundancy Check (CRC) value that is embedded in the bitstream and compared against an internally calculated CRC value. This value is used to check for errors in the transmission of the bitstream data, and can be easily tempered with. The reset of the CRC register, and associated logic, is controlled by the CMD register.

Command Register (CMD): The CMD is used to execute commands on the device. Table III presents the most relevant commands available in the FPGA reconfiguration. These commands are executed by loading the CMD register with the respective binary code.

TABLE III

VIRTEX II PRO CMD REGISTER COMMANDS.

| Command | Action |
|---|---|
| RCRC | Reset CRC Register |
| SWITCH | Change clock frequency |
| WCFG | Write Configuration Data |
| RCFG | Read Configuration data |
| LFRM | Last Frame Write |
| SHUTDOWN | Begins Shutdown sequence |
| START | Activates the reconfigurable hardware |
| MFWR | Activate Multiple Frame Write mode |

### D. Reconfiguration Data

The reconfiguration bitstream is composed of a sequence of packets. Each packet has a header field as well as a content field. The header field contains information regarding the destination and the size of the packet. The header addresses one of the registers of the reconfiguration logic. A header can either exist out of one or two words. Usage of either the short or the long header depends on the length of the content. The two word header is used when the content length is too large to express it using the bits reserved in the type 1 header, for small content the type 1 header is suitable. Most packets are rather small, e.g. a packet to write the frame address register is only one word long, except for the packet containing the frame data, i.e. the data written to the reconfiguration memory. So the long headers are mainly used when writing to the frame data register. The type 2 header is a type 1 header extended with an extra word, when using a type 2 header the length field of the second word contains the length. Type 1 header use 10 bits to express the length, while the type 2 header uses 26 bits, as depicted in , see Figure 1 and Figure 2. Type 1 packets allow a maximum of $2^{11} - 1$ data packets and a type 2 packet $2^{27} - 1$ data packets.

| Packet Header | Type | Operation (Write/Read) | Register Address | Byte Address | Word Count (32-bit data words) |
|---|---|---|---|---|---|
| Bits | [31:29] | [28:27] | [26:13] | [12:11] | [10:0] |

Fig. 1. Type 1 packets

| Packet Header | Type | Operation (Write/Read) | Word Count (32-bit data words) |
|---|---|---|---|
| Bits | [31:29] | [28:27] | [26:0] |

Fig. 2. Type 2 packets

In addition to the packet headers there are a few other words that carry a specific purpose. These words only occur at the beginning or at the end of the bitstream. In total

there are three of these words outlined hereafter.

*Dummy word:* The start of the bitstream, its purpose is to flush the data buffer of the reconfiguration logic.

*Synchronization word:* Follows the dummy word. After receiving the synchronization word the reconfiguration logic starts interpreting the bitstream.

*De-synchronization word:* Occurs after the last package. After the de-synchronization word the occupation of the reconfiguration logic ends (i.e. The ICAP wouldn't interpret any data until it receives a synchronization word).

### E. Bitstream Storage

In the presented design the reconfiguration bitstream is retrieved from a storage device. However, the Xilinx implementation tools generate a so called bit file which contains the bitstream augmented with some additional data. This data used by Xilinx reconfiguration tools like Impact must not be written to the ICAP. Therefore this data is removed before storing it in the repository. During reconfiguration the reconfiguration unit must be able to determine the end of the bitstream when performing a reconfiguration in order to decide when to stop writing data to the ICAP. Two options are considered [5]: $i$) adding a header to the bitstream, containing either the end address; $ii$) append the length value to the bitstream. We concluded that the length value is most suitable, since the end address is dependent of the memory structure and address format. The length value can be kept constant which is advantageous when considering bitstream distribution through the internet, memory hierarchies or NOC's.

### III. PROPOSED STRUCTURE

In this section, the designs proposed to perform the FPGA partial reconfiguration with region delimitation are presented.

### A. Partial Reconfiguration Management Unit

The Partial Reconfiguration Management Unit (PRMU) is an independent unit designed to enable on-chip partial reconfiguration. The PRMU is capable of performing run-time partial reconfiguration using reconfiguration data, also referred to as bitstream, stored in memory. The PRMU uses the Internal Configuration Access Port (ICAP). The ICAP is an on-chip interface to the reconfiguration logic feature of Xilinx VIRTEX series FPGAs.

Three external signals for control purposes are featured by the PRMU: a signal to start a reconfiguration, a signal to abort the current reconfiguration, and a signal that reports the completion of either one of these operations. An additional bus is used to supply the bitstream address, i.e.

location of the reconfiguration data. As depicted in Figure 3, the PRMU is composed of two main parts, the reconfiguration unit, managing the reconfiguration; and the repository interface, which retrieves the bitstream. A third component referred to as the repository selector is needed when using multiple repositories.
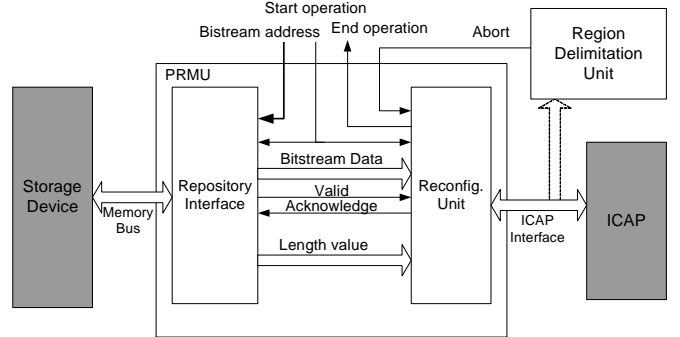


Fig. 3. PRMU and Region Delimitation unit organization.

In order to transfer the bitstream from the repository interface to the reconfiguration unit, a handshaking bus is used. An additional bus is used to transfer the length parameter to the reconfiguration unit. The handshaking bus is composed of three signals: a data transfer bus, a valid signal, and an acknowledge signal. The valid signal is asserted by the repository interface to signal that the current data is valid, while the acknowledge signal is asserted by the reconfiguration unit to acknowledge the current data.

The length bus is used to determine the end of the bitstream, since there is no unique word or sequence that terminates the bitstream. The words written to the reconfiguration logic are counted. This word count is compared against the length parameter to determine the end of the bitstream.

As mentioned earlier a reconfiguration is requested by asserting the start operation signal. On assertion the repository is initialized by means of the repository select signal. During the initialization process, the repository interface makes available both the length parameter and the first word of the reconfiguration data. The reconfiguration unit interacts width the reconfiguration logic through the ICAP. Furthermore, it splits the reconfiguration data into portions compatible with the ICAP, e.g. one byte when using the Virtex II Pro technology. Determination of the bitstream end is also covered by the reconfiguration unit, as well as the execution of an optional abort operation. Figure 4 depicts the internal organization of the PRMU.

The internal structure of the reconfiguration unit is depicted in detail by figure 4. The main controller handles the reconfiguration or abort request and reports their completion. Furthermore, it determines the end of the bitstream
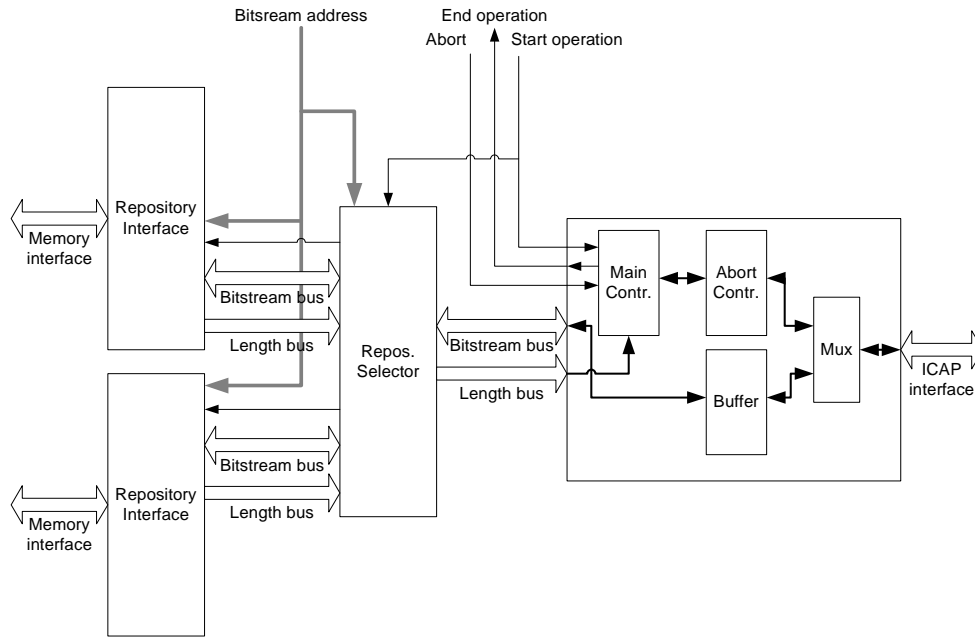
Fig. 4. Internal PRMU organisation.

and controls the data path which is composed of a buffer and a multiplexer. The additional abort controller is used to generate the abort sequence for the ICAP interface.

As mentioned earlier, using multiple repositories implies the integration of the repository selector. The repository selector is a multiplexer, which selects the proper repository when a reconfiguration is requested. The selection is based on the supplied address.

The current design is extendable with a multiple storage devices, the extendibility is due to the usage of repository interfaces. The discussed method not only enables various memories to be used as repository, but also considers a repository interface that can access memory through a NOC (Network On Chip) or one that access the internet to fetch an IPCore.

### B. Region Delimitation unit

The Region Delimitation unit monitors the reconfiguration data sent to the ICAP, to check whether the data is written to the correct region on the chip. In order to assure that a malicious or corrupted bitstream does not write outside the expected region of the FPGA, a region delimitation unit is proposed, to enforce this control. In order to identify which frames of the FPGA are being written, the reconfiguration bitstream has to be interpreted. To detect a write violation outside the delimited region, two things must be known: which frame is being written and when is the frame being written. Since the ICAP does not directly provide this information, the bitstream must be interpreted. In order to assure that all the bitstream

is analyzed, certain commands can not be allowed to be processed by the ICAP, namely the *Shutdown*, the *Multiple Frame Write* (MFWR), and the *Change frequency* (SWITCH) commands, as explained bellow.

To simplify the interpretation of the bitstream, the Region Delimitation unit is organized into two operations. First the 8 bit input stream is filtered, in order to eliminate all data before the synchronization sequence and to align the 8-bit input of the ICAP. The beginning of each packet is identified and 32-bit words are formed by loading each byte into a 32-bit shift register. After receiving 4 new 8-bit inputs to form a 32-bit packet, the $NewPacket$ signal activated, indicating that a new packet is available. The 32-bit packets, of commands or data, are then sent to the bitstream parser that interprets the bitstream data. This parser mostly implements a state machine that identifies which kind of packet is received and their function. Note that the flow of data into the parser is not constant; a delay of at least 3 clock cycles between each packet exists, since 32-bit packets are processed while 8-bit words are received by the ICAP. This means that the new state of the parser machine is only evaluated when a new packet is received, and the outputs are only updated after each state update.

Given that this unit has to identify where the Data Frame is being written to, the writing to the Frame Length Register (FLR), Frame Address Register (FAR), and Frame Data Register Input (FDRI) have to be detected. The parser has its own internal frame register, that is updated every time the FAR packet is sent to the ICAP. The value written to the

FDRI is retrieved and used to calculate which frames will be written by the arriving data packets. To assure that a frame is not improperly written due to a wrong frame size, when the frame size is sent to the device, via a FLR writing, the value is compared with the correct value, which is known for the device in use. The internal frame register of the Region Delimitation unit is initialized to zero, identically to the FAR register in the FPGA reconfiguration logic.

Only when Data frames are sent to the ICAP is the current frame address tested, otherwise during other reconfiguration instructions, an abort signal could be generated. For example, a non initialized FAR register (pointing to zero) would generate an out of bound error for the initial reconfigurations instructions on a valid bitstream sequence.

As mentioned above, it is not sufficient to test this type of instructions; command instruction also have to be interpreted, in order to assure a testable reconfiguration procedure. The following points out which commands are not allowed and why they can not be executed.

SWITCH : by changing the clock frequency, an operating frequency could be set in which the implemented hardware could behave erroneously.
SHUTDOWN : while in shutdown mode, the configured logic in the device is rendered inoperable; consequently, during this period the Region Delimitation unit would by rendered inoperable.
MFWR : the implemented Region Delimitation unit is only able to cope with a frame write at a time. This means that only one of the frames' writes would be within the Region Delimitation hardware testability. The other frames being written could be located anywhere within the device.

Whenever the internal frame address is outside the defined set of allowed frames, or when an illegal command is detected, the *Abort* signal is activated and sent to the PRMU, in order to suspended the reconfiguration process.

Given that the Region Delimitation unit monitors the reconfiguration bitstream simply by snooping the ICAP databus, the integration of these two units is rather simple, as depicted in Figure 3. In fact, the interaction exist of only two signals namely: the *Abort* signal, which is asserted by the delimitation core when it found an illegal addressing, and the end operation signal, which is used by the PRMU core to report the completion of the abort sequence.

Due to the internal structure of the reconfiguration hardware of the FPGA device, a packet is only processed by the ICAP after the next packet is received. This gives time for the Region Delimitation units to analyze a given packet and if necessary to stop/abort the reconfiguration process.

## IV. IMPLEMENTATION RESULTS AND RELATED ART

In order to analyze the functionality and the hardware costs of the proposed units, a Xilinx VIRTEX II Pro (XC2VP30-7) on a XUP prototyping board has been used. Given that on a VIRTEX II Pro the ICAP received 8-bit data at 50 MHz, a maximum reconfiguration throughput of 400Mbit/s is available in this technology.

As presented in Table IV, the PRMU, composed of the reconfiguration unit, repository selector, and repository interface, is capable of achieving an operating frequency of 100 MHz, which is more than sufficient for this technology, using 2% of the available hardware resources. The Region Delimitation unit is capable of operating at 200 MHz, requiring 1% of the available hardware resources to perform the bitstream analysis. The complete

TABLE IV
OBTAINED FIGURES ON A VIRTEX II PRO

| Unit | Slices | Max. Freq. |
|---|---|---|
| PRMU | 336 (2%) | 102 MHz |
| Reg. Delimit. | 175 (1%) | 200 MHz |
| both | 511 (4%) | 102 MHz |

system used to perform the dynamic reconfiguration of the FPGA with region delimitation requires less than 5% of the used device, without imposing any delays to the FPGA dynamic reconfiguration.

Implementation results suggest that the proposed system can also achieve the maximum reconfiguration throughput using the VIRTEX-4 FPGA, which has an ICAP operating at 100MHz with 32-bit input data.

Several self reconfiguring systems using a microcontroller have been proposed. In such approach a microcontroller is used to write the reconfiguration data to the ICAP.

Table V summarizes the reconfiguration throughput (Reconf. ThrPut.) and hardware requirements of the related art.

TABLE V
RELATED ART

| Solution | Reconf. ThrPut. | Slices |
|---|---|---|
| Claus(OPB) [6] | 38 Mbit/s | 933 |
| Claus(PLB) [6] | 400 Mbit/s | 919 |
| Möller(OPB) [7] | 16 Mbit/s | 973 |
| Möller(NoC) [7] | 80 Mbit/s | 1661 |
| Ours | 400 Mbit/s | 336 |

The solutions proposed in the related art have a similar structure, which is composed of a reconfiguration unit and a memory interface. The design proposed in this paper uses a direct memory interface, referred to as repository interface, while the other use a Processor Local Bus (PLB), an OPB, or a NOC as memory interface.

A PLB and an OPB based approach are presented in [6]. Both approaches require, in contrast to ours design, an additional micro-controller to control the data flow. Both approaches require more that 900 slices, 170% more than the proposed design. The OPB approach is significantly slower than ours, while the PLB approach can achieve a reconfiguration throughput of 400 Mbit/s , imposed by the ICAP interface. It should be noted that the throughput of the PLB bus can significantly vary, depending on the utilization of the PLB. The number of slices of [6], depicted in table V, are composed of the values presented in [6] and the slices required for the additional bus and micro-controller. Claus et al. [6] only present the slices required for the reconfiguration unit. The number of slices required for the bus and micro-controller were retrieved from [7].

The NOC based approach proposed by Möller et al. [7] uses the NOC to fetch the bitstream from memory. Identically to our approach, the design using the NOC also does not require a micro-controller; however, the use of the NOC requires 1661 slices and is only able to achieve a reconfiguration throughput of 80 Mbit/s.

The region delimitation unit occupation values have not been considered in this comparison with the related art, since non of the compared reconfiguration units possess this feature.

## V. Conclusions

In this paper a dynamic FPGA partial reconfiguration system, with autonomous memory access is proposed. Implementation results suggest that a compact design with high reconfiguration throughputs can be achieved. When compared with related art, it can be concluded that the proposed design is capable of achieving the maximum reconfiguration throughput imposed by the ICAP interface, while reducing the required reconfigurable resources by 170%. In this paper a novel region delimitation unit is also proposed, allowing the protection of the hardware not being reconfigured, in case an adulterated bitstream is loaded. Implementation results also suggest that the proposed design can be easily exported to other reconfigurable technologies without a performance degradation.

**Future Work**
The following proposes some research direction for the use of the design proposed in this paper.
- Integration of the PMRU in the MOLEN polymorphic processor as the dynamic reconfiguration module [1].
- Add a validation mechanism to the reconfiguration unit to verify the loaded bitstream.
- Use the proposed PMRU with region delimitation and a validation mechanism to assure the attestation of configurable units in the cryptographic processor proposed in [8].
- Development of a generic repository interface to simplify integration of repositories.

## References

[1] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte, "The MOLEN Polymorphic Processor," *IEEE Transactions on Computers*, pp. 1363– 1375, November 2004.

[2] XILINX, *Early access partial reconfiguration user guide for ISE 8.1.01i*, March 2006.

[3] G. Mermoud, *A Module-Based Dynamic Partial Reconfiguration Tutorial*, November 2004.

[4] XILINX, *Virtex-4 FPGA User Guide*, August 2007.

[5] G. Kuzmanov, G. N. Gaydadjiev, and S. Vassiliadis, "Loading rmcode: Design Considerations," in *Proceedings of the Third International Workshop on Computer Systems: Architectures, Modeling, and Simulation*, pp. 11–19, July 2003.

[6] C. Claus, F. H. Müller, J. Zeppenfeld, and W. Stechele, "A new framework to accelerate Virtex-II Pro dynamic partial self-reconfiguration," 2007.

[7] L. Möller, I. Grehs, E. Carvalho, R. Soares, N. Calazans, and F. Moraes, "ANoC-based Infrastructure to Enable Dynamic Self Reconfigurable Systems," 2007.

[8] R. Chaves, G. Kuzmanov, S. Vassiliadis, and L. A. Sousa, "Reconfigurable Cryptographic Processor," in *Workshop on Circuits, Systems and Signal Processing (ProRisc)*, November 2006.