USING AN FPGA IMPLEMENTATION OF THE MULTICYCLE MIPS MACHINE TO TEACH RECONFIGURABLE COMPUTING IN THE NEW EHEA

J. M.GRANADO, M. A. VEGA, J. BALLESTEROS, J. M. SÁNCHEZ, J. A. GÓMEZ
Department of Computer Science, University of Extremadura, Escuela Politécnica, Campus Universitario s/n, 10071, Cáceres, Spain.

Reconfigurable Computing is a very important discipline nowadays. Furthermore, with the new hardware description languages like Handel-C the change to hardware programming is easier for the computer engineers. In this work, we present a proposal of course about Reconfigurable Computing devoted to the training of the new computer engineers within the new European Higher Education Area based on the multicycle MIPS machine implementation using FPGAs and Handel-C.

1. Introduction

Nowadays, FPGAs are very important in hardware design. Their low cost, the possibility of performing multiple implementations (partial and dynamic reconfiguration) and the evolution of the hardware description languages to easier languages for the traditional application programmers, have led the FPGAs to an important place in the computer world. For this reason, the study of these devices has become indispensable in many university studies, particularly in Computer Engineering. In this work, we present a proposal of course about Reconfigurable Computing in order to the new computer engineers within the new European Higher Education Area (EHEA [8]) will know these devices and will use them correctly.

In June 1999, the Ministers of Education of 29 European countries signed the Bologna Declaration, with the purpose of being able to have, in the year 2010, a European Higher Education Area. Spain is one of the countries included in the Bologna Declaration. Our educational proposal arises from the reading of some articles like [2], and our participation in several convocations of actions for the adaptation of our University to the EHEA.

Since we are Computer Science professors, our educational proposal is adapted to advanced course students of Computer Science and Computer Engineering. Particularly, this suggestion is based on the multicycle MIPS machine implementation [1] through FPGAs and Handel-C. In this way, several interesting areas for a computer engineer are combined: computer architecture, FPGAs, Handel-C programming and Visual C++ programming, among others.

This work is organised as follows: in section 2 the multicyle MIPS machine used is described. Section 3 presents important data about the adaptation of our proposal to the EHEA (ECTS needed, competences fomented,…). Then, in section 4, we show some interesting details about the designs performed by the students in the proposed course. Finally, in the last section the conclusions are indicated.

2. The Multicyle MIPS Machine Used

We have selected this machine because it is easy to implement and it also has a reduced instruction set. Of course, in [1] detailed information related to the original multicycle MIPS machine, like its data path and instruction formats, may be found. However, in this section, many of these aspects will be reviewed, since the instruction set of the original implementation has been increased for the development of the proposed educational experiences changing also the data path and the control unit

of the MIPS machine. In conclusion, although some of the data in this section can be redundant, we include them in order to do a complete and exact description of our educational proposal.

2.1. Multicycle MIPS Machine Execution Phases

The execution of each instruction in this machine is divided into five phases. Each phase lasts once clock cycle. The phases of the multicycle MIPS machine are:

- Instruction fetch.
- Instruction decode and register fetch.
- Execution, memory address computation, or branch completion.
- Memory address or R-type instruction completion.
- Write-back.

More details about all these phases can be found in [1]. In short, the machine we have implemented for making the proposed experiences is relatively close to the current market machines, and by that the students will go deep into some interesting aspects for their future work.

2.2. Instruction Formats.

The used machine has three different instruction formats, as we can see in figures 1, 2 and 3. These formats include all instructions we use in our machine. All the formats have 32 bits and the 6 most significant bits belong to the operation code. Now let us see all them.

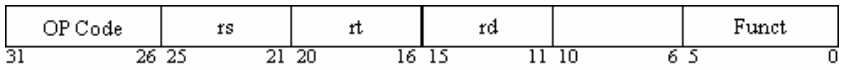

Figure 1. R-type instruction format

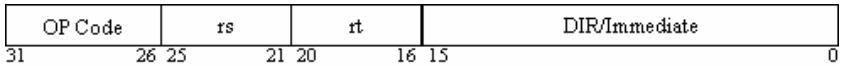

Figure 2. I-type instruction format.

Figure 3. J-type instruction format.

Figure 1 shows the instruction format of R-type instructions. This instruction type includes arithmetic-logic instructions and the unconditional jump JR instruction. These instructions have 0x00 in the operating code field (OP Code), two source registers (rs and rt), and a target register (rd). The Funct field is an extended operating code. All instructions students implement with this format and the value of the corresponding Funct fields are shown in table 1.

Instruction	Funct	Operation
ADD	0x20	rd ← rs + rt
SUB	0x22	rd ← rs – rt
AND	0x24	rd ← rs & rt
OR	0x25	rd ← rs | rt
XOR	0x26	rd ← rs ⊕ rt
NOR	0x27	rd ← ~ (rs | rt)
SLT	0x2A	if rs < rt then rd ← 1 else rd ← 0
JR	0x08	PC ← rs

Table 1. R-type implemented instruction list.

Figure 2 shows the format of the I-type instructions. These instructions use two registers (rs and rt) and a constant value (16 least significant bits). In this format, load and store, arithmetic-logic with

immediate and conditional branch instructions are included. These instructions and the operation they execute are shown in table 2. In this instruction type, when it is necessary rt is the target register.

Instruction	OP Code	Operation
ADDI	0x08	rt ← rs + Immediate
ANDI	0x0C	rt ← rs & Immediate
ORI	0x0D	rt ← rs | Immediate
XORI	0x0E	rt ← rs ⊕ Immediate
SLTI	0x0A	If rs < Immediate then rt ← 1 else rt ← 0
BEQ	0x04	if rs = rt then PC ← PC + DIR*4
BNE	0x05	if rs != rt then PC ← PC + DIR*4
BLEZ	0x06	if rs <= 0 then PC ← PC + DIR*4
BGTZ	0x07	if rs > 0 then PC ← PC + DIR*4
LW	0x23	rt ← MEM[rs + DIR]
SW	0x2B	MEM[rs + DIR] ← rt

Table 2. I-type implemented instruction list.

The last figure (figure 3) shows the J-type instruction format. In this format, unconditional branch instructions are included. These instructions are shown in table 3. In both instructions, the branch address will be computed adding to the high part of DIR field the 4 most significant bits of the PC (Program Counter) and making a double left shift to this value.

Instruction	OP Code	Operation
J	0x02	PC ← (PC[31:28]@DIR)*4
JAL	0x03	Reg_31 ← PC PC ← (PC[31:28]@DIR)*4

Table 3. J-type implemented instruction list.

To conclude, the students also implement the EOP (End Of Program) instruction, which is not included in the instruction set seen in [1]. The purpose of this instruction is to indicate the program end. So, the total number of implemented instructions is 21 instead of the 12 instructions that appear in [1].

2.3. Data path and Control Unit of the Multicycle MIPS machine used

Figure 4 shows the data path and control unit of the multicycle MIPS machine performed in our educational experiences.

Next we enumerate every element used, giving those details that are less evident:

- Control unit.
- ALU control: It generates the control signals to indicate what operation must be executed by the ALU.
- PC (Program Counter).
- Memory: It is an 1024x8 bits memory. This component stores both program instructions and program data. Since the memory is 8-bit word wide, memory address must be a multiple of 4.
- Instruction register.
- Register bank: This component has 32 registers. All the registers in this implementation have 32 bits.
- ALU.
- Sign extender: This element will convert a 16-bit datum to a 32-bit datum, extending its sign.
- X-bit left shifters: If the input data have 32 bits, the X most significant bits of the input data will be lost. If the input data have less than 32 bits, the output data size will be equal to the input data size plus X.

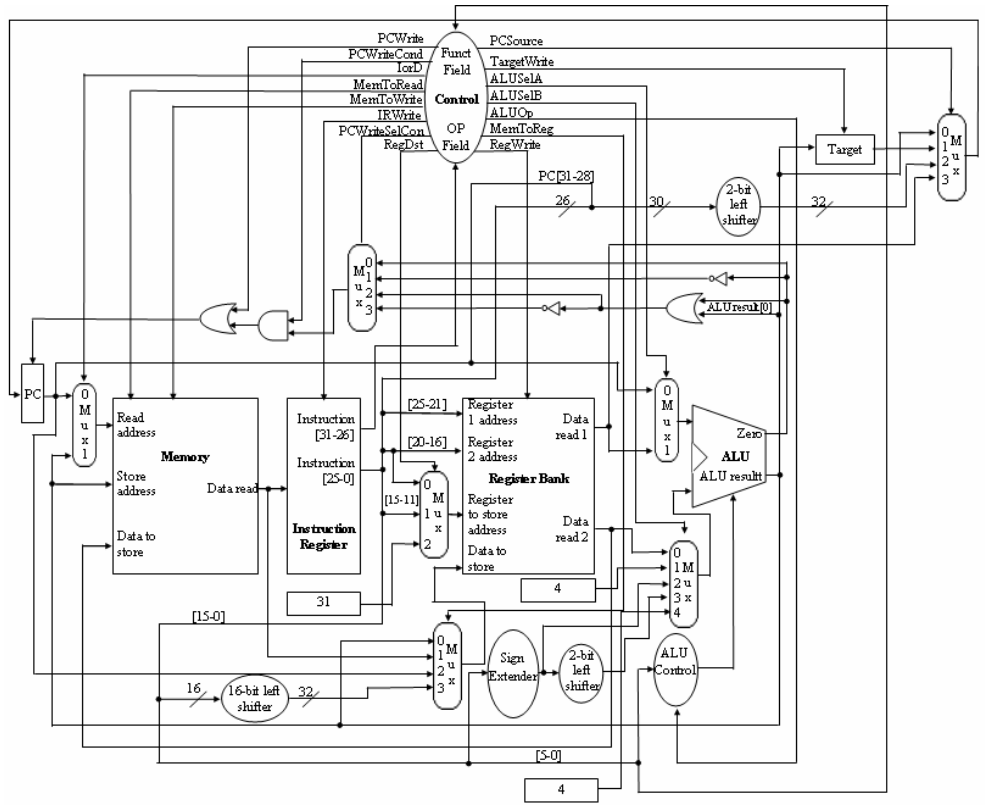

Figure 4. Data path and Control Unit of the Multicycle MIPS machine implemented

3. Methodology and Adaptation to the EHEA

3.1. Distribution of the student workload

It is important to highlight that we are Computer Science professors, therefore, our proposal is focused on advanced course students of Computer Science and Computer Engineering, not including other students like Software Engineering and Information Systems students (see the ACM-IEEE recommendations for more details [9]).

Furthermore, our proposal is adapted to the effective regulations in Spain for the convergence to the EHEA. These regulations can vary lightly from some European countries to others, allowing certain freedom to the corresponding governments and universities. For example, although in the credit system ECTS (European Credit Transfer and Accumulation System [10]) one credit stands for around 25 to 30 working hours within the student workload, in our University, we have used the equivalence 1 ECTS=25 working hours.

In conclusion, we have assigned a total of 3 ECTS (75 hours) to our educational proposal. Remember that student workload in ECTS consists of the time required to complete all planned learning activities such as attending lectures, seminars, independent and private study, preparation of projects and examinations.

In particular, we have divided these 75 working hours like it is described in table 4. Remember that non present activities include all the independent and private work of the students (preparation of classes, exercises, projects, examinations, etc.).

Lectures (10%)	Laboratory (30%)	Tutorships (5%)	Non present act. (55%)
7-8 hours	22-23 hours	3-4 hours	41-42 hours

Table 4. Distribution of the student workload.

3.2. Competences to develop

The academic and discipline competences fomented with our proposal are:

- To know the design cycle: modelling, simulation, synthesis, prototyping, verification and production.
- To study the programmable logic devices: types, architecture and features.
- To use the tools of the design cycle and the platforms of reconfigurable prototyping.
- To design and prototype different digital systems and architectures on reconfigurable hardware.
- To develop products in different programming languages: Handel-C and Visual C++.
- To dominate important concepts about computer architecture.

The personal and professional competences we look for to develop are:

- Analysis and synthesis capability. It is important for the laboratory classes, where we offer wide documental sources to the students.
- Organization and planning capability. It is important in order to perform correctly the implementation of the multicycle MIPS machine using FPGAs.
- Oral and written communication in the native language (Spanish). At the end of the project, the students must deliver the corresponding documentation.
- Knowledge of a second language (English). In this field many technical reports are in English.
- Computer Science knowledge related to the study environment.
- Information management capability (information capture and analysis). Important competence to obtain a good result in the development of the laboratory classes.
- Resolution of problems. Problems during the development of the project and other practical problems.
- Taking of decisions. The student must control specifically certain stages of the design flow, taking some decisions.
- Working in team. We group the students in two-people teams.
- Autonomous learning. There are many no present hours.
- Adaptation to new situations. Many concepts are new for the students: new devices (FPGAs), new computing paradigm (Reconfigurable Computing), new programming languages (like Handel-C) and new technologies.
- Creativity. In order to resolve problems or to conclude some projects with success.
- Initiative and venturesome spirit. It is fundamental to propose improvements or optimizations in the student projects.
- Motivation for the quality. The work delivered by the students must fulfil certain quality requirements.

3.3. Course contents

On the one hand, the course has only 7-8 lectures of one hour. These lectures cover the following aspects:

- The design cycle.
- Programmable Logic Devices. FPGAs.
- Design and synthesis tools.
- Handel-C and DK.
- Xilinx ISE.
- Visual C++.

- Reviewing the multicycle MIPS machine.

Observe that the lectures only introduce the theoretical concepts, which students must study at home and apply during the laboratory sessions.

On the other hand, the lab classes are divided into two-hour sessions. That is, we have 11-12 lab sessions of two hours. This is our content distribution in sessions:
- Designing an ALU in Handel-C (Guided session).
- Generating the corresponding .BIT: DK + Xilinx ISE (Guided session).
- Checking the result with a small Visual C++ application and the FPGA (Guided session).
- Designing the memory, register bank, instruction register and PC (Program Counter) in Handel-C (None guided session).
- Designing the control unit and the ALU control in Handel-C (None guided session).
- Designing in Handel-C other components: shifters, sign extender, multiplexers, etc. (None guided session).
- Generating the .BIT and debugging possible errors (None guided session).
- Development of the Visual C++ application I (None guided session).
- Development of the Visual C++ application II (None guided session).
- Checking the final MIPS implementation (Visual C++ & .BIT & FPGA) (None guided session).
- Extra session for debugging and correcting errors (None guided session).

Note that we need some introductory lectures before beginning with the lab sessions. Also, only the first three lab sessions are guided, in order to guide student along all the process (Handel-C + DK + ISE + Visual C++). The rest of sessions are non guide, so the groups of pupils can advance at their own learning pace. In any case, during any session, students can consult any doubt to the professors.

Furthermore, it is important to highlight that students must prepare the laboratory sessions at home, so that they can do the work more quickly during the lab sessions.

Finally, before students deliver the final documentation of their projects (MIPS implementation) they have to attend tutorships in order to clarify any pending doubt and inform the professors about the structure of the document they are writing. For each student, one hour of tutorship is reserved to do the final examination of the course. Observe that this option is possible because the number of students is reduced.

Although students do the course in pairs, the examination is done individually. In this examination, each student must answer some questions about his/her MIPS implementation, Visual C++ application, the documentation delivered and some important concepts explained during the lectures.

4. Other Interesting Details

4.1. Material Used

Until now, for implementing the multicycle MIPS machine, the students have used a Celoxica [3] ADMXRC2 board with a Xilinx [4] Virtex XC2V6000 FPGA with a speed grade of -6. This board is used for many research works. Only one board of these characteristics is available. This fact makes the students perform the implementation in the computer laboratory and they use this board only in the physical proof phase, so they share the use of the board. This option is possible because the number of students is reduced. If this number increases in future academic years, a greater number of boards will be needed and consequently, other cheaper reconfigurable hardware platforms will be required (such as the Virtex FPGA boards of XESS [5]).

The hardware circuit description is done with the Handel-C language [3] integrated in the DK environment [6]. The Xilinx ISE environment is used to generate the bitstream (.bit). Finally, the students build a little application by means of Visual C++ [7] to check the correct performance of their experiments.

4.2. Difficulties of the Experience

We think the difficulty of the experience is appropriated for advanced course students of Computer Science and Computer Engineering, who are the students that have participated in our first experiences. By way of illustration, the ALU description by means of Handel-C code is shown. As we can see, it is simple and easy to understand the code for any student because Handel-C is based on ANSI-C.

```
macro proc ALU(Datum1, Datum2, Zero, Result, AluControl)
{
  switch (AluControl) {
    case 0: *Result=Datum1 & Datum2; break;
    case 1: *Result=Datum1 | Datum2; break;
    case 2: *Result=Datum1 + Datum2; break;
    case 3: *Result=Datum1 ^ Datum2; break;
    case 4: *Result=-(Datum1|Datum2)-1; break;
    case 5: *Result=Datum1; break;
    case 6: *Result=Datum1 - Datum2; break;
    case 7: *Result=0@(signed)(Datum1<Datum2); break;
    default: delay;
  }
  *Zero=(*Result==0);
}
```

4.3. Common Results of the Implementation

In order to give more information about the complexity of these experiences, the most common results obtained by the students (average case data) are shown in table 5. These results have been obtained from the reports generated by Xilinx ISE tools after the implementation of the hardware circuit.

Occupation (Slices)	Maximum Frequency (MHz)	Minimum Period (ns)	Memory Used (bits)
3451 (10% of 33792)	18.624	53.693	8192

Table 5. Results obtained in the multicycle MIPS machine implementation.

4.4. Checking the Correct Performance of the Experiences

To check the correct performance of the experiences, the students have to make a Visual C++ application. This application configures the FPGA and takes charge of sending the memory content of the multicycle machine (we should take into account that the memory stores both data and instructions). Besides, the application will get back the resulting values from memory, register bank and program counter (PC).

In table 6, we can see a program example used by the students to check the experience quality, and so, the correct implementation of the multicycle MIPS machine in the FPGA. It is a very simple program which adds the constant 13 to each element of a 10-element array. We have tried that a great number of different instructions of every format type (R, I, J) appears in this program. Table 6 shows also the memory content for this program. In this case, the first nine positions store all the program instructions, and the next ten positions contain the array elements. After the program execution, each array element (addresses from 0x24 to 0x48) must have been increased in 13.

Mem. Addr.	Content	Content Explanation	
0x00	0x00421026	XOR $2, $2, $2	# R2 ← 0
0x04	0x34430028	ORI $3, $2, 0x28	# R3 ← 0x28
0x08	0x10430005	BEQ $2, $3, 5	# IF R2 = R3 ➔ PC←PC + (5 * 4)
0x0C	0x8C440024	LW $4, 36($2)	# R4 ← MEM [R2 + 0x24]
0x10	0x2084000D	ADDI $4, $4, 13	# R4 ← R4 + 13
0x14	0xAC440024	SW $4, 36($2)	# MEM [R2 + 0x24] ← R4
0x18	0x20420004	ADDI $2, $2, 4	# R2 ← R2 + 4
0x1C	0x08000002	J 2	# PC ← 2 * 4
0x20	0xFC000000	EOP	# FIN
0x24	0x00000005	Content on the first array position	
0x28	0x00000006	Content on the second array position	
0x2C	0x00000007	Content on the third array position	
0x30	0x00000008	Content on the fourth array position	
0x34	0x00000009	Content on the fifth array position	
0x38	0x00000000	Content on the sixth array position	
0x3C	0x00000001	Content on the seventh array position	
0x40	0x00000002	Content on the eighth array position	
0x44	0x00000003	Content on the ninth array position	
0x48	0x00000004	Content on the tenth array position	

Table 6. Example program to check the correct performance of the experiments.

5. Conclusions

In this work, an educational proposal has been presented to elaborate a course about Reconfigurable Computing to be included in advanced studies of Computer Science and Computer Engineering within the new European Higher Education Area. We have also described the methodology followed and other interesting details. These experiences are based on the multicycle MIPS machine implementation and using FPGAs and Handel-C. In this way, they combine several interesting aspects for a computer engineer: the use of FPGAs, FPGAs programming by means of a high level language like Handel-C, reviewing the learnt concepts about computer architecture, Visual C++ programming, etc. The practical implementation of the MIPS machine forces students to know in depth the theoretical considerations involved. In conclusion, they acquire a better and larger knowledge about these subjects.

Our proposal not only produces an improvement of the quality in education, but also is a clear example of how the results of the developed research can revert in the teaching.

Finally, although our experience has been developed with students of Computer Science and Computer Engineering, we think that this kind of educational experiences is also adequate for students of other curricula: Electronic and Electrical Engineering, etc.

References

[1] Patterson, D. A.; Hennessy, J. L.: "Computer Organization and Design. The Hardware/Software Interface". Morgan Kaufmann, 3rd edition (2004).

[2] Šulík, D.; Vasilko, M.; Ďuračková, D.; Fuchs, P.: "Design of a RISC Microcontroller Core in 48 Hours". Embedded Systems Show 2000, London Olympia, (May 2000).

[3] Celoxica Ltd. http://www.celoxica.com (2005).

[4] Xilinx Inc. http://www.xilinx.com (2005).

[5] XESS Corp. http://www.xess.com (2005).

[6] DK Design Suite User Manual. Celoxica Limited (2003).

[7] Pappas, C. H.; Murray III, W. H.: “Visual C++ .NET. The Complete Reference”. McGraw-Hill, 2nd edition (2002).
[8] Haug, G.; Tauch, C.: “Towards the European Higher Education Area: Survey of Main Reforms from Bologna to Prague”. Available at: http://www.eua.be/eua/jsp/en/upload/OFFDOC_BP_trend_II.1068715483262.pdf (2001).
[9] ACM-IEEE: “Computing Curricula 2001”. Available at: http://www.computer.org/education/cc2001 (2001).
[10] European Commission: “ECTS – European Credit Transfer and Accumulation System”. Available at: http://europa.eu.int/comm/education/programmes/socrates/ects_en.html (2005).