

# Data memory management in partial dynamically reconfigurable systems

A. Montone, Vincenzo Rana and M.D. Santambrogio  
DEI - Politecnico di Milano

*e-mail:* alessio.montone@dresd.org

*e-mail:* {rana, santambr}@elet.polimi.it

## ABSTRACT

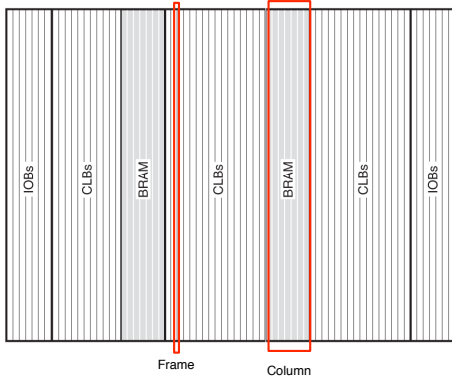
This paper aims at introducing a novel approach for the management of processor data memory in reconfigurable systems. The proposed approach allows the individual management of separated data and the dynamic update of the memory with a partial bitstream. By following this way it is possible to create a system in which several master components (e.g., soft-processors or hard-processors) can be dynamically configured and in which their memory can be dynamically changed. In the first section the reconfigurable scenario and its problems concerning to memory management will be introduced. The following sections will describe the state of the art and the development details of the tool that implements the proposed approach. Finally, a set of experimental results will be presented and conclusive remarks will be drawn.

## 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 permitting 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. In a such MPRE architectures, partial dynamic reconfigurability can be exploited in order to create a more flexible system, examples can be found in different works i.e., [3–5]. Partial dynamic reconfigurability allows to change part of a systems without preventing the rest of the system from continuing the execution of its tasks. In a Multi Processing Reconfigurable Elements (MPRE) 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. throughput, response time and so on). The reconfiguration driver can take several inputs from the environment and from the system's runtime conditions and, using different policies, it can decide which Processing Elements (PEs) will be changed and when. In such a scenario, a larger number of complex components can be mapped at the same time into the same device. 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. 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 [6] 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 [7]. Equations for Virtex II (Pro) are presented here, but they allow only to address frames (also according with [8]) 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. This paper proposes a novel approach for BRAM-Block content manipulation for Xilinx Virtex II and Virtex II Pro FPGAs families, [8]. It consists in 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 [9] and 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



**Figure 1: Virtex II Pro Family Configuration Bitstream Structure.**

(as previously underlined, memory content consists of code and data segments of a software running on a processor).

This paper is organized as follow. The next section presents the general bitstream structure, with particular attention to the Virtex II Pro FPGAs family configuration. Section 2 provides a short overview of the bitstream structure while Section 3 describes the proposed approach, that consists of a mapping procedure and a tool, the marBram tool, that implements the proposed procedure. Section 4 introduces a set of experimental results useful to validate the marBram tool and the whole proposed methodology. Finally, conclusions are drawn in Section 5.

## 2. BITSTREAM STRUCTURE

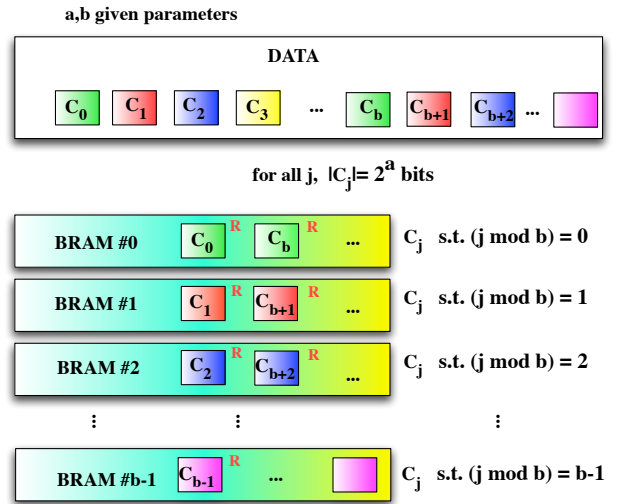
As introduced in Section 1, the approach proposed in this paper allows the creation of a bitstream able to modify only BRAM-Block content without changing the other configurable element of the FPGA. In order to understand how the approach is technically feasible, the structure of Virtex II - Pro FPGAs and of their configuration bitstreams will be briefly described in this section.

### 2.1 Virtex II Pro Configuration

According to [8], Virtex II FPGAs are configured by columns [10, 11], as shown in Figure 1, where each bitstream column configures one physical device column which is divided in several frames [12]. 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.

## 3. 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



**Figure 2: Memory content splitting.** The red “R” stands for segment reversing.

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

### 3.1 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  $k$ -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 2).

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  $j$ -th bit of this list. 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 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  $\text{byte\_offset}(c_j)$ , 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:

$$\begin{aligned} \delta(l) &:= \{6, 4, 2, 0, \\ &\quad -4, -6, -8, -10, \\ &\quad -16, -18, -20, -22, \\ &\quad -26, -28, -30, -32\} \\ \varphi(l) &:= \{3, 2, 1, 0, 4, 5, 6, 7\} \\ \gamma &= (\text{BRAM\_Rows} - \text{BRAM\_Y}) * 40 \\ &\quad + 80 + K \\ MJA &= \text{BRAM\_X} \\ MNA &= j \text{ div } 256 \\ \text{cpbo} &= (j \text{ div } 32) \text{ mod } 8 \\ \text{byte\_offset}(c_j) &= \gamma + \delta(j \text{ mod } 16) \\ &\quad + (((j \text{ div } 16) \text{ mod } 2) = 0)?0 : -1 \\ &\quad + MNA \times \text{Frame\_Len} \times \text{Word\_Len} \\ \text{bit\_offset}(c_j) &= +(((j \text{ div } 16) \text{ mod } 2) = 0)? \\ &\quad \varphi(\text{cpbo}) : 7 - \varphi(\text{cpbo}) \end{aligned}$$

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

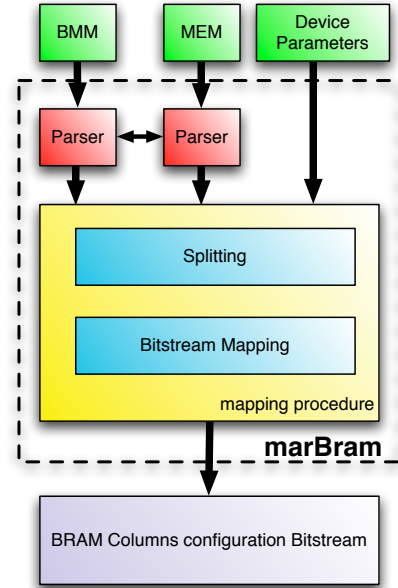


Figure 3: marBram tool data flow

### 3.2 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 three parameters: the *BMM* file used to describe how different BRAM-Blocks create a contiguous memory interval, the *Device Parameters* file which contains device dependent information and the *MEM* file containing all the 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 3 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 3). 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 procedure on different families or different memory content splitting procedure for different architectures).

After the execution of the mapping procedure, columns and frames are wrapped by commands (according to Xilinx Documentation [8]). 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 3) in order to flush configuration pipelines. More than 90% of the commands overhead (see 4 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 send to the FPGA through one of the configuration interfaces.

## 4. EXPERIMENTAL RESULTS

The mapping procedure and marBram tool 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 [14]. All the architectures have been developed with Xilinx Embedded Development Kit (EDK) [15]. 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                 | ≈ 1.5%            | 56 Kbytes      |
| VP7         | PowerPC-405 | 8            | 3                      | 203 ms                 | ≈ 1.5%            | 84 Kbytes      |
| VP7         | Microblaze  | 8            | 5                      | 263 ms                 | ≈ 1.5%            | 136 Kbytes     |
| VP20        | PowerPC-405 | 8            | 3                      | 248 ms                 | ≈ 1.5%            | 112 Kbytes     |
| VP20        | Microblaze  | 8            | 5                      | 326 ms                 | ≈ 1.5%            | 160 Kbytes     |
| VP20        | Microblaze  | 16           | 5                      | 326 ms                 | ≈ 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.

## 5. CONCLUSIONS

The approach proposed in this paper has been proved to be a feasible solution for memory management in a system composed by a set of master components. Results presented in Section 4 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 wick 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.

## 6. 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] Edson L. Horta, John W. Lockwood, and David Parlour. Dynamic hardware plugins in an fpga with partial run-time reconfiguration. pages 844–848, 1993.
- [4] 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.*
- [5] 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.
- [6] *Virtex Series Configuration Architecture User Guide.* Xilinx Inc., 24 March 2003.
- [7] 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.
- [8] *Virtex-II Pro and Virtex-II ProX Virtex-II Pro and Virtex-II Pro X FPGA User Guide.* Xilinx Inc., 28 March 2007.
- [9] *Development System Reference Guide 8.1i.* Xilinx Inc., 2006.
- [10] Xilinx Inc. Two Flows of Partial Reconfiguration: Module Based or Difference Based. Technical Report XAPP290, Xilinx Inc., November 2003.
- [11] Xilinx Inc. *Early Access Partial Reconfiguration Guide.* Xilinx Inc., 2006.
- [12] S. Kelem. Virtex series configuration architecture user guide. *Xilinx XAPP151*, 2003.
- [13] IBM corporation. *The CoreConnect Bus Architecture, white paper.* International Business Machines Corporation., 2004.
- [14] Inc. Xilinx. Microblaze processor reference guide. *Embedded Development Kit, EDK 8.2i. Xilinx User Guide v6.0*, June 2006.
- [15] Xilinx Inc. *Embedded Development Kit EDK 8.2i.* Xilinx Inc., 2006.