

HARPE: a Harvard-based Processing Element Tailored for Partial Dynamic Reconfigurable Architectures

Alessio Montone, Vincenzo Rana, Marco D. Santambrogio, Donatella Sciuto

Politecnico di Milano
Dipartimento di Elettronica e Informazione
Via Ponzio 34/5
20133 Milano, Italy

{santambr,rana,sciuto}@elet.polimi.it, alessio.montone@dresd.org

ABSTRACT

Aim of this paper is to propose a Reconfigurable Processing Element based on a Harvard Architecture, called HARPE. HARPE's architecture includes a MicroBlaze soft-processor in order to make HARPEs deployable also on devices not having processors on silicon die. In such a context, this work also introduces a novel approach for the management of processor data memory. The proposed approach allows the individual management of data and the dynamic update of the memory, thus making it possible to define Partially Dynamical Reconfigurable Multi Processing Element Systems, that consist of several master (e.g., soft-processors, hard-processors or HARPE cores) and slave components. Finally, the proposed methodology enables the possibility of creating a system in which both HARPEs and their memories (data and code) can be separately configured at run time with a partial configuration bitstream, in order to make the whole system more flexible with respect to changes occurring in the external environment.

1. INTRODUCTION

In recent years, the evolution of reconfigurable devices has brought to significantly increase their size, capacity and performance [1,2]. The most commonly used reconfigurable devices are *Field Programmable Gate Arrays*, FPGAs, employed both as a component of a more complex system (playing the role of a co-processor), and as main architecture itself integrating most of the components of the system. In addition to allowing an unlimited number of reconfigurations, FPGAs can also be partially and dynamically reconfigured: this means that single portions of the architecture can be modified without blocking the entire system execution.

FPGAs can be used to create hardware/software platforms that keep their flexibility after deployment, allowing the development of complex *System-on-Chip* (SoC) and reducing the overall number of physical components, since many resources can be configured on request, replacing unused ones. Several research groups, [3–6] have built reconfigurable computing engines to obtain high performance at low cost by specializing the computing engine to the computation task.

In such a scenario, a larger number of complex components can be mapped at the same time into the same device [7]. An emerging design pattern is based on Multi Processing Element [8,9]. This paper mainly focuses on the

definition of a single Processing Element (PE) tailored for a partially dynamically reconfigurable architecture. Target devices have been chosen among Xilinx's products, particularly Virtex-II and Virtex-II Pro FPGA's Families [10]. In [8,9] the authors proposed a multi processing element architecture showing how the reconfiguration may provide benefits if applied within this context. We extend these works, considering a dynamic architecture where the number of processors can be modified at runtime to better satisfy new demands and allowing the modification of the information stored in the memory of the new cores that have to be mapped via partial bitstreams, without leaving this task to one of the processors already configured on the reconfigurable device.

Furthermore, previous works related to the manipulation of bitstreams content mainly focused their attention on Xilinx FPGAs devices. Inside Virtex family documentation [11] there are equations for single BRAM-Block content memory manipulation. These equations allow to find, to insert or to modify memory information inside a configuration bitstream, but they are not valid for Virtex II FPGAs families and more recent ones (e.g., Virtex-4 [1] and Virtex-5 [2]). Furthermore these documents do not explain how several BRAM Block create a contiguous addressable memory address space. A more recent work about bitstreams manipulation can be found in [12]. Equations for Virtex II (Pro) are presented here, but they only allow to address frames (also according with [10]) and they do not provide information about single bit position and contiguous memory space of BRAM-Blocks set. No other studies have been performed on bitstream analysis tailored for BRAM content manipulation on more recent Xilinx FPGAs families.

The innovative contribution of this work can be found:

- in the definition of a novel reconfigurable processing element, named HARPE, based on a Harvard architecture (Section 2);
- in the formalization and in the corresponding implementation of a methodology that allows to replace a generic soft/hard-core, or its memory contents at runtime without preventing the rest of the system from correctly working (Section 4);
- in the implementation of a Multi-Processing Reconfigurable Architecture¹ where both memories and num-

¹Where each processing element is defined using a HARPE

ber of processors (e.g., HARPE components) can be reconfigured to meet run-time system needs (Section 5).

Section 2 presents the single PE, while Section 3 briefly introduces the configuration process of a Virtex II FPGA. Section 4 describes the methodology employed for the creation of a memory content configuration bitstream. Results will be provided in Section 5 and conclusion will be drawn in Section 6.

2. THE RECONFIGURABLE ELEMENT

Before going into details of the proposed reconfigurable processing element, let briefly describe the target architecture. We consider an architecture supporting partial dynamic reconfigurability and composed by a fixed logic and a set of reconfigurable areas (RA). Reconfigurable PEs are configured inside the RAs. In this paper the Early Access Partial Reconfiguration design flow [16] is considered, hence every reconfigurable area can provide links to the static logic regardless its position within the FPGA chip area. From a logical point of view the architecture's fixed logic is responsible for managing inter-PE communication implementing a communication infrastructure (like bus-based, NoC-based, point to point and so on). A logical overview of the target architecture is drawn in Figure 1. In this paper we propose

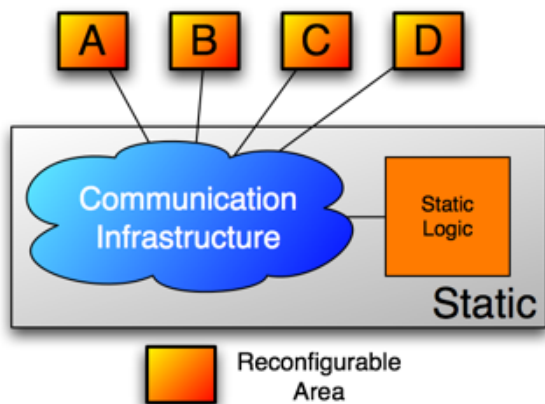


Figure 1: Logical view of target architecture. The architecture is composed by a set of reconfigurable areas and static logic. Communication infrastructure is provided by static logic.

a processing element that can be configured inside an RA of a such architecture.

The logical view of the proposed processing element is shown in Figure 2. It is a standard architecture based on a MicroBlaze soft processor.

The processor's bus is an OPB bus belonging to the IBM CoreConnect technology [13]. In the typical configuration it is arbitrated and it has a 32 bit bus both for data and address.

Due to the standardized interface, also user logic can be attached to the bus, hence processor can use them. Typ-
core

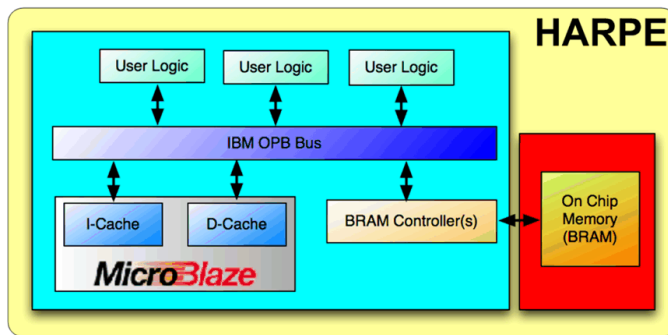


Figure 2: The HARPE processing element

ically, processors perform control tasks while a hardware-implemented user-logic perform the calculation. In such a way the flexibility of the software is coupled with the speed of the hardware.

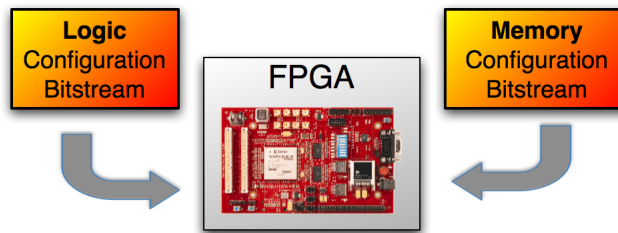


Figure 3: Decoupling logic from memory content. The novelty of the proposed approach is to decouple logic configuration from on-die memory configuration

While a coupling of software flexibility and hardware speed is needed, a configuration flexibility is also required. This has been achieved with the introduction of a novel approach for BRAM-Block content manipulation for Xilinx Virtex II and Virtex II Pro FPGAs families, [10]. It consists in the decoupling of the configuration of logic from the configuration of BRAM (i.e. code and data). Currently there are softwares that initialize memory information on complete configuration bitstreams (i.e. bitstreams configuring the entire FPGA), but they do not work on partial configuration bitstreams. Considering the fact that a PE is just a system's part that can be modified and configured separately from each other, then it seems obvious the need of a memory initialization framework able to work with partial bitstreams.

The main issue is a mapping procedure that allows the creation of a bitstream able to configure only BRAM Blocks, leaving the rest of the logic unaltered. This procedure takes as input the data needed to fill the memory and the list of BRAM-Blocks that have to be used. Therefore, the proposed approach decouples logic configuration from the BRAM memory content configuration. Such a feature may result in the partitioning of configuration bitstreams in two parts (Figure 3):

- a bitstream able to configure the logic [14];
- a second bitstream able to configure just the BRAM content.

Due to this approach either the logic can be changed in order to perform different computation on the data stored in BRAMs or it can remain unchanged while the memory content can be updated in order to perform the same computation on different input data (as previously underlined, memory content consists of code and data segments of a software running on a processor).

3. BITSTREAM STRUCTURE

As introduced in Section 2, one of the main contributions of this paper is the definition of a methodology for the dynamic reconfiguration of memory content. This is possible due to a partial bitstream able to modify only BRAM-Block content without changing the other configurable elements of the FPGA, thus not interfering with the part of the system that is not directly involved in the reconfiguration process. In order to understand how the approach is technically feasible, the structure of Virtex II - Pro FPGAs and of their configuration bitstreams will be briefly described in this section.

3.1 Virtex II Pro Configuration

A bitstream is a binary file containing the information of the logic that will be downloaded onto the FPGA; it is structured as a sequence of 32-bits words. A bitstream is composed of configuration commands and data. Commands specify the type of the operation (read or write) and the referred configuration register; data words contain the actual logic of the hardware core.

According to [10], Virtex II FPGAs are configured by columns [15, 16], as shown in Figure 4. Each bitstream column configures one physical IOBs, CLBs, BRAM-Block Interconnections or BRAM-Content column. Each column is divided in several frames ([17]) and each frame is addressed by a Frame Address containing information about column type BA (CLB-IOB, BRAM Interconnection, BRAM Content), column number (Major Address, MJA), frame address inside current column (Minor Address, MNA). Each frame can be configured independently from each other frame; in particular, it is possible to configure BRAM-Content frames (and columns) independently of the real logic (CLBs, IOBs, et cetera). The Frame length is a device dependent parameter and, in each bitstream, columns and frames are sequentially written.

4. THE PROPOSED METHODOLOGY

Aim of the proposed methodology is the definition of a systems in which both HARPE elements (based on MicroBlaze soft-processor, as described in Section 2) and their memories can be dynamically reconfigured without interfering with the rest of the system. For instance, it is possible that, for a particular configuration of the reconfigurable device, there is not enough available space for a dedicated core that is necessary for the fulfillment of the functionality of the whole system.

Table 1 shows the area usage of two versions of the MicroBlaze soft-processor and, as presented in [18], of a set of dedicated cores that implement image processing operators [19]; in particular, *open* is actually the combination of erosion and dilatation (even if it runs two time faster than the individual routines), while in *wavelet* and *tridiagonal* the loops are fully unrolled to exploit more parallelism. In all

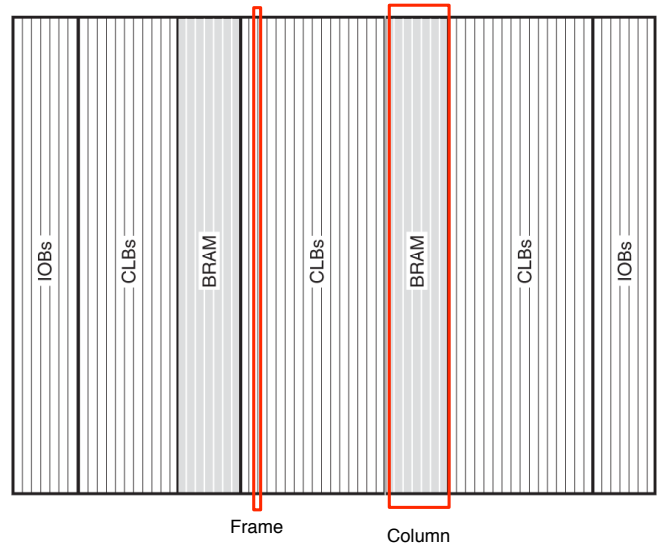


Figure 4: Virtex II Pro Family Configuration Bitstream Structure.

Table 1: Area usage comparison between soft-core processors and dedicated cores on a Xilinx Virtex-II pro device (XC2VP7)

Core	Area usage (LUTs)	Area usage (%)
MicroBlaze (Area optimized configuration)	1566	15.9
MicroBlaze (Performance optimized configuration)	1679	17
Sobel mag	1943	19.7
Convolution	1783	18.1
Convolution 5	3235	32.8
Convolution sm	2609	26.5
Prewitt mag	1815	18.4
Open	4500	45.7
Close	4500	45.7
Wavelet	2172	22
Tridiagonal	7476	75.6

these cases, dedicated cores are larger than a simple HARPE element.

Figure 5 shows an example in which a reconfigurable device has to be configured with two different dedicated cores: Convolution 5 (C) and Open (O). In the first step, just the static part (S) is configured on the device, while the reconfigurable part is completely empty. This makes it possible to configure either the Convolution 5 or the Open core (but not both the cores, since the sum of their area usage exceeds the free space). In the second step, the Convolution 5 has been configured on the device, but there is not enough free space to configure the Open core, so it is possible to follow two different ways:

- A) it is possible to stall the configuration process until the device will free the amount of resources required for the configuration of the dedicated core (in other words it is necessary to wait the the Convolution 5

core ends its computation); or

- B) it is possible to configure a HARPE core (H) in order to perform (with the MicroBlaze soft-processor) the same computation of the dedicated core.

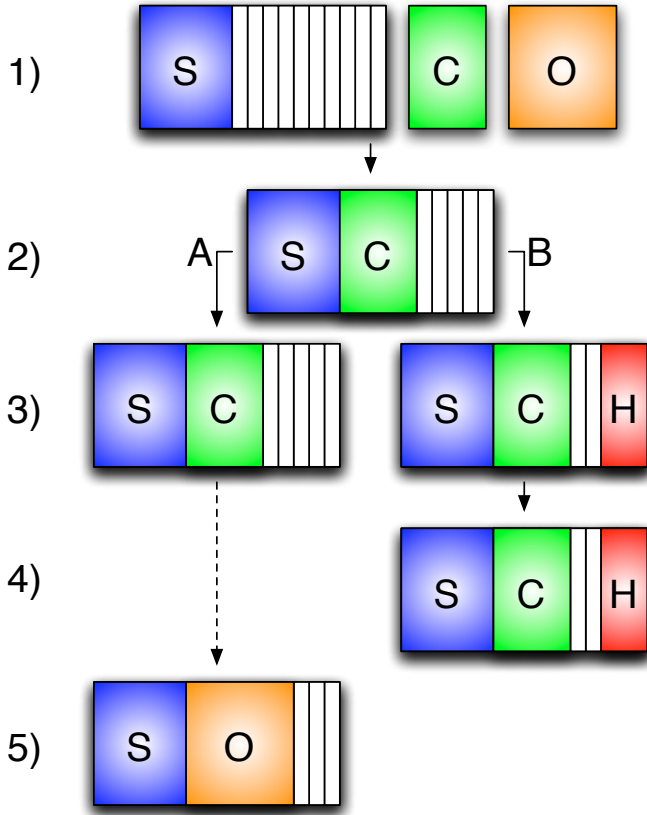


Figure 5: Example of HARPE core configuration

The second way has to be preferred when there is no precise information about the status of the available resources of the device in the next future or when the requested resources will not be free in a short time interval. In these cases, even if the computation on the MicroBlaze will obviously take more time than the computation with the dedicated core, it is possible to improve timing performance since the configuration can take place immediately, without waiting for other resources.

4.1 Mapping Procedure

While a dynamic configuration of HARPE modules requires a simple partial reconfiguration process that involves just the logic of HARPE modules, for what concerns their memories it is necessary to generate a partial bitstream that configures a subset of total BRAM Blocks leaving unchanged the other columns. The core of the partial bitstream creation phase is a mapping procedure. For each bit to be put in memory, i.e. a bit of data or even a bit of code running on a processor, the mapping procedure returns the column identifier (Major Address) and the target position offset inside the column.

This procedure is divided in two parts: memory content splitting and bitstream mapping. There are several BRAM

Blocks on a Virtex II FPGA die and a subset of them, depending on the implemented architecture, creates the on-chip memory space. Given a list of bits to be put in BRAM, the memory content splitting associates each bit with the correct BRAM-Block used by the implemented architecture, while the bitstream mapping maps each BRAM-Block content inside the bitstream column configuring such block.

During the *memory content splitting* phase, data is divided in 2^a bit segments (where a is a given parameter). We will refer to the k th segment as C_k . Each of this segment is reversed (most significant bit becomes the least significant bit and so on) and circularly assigned to one of the b BRAM-Blocks (i.e. segment C_j is assigned to the block $j \bmod b$, as shown in Figure 6).

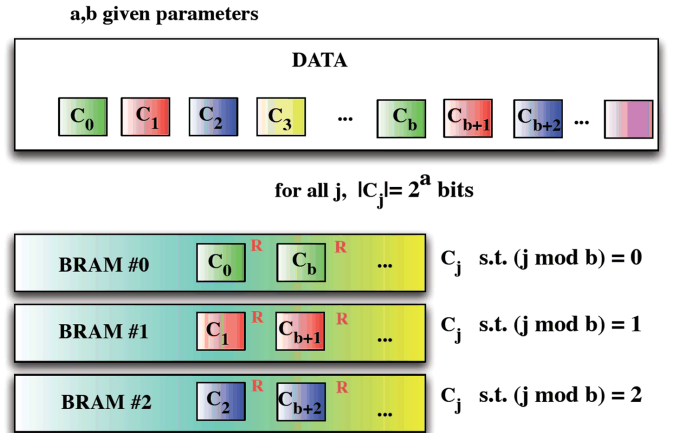


Figure 6: *Memory content splitting*. The red R stands for segment reversing.

Let us consider now a single BRAM-Block j and all the segments associated with it. Let c be an ordered list of the bits assigned to the block and c_j be the j th bit of this list (as shown in Figure 7).

This list c goes through the second phase, called *bitstream mapping*. Each one of these bits has to be inserted inside a frame and a column of the final configuration bitstream; this is done by the bitstream mapping phase.

Before seeing the equations ruling the calculus of addresses and offsets, some parameters and terms need to be introduced.

- *BRAM_Rows* is the total number of Block RAM rows in the FPGA die. It is a device dependent parameter and can be found in the FPGA's datasheet.
- *Frame_Len* is the length of a frame expressed in words per frame. It is a device dependent parameter and can be found in the FPGA's datasheet.
- *Word_Len* is the length of a word in bytes. It is constant equal to 4 bytes per word.
- *BRAM_X* is the X-position of the involved Block RAM.
- *BRAM_Y* is the Y-position of the involved Block RAM.
- K is a device dependent offset. It can be evaluated by positioning a bit inside BRAM Columns with Xilinx tools and then inverting mapping equations.

- $\text{byte_offset}(c_j)$ gives the offset, starting from the beginning of the column identified by MJA, of the byte where c_j has to be positioned.
- $\text{bit_offset}(c_j)$ gives the offset, starting from the beginning of the byte identified by $\text{byte_offset}(c_j)$, of the bit where c_j has to be positioned.

In the equations, the following short hand notations will be used for *if* sentences:

$$a?b : c \Leftrightarrow \begin{cases} b & \text{if } a \\ c & \text{if not } a \end{cases}$$

and for constant arrays:

$$\alpha(l) := \{\xi_0, \xi_1, \xi_2\} \Leftrightarrow \alpha(0) = \xi_0, \alpha(1) = \xi_1, \alpha(2) = \xi_2, \\ \alpha \text{ undefined elsewhere}$$

At this point, equations that give the bitstream position of each c_j bit can be provided:

$$\begin{aligned} \delta(l) &:= \{6, 4, 2, 0, \\ &\quad -4, -6, -8, -10, \\ &\quad -16, -18, -20, -22, \\ &\quad -26, -28, -30, -32\} \\ \varphi(l) &:= \{3, 2, 1, 0, 4, 5, 6, 7\} \\ \gamma &= (BRAM_Rows - BRAM_Y) * 40 \\ &\quad + 80 + K \\ MJA &= BRAM_X \\ MNA &= j \text{ div } 256 \\ \text{cpbo} &= (j \text{ div } 32) \text{ mod } 8 \\ \text{byte_offset}(c_j) &= \gamma + \delta(j \text{ mod } 16) \\ &\quad + (((j \text{ div } 16) \text{ mod } 2) = 0)?0 : -1 \\ &\quad + MNA \times \text{Frame_Len} \times \text{Word_Len} \\ \text{bit_offset}(c_j) &= +(((j \text{ div } 16) \text{ mod } 2) = 0)? \\ &\quad \varphi(\text{cpbo}) : 7 - \varphi(\text{cpbo}) \end{aligned}$$

The entire mapping procedure described here can be used, with data that has to be written in memory, to create bitstreams that modify the BRAM-Content Columns of Xilinx Virtex II (Pro) FPGAs. Next section describes the marBram framework, that implements such mapping procedure.

4.2 The marBram Framework

The marBram framework provides an implementation of the mapping procedure. This framework takes as input the following data:

- BMM file: it describes how different BRAM-Blocks create a contiguous memory interval. It contains a , b , $BRAM_X$ and $BRAM_Y$ parameters.
- Device Parameters file: it contains device dependent information, most of them derived from Xilinx Datasheets (Frame Length, number of frames in BRAM-Content column, device id)
- MEM file: it is a file with memory data in hexadecimal notation using Verilog memory simulation syntax. This file can be easily extracted from compiled code (ELF file) in order to fill in BRAM with code.

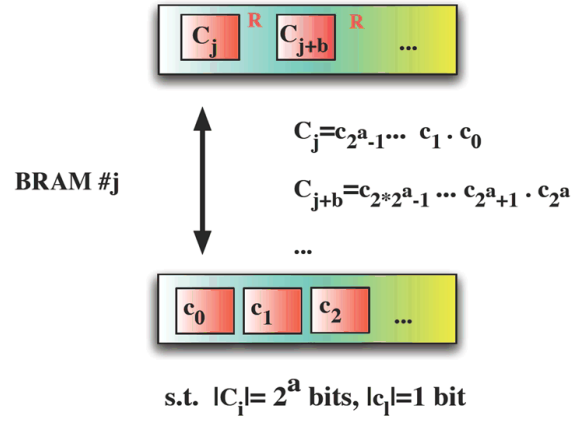


Figure 7: Bits assigned to a given BRAM-Block.

The output of marBram is a bitstream (ready to be downloaded on the device) that configures only the content of BRAM Columns containing BRAM-Blocks specified inside the ELF file.

Figure 8 presents the marBram framework flow. MEM and BMM files have to be parsed to generate all the necessary information to the core of the marBram framework that implements the previously described mapping procedure. Both parsers consist of lexical analyzers built with FLEX. The BMM parser reads the input file and returns the following parameters:

- a (2^a is the number of bits of each segment in which memory data are split during the mapping procedure);
- b (the number of BRAM-Blocks);
- the position inside the FPGA (X and Y) of each BRAM-Block;
- the base address and the high address of the entire addressable space.

The MEM parser reads the input file returning a list of bytes that have to fill adjacent memory positions. The first byte of the returned list is associated with the base address returned by BMM parser (that is why there is an interaction between the two parsers in Figure 8). Gaps in memory data (i.e. addresses between base address and high address without any data associated) are considered as filled with zeros.

The two phases of the mapping procedure have been implemented separately in order to be reusable for future developments (different bitstream mapping procedures on different families or different memory content splitting procedures for different architectures). After the execution of the mapping procedure, columns and frames are wrapped by commands (according to Xilinx Documentation [10]). Some commands are needed to initialize and terminate the partial dynamic configuration sequence, while others to specify, for each modified BRAM Content column, the Frame Address of column's first frame and the length of data written in the addressed column. Padding frames are inserted at the end of each column configuration (according to Figure 8) in order to flush configuration pipelines. More than 90% of the commands overhead (see Section 5 for further details) are made of such padding frames. The final bitstream is written

on the output file that is straightforward downloadable on the target FPGAs, i.e. can be sent to the FPGA through one of the configuration interfaces. In the following section the produced bitstream will be described in details.

4.3 Memory configuration bitstream

According to Xilinx Documentation [10] partial dynamically reconfiguration bitstreams have a well defined structure. The bitstream is made of a sequence of commands and operands. Most of the commands set the destination register of the operands, while other commands are provided without operands (e.g. command for the startup of the device). The partial dynamic reconfiguration bitstream is characterized by the absence of global startup command and contains only information regarding the startup process. In order to configure just BRAM columns, the bitstream has to provide only the related commands. The structure of the bitstream generated by marBram is the following:

```

<Bitstream> → <align> <CRCReset> <DeviceID>
              <WriteConfig>
              (<FarAddress> <FDRI>
               <FrameData>)+
              <DefCRC> <CRCReset>
              <CRC> <DefCRC>
              <Detach> <DummyWord>4

<align> → FFFFFFFF AA995566
<CRCReset> → 30008001 00000007
<DeviceID> → 3001C001 xxxxxxxx (ref. Datasheet)
<WriteConfig> → 30008001 00000001
<FarAddress> → 30002001 02 (MJA · 2)80000
<FDRI> → 30004000 3yyyyyyyy (Length in words)
<DefCRC> → 0000DEF C
<CRC> → 30000001 0000DEF C
<Detach> → 30008001 0000000D
<DummyWord> → 20000000

```

The $\langle FDRI \rangle$ contains the length of the frame data in words, while the $\langle DeviceID \rangle$ contains a value that is unique for each FPGA device and that can be found in Xilinx's Datasheet. $(MJA \cdot 2)_8$ means that the Major Address, given by the equations, has to be coded in 7 bits adding a zero on the lowest significant bit, hence it is the same as coding $MJA \cdot 2$ in 8 bits. The position of the Dummy words is critical because its erroneous positioning leads to a non complete flushing of configuration pipelines and consequently of a non correct memory content configuration.

5. EXPERIMENTAL RESULTS

The mapping procedure and marBram framework have been validated using several architectures, based on IBM CoreConnect Technology [13] and using either a PowerPC-405 hard processor or a MicroBlaze soft processor [20] (even if the proposed processing element is based on a MicroBlaze, the memory content mapping algorithm is also compatible with the on die PowerPC-405 processor). All the architectures have been developed with Xilinx Embedded Development Kit (EDK) [21]. Tests have been performed on Xilinx Virtex II Pro VP7 and VP20 [10] and consisted in modify-

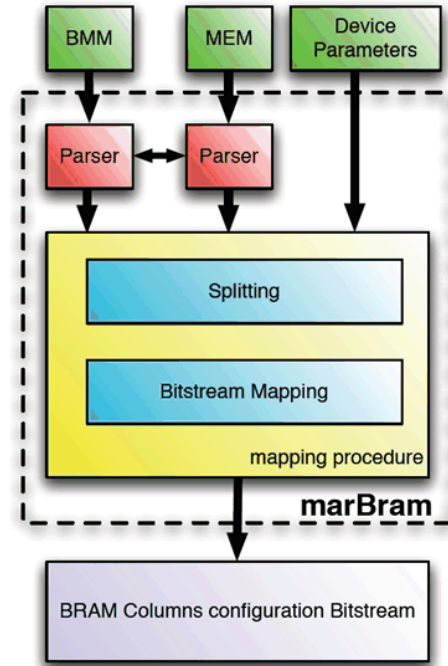


Figure 8: marBram framework data flow

ing only the BRAM Memory content (without modifying CLBs, memory interconnections, etc) with the bitstream created using the marBram framework (giving in input different code and data segments) and resetting the processor in order to start fetching the new code. For instance, Figure 9 presents an implementation of the HARPE architecture on a VIIP7 device where the BRAMs have been assigned trying to maximize their placement inside the HARPE area.

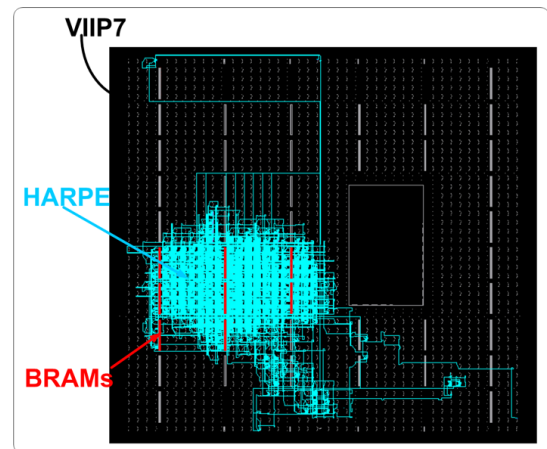


Figure 9: HARPE architecture implemented on a VIIP7 device.

In Table 2 a collection of results of the execution of marBram framework is provided.

It can be noticed that the size of the BRAM configuration bitstream only depends on the number of columns involved and not on the number of BRAM Blocks. The overhead, i.e. the ratio between device commands length in the bitstream

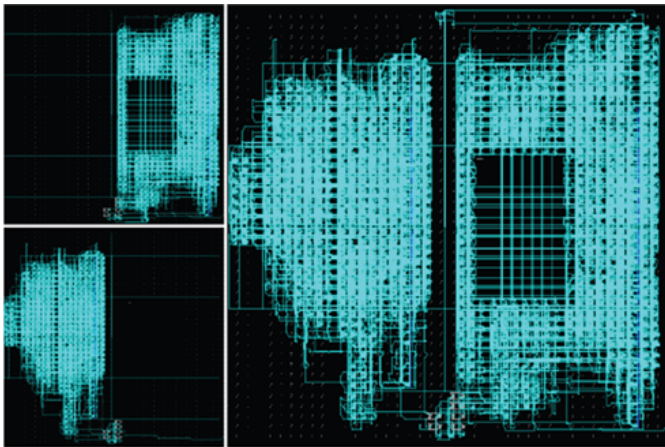
Table 2: Results of the execution of marBram framework

Target FPGA	Processor	#BRAM Blocks	#BRAM columns involved	marBram execution time	commands overhead	bitstream size
VP7	Microblaze	4	2	179 ms	$\simeq 1.5\%$	56 Kbytes
VP7	PowerPC-405	8	3	203 ms	$\simeq 1.5\%$	84 Kbytes
VP7	Microblaze	8	5	263 ms	$\simeq 1.5\%$	136 Kbytes
VP20	PowerPC-405	8	3	248 ms	$\simeq 1.5\%$	112 Kbytes
VP20	Microblaze	8	5	326 ms	$\simeq 1.5\%$	160 Kbytes
VP20	Microblaze	16	5	326 ms	$\simeq 1.5\%$	160 Kbytes

and the data length that configures the FPGA, inversely depends upon the number of frames per columns (for VP7 and VP20 it is the same value). Finally the execution time mainly depends on the total columns number of the reconfigurable device.

Tests on the embedded partial reconfiguration show that the baud rate changes according to the processor working frequency. With the proposed architecture, the reconfiguration baud rate is 1,02 MByte/s when the processor works at 100 MHz, 1,15 MByte/s at 200 MHz, and finally 1,22 MByte/s at 300 MHz. As shown in Table 3, the baudrate is independent of the bitstream size.

A real Multi-Processing Reconfigurable Architecture, where each processing element is defined using a HARPE core, have been implemented on different target devices, such as Xilinx VIIP7 and VIIP20. Figure 10 shows the synthesis results for the Xilinx VIIP7.

**Figure 10: HARPE scenarios on a Xilinx VP7 device**

In particular, the left side of Figure 10 presents a couple of reconfigurable components (HARPE instances), that can be implemented and used independently, while the right side shows a scenario where both the HARPEs have been configured together. In Table 4 the resources requirement for a dual-HARPE architecture are presented. The architecture has been verified using two different FPGAs: a Xilinx Virtex II Pro 20 and a Xilinx Virtex II Pro 7. The second solution is the same one that has been presented in Figure 10.

6. CONCLUSIONS

Several research groups, [3–6] have built reconfigurable computing engines to obtain high performance at low cost by

Table 4: Resources requirement for a dual-HARPE architecture implemented on a Xilinx Virtex II Pro 20 and 7. S: Static Component, H1: HARPE Core 1, H2: HARPE Core 2.

		VP20			VP7		
		S	H1	H2	S	H1	H2
Area		4x112	40x112	44x112	-	32x80	32x80
BufG	Total	3	1	1	3	1	1
	LOCed	2	1	1	2	1	1
DCM		1	-	-	1	-	-
I/O	Total	12	12	12	12	12	12
	LOCed	2	4	2	2	4	10
Mult. 18x18		-	3	3	-	3	3
BRAM		-	8	8	-	8	8
Slice	Total	32	1.207	1.229	48	1.222	1.245
	LOCed	-	1.175	1.197	-	1.174	1.197

specializing the computing engine to the computation task. What seems to be neglected so far is the full exploitation of the ability to partially reconfigure the FPGA at runtime even if some results have been published i.e., [22–24]. The approach proposed in this paper has been proved to be a feasible solution for memory management in a system composed of a set of master components, the HARPE architecture. Results presented in Section 5 have been used to validate the whole methodology and have shown that the overhead introduced by the marBram framework is very small with respect to the whole bitstream size. The framework has been validated, both with the Microblaze soft-processor and with the PowerPC-405 hard-processor, with a collection of bitstreams which size ranges from 56 Kbytes to 160 Kbytes, while the execution time on this experiments ranges from 179 ms to 326 ms. In all the proposed examples, the marBram framework generated a partial bitstream able to individually configure the memory of one master component, leaving unaltered the memory of the others master components. This memory update can be performed in a dynamic way, since it is based on a partial bitstream, and then, by using the proposed solution, it is possible to develop a reconfigurable system in which both components and component memories can be reconfigured at run-time. One of the possible extensions of this work could be the creation of a system in which several master components can operate at the same time, but in which just a subset of them is responsible for the management of the data memory of all the master components. In such a scenario, then, the developed system is able to autonomously adapt its functionalities to the changing environment by modifying either the number and the

Table 3: Reconfiguration Times

Number of frames	Bitstream size (Byte)	Time (ms)	Baudrate (MByte/s)	Time (ms)	Baudrate (MByte/s)	Time (ms)	Baudrate (MByte/s)
		System at 100 MHz		System at 200 MHz		System at 300 MHz	
1	1454	1,4256	1,0199	1,2655	1,1489	1,1928	1,2190
2	2318	2,2723	1,0201	2,0172	1,1491	1,9013	1,2192
3	3182	3,1191	1,0202	2,7689	1,1492	2,6098	1,2193
14	10610	10,3985	1,0203	9,2325	1,1492	8,7024	1,2192
16	12338	12,0919	1,0204	10,7352	1,1493	10,1198	1,2192

type of master components or the data memories on which they have to operate.

7. REFERENCES

- [1] Xilinx Inc. Virtex-4 configuration user guide. Technical Report ug71, Xilinx Inc., January 2007.
- [2] Xilinx Inc. Virtex-5 configuration user guide. Technical Report ug191, Xilinx Inc., February 2007.
- [3] Jason Miller David Wentzlaff Fae Ghodrati Ben Greenwald Henry Hoffman Paul Johnson Jae-Wook Lee Walter Lee Albert Ma Arvind Saraf Mark Seneski Nathan Shnidman Volker Strumpfen Matt Frank Saman Amarasinghe Michael Bedford Taylor, Jason Kim and Anant Agarwal. The raw microprocessor: A computational fabric for software circuits and general-purpose programs.
- [4] H. F. Silverman. Processor reconfiguration through instruction-set metamorphosis. *IEEE Computer*, March 1993.
- [5] Mihai Budiu Srihari Cadambi Matt Moe Seth Copen Goldstein, Herman Schmit and R. Reed Taylor. PIPERENCH: A reconfigurable architecture and compiler. In *Computer*.
- [6] Guangming Lu Nader Bagherzadeh Fadi J. Kurdahi Eliseu M. C. Filho Ming-Hau Lee, Hartej Singh and Vladimir Castro Alves. Design and implementation of the morphosys reconfigurable computing processor. In *J. VLSI Signal Process. Syst.*
- [7] Markus Koester Mario Porrmann Jens Hagemeyer, Boris Kettelhoit. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms, ERSAs 2007*.
- [8] Michael Ullmann, Wansheng Jin, and Jurgen Becker. Hardware enhanced function allocation management in reconfigurable systems. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 3*, page 156.1, Washington, DC, USA, 2005. IEEE Computer Society.
- [9] Michael Hubner, Katarina Paulsson, and Jurgen Becker. Parallel and flexible multiprocessor system-on-chip for adaptive automotive applications based on xilinx microblaze soft-cores. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 3*, page 149.1, Washington, DC, USA, 2005. IEEE Computer Society.
- [10] *Virtex-II Pro and Virtex-II ProX Virtex-II Pro and Virtex-II Pro X FPGA User Guide*. Xilinx Inc., 28 March 2007.
- [11] *Virtex Series Configuration Architecture User Guide*. Xilinx Inc., 24 March 2003.
- [12] E. de la Torre Teresa Riesgo Yana E. Krasteva, Didier Joly. Virtex ii bitstream manipulation: Application to reconfiguration control systems. In *Proc. of 16th IEEE Intl. Conference on Field Programmable Logic and Applications*, pages 717–720, August 2006.
- [13] IBM corporation. *The CoreConnect Bus Architecture, white paper*. International Business Machines Corporation., 2004.
- [14] *Development System Reference Guide 8.1i*. Xilinx Inc., 2006.
- [15] Xilinx Inc. Two Flows of Partial Reconfiguration: Module Based or Difference Based. Technical Report XAPP290, Xilinx Inc., November 2003.
- [16] Xilinx Inc. *Early Access Partial Reconfiguration Guide*. Xilinx Inc., 2006.
- [17] S. Kelem. Virtex series configuration architecture user guide. *Xilinx XAPP151*, 2003.
- [18] Robert Rinker Fadi J. Kurdahi Dhananjay Kulkarni, Walid A. Najjar. Fast area estimation to support compiler optimizations in fpga-based reconfigurable systems. In *Proceedings of Field-Programmable Custom Computing Machines*, pages 239– 247, 2002.
- [19] W. Najjar J. Hammes R. Rinker M. Chawathe A. P. W. B ohm, B. Draper and C. Ros. One-step compilation of image processing algorithms to fpgas. In *Proceedings of Field-Programmable Custom Computing Machines*, pages 239– 247, 2001.
- [20] Inc. Xilinx. Microblaze processor reference guide. *Embedded Development Kit, EDK 8.2i. Xilinx User Guide v6.0*, June 2006.
- [21] Xilinx Inc. *Embedded Development Kit EDK 8.2i*. Xilinx Inc., 2006.
- [22] Edson L. Horta, John W. Lockwood, and David Parlour. Dynamic hardware plugins in an fpga with partial run-time reconfigurition. pages 844–848, 1993.
- [23] David E. Taylor, John W Lockwood, and Sarang Dharmapurikar. Generalized rad module interface specification of the field programmable port extender (fpx). *Washington University, Department of Computer Science. Version 2.0, Technical Report*.
- [24] Edson Horta and John W. Lockwood. Parbit: A tool to transform bitfiles to implement partial reconfiguration of field programmable gate arrays (fpgas). *Washington University, Department of Computer Science, Technical Report WUCS-01-13*, July 2001.