DESIGN OF AN INTEGRATED ENVIRONMENT FOR A DIDACTIC PROCESSOR

Valter C. Filho UFERSA
Leonardo A. Casillo UFERSA
Sílvio R. Fernandes UFERSA
Argemiro S. Neto UFERSA

ABSTRACT

This paper presents the design and implementation of an integrated environment designed for educational purposes RISC architectures. This environment includes a VHDL description of the SIC (Simplified Instructional Computer) processor and a software with a set of useful tools for the executable programs development for that processor. The software tools allow assembling, linking, loading and simulating the execution of programs. Additionally, this environment does an automatic generation of a file memory initialization with the executable program that can be used in a FPGA.

1. INTRODUCTION

There are many disciplines in undergraduate computing that involves the study of architectural details of processors and the system software related to them. In this study two ideas can be used: real processors architectures or simulators.

The real architectures has several tools that assist in developing and debugging applications, however studying every architecture's detail can be extremely complex. Simulators, on the other hand, can provide more abstraction of the fine architecture's details, but on the other, can reduce the accuracy of execution time, chip area occupied on or power consumption.

Alternatively arise didactic architectures, in order to address the key theoretical concepts related to these disciplines. The architecture SIC was proposed by Beck [3] for the study of system software, however it is a virtual architecture. Therefore, we propose a VHDL description of this architecture and additionally implement its system software, creating an integrated educational environment.

Section 2 presents the methodology adopted for the description of the architecture. Section 3 describes the details of the SIC processor. Section 4 presents the software with the simulation tools and application development, section 5 presents the results and section 6 the conclusions and future works.

2. METHODOLOGY

The design of an architecture basically involves the following topics: relate characteristics of the target processor, describe the microprograms [6] by using flowcharts, data path design, the state machine description, description in HDL and processor's testing through simulation. This is an own methodology [9] and can be used for the development of any architecture.

2.1. Characteristics of the Target Processor

Initially we must consider the characteristics of the target architecture. Thus, we have the understanding of which components have the processor internally. At this stage, components such as registers, arithmetic logic unit, address bus size, data bus size, control bits, among others, should be considered and described with their specifications.

2.2. Describing Microprograms

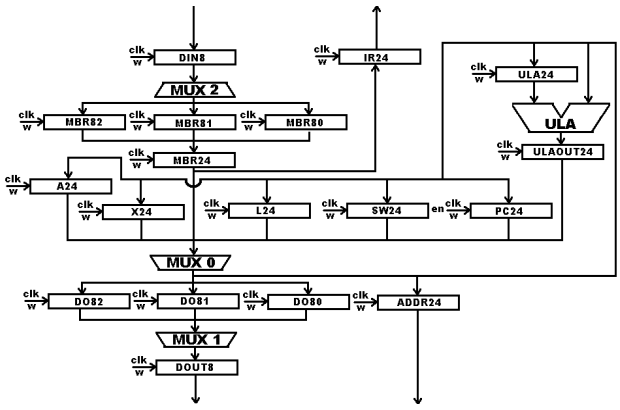
Using flowcharts, we can describe these microprograms to recognize which processor's structures are used for performing the instruction set and in what sequence the tasks are performed.

For each instruction, there are several flowcharts covering some aspects related to its execution, for example: fetch, decode, execution and storage.

2.3. Designing Datapath

The completion of the flowcharts is necessary for determining the data path configuration. Each interaction observed in the flowchart is considered to determine what logical structures are necessary for that operation. By uniting all the structures and excluding redundancies, one can possibly obtain the data path. The aim is to have a unique, complete and logical structure only capable of performing all the operations described by microprograms within the processor.

Figure 1 - Datapath of the SIC

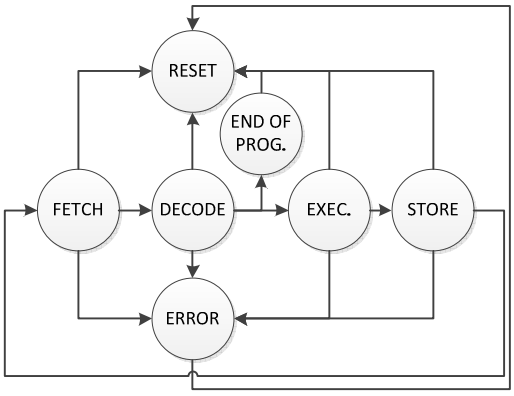


2.4. Compiling the State Machine

The development of a state machine (FSM) is required to describe the behavior of the control unit. This FSM describes what actions will be taken and what time they should be executed.

It is based on flowcharts and data path that one can describe the state machine to model the logical treatment of the architecture's instructions. In Figure 2 - Simplified State Machine we can see a simplified example of a state machine. Each state, except the states of RESET, ERROR and END OF PROGRAM, should be developed, resulting in an even greater FSM.

Figure 2 - Simplified State Machine



2.5. Describing Hardware

For the description of the processor, you should begin production of smaller components, combining them into larger structures. Initially, we describe the components of the operating unit of the processor (register, multiplexer, etc.). At the end you have enough parts to combine them in a unique data path. The control unit, by contrast, is an extra component part and it will contain all states of the FSM. After completion of the control unit and data path, the next step is to integrate these two structures into a single entity, the target processor.

2.6. Simulation and Test

This step is necessary to describe the main memory and create a superior entity that connects this new component to the processor. In memory, a machine language program must be loaded and a test file for the simulation of the description proposed. This program must use all the instructions in the ISA. The goal is to scan the whole state machine and empty the possibilities of system failure.

3. SIC PROCESSOR

3.1. Characteristics

The target processor considered in this work is the SIC - Simplified Instructional Computer [3]. This didactic processor consists of 5 internal registers (A, X, L, PC and SW) all 24-bit and with well specified functions. The A register is used as the operator's work, X is commonly used in indexing operations, the L is used for diversion and return of subroutines, the PC is the program counter and SW is the status word which contains status information.

Although main memory is addressed by a word of 20 bits, 12 bits are used only addressing 4096 bytes. SIC has two types of addressing: indexed and direct. The word size is 24 bits and is organized in memory using the big Indian scheme.

An important SIC's feature is the address and data bus is common between the memory devices and I/O. However the address space of the main memory is different from the one used for input and output devices. Thus, an I/O unit is necessary to connect the devices to the processor.

Due to lack of available interrupts in SIC natively, it's considered a monoprogramable processor. Thus, a switch of tasks is not possible to be executed by the processor through an operating system.

The Instruction Set Architecture (ISA) consists of 26 instructions that perform basic operations, which characterizes it as RISC (Reduced Instruction Set Computer) processor. According to Table 1, the instructions were divided into arithmetic, logical, comparison, diversion, load, unload and I/O. This distribution is critical for the proper identification of the steps and location of results obtained in each processing step: fetch, decode, execution and storage.

Table 1 - SIC's Instruction Set Architecture.

Category	Instruction	Efect
Arithmetic	ADD m	A<-A+m..m+2
	SUB m	A<-A-(m...m+2)
	MUL m	A<-(A)*(m...m+2)
	DIV m	A<-(A)/(m...m+2)
Logic	OR m	A<-(A)|(m...m+2)
	AND m	A<- A&m...m+2
Comparison	COMP m	(A):(m...m+2), CC
	TIX m	X<-(X)+1; (X):(m...m+2)
Diversion	JSUB m	L<-(PC); PC<-m
	RSUB	PC<-L
	J m	PC<-m
	JEQ m	PC<-m if CC is =
	JGT m	PC<-m if CC is >
	JLT m	PC<-m if CC is <
Load	LDA m	A<-(m...m+2)
	LDCH m	A [LSB] <- (m)
	LDL m	L <- (m ... M+2)
	LDX m	X <- (m ... M+2)
Unload	STA m	m...m+2 <-A
	STCH m	m...m+2 <-A [LSB]
	STL m	m...m+2 <- L
	STSW m	m...m+2 <- SW
	STX m	m...m+2 <- X
I/O	TD m	Test of device.
	WD m	Device m <- (A) [LSB]
	RD m	A [LSB] <- Device m

3.2. Hardware Describing

SIC was designed based on components. Initially, basic parts of architecture were described and tested as registers, counters, multiplexers, demultiplexers, clock dividers and arithmetic logic unit. In a second step these components have been integrated into a structure called the data path. Here we have the basic structure for performing the operations available on the ISA. At this point, we must describe the control unit, responsible for coordinating the use of the data path structure.

The processor's description finishes in an entity that performs the connections between the control unit and data path. It was then described an entity representing the memory with the loaded initialization file. This file is the object code for program's execution, described in MIF format (Memory Initialization File).

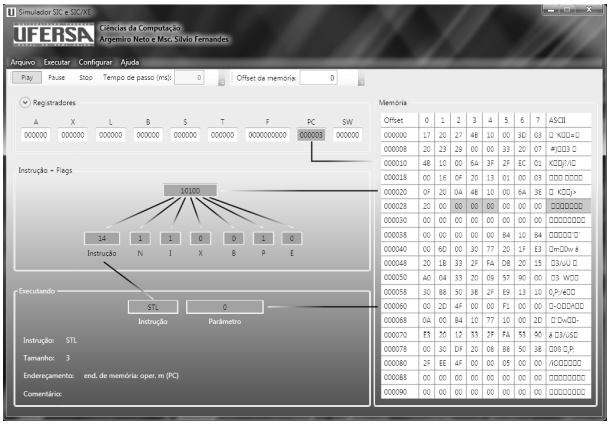
To test and validate the architecture we used the DE2 board education as an interface I/O. For this we developed an LCD controller, responsible for mapping the characters received in their corresponding LCD addresses sent by the processor.

4. SIMULATOR ENVIROMMENT

4.1. Software

The simulator combines all the tools developed in an environment with full graphical interface, where the user can set up object programs. Figure 3 shows the software interface.

Figure 3 – Simulator interface.



This software simulator was developed in C #, based on the technologies .NET Framework 4 and is compatible only with Windows operating system, in Windows Vista and later versions.

The user can choose between assembling standard object SIC or SIC/XE (extended version). The file to be assembled should be in the format ".asm" and be compatible with the syntax of assembly language of the SIC.

The assembled files will be saved with the name of the program contained in the assembly and not the name of the file read. If there are sections in the program, each section is saved in a different file with the name of its section.

There is also the option to save the original file with the assembly code produced by the assembler. In this case, the filename will be "NAME_OF_THE_ PROGRAM.asm." The software also allows the automatic generation of a file in MIF (Memory Initialization File) format with the assembled program.

4.2. Sample Program

The sample program for demonstration of the architecture is described as shown in Figure 4. This program in three-address code has the function to write the string "Hello World" on the top line of the LCD.

Figure 4 – SIC's sample code.

```
TESTIO  START   0
FIRST   LDX     ZERO
READ    LDCH    TEXTO,X
RLOOP   TD      LCD
        JEQ     RLOOP
        WD      LCD
        TIX     LENGHT
        JLT     READ
        SVC     4
TEXTO   BYTE    X'00'
        BYTE    X'20'
        BYTE    X'01'
        BYTE    X'20'
...
        BYTE    X'1F'
        BYTE    X'20'
ZERO    WORD    0
LENGHT  WORD    64
LCD     BYTE    X'00'
        END     0
```

4.3. Generating MIF

After submitting the assembly code, the software generates the object file, which can be used to simulate the processor SIC execution. You can also generate the memory initialization file with extension ".mif", as shown in Figure 5. This could be included in the VHDL project of the processor and used as an application that initializes the memory of the FPGA.

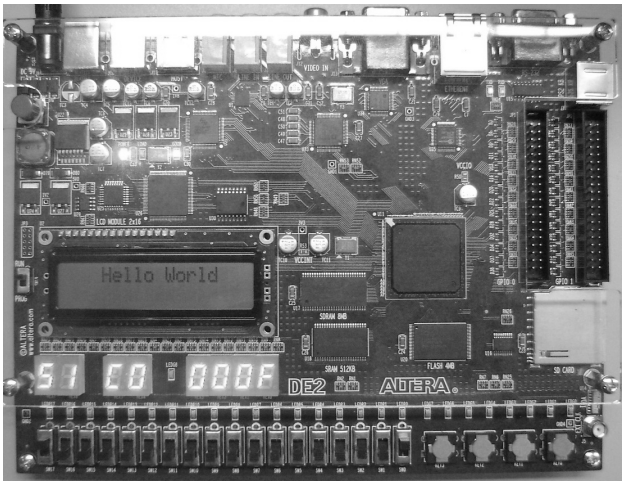
Figure 5 - .MIF generated by the simulator.

```
WIDTH=8;
DEPTH=4096;

ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;

CONTENT BEGIN
        0  :   4;
        1  :   0;
        2  :   58;
        3  :   50;
        4  :   80;
...
        59  :   0;
        5A  :   0;
        5B  :   0;
        5C  :   0;
        5D  :   40;
        [5E...1000] :   0;
END;
```

5. RESULTS

To validate the proposed environment in this article, it was developed an example of a simple "Hello World" as described in Figure 4, which was submitted to the software to generate the equivalent object code and the file format ".mif", as shown in Figure 5. This file was compiled with the project and downloaded to the DE2 board, so the message "Hello World" appears on the LCD display, as shown in Figure 6. The same figure shows the value "51C0000F" in the 7-segments display, meaning end of program.

Figure 6 - Hello World in the DE2 Board



The project's time analysis undertaken by the Quartus II indicates that the maximum operation frequency is 76.07 MHz. Another strength of this study is the low consumption of logic gates and registers for the development of this processor. The number of registers used was 601, using only 930 logic gates, that means 3% of the total available on the chip EP2C35F672C6 of the Cyclone II family from the manufacturer Altera. This result underscores the low consumption of chip area and, probably, low power.

6. CONCLUSION AND FUTURE WORKS

This paper presented an integrated software/hardware environment for a didactic RISC processor, the SIC. In this environment is possible to assemble, link, load, simulate the execution and generate a file in ".mif" format, an application using the assembly language syntax of the SIC. The methodology of development adopted to design and describe the processor in VHDL was also presented.

A good result of this methodology is a highly modular project that can be adapted or incremented with new instructions more easily and directly. It's also easy and intuitive to identify possible errors and inconsistencies in the program loaded into memory and in their own state machine that describes the control unit. Besides these points, we can note that the amount of documentation created in the process is complete and covers all the necessary aspects for identifying inconsistencies in the testing phase.

The synthesis results show the small area occupied by the processor chip and the maximum frequency of operation.

For validation we used the DE2 education board, which demanded the development of an LCD display controller used as standard output.

For further work, the SIC processor will be expanded to SIC/XE processor, more complete, multiprogrammable, enables interrupts, including a processor pipeline, development of a C compiler, complementing the simulation environment and providing the development of libraries and softwares for this platform. Other I/O drivers will be developed in order to communicate the processor with other interfaces available on the DE2 board.

7. REFERENCES

[1] Aho, Alfred V., R. Sethi, Compiladores: princípios, técnicas e ferramentas, Addison Wesley, São Paulo, 2008.

[2] Amore, Robert d'. VHDL: descrição e síntese de circuitos digitais, LTC, Rio de Janeiro, 2005.

[3] Beck, Leland L., System Software: an introduction to systems programming. Addison Wesley Longman, 1997.

[4] Montenegro, Toni F., A. Girardi, "Implementação em VHDL do Processador Educacional BIP I," Iberchip XV Workshop, Buenos Aires, 2009.

[5] T. F. Oliveira, I. S. Silva, "Cabare: An Educational Reconfigurable General Purpose Processor," Sforum, Natal, 2009.

[6] Stallings, W., Arquitetura e Organização de Computadores, Prentice Hall, Rio de Janeiro, 2008.

[7] Tanenbaum, A. S., Sistemas Operacionais Modernos, Prentice Hall, Rio de Janeiro, 2007.

[8] Tocci, R. J., N. S. Widmer, Sistemas Digitais: Princípios e Aplicações, Prentice Hall, Rio de Janeiro, 2007.

[9] Costa, R. V., L. A. Casillo, S. F. Araújo, J. P. Pereira, "Metodologia para Projeto de um Processador RISC Didático," WTCC 2011, Mossoró, 2011.