Journal of Universal Computer Science, vol. 19, no. 3 (2013), 301-324
submitted: 19/8/11, accepted: 28/1/13, appeared: 1/2/13 © J.UCS

Fast Self-Reconfigurable Embedded System on Spartan-3

Enrique Cantó, Mariano Fons, Francesc Fons
(University Rovira i Virgili, Tarragona, Spain
ecanto@etse.urv.es, {mariano.fons, francisco.fons}@estudiants.urv.cat)

Mariano López, Rafael Ramos
(Technical University of Catalonia, Barcelona, Spain
{mariano.lopez, rafa.ramos}@upc.edu)

Abstract: Many image-processing algorithms require several stages to be processed that cannot be resolved by embedded microprocessors in a reasonable time, due to their high-computational cost. A set of dedicated coprocessors can accelerate the resolution of these algorithms, although the main drawback is the area needed for their implementation. The main advantage of a reconfigurable system is that several coprocessors designed to perform different operations can be mapped on the same area in a time-multiplexed way. This work presents the architecture of an embedded system composed of a microprocessor and a run-time reconfigurable coprocessor, mapped on Spartan-3, the low-cost family of Xilinx FPGAs. Designing reconfigurable systems on Spartan-3 requires much design effort, since unlike higher cost families of Xilinx FPGAs, this device does not officially support partial reconfiguration. In order to overcome this drawback, the paper also describes the main steps used in the design flow to obtain a successful design. The main goal of the presented architecture is to reduce the coprocessor reconfiguration time, as well as accelerate image-processing algorithms. The experimental results demonstrate significant improvement in both objectives. The reconfiguration rate nearly achieves 320 Mb/s which is far superior to the previous related works.

Keywords: FPGA, Spartan-3, partial reconfiguration, embedded system, image-processing, reconfigurable coprocessor, hardware accelerator
Categories: B.6.1, B.6.3, B.6.m, C.1.3, C.1.m

1 Introduction

Complex algorithms, such as those used for extracting biometrics features (fingerprint, iris, face, etc.), require several image-processing stages. Their resolution is generally carried out by software, executing a program on a high-performance computer characterized by its high cost and power consumption. Embedded systems are usually built around a low-cost and low-power consumption microprocessor, which does not have adequate performance for processing intensive operations in a reasonable time. Thus, when these algorithms are ported to embedded platforms, they are usually modified to reduce their computational requirements. However, these modifications imply increasing the biometric error rates measured in terms of False Acceptation Ratio/False Rejection Ratio (FAR/FRR). A better approach is to accelerate the most time-consuming stages by using dedicated coprocessors, in order to improve the response time. The main drawback is that the more coprocessors there are, the more area is needed for their implementation. Usually, this could be a critical

issue in FPGAs, since the algorithm consists of several image-processing stages, each one executed by a coprocessor that only is active during a slot time, being the remainder on standby awaiting the conclusion of the previous stage. A reconfigurable coprocessor mitigates this drawback, since it can dynamically accommodate a new coprocessor that performs a different computation in the same area. This objective is achieved by reconfiguring a section of the FPGA by retrieving a bit-stream from a memory. The number of theoretical coprocessors, which can be mapped on the reconfigurable section, is limited by the capacity of the memory that stores the set of bit-streams (each one related to one of these coprocessors) without increasing the area needed to implement the overall system. Although it is out of the scope of this work, another advantage of reconfigurable systems is that power consumption can be reduced, since it decreases in FPGAs with smaller logic capacity. Their main disadvantage is the reconfiguration time overhead, which must be small enough in order not to significantly degrade the overall computational performance.

Usually, self-reconfigurable systems are mapped on partially reconfigurable FPGAs, such as Virtex-2/4/5 FPGAs, due to the design flow support and internal architecture. The main drawback of these Xilinx devices is their high cost when they are compared with low-cost families such as Spartan-3. Implementing self-reconfigurable systems on Spartan-3 is a great challenge, since it does not officially support partial reconfiguration, which is the reason why there are few research works published on this subject.

The main contributions of this paper are focused on reducing the reconfiguration time of a self-reconfigurable embedded system implemented in a Spartan-3, which accelerates image-processing algorithms by means of a dynamically reconfigurable coprocessor. In order to greatly reduce the reconfiguration time, the system embeds two new controllers, a fast reconfiguration controller and a high-bandwidth memory controller that performs the reconfiguration retrieving bit-streams directly from an external FLASH. Moreover, the memory controller also provides direct access from the coprocessor to an external SRAM, improving the execution time of image-processing stages. The paper also addresses the design flow followed to build a successful reconfigurable system on a Spartan-3.

Section 2 summarizes the main features of partial reconfiguration on Spartan-3 that have been considered during our system design. Section 3 presents the state of the art of self-reconfigurable embedded systems on Spartan-3 FPGAs, focusing on their main drawbacks. The next section describes the system architecture, including the memory and reconfiguration controllers developed. Section 5 overviews the design flow used to build the system. The experimental results are presented in Section 6, and the conclusions and future work are presented in the last two sections.

2 Partial Reconfiguration on Spartan-3

The section presents the limitations related to the partial reconfiguration of Spartan-3 that are taken into account in the proposed reconfigurable system. In contrast to FPGAs such as Virtex-2/4/5, Xilinx does not officially support partial reconfiguration on the Spartan-3 devices. At the design flow level, these devices are not supported by the PlanAhead with Partial Reconfiguration tool [Dorairaj, 05]. Additionally, at the

internal architecture level, Spartan-3 are characterized by coarse-granularity and non-glitchless reconfiguration, and they lack the internal reconfiguration port [Xilinx, 08].

The parallel SelectMAP and the serial JTAG are two external configuration ports available in Xilinx FPGAs that are used for programming the device from an external controller during the boot-up sequence. Moreover, high-end FPGA families, such as Virtex-2/4/5, provide the Internal Configuration Access Port (ICAP) which is used to perform partial reconfiguration at run-time. Although the ICAP circuitry is not available in Spartan-3, an embedded system mapped on the device can control the reconfiguration of the FPGA through some I/O pins that are externally connected to the SelectMAP or JTAG ports by simple wires.

A bit-stream is composed of a large set of configuration bits that are grouped in frames, and each frame is composed of a fixed number of 32-bit configuration words. The reconfiguration granularity is the minimum set of bits required to update the FPGA resources. The granularity on a Virtex-2 is a single column frame whose size is proportional to the number of rows of Configurable Logic Blocks (CLB). This feature only allows the reconfigurable sections to be arranged horizontally. Virtex-4/5 devices have a finer granularity which is independent of the device size. Since each frame configures a bit-wide column with a height of 16 CLBs, it is also possible to arrange reconfigurable sections vertically. By contrast, the Spartan-3 FPGAs are featured by a coarse reconfiguration granularity defined by a configuration column. Each configuration column is composed of a set of frames that configures different hardware resources, depending on the column type. A CLB column configures the CLBs from the top-side to the bottom-side of the FPGA, as well as the neighbouring routing lines and the associated Input/Output Blocks (IOB). The clock column configures the global routing lines that are usually dedicated to the distribution of the system clock, and it is physically located at the central position of the Spartan-3 columns. The coarse granularity of Spartan-3 means that the device must be reconfigured by configuration columns. Therefore, a reconfigurable system on Spartan-3 must be partitioned into sections that are horizontally arranged, since each section is constrained to the hardware resources that are configured by whole columns, and one of these sections contains the clock column.

The reconfiguration on Virtex-2/4/5 is free of glitches (or glitchless). The glitchless reconfiguration guaranties that if a configuration bit has the same value before and after the reconfiguration, the hardware resource controlled by this bit does not undergo any discontinuities in its operation. However, Spartan-3 is non-glitchless and configuration bits that are going to be reconfigured are initially cleared. Therefore, routing lines connecting a reconfigurable section to the rest of the system (including IOBs attached to external memory) are temporally disconnected and affected by glitches during the reconfiguration process. Moreover, if the central column of the FPGA is reconfigured, the clock signal, which is distributed by the global routing lines, is also temporally disconnected and affected by glitches.

Virtex-2 provides internal tri-state buffers and routing lines usually used in reconfigurable systems to ensure that the interconnecting busses between sections are unaltered between different configurations. In the rest of Xilinx FPGAs, including Spartan-3, the solution that is usually adopted is bus-macros.

3 State of the Art of Self-Reconfigurability on Spartan-3

This section summarizes the drawbacks of the previous related publications about self-reconfigurable embedded systems on Spartan-3 devices. The presented drawbacks are addressed in the proposed system.

The lack of official support for partial reconfiguration on Spartan-3 (in terms of design flow and internal architecture) has promoted a limited number of papers about reconfigurable systems mapped on this low-cost family. The previous research papers dealing with self-reconfigurable embedded systems on Spartan-3 have the following drawbacks.

The work by Gonzalez et al. [Gonzalez, 07][Gonzalez, 08] covered the design flow and architecture of a self-reconfigurable system for accelerating cryptographic applications. The hardware resources of the FPGA are partitioned in two sections: the dynamic section, which allocates a reconfigurable coprocessor, and the static section to implement the rest of the system (i.e. the microprocessor, memory controller and peripherals). The system is based on Microblaze, the soft-core microprocessor provided by Xilinx to build embedded systems. The Microblaze controls the reconfiguration process, driving a General Purpose Input Output (GPIO) peripheral which is externally connected to the SelectMAP port. While the reconfiguration process is running, the microprocessor retrieves the configuration bit-stream from an external DRAM. The I/O pins attached to the DRAM are allocated in the static section, in order to assure the proper communication with the external memory during the reconfiguration. Moreover, the central column is placed in the static section, in order to avoid the disconnection of the global routing lines that distribute the system clock due to the non-glitchless reconfiguration. Therefore, the dynamic section occupies a small area of the FPGA (less than 50%) which is a negative constraint when implementing coprocessors that may need many logic resources. The reconfigurable coprocessor is attached to Microblaze through dedicated links called Fast Simplex Links (FSL). The microprocessor has to execute the specific read/write instructions to exchange data with the coprocessor through the FSLs. In these systems, glitches do not affect the FSL operations, since the microprocessor stops communication with the coprocessor during the reconfiguration process.

The publications by Paulsson et al. [Paulsson, 07][Paulsson, 08] show an embedded system provided with an ICAP version. This system is designed to be externally connected to the JTAG serial port, although the reconfiguration time is significantly increased. Moreover, the system has the same drawbacks: a small area for the dynamic section, and the I/O pins and the clock column must be both allocated in the static section.

Research by Cantó et al. [Cantó, 08][Cantó, 08b] focused on the system design and design flow to map a self-reconfigurable embedded system whose bit-streams are retrieved from an external SRAM memory. The memory capacity of SRAMs is smaller than for DRAMs, however they provide simpler control and lower power consumption when they are in standby mode. The main contributions are that there are no constraints on the placement of I/O pins and that the dynamic section occupies more than 50% of the FPGA area. These characteristics were achieved by executing a reconfiguration routine on the microprocessor and distributing the clock in different routing lines for static and dynamic sections. The routine analyzes bit-streams from

the external SRAM and separates them into individual column frames. Each column is temporarily stored in the internal FPGA memory, i.e. the Block RAM (BRAM), in order to reconfigure the large dynamic section column by column. Therefore, the microprocessor communication with the internal BRAM is not affected during reconfiguration, while communication with the external SRAM is re-established once the reconfiguration of a column is completed. To improve the poor performance of the GPIO, the authors developed a faster controller able to drive the 8-bit wide SelectMAP port efficiently. The main disadvantages of this system are the time needed by the microprocessor to execute the routine and the large number of BRAMs used as temporal memory. Although the reconfiguration time is acceptable for many applications, it is desirable to accelerate it to improve the overall performance of more complex algorithms. Another important difference compared with the previous works is that the reconfigurable coprocessor is attached to the microprocessor using the On-chip Peripheral Bus (OPB), a bus shared with the rest of peripherals. During the reconfiguration, unexpected glitches generated in the OPB bus could hang-up the system. In order to prevent this problem, an isolation circuitry disables the OPB signals that arrive from the dynamic section during the reconfiguration.

We should also mention the Parallel Configuration Access Port (PCAP) and Compressed PCAP (cPCAP), two fast reconfiguration controllers for Spartan-3 published by Bayhar et al. [Bayhar, 08] [Bayhar, 08b]. These controllers are designed to read a bit-stream already loaded into the internal BRAM, and able to achieve a 400 Mb/s reconfiguration rate. However, they can only reconfigure a very small area of the FPGA due to the small capacity of the BRAM. Moreover, the controllers are stand-alone and cannot be attached to an embedded system; therefore, updating a bit-stream stored in the BRAM requires an external PC connected through JTAG to program the whole FPGA. There are unresolved issues in these reconfigurable systems, such as how to reconfigure larger bit-streams that cannot fit in the internal BRAM, or how to dynamically update BRAM contents without a FPGA programmer. A hypothetical embedded system attached to these controllers would significantly decrease the reconfiguration rate, due to the time the microprocessor devotes to retrieving a bit-stream portion from an external memory (or host) and to copying it in the BRAM, before enabling the PCAP reconfiguration.

None of the previous embedded systems permits direct access to an external memory from the reconfigurable coprocessor. As a consequence, the microprocessor has to execute a routine to read/write data from/to the memory and exchange it with the coprocessor through FSL or OPB. Image-processing algorithms usually require a large amount of data to be accessed from an external memory, and thus an architecture that permits direct access to the memory at high-bandwidth can accelerate the algorithm resolution. Moreover, previous systems must copy a bit-stream from a host to SRAM or DRAM before performing a reconfiguration.

This paper aims to improve the main drawbacks of previous works related to self-reconfigurable embedded systems. The properties of our proposed system can be summarized as follows:

1. The size of the dynamic section is larger than the static one, in order to increase the hardware resources available on the reconfigurable coprocessor.
2. The system clock is distributed differently for the static and dynamic sections, in order to avoid the clock disconnection for the static section

during the reconfiguration, and to prevent malfunction of the coprocessor due to a clock-skew in the larger FPGA area.

3. The reconfiguration rate has been improved to reduce the reconfiguration time overhead.
4. Bit-streams are retrieved from an external non-volatile memory (FLASH) during the reconfiguration, in order to save time f they were previously copied to the volatile memory (SRAM).
5. In order to accelerate the image-processing stages, the reconfigurable coprocessor accesses to the external SRAM through a high-bandwidth memory controller.

4 Architecture of the Self-Reconfigurable Embedded System

The section presents the architecture of the embedded system which is designed to reduce the reconfiguration time of the dynamic coprocessor. This way, the embedded system can rapidly switch the coprocessor architecture, in order to efficiently resolve the different stages of an algorithm.

The system, depicted in Fig. 1, is partitioned into two sections that are linked using a set of bus-macros. The bus-macros maintain the connectivity of the nets that attach the static and the dynamic sections, once the reconfiguration is completed. A bus-macro [Carvey, 09] is composed of two Look-Up Tables (LUT) interconnected through a routing line which crosses the partition between the sections. Each LUT is placed in a different section, and depending on the directionality of the bus-macro, one of the LUTs provides an input port and the other one provides the output port. The static section (instance sys) includes the microprocessor and peripherals, whereas the dynamic section (instance drip) includes the reconfigurable coprocessor. The static and dynamic sections occupy 34% and 66% of the area in a XC3S1500 device, respectively. This FPGA provides two columns of BRAM and 18x18 multipliers (MULT18x18). Each BRAM+MULT18x18 column is placed near the left or the right edges of the device [Xilinx, 08c], thus half the BRAM and MULT18x18 resources belong to the static section, and the other half belong to the dynamic section. The clock column belongs to the larger section, since it is located at the central position of the FPGA columns. Hence, to avoid the system clock disconnecting from the microprocessor during the non-glitchless reconfiguration, the input clock pin (clk) was changed from the default location in the dynamic section to another pin located in the static section. The clock is distributed separately on local low-skew routing lines in the static section (sys_clk_local), and on global routing lines (drip_clk_global) in the dynamic section through a BUFGMUX buffer.

In order to perform the self-reconfiguration, the SelectMAP interface of the FPGA is linked through wires to the devoted ports of the system (rcf_*). The system is also attached to an external SRAM to allow the microprocessor to execute large programs. In addition, the system is also connected to a FLASH memory that stores the set of bit-streams of the reconfigurable coprocessor. In order to prevent the microprocessor executing the reconfiguration routine [Canto, 08], which increases reconfiguration time, the I/O pins linked to the external memory (emc_*) are placed in the static section. The rest of the I/O pins, i.e. input reset (rst) and the serial

transmission/reception (TX, RX), can be placed in the dynamic sections, transferring the signals towards the static section through bus-macros.

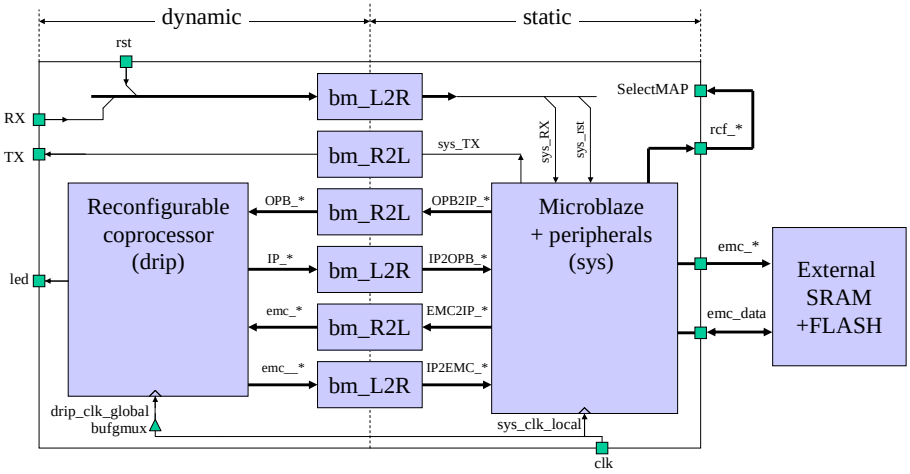

Figure 1: Top-level architecture of the self-reconfigurable embedded system

The static section, depicted in Fig. 2, allocates the Microblaze, peripherals and busses. The peripherals UART, MDM and Timer are provided by Xilinx's Embedded Development Kit (EDK), and they are connected as slaves of the OPB for debugging and testing purposes. The Instruction and Data Local Machine Busses (ILMB, DLMB) connect the microprocessor with the internal BRAM. Instead of using the OPB External Memory Controller (OPB-EMC) provided by the EDK, we developed a faster LMB-EMC, a dedicated EMC-Bus and its arbiter to permit time-multiplexed access to the external memory from the EMC-Bus masters.

Our ReConFiguration (RCF) and Direct Memory Access (DMA) controllers are connected as masters of the EMC-Bus, which allows direct access to external memory. They can be controlled by the microprocessor, since they are also linked as slaves of the OPB. The reconfigurable coprocessor can also access external memory directly through the LMB-EMC, in the same way as the RCF and DMA controllers, to accelerate the resolution time of image-processing algorithms. Therefore, the architecture provides three EMC-Bus masters (RCF, DMA and drip) that can access the external memory through the LMB-EMC controller.

The static and dynamic sections are isolated by using specific interfaces, which avoids the harmful effect of glitches on busses that can produce the improper operation of the system during the reconfiguration.

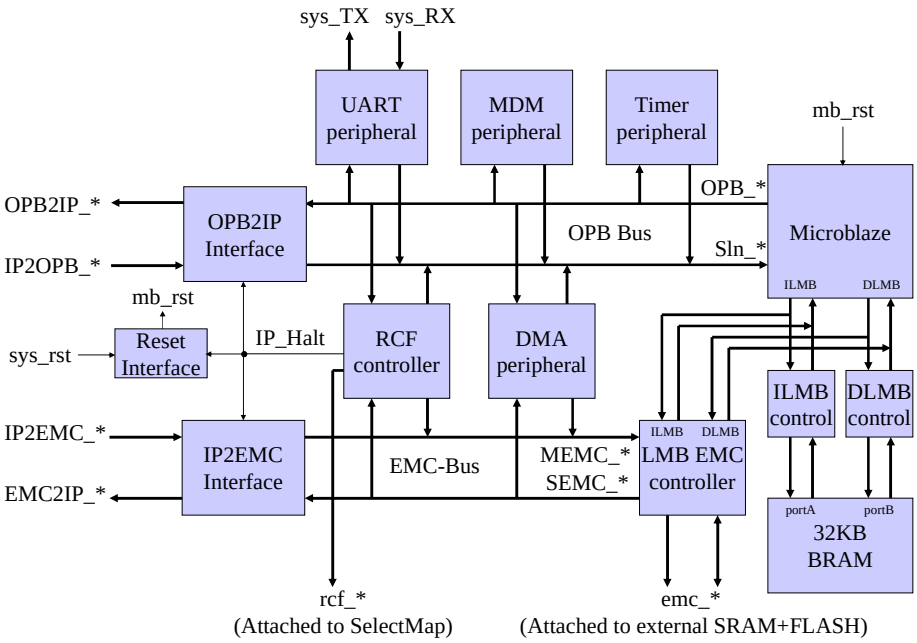

Figure 2: Architecture of the static section

During the reconfiguration, the RCF controller asserts the IP_Halt signal which drives the isolation interfaces (see Fig. 2). The isolation interface of the OPB (OPB2IP Interface) is depicted in Fig. 3(a). The main functionality of this interface is to de-assert the arriving OPB signals (IP2OPB_*) from the dynamic section when the IP_Halt is asserted, in order to prevent unexpected OPB glitches that would hang-up the system. It provides two additional functionalities: holding the dynamic section in reset state during its reconfiguration (signal OPB2IP_Rst), and incorporating a partial address decoder (signal OPB2IP_Select) in order to reduce the number of bus-macros needed by the OPB address bus (signal OPB2IP_Addr). Fig. 3(b) shows the isolation interface of the EMC-Bus (IP2EMC Interface), which is simpler since it only de-asserts the start request signal to memory access from the reconfigurable coprocessor (Memc_strtreq(2)) when the IP_Halt is asserted. The rest of the EMC-Bus signals (Mecm_strack, Memc_* and Semc_doutack*) are de-asserted by the bus arbiter, as shown in Fig. 3(d), since it controls the priority of accesses to external memory from the three EMC-Bus masters. The reset I/O pin (rst) is placed in the dynamic section and transferred to the static section (sys_rst) through a bus-macro (see Fig. 1); hence, there is also a very simple isolation circuit (Reset Interface) to prevent an undesired reset (signal mb_rst) due to the glitches during the reconfiguration, as depicted in Fig. 3(c).

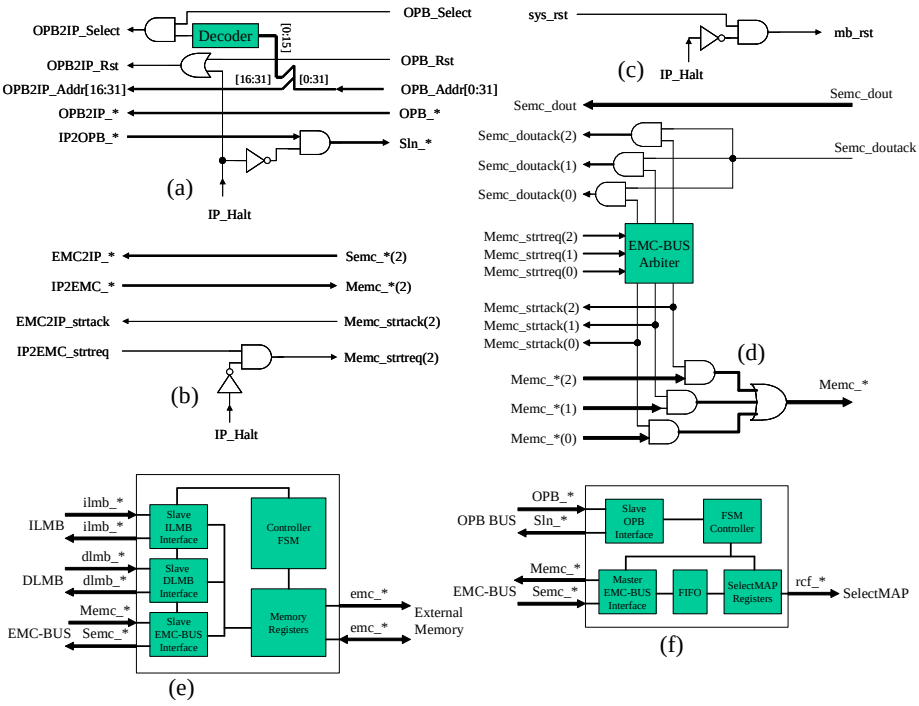

Figure 3: Isolation interfaces of the OPB (a), EMC-Bus (b), and reset signal (c). Schematic of the EMC-Bus arbiter (d), the LMBLMB-EMC) (e) and the RCF (f)

4.1 The LMB External Memory Controller (LMB-EMC)

This subsection presents the memory controller which connects the system with the external SRAM and FLASH memories. Usually, Microblaze accesses external memory through the OPB memory controller, which is not as fast as the LMB due to the more sophisticated bus handshake. In order to improve the speed to access external memory from the microprocessor, the new memory controller shares the LMB busses with the internal BRAM memory controllers.

Microblaze is connected to the internal BRAM memory through dedicated DLMB/ILMB busses and BRAM controllers. The LMB busses are designed to link Microblaze with the internal dual-port synchronous BRAM, and they allow reading/writing in 1 T_{CLK} (system clock cycles) along with simultaneous access to data and instruction memory. The Xilinx EDK OPB-EMC facilitates designing embedded systems attached to an external memory, due to the very limited capacity of the internal BRAM. However, programs executed on Microblaze from an external SRAM undergo a large decrease in speed, due to the poor performance of the OPB and the OPB-EMC. We designed a new external memory controller (LMB-EMC) which optimizes the number of clock cycles required to read/write data from/to the asynchronous memory. It provides three interface channels: DLMB, ILMB and EMC-Bus. The DLMB and ILMB channels are used by the Microblaze, and the EMC-Bus

channel is used by the rest of the system. The LMB-EMC is composed of the channel interfaces, the registers that drive the external memory, and a Finite State Machine (FSM) which performs the control tasks, as depicted in Fig 3(e).

Figure 4 (top) depicts the waveforms when the OPB-EMC is configured to fit the characteristics of the Spartan-3 development kit [Avnet, 04] clocked at 40 MHz (T_{CLK} =25 ns), which uses a 12-ns asynchronous 32-bit wide SRAM [Cypress, 03]. These waveforms show one data writing cycle (data d2 written to the memory address a2) between two reading instruction cycles (data d1 and d3 read from the memory addresses a1 and a3, respectively). It depicts two groups of signals: the OPB signals (denoted as opb_*) and the I/O pins connected to the external memory (denoted as emc_*). In order to start a new memory access, Microblaze waits for the completion of the previous OPB access, checking if the transfer acknowledge signal (opb_xferack) is asserted at every rising edge of the clock signal (clk). At the beginning of a read cycle, Microblaze requests an instruction read from the internal BRAM through the ILMB. If the address is not in the BRAM range, the instruction read is requested to the external memory controller through the OPB, asserting the select signal (opb_select) and driving the address bus (opb_abus). Then, the OPB-EMC drives the SRAM address bus (emc_addr) and enables the read signal (emc_noe). The asynchronous SRAM drives the data bus (emc_data) with the valid read data after 12 ns. Finally, the OPB-EMC stores the read data into a register which drives the OPB data bus (opb_dbus), and asserts the transfer acknowledge signal (opb_xferack) to indicate the end of the OPB access cycle. A read cycle takes 12 T_{CLK} (or 10 T_{CLK} if there is a previous write cycle), of which 4 T_{CLK} are devoted to reading the word from the SRAM, and the rest are used to handshaking and registering data between ILMB, OPB and OPB-EMC. A write cycle to external memory is quite similar, but Microblaze starts requesting the data to be written into the BRAM through the DLMB. Since the address is not in the BRAM range, the data is transmitted to the external memory controller through the OPB data bus (opb_dbus). Then, the OPB-EMC drives the address bus (emc_addr) and enables the SRAM write signal (emc_nwe) meanwhile it drives its data bus (emc_data) during more than 12 ns. The write cycle takes 8 T_{CLK}, of which 3 T_{CLK} are devoted to writing the word into the SRAM, and the other 5 T_{CLK} are devoted to handshaking between DLMB, OPB and OPB-EMC.

Figure 4 (bottom) depicts the same three SRAM accesses as the previous case (two instruction reads and a data write) through the LMB-EMC. It shows three groups of signals: the ILMB signals (denoted as ilmb_*), the DLMB signals (denoted as dlmb_*), and the I/O pins connected to the external memory (denoted as emc_*). In order to start a new memory access, the Microblaze must acknowledge the completion of the previous access by checking the assertion of the ready signal of the ILMB or DLMB (ilmb_ready or dlmb_ready) at every rising edge of the clock (clk). The Microblaze starts requesting an instruction read, driving the ILMB address bus (ilmb_abus) and asserting its strobe signal (ilmb_addrstrobe). In a similar way, the microprocessor requests a data write asserting the strobe signal (dlmb_addrstrobe) and driving the address bus (dlmb_abus) and the data bus (dlmb_dbus). The LMB-EMC drives the external memory I/O pins (emc_*) in order to read or write data at the requested address and asserts the ILMB/DLMB ready signal (ilmb_ready/dlmb_ready) when it completes. A write access requires 2 T_{CLK}, while a

read access requires just 1 T_{CLK} for the same SRAM. The new memory controller can read a new 32-bit word (4 bytes) from SRAM every clock cycle (40 MHz), therefore it can achieve a read bandwidth of 160 MB/s (40 MHz*4 bytes/1 T_{CLK}), which is much superior to the 13.3 MB/s (160 MB/s /12) to 16 MB/s (160 MB/s /10) of the OPB-EMC provided by the EDK.

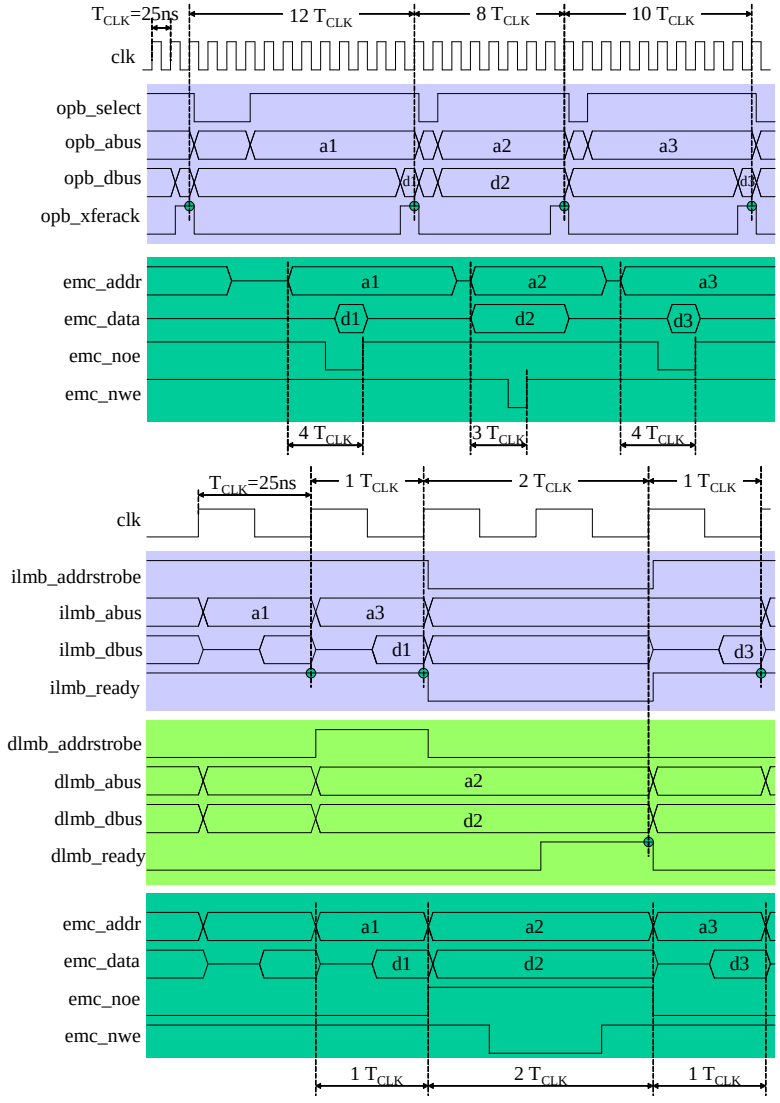

Figure 4: Read and write access to an external SRAM (12 ns) from Microblaze (40 MHz), using the OPB-EMC (top) and the LMB-EMC (bottom) controllers

The Xilinx EDK also provides the MultiChannel (MCH) OPB-EMC module to accelerate the execution of programs from an external SRAM when Microblaze incorporates cache memory. However, our embedded system provided with the LMB-EMC is faster and it occupies less FPGA area, since it does not dedicate BRAM and logic resources for implementing the microprocessor's cache memory. Moreover, the implementation of cache memory for the coprocessor would increase the number of resources needed as well as the design complexity. However, our system architecture permits efficient access to an external memory from the dynamic coprocessor through the LMB-EMC.

On the other hand, the LMB-EMC provides fast page reading from a FLASH memory, a feature not available in the (MCH) OPB-EMC offered by the EDK, which improves the reconfiguration time when bit-streams are retrieved from this memory. In order to achieve the fast page reading, the LMB-EMC decreases the number of T_{CLK} required to read a new word from FLASH if it is in the same page of the previous word read. An internal register stores the address of the last completed reading from the FLASH, and the LMB-EMC compares the address of any read access from FLASH against the stored address, detecting when the page remains unchanged. Our development board is provided with an Intel StrataFlash J3 [Intel, 05] which requires 120 ns for the first word reading from a new page, and 25 ns for the following word readings within the same page (a page contains 4 words). Hence, as the embedded system is arranged with a 40 MHz clock, the retrieval of the initial word takes 5 T_{CLK}, whereas each of the following 3 words needs only 2 T_{CLK}. Thus, an averaged read cycle takes 2.75 T_{CLK} when data is sequentially retrieved from FLASH. Therefore, our system achieves a read bandwidth of 58.2 MB/s from this non-volatile memory.

4.2 The Reconfiguration Controller (RCF)

This subsection presents the controller which performs the reconfiguration of the dynamic coprocessor. The reconfiguration time is reduced, since the controller drives the SelectMAP interface according to the bit-stream which is retrieved from the external memory, independently of the microprocessor.

The reconfiguration task executed by the microprocessor is simplified and it consists on writing in the control registers of the RCF for setting the start address and size of a bit-stream. Then, the RCF retrieves the bit-stream from the SRAM/FLASH through the LMB-EMC, and drives the SelectMAP interface according to the configuration words. The RCF is composed of several building blocks, as depicted in Fig. 3(f). The Microblaze writes on the control registers of the RCF through the OPB slave interface. The RCF accesses the external memory, where the bit-stream is stored, through the master interface of the EMC-Bus. The RCF stores a new configuration word (32-bit wide) into a register which slices it on bytes, in order to drive synchronously four times the data bus (8-bit wide) of the SelectMAP. Therefore, the RCF takes 4 T_{CCLK} (configuration clock cycles) to write a single configuration word to the SelectMAP. The maximum recommended frequency of the input CCLK is 50 MHz to avoid non-continuous data loading to the SelectMAP [Xilinx, 08b].

The RCF has a dedicated First Input First Output (FIFO) buffer which stores a queue of configuration words to increase the Reconfiguration Rate (RR). The RCF

stores the last configuration word retrieved from memory into the FIFO, and concurrently drives the SelectMAP according to the first available word in the queue. The RR is limited by the time required to carry out these two actions being its theoretical expression defined in Mb/s by:

$$RR(Mb/s) = 32 \cdot min\left\{ \frac{f_{CCLK}(MHz)}{4}, \frac{f_{CLK}(MHz)}{N_{32b-READ}} \right\} \quad (1)$$

where f_{CCLK} and f_{CLK} are the CCLK and system clock frequencies, respectively. The $N_{32b\text{-}READ}$ is the averaged number of T_{CLK} cycles needed to read a configuration word (32-bit wide) from the memory. The $f_{CLK}/N_{32\text{-}READ}$ and $f_{CCLK}/4$ are the throughputs (expressed in 32-bit words per second) of reading configuration words from memory and driving them to the SelectMAP, respectively. In our system $f_{CCLK}=f_{CLK}=40$ MHz, since it provides a single clock domain to avoid the additional complexity of the communication channels required to synchronize data [Varela, 08]. The $N_{32b\text{-}READ}$ for the SRAM or FLASH of the development board are 1 and 2.75 T_{CLK}, respectively. By substituting these data in (1), the system achieves a RR of 320 Mb/s in both cases.

Figure 5 shows the initial bit-stream retrieval from the external SRAM (top) or from the FLASH (bottom) (signals denoted as emc_*), along with the SelectMAP control lines that come from the RCF controller (denoted as rcf_*). The RCF starts reading the bit-stream words (d1, d2) at sequential addresses (a1, a2) from the external memory through the LMB-EMC until de FIFO is filled up. The first configuration word stored in the FIFO (d1) is transferred to the SelectMAP controller which slices it into bytes (d1.1, d1.2, d1.3, d1.4) and drives the 8-bit data bus (rcf_d) synchronously with the configuration clock (rcf_cclk). Concurrently, the RCF requests to the LMB-EMC the next word retrieval (d3) at the address (a3), taking 1 T_{CLK} to read it from the SRAM. In order to improve the bit-stream retrieval from FLASH, the LMB-EMC takes advantage of the fast page reading. The first word from a page is read in 5 T_{CLK}, and each of the remainder three words takes 2 T_{CLK}. The average time taken to read a configuration word from the SRAM or FLASH is smaller than the 4 T_{CLK} required for driving the SelectMAP interface. Although the SRAM is faster than the FLASH, the RR achieved is 320 Mb/s in both cases, once the FIFO buffer is filled up. The reconfiguration rate is limited by the SelectMAP interface, instead of being constrained by the bit-stream retrieval from the external memory. Therefore, it is unnecessary copying the bit-streams from the non-volatile FLASH to the SRAM during the boot-up sequence, in order to accelerate the reconfiguration.

5 Design Flow of the Self-Reconfigurable Embedded System

This section covers the design flow which is used to implement the reconfigurable embedded system on Spartan-3. The implemented system must accomplish some requirements about the partitioning and the distribution of the clock network due to the reconfiguration features of the Spartan-3. Moreover, the design flow uses a custom tool for merging partial layouts, since the Xilinx tools do not support the modular design on Spartan-3.

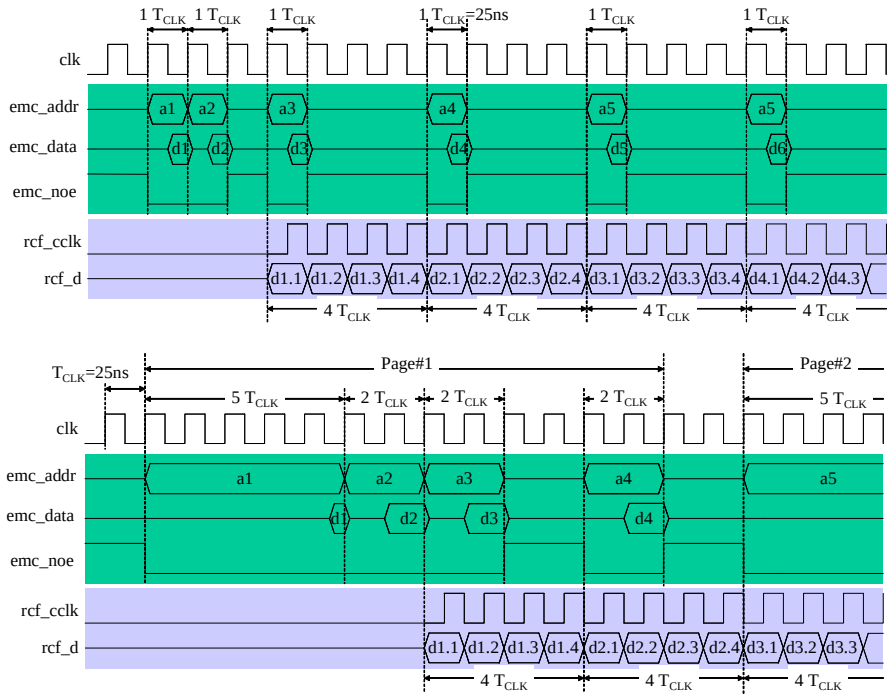

Figure 5: Initial bit-stream retrieval from the external SRAM (top) and FLASH (bottom) and the SelectMAP control

As we mentioned previously in Section 2, PlanAhead with the Partial Reconfiguration patch neither support Spartan-3, so it was necessary to use a design flow based on the Integrated Software Environment (ISE) by Xilinx. The flow executes the ISE tools in modular design mode [Xilinx, 04], but some steps are modified due to our system requirements. The modular design starts from the netlist files of the modules that build the reconfigurable system. Therefore, the design flow starts synthesizing the modules, it continues implementing the layouts using modular design, and it finishes generating the bit-streams, as depicted in Fig. 6 (top).

The reconfigurable system is composed of several modules. The first module corresponds to the static section of the embedded system (according to Fig. 2), and it is synthesized using the EDK flow to get the static module netlist (sys netlist). The dynamic module netlists are the set of coprocessors that can be mapped in the dynamic section. A netlist file is obtained for each dynamic module with a synthesizer (such as Xilinx XST or Leonardo) from its VHDL description. Figure 6 depicts the design flow for two dynamic modules: dummy and copro1. The dummy coprocessor is an empty design (its use is explained during the bit-stream generation), and copro1 is an image-processing coprocessor. Finally, the modular design requires a system top-level netlist (top netlist) which is synthesized from a VHDL file. The top-level VHDL describes the system which is composed of two black-boxes (the static and dynamic modules) interconnected using bus-macros (according to Fig. 1). It also

instantiates the I/O pins and the BUFGMUX buffer which is used to distribute the system clock on global routing lines.

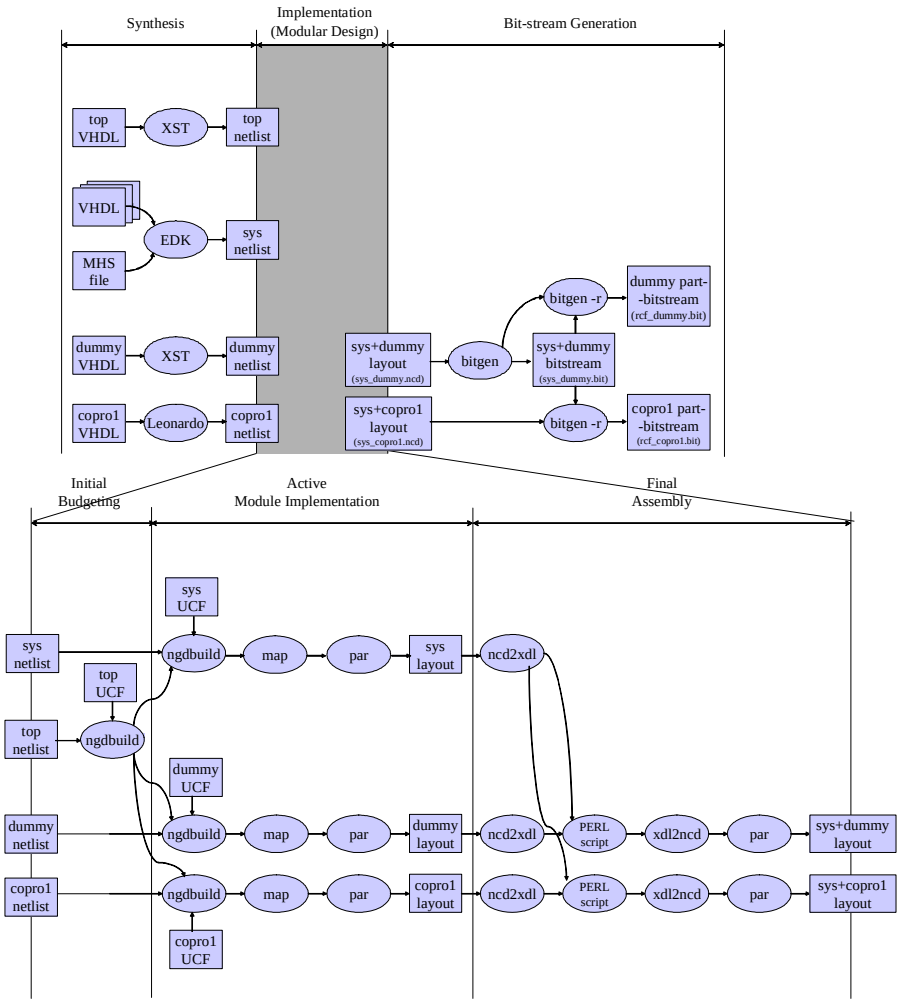

Figure 6: Steps of the design flow (top): synthesis, implementation, and bit-stream generation. Phases of the implementation in modular design mode (bottom): initial budgeting, implementation of modules, and final assembly.

There are three phases in the modular design, as depicted in Fig. 6 (bottom). The first phase is the initial budgeting, which describes the top-level design as a set of interconnected modules and their associated constraints. The second phase is the active module implementation that computes the partial layout of each module separately. The last phase is the final assembly, which merges partial modules to build the complete system layout.

The initial budgeting phase implements the top-level layout which is composed of the two modules, the bus-macros, the I/O pins and the BUFGMUX buffer. It runs the ngdbuild tool to annotate the placement of the top-level components declared in the User Constraints File (UCF) [Xilinx, 07b]. Code 1 shows the main constraints for the XC3S1500 FPGA which consists of an array of 104-width and 128-height of CLB slices, and two columns of 16-height BRAM+MULT18x18 resources. The first two sections of the UCF declare the FPGA resources reserved to the dynamic and the static modules (instances drip and sys, respectively). The modules are horizontally arranged, since the first 68 whole columns of CLB slices (left side of the FPGA) are devoted to the dynamic module, and the rest of columns to the static one. It also dedicates a whole BRAM+MULT18x18 column to each of the modules. The third section of the UCF sets the placement of the bus-macros along the edges of the partition. The last section of the UCF sets the placement of the input clock pin (clk) in the static section, and the placement of the BUFGMUX (instance bufgmux_drip_clk) which distributes the clock signal in the dynamic module (net drip_clk_global) using the global routing lines. Finally, the UCF configures the routing of the clock in the static module (net sys_clk_local) on low-skew routing lines, in order to prevent the malfunction of the static module due to the incremented delays of the local routing lines.

```
INST "drip"            AREA_GROUP="DYNAMIC";
AREA_GROUP "DYNAMIC"   RANGE=SLICE_X0Y0:SLICE_X67Y127;
AREA_GROUP "DYNAMIC"   RANGE=RAMB16_X0Y0:RAMB16_X0Y15;
AREA_GROUP "DYNAMIC"   RANGE=MULT18X18_X0Y0:MULT18X18_X0Y15;
...
INST "sys"             AREA_GROUP="STATIC";
AREA_GROUP "STATIC"    RANGE=SLICE_X68Y0:SLICE_X103Y127;
AREA_GROUP "STATIC"    RANGE=RAMB16_X1Y0:RAMB16_X1Y15;
AREA_GROUP "STATIC"    RANGE=MULT18X18_X1Y0:MULT18X18_X1Y15;
...
INST bm1               LOC=SLICE_X67Y118;
INST bm2               LOC=SLICE_X67Y100;
INST bm3               LOC=SLICE_X67Y94;
...
NET clk                LOC=AB26;
INST bufgmux_drip_clk  LOC=BUFGMUX1;
NET sys_clk_local      USELOWSKEWLINES;
```

Code 1: Main constraints of the top UCF file

The second phase is the active module implementation which computes the layout of each module and its connection with the top-level components, running the MAPping (map) and the Placement And Routing (par) tools [Xilinx, 07]. This phase also requires annotating a UCF file, composed of the top-level UCF plus the particular constraints for the implemented module. The output of this phase is the partial layout of the static module (sys layout) and the partial layout of the dynamic modules (layouts dummy and copro1, in Fig. 6), according to the same top-level constraints.

The last phase of the modular design is the final assembly to compute the complete layout, which merges the partial layouts of the static module and one of the dynamic modules. The assembly is executed for every dynamic module that can be

mapped on the reconfigurable coprocessor. Unfortunately, the final assembly phase cannot be completed using the ISE tools for Spartan-3 devices, because the top-level components (the global clock net, the BUFGMUX, the bus-macros, and IOBs) are declared in both partial layouts. Another undesired behaviour in the layout of the dynamic module is that the par tool routes the global clock net (drip_clk_global) on local lines instead of global lines. In order to solve these two problems, we developed a new PERL script based on the one developed by Gonzalez et al. [Gonzalez, 07], but modified for our system requirements. The modified script is able to place the IOBs either in static or dynamic partitions, and to devote a large FPGA area to the dynamic section, since it permits the clock column to be allocated in the dynamic section. The script cannot interpret layout files given in the proprietary format Native Circuit Description (NCD), but it can deal with the format Xilinx Description Language (XDL). Hence, it is necessary to execute a translator program before and after its execution (ncd2xdl and xdl2ncd tools, respectively). A layout file is composed of a list of instances of FPGA components (LUT, IOB, BUFGMUX, BRAM, MULT18x18) and interconnection nets, including the configuration and placement details. The script basically performs the following steps:

1. Delete the declaration of the drip_clk_global net in the partial layout of the static module, since it is already declared in the partial layout of the dynamic module.
2. Clear the configuration of the drip_clk_global net in the partial layout of the dynamic module, since the par tool routed it on local routing lines.
3. Delete the top-level nets and instances that are placed in the dynamic module, but they are already declared in the partial layout of the static module.
4. Delete the top-level nets and instances that are placed in the static module, but they are already declared in the partial layout of the dynamic module.
5. Add a prefix onto the names of nets and instances that are declared in the partial layout of the dynamic module, in order to prevent name duplicities in the next step.
6. The merged layout is composed by simply copying the list of instances and nets of the partial layouts of the two modules.

The merged layout is not entirely completed because the configuration of the drip_clk_global net is cleared due to the PERL script. Thus, the par tool is executed for the merged layout, in order to distribute the drip_clk_global net on the global routing lines that are driven by the BUFGMUX. The layout of each dynamic module is merged with the same layout of the static module. The output implementation using the modular design tools is a set of complete layouts in NCD files (sys_dummy.ncd and sys_copro1.ncd, in Fig. 6).

Finally, in order to generate the set of partial bit-streams necessary to reconfigure the dynamic section, we executed the bitgen tool [Xilinx, 08b]. The first execution of bitgen generates the bit-stream of the complete layout (sys_dummy.bit), which consists of the layouts of the static and the dummy modules (sys_dummy.ncd). This bit-stream is used to boot up the FPGA, and is obtained by executing the following command line:

```
bitgen -w –d –g Binary:yes –g Persist:yes sys_dummy.ncd sys_dummy.bit
```

where the Persist option is set to yes to keep the SelectMAP port enabled after the device initialization.

The partial bit-stream of the dynamic module copro1 (rcf_copro1.bit), necessary to reconfigure the dynamic section, is obtained executing the bitgen tool using the following command:

```
bitgen -w -d -g Binary:yes -g Persist:yes -g ActiveReconfig:yes
-g PartialMask0:000001fffffffd -g PartialMask1:1 -g PartialMask2:1
-r sys_dummy.bit sys_copro1.ncd rcf_copro1.bit
```

where the –r sys_dummy.bit option is used to compute the partial bit-stream from the layout composed of the static and the copro1 modules (sys_copro1.ncd). It is also necessary to set the ActiveReconfig option to yes to permit the partial reconfiguration. The options PartialMask0, PartialMask1 and PartialMask2 generate the partial bit-stream that corresponds to the CLB and BRAM+MULT18x18 columns assigned to the dynamic section. The same procedure is executed to create the partial bit-stream of the dummy module (rcf_dummy.bit), which is used to void the reconfigurable coprocessor.

The rest of the dynamic modules, and their partial bit-streams, are obtained following the same steps used to obtain the partial bit-stream of copro1. This set of partial bit-stream files are programmed into the FLASH memory. Once the Spartan-3 is booted-up, the embedded system can retrieve a partial bit-stream from the non-volatile memory to reconfigure the dynamic section when it is required by the microprocessor.

Figure 7 (left) shows the complete layout when the dynamic section maps the copro1. As it can be seen, the dynamic section is larger than the static one, providing a large number of free hardware resources available to map complex dynamic modules. Both sections are interconnected by the bus-macros, and each one contains one of the BRAM+MULT18x18 columns. The clock column (the central column of the FPGA), which configures the global routing lines, belongs to the dynamic section, and the clock input pin is placed into the static section. The clock is distributed in the static section on local routing lines (sys_clk_local), as depicted in Fig. 7 (right-top). In the dynamic section, the clock is distributed using the global routing lines (drip_clk_global) through the BUFGMUX, as presented in Fig. 7 (right-bottom).

6 Experimental Results

The experimental results were obtained by implementing a fingerprint feature extraction algorithm [Fons, 07]. The fingerprint images are 512*280 pixels (8-bit grey levels). The algorithm executes three different image-processing stages: segmentation, normalization plus filtering, and field orientation. The results were obtained using the AVNET Spartan-3 Development Kit [Avnet, 04], which allocates an XC3S1500 FPGA, and the SRAM and FLASH memories.

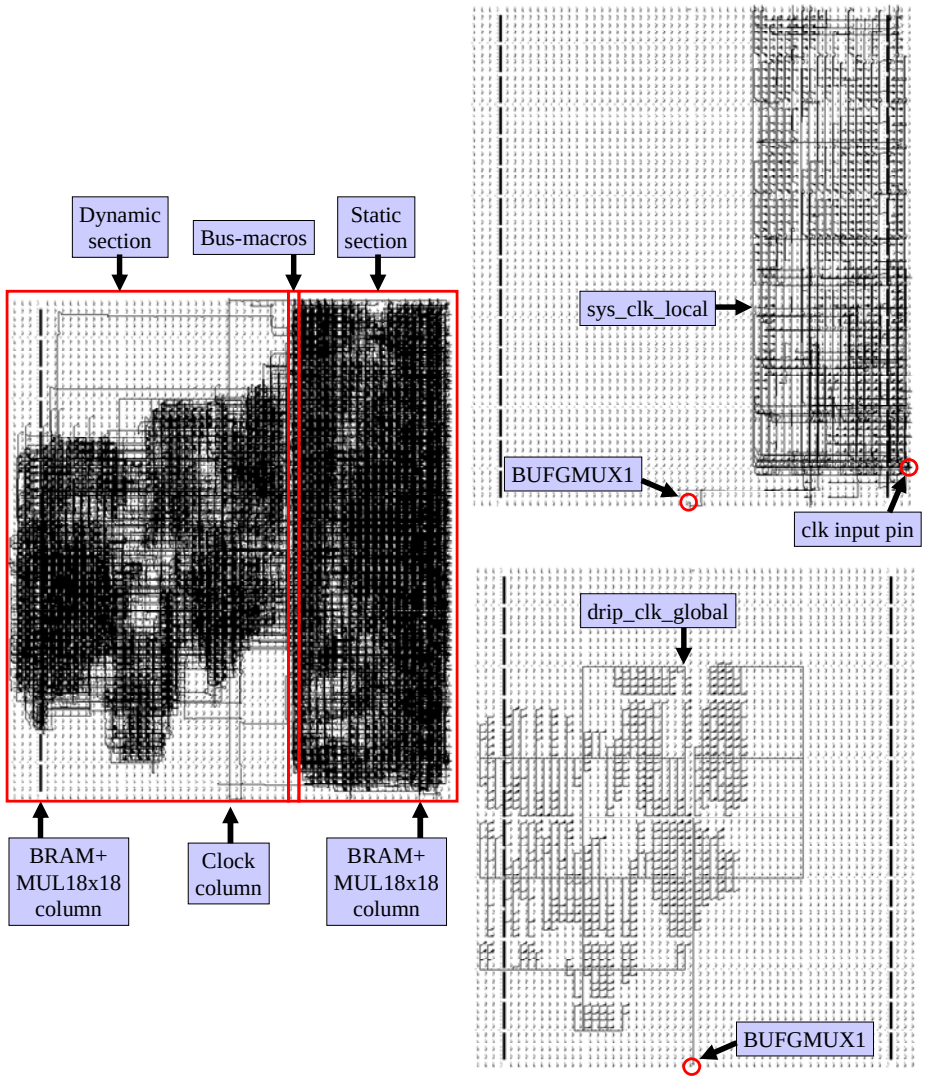

Figure 7: Layout of the static section and one coprocessor (left), and the clock distribution in the static section (right-top) and in the dynamic section (right-bottom)

The first set of experimental results tests the performances of our LMB-EMC, in terms of speed and area. The LMB-EMC is compared against the two SRAM controllers provided by the Xilinx EDK: the OPB-EMC and the MCH OPB-EMC. The algorithm is implemented in software and executed by the Microblaze. The memory size required by the software is larger than the capacity of the internal BRAM, so the program is stored in the external SRAM. We built three different

embedded systems depending on the memory controller. Table 1 shows the execution time and the synthesis results of each one. Our LMB-EMC controller significantly reduces the number of registers due to the design optimization and the LMB interface. The LMB-EMC achieves a great speed improvement compared with the OPB-EMC, as it is about 9 times faster. When the system is designed using the MCH OPB-EMC controller, Microblaze implements cache logic on some additional CLBs, and 10 BRAMs are devoted to cache memory (8 KB for data +8 KB for instructions). The LMB-EMC is about 11% faster and requires 70% less area when compared against the MCH OPB-EMC. Moreover, using our controller, Microblaze is synthesized without the cache, saving 10% of CLBs dedicated to the cache logic and the BRAMs devoted as cache memory. In addition, the main advantage of the LMB-EMC is that coprocessors do not need to incorporate cache logic and memory to speed-up the processing time from external SRAM, which simplifies their design and devoted area.

		OPB-EMC	MCH OPB-EMC	LMB-EMC
Execution	Segmentation (s)	6.1	0.65	0.63
	Norm+Filter (s)	79.3	9.37	8.65
	Field Orient. (s)	13.1	1.86	1.37
	Total time (s)	98.5	11.9	10.7
Synthesis	CLB Slices	196	616	185
	Flip-flops	323	730	171
	LUTs	135	705	347
	Frequ. (MHz)	136	71	125

Table 1: Execution time on Microblaze with the different SRAM controllers, and their synthesis results

Table 2 shows the reconfiguration rate of our system when a bit-stream is retrieved from the external FLASH, and compares it with the reconfigurable embedded systems described in Section 3 that retrieve bit-streams already loaded in a SRAM or DRAM. It also shows the measured reconfiguration time for our reconfigurable coprocessor, in which the bit-stream size of the dynamic section is 409 KB (66% of the area of the XC3S1500). In our proposed system the measured reconfiguration rate is quite close to the theoretical 320 Mb/s, demonstrating the high efficiency of the LMB-EMC and RCF controllers. The reconfiguration rate achieved is also better than other self-reconfigurable embedded systems mapped on higher cost devices, such as the 41 Mb/s [Braun, 08] on a Virtex-2, or the 26.2 Mb/s [Llamocca, 09] on a Virtex-4.

	[Gonzalez,08]	[Paulsson,08]	[Canto,08]	Proposed system
RR (Mb/s)	16.1	2	36.3	319.8
Time (ms)	203	1636	90	10.2

Table 2: Reconfiguration rate and time (409 KB bit-stream)

Table 3 shows the execution time of the image-processing stages when they are implemented as hardware coprocessors. It also reports the number of occupied CLB slices, BRAM and multiplier blocks. As can be seen, the execution time is greatly improved compared with the Microblaze execution time, since the coprocessors access to the SRAM directly and execute several computations in parallel. The last two rows of this table show the execution time and area of the three coprocessors when they are statically implemented or dynamically mapped on the reconfigurable coprocessor. The static implementation requires that the three coprocessors are attached to the rest of the system; therefore, the total area is the sum of the individual coprocessor areas. The processing time is also the sum of the processing times, since a computation stage cannot start until the previous one has been completed. In the dynamic implementation, the area needed by the reconfigurable coprocessor can be constrained by the maximum area of the individual coprocessors, while the processing time is exactly the same as that of the static implementation with the addition of the reconfiguration time of three reconfigurations. The self-reconfigurable embedded system accelerates the processing time by about 145 times compared with the software execution on the embedded system using Microblaze with the cache memory. The processing time is quite competitive even if it is compared with the execution on a high-performance PC (Intel Core Duo 2GHz) which takes 16.7 ms.

	Time (ms)	CLB Slices	BRAM	MULT18x18
Segmentation	11.7	2489	2	4
Norm+Filter	23.6	6519	4	4
Field Orient.	12.0	2468	2	8
STATIC	50.8	11476	8	16
DYNAMIC	81.4	6519	4	8

Table 3: Execution time and area of the three coprocessors, and their static and dynamic implementation

7 Conclusions

This work demonstrated that the low-cost family Spartan-3 of Xilinx is able to implement fast self-reconfigurable embedded systems to accelerate the computational stages of image-processing algorithms. A reconfigurable coprocessor, which maps a set of coprocessors multiplexed in time, speeded up the processing of the stages

involved in algorithms. The number of such coprocessors is limited by the external FLASH which allocates the set of partial bit-streams, but not by the device size. The proposed architecture included a new external memory controller (LMB-EMC) that permits direct access to SRAM from the reconfigurable coprocessor, in order to improve computational performance. Moreover, the reconfiguration time was reduced by designing a new reconfiguration controller (RCF) which efficiently retrieves bit-streams from FLASH memory through the LMB-EMC. The reconfiguration rate of the presented system achieved 319.8 Mb/s, widely outperforming the experimental results obtained from the other previous related works. Therefore, despite the reconfiguration time required by the stages, the reconfigurable coprocessor accelerated 145 times the computational time of a fingerprint feature extraction algorithm.

8 Future Work

The reconfiguration rate is not limited by the retrieval of bit-streams from the external memory, but rather by the SelectMAP interface. In order to improve the reconfiguration rate of the system, the configuration clock should be increased. A straightforward way of achieving this aim is to increase the system clock to 50 MHz to obtain the maximum RR (400 Mb/s). However, we could not complete the design due to the timing problems in the placement and routing of Microblaze. An embedded system with two clock domains can solve the problem, since the system and the SelectMAP can run at 40 MHz and 50 MHz respectively. However, the interface between clock domains will complicate the design. Moreover, the RCF can run at higher frequencies if the data loading into SelectMAP is non-continuous, although it complicates the RCF design.

The static section of our system allocates the Microblaze configured with an FPU (Floating Point Unit). The FPU accelerates floating-point operations executed by the microprocessor, although it occupies a large area of the static section. A new FPU designed to be mapped on the reconfigurable coprocessor will allow the static section to be reduced, and consequently a larger area can be assigned to the dynamic section. The FPU could be dynamically mapped on the reconfigurable coprocessor when Microblaze requires computing a large number of floating-point operations.

Another point of interest is the study of the system's power consumption. The CMOS static power has a great impact on advanced submicron technologies that increases according to the silicon area occupied by larger FPGAs. The reconfigurable system can map a large number of coprocessors multiplexed in time, reducing the FPGA size and, hence, the static power. The analysis of the dynamic power is more complicated. This consumption is incremented due to the reconfigurations of the dynamic section. However, on the other hand, the size of the clock distribution network decreases in smaller devices, and it is one of the most important terms of dynamic power in FPGA technologies [Wang, 09]. Moreover, dynamic reconfiguration permits the reconfiguration of the clock managers [Paulsson, 09], which can be used to dynamically scale the clock frequency to fulfil the power requirements.

Acknowledgements

The authors would like to thank Ministerio de Economía y Competitividad of Spain, for financial support under grant TEC2012-38329-C02-02.

References

[Altera, 08] Altera Corp., "40-nm FPGA Power Management and Advantages", White Paper 01059, December 2008

[Avnet, 04] AVNET Design Services, "Xilinx Spartan-3 Development Kit. User's Guide", Oct. 2004

[Bayhar, 08] S. Bayhar, A. Yurdakul,: Dynamic Partial Self-Reconfiguration on Spartan-III FPGAs via a Parallel Configuration Access Port (PCAP), Proc. of the HiPEAC Workshop on Reconfigurable Computing, Jan.2008.

[Bayhar, 08b] S. Bayhar, A. Yurdakul,: Self-Reconfiguration on Spartan-III FPGAs with compressed Partial Bitstreams via a Parallel Configuration Access Port (cPCAP) Core, 4th Conference on PhD Research in Microelectronics and Electronics, pp. 137-140, June 2008

[Braun, 08] L. Braun, K. Paulsson, H. Kröner, M. Hübner, J. Becker, "Data Path Driven Waveform-Like Reconfiguration", Field Programmable Logic and Applications (FPL'2008), pp. 607-610, Sept. 2008

[Cantó, 09] Cantó Navarro, E., López García, M., Fons Lluís, F.,Ramos Lara, R., "Sistema Embebido de Rápida Auto-Reconfiguración sobre Spartan-3", IX Jornadas de Computación Reconfigurable y Aplicaciones (JCRA'2009), pp. 183-192, Sept. 2009

[Cantó, 08] E. Cantó, M. López, F. Fons, "Self-Reconfiguration of Embedded Systems Mapped on Spartan-3", 4th International Workshop on Reconfigurable Communication Centric SoCs (ReCoSoC'2008), pp. 117-123, July 2008

[Cantó, 08b] E. Cantó, M. López, F. Fons, "Self-Reconfigurable Embedded System on Spartan-3", Field Programmable Logic and Applications (FPL'2008), pp. 571-574, Sept. 2008

[Carvey, 09] J. M. Carver, R. N. Pittman, A. Forin, "Automatic Bus Macro Placement for Partially Reconfigurable FPGA Designs", Proc. of the ACM/SIDGA International Symposium on Field-Programmable Gate Arrays (FPGA'2009), pp. 269-272, Feb. 2009

[Cypress, 03] Cypress Semiconductor Co., "CY7C1062AV33", Doc#38-05137, Feb. 2003

[Dorairaj, 05] N. Dorairaj, E. Shiflet, M.Gossman, "PlanAhead Software as a Platform for Partial Reconfiguration", XCELL Journal, 4th Quarter 2005

[Fons, 07] F. Fons, M.Fons, E. Cantó, M. López, "Flexible Hardware for Fingerprint Image Processing", Conf. on PhD Research in Microelectronics and Electrics, Vol 10.3, Sept. 2007

[Gonzalez, 07] I. Gonzalez, E. Aguayo, S. López Buedo, "Self-Reconfigurable Embedded Systems on Low-Cost FPGAs", IEEE Micro, Vol.27, Issue 4, pp.49-57, July-Aug. 2007

[Gonzalez, 08] I. Gonzalez, S. López-Buedo, F.J. Gómez-Arribas "Implementation of Secure Applications in Self-Reconfigurable Systems", Microprocessors and Microsystems, Vol.31, Issue 1, pp.23-32, Feb. 2008

[Intel, 05] Intel, "Intel StrataFlash (J3) Datasheet", March 2005

[Llamocca, 09] D. Llamocca, M. Pattichis, A. Vera, "A Dynamically Reconfigurable Parallel Pixel Processing System", Field Programmable Logic and Applications (FPL'2009), pp. 462-466, Aug-Sept. 2009

[Paulsson, 07] K. Paulsson, M. Hübner, G. Auer, M. Dreschamann, L. Chen , J. Becker, "Implementation of a Virtual Configuration Access Port (JCAP) for Enabling Partial Self-Reconfiguration on Xilinx Spartan III FPGAs", Field Programmable Logic and Applications (FPL'2007), Aug. 2007

[Paulsson, 08] K. Paulsson, U. Viereck, M. Hübner, J. Becker, "Exploitation of the External JTAG Interface for Internally Controlled Configuration Readback and Self-Reconfiguration of Spartan 3 FPGAs", IEEE Computer Society Annual Symposium on VLSI (ISLSI'2008), pp.304-309, April 2008

[Paulsson, 09] K. Paulsson, M. Hübner, J. Becker, "Dynamic Power Optimization by Exploiting Self-Reconfiguration in Xilinx Spartan 3-Based Systems", Microprocessors and Microsystems, Ed. Elsevier, Vol. 33, Issue 1, pp.46-52, Feb. 2009

[Varela, 08] J. Varela, J. Freijedo, J. Andina, C. Leong, J. P. Teixeira, "Robust Solution for Synchronous Communication Among Multi Clock Domains", IEEE Asia Pacific Conference on Circuits and Systems, pp. 1107-1110, Dec. 2008

[Xilinx, 04] Xilinx Inc., "Two Flows for Partial Reconfiguration: Module Based or Difference Based", XAPP290, 2004

[Xilinx, 07] Xilinx Inc., "Development System Reference Guide", 2007

[Xilinx, 07b] Xilinx Inc., "Constraints Guide", 2007

[Xilinx, 08] Xilinx Inc., "Spartan-3 Advanced Configuration", XAPP452, June 2008.

[Xilinx, 08b] Xilinx Inc, "Spartan-3 Generation Configuration User Guide", UG332, July 2008.

[Xilinx, 08c] Xilinx Inc, "Spartan-3 FPGA Family. Data Sheet", DS099, June 2008

[Wang, 09] Q. Wang, S. Gupta, J. Anderson, "Clock Power Reduction for Virtex-5 FPGAs", Proc. of the ACM/SIDGA International Symposium on Field-Programmable Logic Arrays (FPGA'2009), pp. 13-22, Feb. 2009