# Teaching Microprocessors Design Using FPGAs

Joaquín Olivares, José Manuel Palomares, José Manuel Soto, Juan Carlos Gámez
Dept. of Computer Architecture, Electronics, and Electronics Technology
University of Córdoba
Córdoba, Spain
olivares@uco.es

*Abstract*— **Microprocessors is a typical subject within the Computer Architecture field of scope. It is quite common to use simulators in practical sessions, due to the complexity of its contents. In this paper a new methodology based on practical sessions with real devices and chips is proposed. Simple designs of microprocessors are exposed to the students at the beginning, rising the complexity gradually toward a final design with a multiprocessor integrated in a single FPGA chip. Finally, assessment results are shown.**

***Keywords: Learning Experiences; Laboratory Experiences; Teaching Engineering; Computer Architecture***

## I. INTRODUCTION

Traditional laboratory practical sessions to teach microprocessors are based on simulators, in this paper an experience implementing real microprocessors is shown. The main purpose is to encourage the student interest and to improve the assessment. This proposal is useful for many subjects within several engineering degrees. This activity was carried out in particular in the Microprocessors subject, in the degree in Computer Sciences at the University of Córdoba.

In most cases, computer architecture has been taught with software simulators [1], [2]. These simulators are useful to show: internal values in registers, memory accesses, cache fails, etc. However, the structure of the microprocessor is not visible, and students are not aware learning a real processor. Recently [3], [4], digital design is being teaching using real Programmable Logic Devices (PLD), showing it attractiveness for the students. Also recent works shown how the learning throw projects and games based on FPGAs are very useful [5].

In this work, a methodology for easy design and real implementation of microprocessors is proposed, in order to provide students with a user-friendly tool. Simple designs of microprocessors are exposed to the students at the beginning, rising the complexity gradually toward a final design with two processors integrated in an FPGA; each of which has an independent memory system, and are intercommunicated with a unidirectional serial channel. Furthermore, an introduction to the architecture of a T1 SUN OpenSparc system with 8 processors, 4 thread/processor plus one MicroBlaze is introduced at the end of the semester while students are working on their projects, this final seminar is useful because students are encouraged when see that a high performance parallel architecture is suitable of being implemented on a single FPGA Virtex5 [6].

In this paper the methodology to design and implement a microprocessor or multiprocesors is presented. To illustrate it with high detail and in a useful way, how to design the most complex practical session is shown. In section I, other methodologies based on simulators are referenced. The software platform used to implement real processors is presented in section II. Features of MicroBlaze processor are introduced in section III. The course practical content is described in section IV. To illustrate how a processor is designed and implemented, section V presents a brief guide of how to build a biprocessor, this is the most complex practical session. Finally, conclusions are presented in section VI.

## II. SOFTWARE TOOL

The Xilinx Platform Studio (XPS) is used to design MicroBlaze processors. XPS is a graphical IDE for developing and debugging hardware and software. XPS simplifies the procedure to the users, allowing them to select, interconnect, and configure components of the final system. Dealing with this activity, the student learns to add processors and peripherals, to connect them through buses, to determine the processor memory extension and allocation, to define and connect internal and external ports, and to customize the configuration parameters of the components. Once the hardware platform is built, the students learn many concepts about the software layer, such as: assigning drivers to peripherals, including libraries, selecting the operative system (OS), defining processor and drivers parameters, assigning interruption drivers, establishing OS and libraries parameters.

Students have deeper documentation available in [7], [8], and other useful resources recommended are [9], and, [10].

An embedded system performed with XPS can be summarized as a conjunction of a Hardware Platform (HWP) and a Software Platform (SWP), each defined separately.

### *A. The Hardware Platform*

The HWP is described in the Microprocessor Hardware Specification (MHS) file; it contains the description of the system architecture, the memory map and the configuration parameters. HWP can be defined as one or more processors connected to one or more peripherals through one or more buses. The definition of the activity follows this sequence:

- To add processors and peripherals.
- To connect them through buses.

978-1-4244-6571-2/10/$26.00 ©2010 IEEE

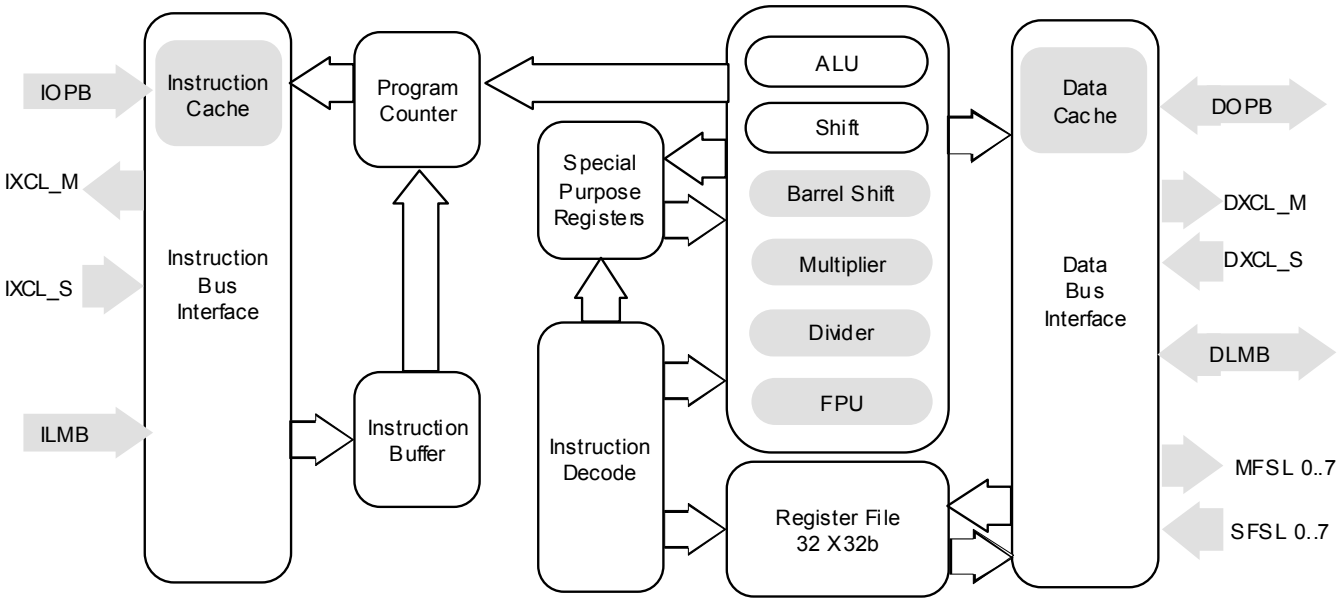


Figure 1. MicroBlaze architecture. In gray, reconfigurable components

- To determine the processor memory allocation.
- To define and connect internal and external ports.
- To customize the configuration parameters of the components.

### B. *The Software Platform*

The SWP is described in the Microprocessor Software Specification (MSS) file; it contains the description of drivers, component libraries, configuration parameters, standard input/output devices, interruption routines and other software features. The sequence of activities needed to define the SWP is the following:

- To assign drivers to peripherals.
- To assign interruption drivers.
- To establish OS and libraries' parameters.

## III. THE MICROBLAZE PROCESSOR

MicroBlaze is a 32-bit specific purpose processor developed by Xilinx in VHDL. It can be parameterized using XPS to obtain an *à-la-carte* processor. It is a RISC processor, structured as a Harvard architecture with separated data and instruction interfaces.

MicroBlaze components are divided into two main groups depending on their configurability as shown in Fig.1.

Some fixed feature components are:

- 32 general purpose registers sized 32-bit each.
- Instructions with 32 bits word-sized, with 3 operands and 2 addressing modes.
- 32 bits address bus.
- 3-stage Pipeline.

Some of the most important configurable options are:

- An interface with OPB (On-chip Peripheral Bus) data bus.
- An interface with OPB instruction bus.
- An interface with LMB (Local Memory Bus) data bus.
- An interface with LMB instruction bus.
- Instruction cache.
- To include EDK libraries.
- To select the operative system (OS).
- To define processor and drivers' parameters.
- Data cache.
- 8 Fast Simplex Link (FSL bus) Interfaces.
- CacheLink bus support.
- Hardware exception support.
- Floating Point Unit (FPU).

## IV. PRACTICAL DESIGNS

Practical sessions introduce gradual learning, allowing the fast design based on previous sessions. Essential problems in hardware programming will be raised:

- Hyperterminal serial communication.
- Using IO ports.
- Memory controller.
- Interruption routines and priority.

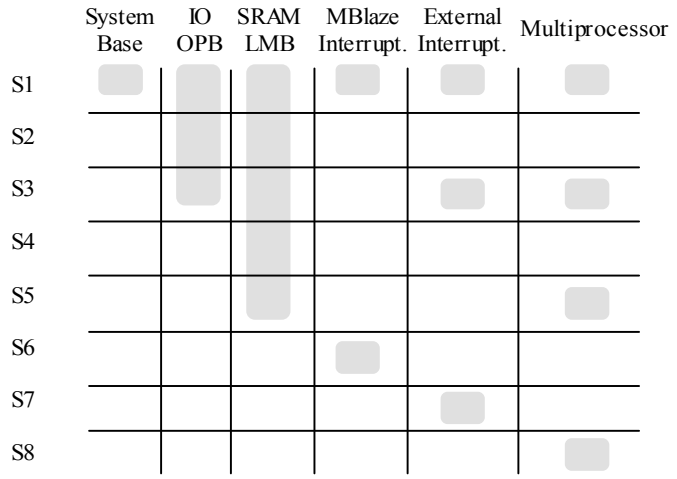


Figure 2. Sessions and contents.

- Message passing in multiprocessors communication.

The practical content of the subject is composed of 8 projects. In the first session, students make a basic system which will be used in following sessions as the base core system. Second and third sessions are used to introduce the input/output flow and the communication with external peripheral through the On-chip Peripheral Bus, for general purpose. SRAM external memory is added to the system at fourth session. Next session is dedicated to the External Memory Controller and how to split the bus. MicroBlaze interruptions are added in the sixth session, and external interruptions using the interruption controller are included in the seventh session. Finally, students build a biprocessor, using the Fast Simple Link channel at session eight. In fig. 2 the relation between practices is shown. For instance, 5$^{th}$ session is based on all previous sessions, 7$^{th}$ session is based on 3$^{rd}$ and 1$^{st}$ session.

## V. BIPROCESSOR SYSTEM DESIGN

The last and most complex practical session is the design and implementation of a biprocessor. A computational system composed of two MicroBlazes will be designed. Both MicroBlazes will be interconnected using message-passing protocol. Each MicroBlaze has its own non-shared memory for instructions and data.

In the Fig. 3 a diagram with the structure of the design is shown. In it, the buses and components used have been detailed. It also includes how they are interconnected

At first, following the logical sequence exposed previously, a HWP will be created. This HWP will include the configuration of the components and buses, their interconnection, the memory map, ports and other parameters.

In the following subsection, the steps needed to configure the system will be described. Trivial stages, such as the creation of the project, will not be included in this paper. The parameters shown in this section depends on the FPGA chip, in this case the Spartan 3 board [11].

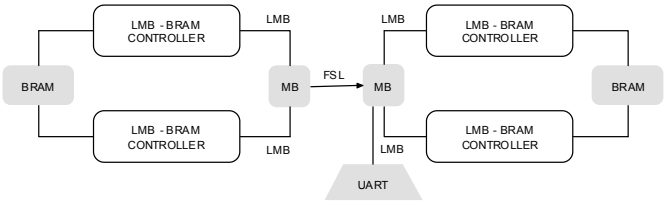


Figure 3. Biprocessor system diagram.

### A. Hardware Platform Specifications

This stage is described in the MHS file. Following, the components specified in the structure of the system are enumerated:

- Two MicroBlaze processors.
- Two on-chip RAM memory blocks (BRAM), one for each processor.
- One UART.
- One OPB bus, to connect the UART with the slave processor.
- Two LMB buses to communicate each processor with their respective data memory controller; and another two LMB buses to interconnect the processors with their instruction memory controller.
- One FSL channel to intercommunicate each processor with the other.

After that, the interconnection of buses and components is defined. The connection of the memory ports are also set at this point. The student has to specify in the connection matrix which components are linked to which buses and with which kind of connection.

In the exposed case, four LMB buses are needed to access local memory, two for each MicroBlaze, because each processor has its own memory subsystem.

Also, one FSL channel which connects both processors.

Each BRAM has been designed with 4 different ports. Each MicroBlaze reaches its memory block through two different interfaces (instructions and data).

After that, it is necessary to map the components inside the configuration memory of the processors. XPS provides a functionality which is able to compute automatically a valid configuration memory map for a monoprocessor system structure. However, as the system proposed is a biprocessor one, this functionality cannot be used.

Each MicroBlaze looks for the first instruction in its program at memory address 0x0.

The next step is to define the internal and external ports. Most of the internal ones are configured by XPS with default settings. It is also necessary to define and to connect some of the internal ports to make the system works: those ports related to the reset and clock signals must be forwarded to all of the subsystems and components. Four external ports are mandatory: clock, reset, UART in and UART out. With these

ports, the student sends commands and synchronization information to the system. Finally, the components are configured. The parameters for each component and their meaning are described thoroughly in the documentation included in the XPS platform.

Particularly, MicroBlaze includes a parameter which selects the amount of FSL interfaces used. Thus, both processors have to set this configuration value to one to allow the communication between them. The configuration of this parameter is done by changing C_FSL_LINKS. This parameter has to be set to a numerical value, representing the amount of FSL interfaces to be included in the core.

Another interesting configuration to be mentioned is the UART operational configuration. The student has to determine the operational frequency, the application of the parity bit checking, working bauds, etc. A valid set of parameters for the UART and MicroBlaze are the following:

*1) UART parameters.*

*a) C_CLK_FREQ = 50_000_000.* Set the frequency of the OPB bus, connected to the UART. It has to coincide with the operational system speed.

*b) C_BAUDRATE = 19200.* Set the bauds for the UART. The terminal used to receive characters has to be configured at the same baud rate.

*c) C_USE_PARITY = 0.* Set whether the UART should work with parity bit or not.

*2) MicroBlaze parameters.*

*a) C_FSL_LINKS = 1.* In order to communicate between the two processors, at least, one FSL channel has to be defined.

After the HWP is defined, the netlist files and the support files can be generated.

### B. Constraints File

The constraints file specifies how external ports from the designed system correspond with the Spartan-3 Board [ ] pins:

```
# Clock signal.
Net sys_clk LOC=T9;
Net sys_clk TNM_NET = sys_clk;
TIMESPEC TS_sys_clk = PERIOD
sys_clk 20000 ps;
# Reset button
Net sys_reset LOC=l14;
Net sys_reset TIG;
# UART
NET TX LOC = R13;
NET RX LOC = T13;
```

Once the constraint file and the HWP are ready, XPS has enough information to create the internal bitstream file.

### C. Software Platform Specification

The SWP corresponds with the Microprocessor Software Specification (MSS). The first step is to select the drivers for the components, the SWP libraries and the OS for each processor. In this project, the standalone OS is selected, no libraries are used and the drivers for the components are established by default.

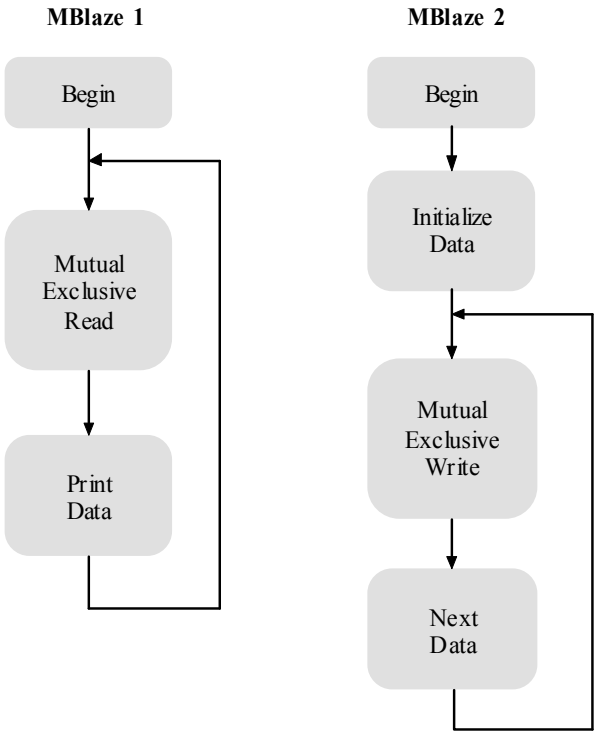


Figure 4. Software application flowchart.

In the following step, the configurable parameters from processors and drivers must be edited; also the interruption routines must be assigned if needed; for instance, in the system under explanation, the interruption routines are not used. In this case, all fields are configured by default except for the processors, which must be configured to work at 100 MHz.

In the final step, the OS libraries configuration parameters are edited and configured. The standalone OS has a minimum size software layer with some primitives for input/output, control, memory allocation, etc.

In the proposed system, the student must specify the UART as the input/output standard peripheral. This is established by assigning the UART to the OPB master MicroBlaze.

Once SWP is configured, it is possible to program the processors software code. However, it is necessary to generate previously the libraries and the Board Support Package (BSP).

### D. Software Application

To carry out the goals of this work it is necessary to create two SW projects, one for each MicroBlaze. With these, each Microblaze will execute its own program.

Some primitives are used to allow a fluid message passing between both processors through the FSL channel.

The writer processor generates the data that will be sent to the reader processor through the FSL channel using mutual exclusion primitives for message passing. The OPB bus master sends the data received by the reader processor to the UART.

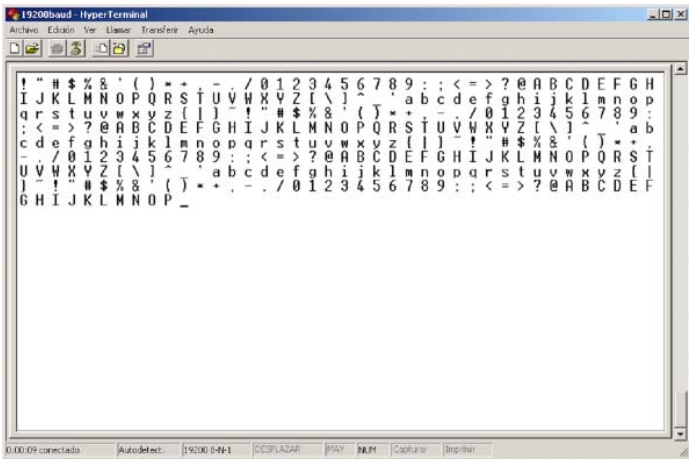

Figure 5. Hyperterminal capture while the both processors are i

After the compilation of the software application, the FPGA configuration file is generated.

Finally, the configuration file is loaded into the FPGA. After all these steps, the students execute the global system and they verify that both processors work properly: The reader processor reads data generated from the writer processor, and the writer processor generates successive data. This process is shown in Fig. 4.

Following the code based on C language for both MicroBlaze processors is shown. Some Xilinx libraries are required to optimize the code and to use some resources:

**MicroBlaze 1 Code. Reading data from MicroBlaze 2 and showing them in the UART**

```
#include <mb_interface.h>
int main( )
{
  int i;
  while(1) {
    microblaze_bread_datafsl(i,0);
    xil_printf("%c ",i); }
  return;
}
```

**MicroBlaze 2 Code. Generating consecutive ascii characters and sending them to MicroBlaze 1 through FSL**

```
#include <mb_interface.h>
int main()
{
  int i = 33;
  int c;
  while(1) {
    for(c=0;c<10000000;c++);
    for(c=0;c<10000000;c++);
    icroblaze_bwrite_datafsl(i,0);
    i++;
    if(i>126) i=33; }
  return;
}
```

Hyperterminal screen was captured and is shown in Fig. 5 to show the results when the both processors are running.

## VI. CONCLUSIONS

XPS can be used as an excellent tool for the students to design and build complex architectures avoiding implementation details. Otherwise, they would spend a lot of time until they master concepts and techniques to develop those systems. Students understand the functionality and the structure of the different components, in order to interconnect all of them to build either a monoprocessor or a biprocessor.

A notable improvement of the qualifications compared to the average of the previous five years was obtained. In particular, in 2008 an improvement of 21% was obtained in front of the mean of the assessment for 2005-07 period. In 2009 the improvement in the assessment was 22%.

As a result of a survey, students are more motivated using real devices than using just simulators.

Furthermore, the configuration of each of the component parameters contributes to a better understanding of the developed architecture. And they are able to test how different values for those parameters influence the performance of the whole system. Finally, a guide on how to implement several processors systems on a single FPGA chip has been provided.

## ACKNOWLEDGMENT

Authors wish to remark the great task carried out by the Xilinx University Program, XUP; and the Sun Microsystems OpenSPARC University Program, which have donated software and materials which have been very useful to partially finance this work.

## REFERENCES

[1] R. D. Williams, R. H. Klenke, and J.H. Aylor. "Teaching Computer Design Using Virtual Prototyping," *IEEE Trans. on Education, vol. 46, no. 2, 296–301,* 2003.

[2] Simuproc. http://simuproc.softonic.com/ . Last accessed on 1st September 2009.

[3] T. Weng, Y. Zhu, and C.-K. Cheng. "Digital Design and Programmable Logic Boards: Do Students Actually Learn More?," *in Proc. ASEE/IEEE Frontiers in Education Conference,* 2008.

[4] Y. Zhu, T. Weng, and C.-K. Cheng. "Enhancing Learning Effectiveness in Design Courses Through the Use of Programmable Logic Boards," *IEEE Trans. on Education, vol. 52, no. 1, 151–156,* 2009.

[5] V. Sklyarov, and I. Skliarova. "Teaching Reconfigurable Systems: Methods, Tools, Tutorials, and Projects," *IEEE Trans. on Education, vol. 48, no. 2, 290–300,* 2005.

[6] OpenSPARC http://www.opensparc.net/edu/university-program.html. Last accessed on 8th November 2009.

[7] -. "Platform Studio User Guide," Application notes, Xilinx, 2005.

[8] -. "Microblaze Processor Reference Guide," Application notes, Xilinx, 2005.

[9] -. "Embedded System Tools Reference Manual," Application notes, Xilinx, 2008.

[10] -. "OS and Libraries Document Collection," Xilinx, Application notes, September 2007.

[11] Spartan-3 Board. http://www.digilentinc.com/ . Last accessed on 30th October 2009.