Politecnico di Milano
MicroLAB
HPPS Project Presentation

CITiES

A Harvard-based Processing Element for Partial Dynamic Reconfigurable Architectures

Tutor: M.D. Santambrogio

Project Author:
Alessio Montone
706983

A.A. 2006/07

Contents

1 Introduction **3**

2 Rationale **4**

3 The processing Element **5**

4 Bitstream Structure **7**
4.1 Virtex II Pro Configuration 7

5 The Proposed Methodology **7**
5.1 Mapping Procedure 8
5.2 marBram tool 11
5.3 Memory configuration bitstream 13

6 Experimental Results **14**

7 Conclusions **15**

Abstract

This project focuses on the Processing Element (PE) of a Multing Processing Elements SoC Architecture. This work will be part of the DRESD Project whose aim is proposing solutions for reconfigurable systems. Currently Field Programmable Gate Arrays (FPGA) are the state of the art of reconfigurable devices. Hence project's target devices have been chosen inside Xilinx's offer (Xilinx is the largest producer of programmable logic), particularly Virtex and Spartan FPGA's Families. Aim of this project is to propose, implement and test a PE based on a Harvard Architecture with a MicroBlaze soft processor in order to make PE deployable also on devices not having processors on silicon die. The largest part of the project consists of developing a tool for manipulating partial configuration bitstreams (i.e. the binary file configuring only a part of the FPGA). This tool will be able to initialize memory (data and code) information included in partial configuration bitstreams. Currently there are softwares that initialize memory information on complete configuration bitstream(i.e. bitstreams configuring the entire FPGA), but they do not work on partial configuration bitstreams. Considering the fact that a PE is only a system's part that can be modified and configured separately from each other, then it seems obvious the need of a such memory initialization tool.

1 Introduction

In recent years, the evolution of reconfigurable devices has brought to significantly increase their size, capacity and performance [8, 9]. In such a scenario, a larger number of complex components can be mapped at the same time into the same device. On of the emerging design pattern is a modular one called Multi Processing Element. According to this design pattern, the resulting architectures are made of several modules, each one implementing a different function, and connected each other with an application-dependent communication infrastructure. This project focuses on the Processing Element (PE) tailored for a partially dynamically reconfigurable architecture. Currently Field Programmable Gate Arrays (FPGA) are the state of the art of reconfigurable devices. Hence target devices have been chosen inside Xilinx's offer, particularly Virtex-II FPGA's Families. Hence aim of this paper is to propose a processing element containing a Harvard soft processor and such PE can be replace at runtime by another PE without preventing the rest of the system from correctly working. In section 2 will be described the rationale of this work providing some references to the state of the art, while in section 3 will be described the single processing element. In section 4 will be briefly described the configuration process of a Virtex II FPGA while in section 5 will be deeply

described the method used for the creation of a memory content configuration bitstream. At the end results will be provided in section 6 and conclusion will be drawn in section 7.

2 Rationale

With the growth of design methodologies and architectures, like happened in software engineering in the 80s, system-on-chip architectures are becoming less monolithic and more modular. Modularization allows the exploitation of code re-usage and simplify debug process of the entire architecture (top down compositional debug techniques can be used). These two components impacts directly on time to market for SoC-Architectures.

Multi-processing-element architectures are one of the proposed approaches answering to the increasing demand of modularization: in such architecture several peer modules (Processing Elements, PE) are linked together, each of these module provides a given functionality. Depending on the interconnections betweene the PEs classical architectures can be implemented: connecting in a sequential way the modules a pipelined architecture can be implemented, while connecting duplicated PEs with a mesh allows the exploitation of parallelism given under the assumption of input data independency.

The spread of scalable algorithms in several application field of computer engineering gives a further motivation for research in Multi PE architectures. Examples of such algorithms can be found in dymanic systems simulation (for physical of financial applications), biomedical applications (image processing, DNA and other molecular analysis). All these algorithms involves matrix calculation that, generally, are easily scalable, with respect to the number of PEs, by a matrix splitting.

In a such multi PEs SoC architectures partial dynamic reconfigurability can be exploited in order to create a more flexible system. Partial dynamic reconfigurability allows to change part of a systems without preventing the rest of the system from continuing the execution of the tasks. In a multi PEs architecture the most obvious reconfiguration is the replacement of a PE with another one. Hence according to the system requirements one (or more) processing element can be replaced by another in order to optimize an objective function (e.g. througput, response time and so on). The reconfiguration driver can take several inputs from the environment and the system and, applying a policy, decide which PEs will be changed and when.

Inside a complex system hardware-software codesign questions are often issued. In order to make more flexible the system and exploit Hw-Sw codesign

results, this project will describe a processing element containing a softprocessor. In such a way the single processing element can couple the flexibility of the software with the speed of the hardware. Hence the single PE will be able to exploit Hw-Sw codesign results, while in the past the entire system exploits such results. The presence of a software running on a processor allows to implements part of the functionality via software resulting in a simplier debugging and in a decreased time-to-market of the system.

One of the main drawback of this context can arise when several master components have to work at the same time on the same device, since it is necessary to individually manage their memory data to make each processing element independent from the others. In order to take advantages from the high flexibility of a reconfigurable system each component has to be individually configured with its memory (data and/or instruction), but this is not possible with standard tools for the design of reconfigurable systems, since they are able to create a configuration bitstream for the data configuration just in complete bitstreams, and not in partial ones. For this reason it is impossible to adopt these tools for the design of partially dynamically reconfigurable systems in which memory data has to be partially and dynamically changed.

Previous works in this area mainly focused their attention on the manipulation of bitstreams content on Xilinx FPGAs devices. Inside Virtex family documentation [2] 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 [8] and Virtex-5 [9]). Furthermore these documents do not explain how several BRAM Block create a contiguos 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 allow only to address frames (also according with [3]) and provide no 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. Having analyzed the state of the art reguarding bitstream manipulation in the next section will be described the proposed processing element.

3 The processing Element

The logical view of the proposed processing element is shown in figure 1. It is a standard architecture based on MicroBlaze soft processors.

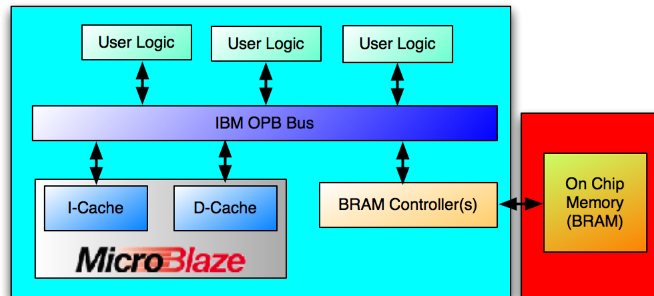

Figure 1: The processing element

The processors's bus is an OPB bus belonging to the IBM CoreConnect technology [4]. In the tipical configuration it is arbitrated, has a 32 bit address bus and 32 bit data bus and some control signals. The memory controller (or controllers) is also attached on this bus, hence code and data flows through such OPB bus.

Due to the standardized interface also user logic can be attached to the bus, hence processor can use them and hardware-software codesign results can be exploited. Tipically processors performs 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.

While it is needed a coupling of software flexibility and hardware speed, a configuration flexibility is also required. Thus this project proposes a novel approach for BRAM-Block content manipulation for Xilinx Virtex II and Virtex II Pro FPGAs families, [3]. It consists in the decoupling of the processing element logic configuration from the configuration of BRAM (i.e. code and data). Currently there are softwares that initialize memory information on complete configuration bitstream(i.e. bitstreams configuring the entire FPGA), but they do not work on partial configuration bitstreams. Considering the fact that a PE is only a system's part that can be modified and configured separately from each other, then it seems obvious the need of a such memory initialization tool.

The main issue is a mapping procedure that allows the creation of a bitstream able to configure only BRAM Blocks, leaving other logic unaltered. This procedure takes in input the data needed to fill the memory and the list of BRAM-Block that have to be used. Therefore, the proposed approach allows to decouple logic configuration from BRAM memory content configuration (e.g. one can change code executed by a processor without having to configure the processor logic again). Such a feature may result in the partitioning of configuration bitstream in two parts:

- a bitstream able to configure just the logic [1]

- a second bitstream able to configure just the BRAM content

Due to this approach 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).

4 Bitstream Structure

As introduced in Section 3, the approach proposed in this project allows the creation of a bitstream able to modify only BRAM-Block content without changing the other configurable element of the FPGA. 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.

4.1 Virtex II Pro Configuration

According to [3], Virtex II FPGAs are configured by columns [5, 6], as shown in Figure 2, where each bitstream column configures one physical device column which is divided in several frames [10]. Each bitstream column configures one physical IOBs, CLBs, BRAM-Block Interconnections or BRAM-Content column. Each column is divided in several frames 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 from 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.

5 The Proposed Methodology

Aim of the proposed methodology is to create a bitstream that configures only a subset of total BRAM Blocks leaving unchanged the other columns. This methodology has been implemented in our tool marBram. The core of the methodology is a mapping procedure. For each bit to be put in memory, i.e. a bit of data or even a bit of a code running on a processor, the mapping procedure returns the column identifier (Major Address) and the target position offset inside the column.

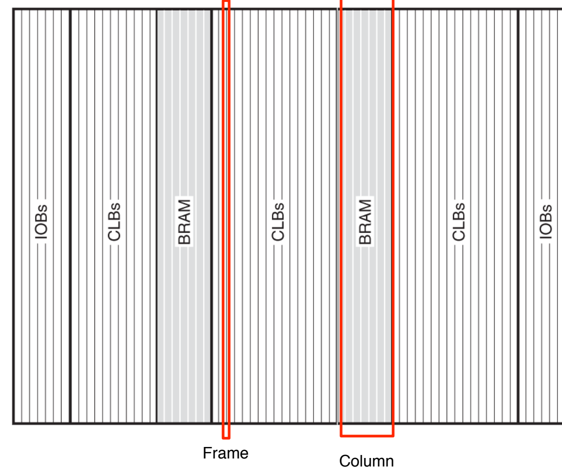

Figure 2: Virtex II Pro Family Configuration Bitstream Structure.

In this section the mapping procedure will be described and then marBram tool will be presented.

5.1 Mapping Procedure

Memory data is not consequently written inside BRAM-Content frames, but follows a well defined mapping procedure. 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 to 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 $k-\mathrm{th}$ 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 $j \mathrm{mod} b$, as shown in Figure 3).

Let consider now a single BRAM-Block j and all the segments associated to it. Let c be an ordered list of the bits assigned to the block and c_j be the $j-\mathrm{th}$ bit of this list (as shown in Figure 4).

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.

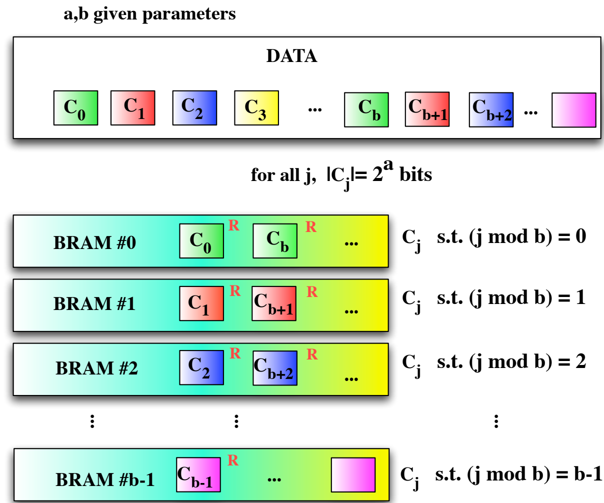

Figure 3: *Memory content splitting*. The red “R” stands for segment reversing.

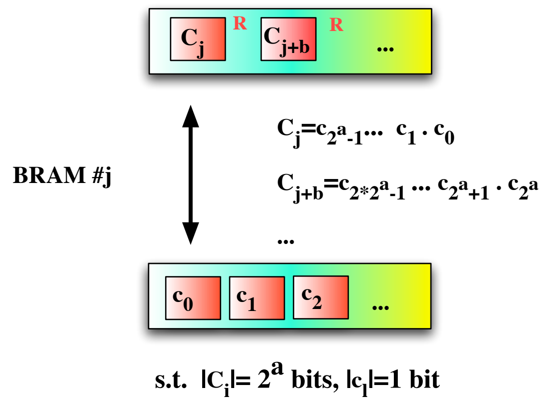

Figure 4: *Bits assigned to a given BRAM-Block.*

Before seeing the equations ruling the calculus of addresses and offsets, some parameters and terms will be now briefly described.

- **BRAM_Columns** it the total number of Block RAM columns in the FPGA die. It is a device dependent parameter and can be found on FPGA's datasheet.
- **BRAM_Rows** is the total number of Block RAM row in the FPGA die. It is a device dependent parameter and can be found on 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 on FPGA's datasheet.
- **Word_Len** is the length of a word in byte. 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 It 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 $byte_offset(c_j)$, of the bit where c_j has to be positioned.

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

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

and for constant arrays:

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

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

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

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

5.2 marBram tool

The marBram tool provides an implementation of the mapping procedure and of the entire proposed methodology allowing to obtain a BRAM configuration bitstream starting from memory data to be put into BRAMs. This tool takes in input the following data:

- BMM file: it describes how different BRAM-Blocks create a contiguous memory interval. It contains a, b parameters and also *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)

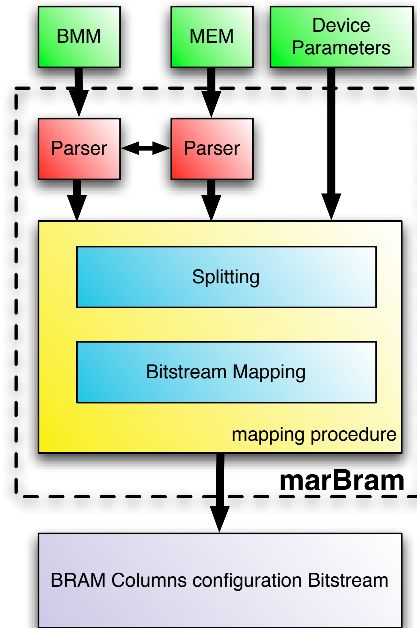

Figure 5: marBram tool data flow

- MEM file: it is a text file with memory data in hexadecimal notation using Verilog memory simulation syntax. This file can also be easily extracted from compiled code in order to fill in BRAM with code.

The output of marBram is a downloadable bitstream that configures only the content of BRAM Columns containing BRAM-Blocks specified inside the ELF file.

Figure 5 presents the flow followed by the marBram tool. MEM and BMM files go through the parsers, returning information to the core of the marBram tool 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 splitted during mapping procedure), b (the number of BRAM-Blocks), the position inside the FPGA (X and Y) of each BRAM-Blocks, base address and 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 position. 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 5). Gaps in memory data (i.e. address 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 pro-

cedure on different families or different memory content splitting procudure for different architectures). After the execution of the mapping procedure, columns and frames are wrapped by commands (according to Xilinx Documentation [3]). 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 5) in order to flush configuration pipelines. More than 90% of the commands overhead (see 6 for further details) are made of such padding frames. The final bitstream is written on the output file that is straghtforward downloadable on the target FPGAs, i.e. can be send to the FPGA through one of the configuration interfaces. Now the produced bitstream will be described

5.3 Memory configuration bitstream

According to Xilinx Documentation [3] partial dynamically reconfiguration bitstream have a well defined structure. The bistream is made of a sequence of commands and operands. Most of the commands sets 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 are characterized by the absence of global startup command and contains only information regaarding the startup process. In order to configure only BRAM columns the bitstream has to provide only the related commands. In figure is reported the structure of the bitstream generated by marBram.

$$
\begin{array}{rcl}
\langle Bitstream \rangle & \rightarrow & \langle align \rangle \, \langle CRCReset \rangle \, \langle DeviceID \rangle \\
 & & \langle WriteConfig \rangle \\
 & & (\langle FarAddress \rangle \, \langle FDRI \rangle \, \langle FrameData \rangle)^{+} \\
 & & \langle DefCRC \rangle \, \langle CRCReset \rangle \\
 & & \langle CRC \rangle \, \langle DefCRC \rangle \\
 & & \langle Detach \rangle \, \langle DummyWord \rangle^{4} \\
\langle align \rangle & \rightarrow & FFFFFFFF \; AA995566 \\
\langle CRCReset \rangle & \rightarrow & 30008001 \; 00000007 \\
\langle DeviceID \rangle & \rightarrow & 3001C001 \; xxxxxxxx \; \text{(ref. Datasheet)} \\
\langle WriteConfig \rangle & \rightarrow & 30008001 \; 00000001 \\
\langle FarAddress \rangle & \rightarrow & 30002001 \; 02 \, (MJA \cdot 2)_{8} 0000 \\
\langle FDRI \rangle & \rightarrow & 30004000 \; 3yyyyyyy \; \text{(Length in words)} \\
\langle DefCRC \rangle & \rightarrow & 0000DEFC
\end{array}
$$

$$\langle CRC \rangle \rightarrow 30000001\,0000DEFC$$
$$\langle Detach \rangle \rightarrow 30008001\,0000000D$$
$$\langle DummyWord \rangle \rightarrow 20000000$$

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

6 Experimental Results

The mapping procedure and marBram tool have been validated using several architectures, based on IBM CoreConnect Technology [4] and using either a PowerPC-405 hard processor or a MicroBlaze soft processor [11] (even if the proposed processing element is based on a MicroBlaze, the memory content mapping algorithm is compatible also with the on die PowerPC-405 processor). All the architectures have been developed with Xilinx Embedded Development Kit (EDK) [7]. Tests have been performed on Xilinx Virtex II Pro VP7 and VP20 and consisted in modifying only the BRAM Memory content (without modifying CLBs, memory interconnection et cetera) with the bitstream created using the marBram tool (giving in input different code and data segments) and resetting processor in order to start fetching the new code. In Table 1 a collection of results of the execution of marBram tool is provided.

Table 1: Results of the execution of marBram tool

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

It can be noticed that the size of the BRAM configuration bitstream depends only 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

and the data length that configure FPGAs, 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 column number of the reconfigurable device.

7 Conclusions

The approach proposed in this project has been proved to be a feasible solution for memory management in a system composed by a set of master components. Such approach allows to completely decouple the configuration of the logic (including soft processors) from the configuration of the on-chip memory content (code and data segments). Results presented in Section 6 have been used to validate the whole methodology and have shown that the overhead introduced by the marBram tool is very small with respect to the whole bitstream size. The tool has been tested, both with the Microblaze soft-processor and with the PowerPC-405 hard-processor, with a collection of bitstreams wich 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 tool 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 tool, 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 kind of master components or the data memories on which they have to operate.

References

[1] *Development System Reference Guide 8.1i*. Xilinx Inc., 2006.

[2] *Virtex Series Configuration Architecture User Guide*. Xilinx Inc., 24 March 2003.

[3] *Virtex-II Pro and Virtex-II ProX Virtex-II Pro and Virtex-II Pro X FPGA User Guide*. Xilinx Inc., 28 March 2007.

[4] IBM corporation. *The CoreConnect Bus Architecture, white paper*. International Business Machines Corporation., 2004.

[5] Xilinx Inc. Two Flows of Partial Reconfiguration: Module Based or Difference Based. Technical Report XAPP290, Xilinx Inc., November 2003.

[6] Xilinx Inc. *Early Access Partial Reconfiguration Guide*. Xilinx Inc., 2006.

[7] Xilinx Inc. *Embedded Development Kit EDK 8.2i*. Xilinx Inc., 2006.

[8] Xilinx Inc. Virtex-4 configuration user guide. Technical Report ug71, Xilinx Inc., January 2007.

[9] Xilinx Inc. Virtex-5 configuration user guide. Technical Report ug191, Xilinx Inc., February 2007.

[10] S. Kelem. Virtex series configuration architecture user guide. *Xilinx XAPP151*, 2003.

[11] Inc. Xilinx. Microblaze processor reference guide. *Embedded Development Kit, EDK 8.2i. Xilinx User Guide v6.0*, June 2006.

[12] E. de la Torre Teresa Riesgo Yana E. Krasteva, Didier Joly. Virtex ii bitstream manipulation: Application to reconfiguration control sytems. In *Proc. of 16th IEEE Intl. Conference on Field Programmable Logic and Applications*, pages 717–720, August 2006.