DESIGN OF A CUSTOM CPU ARCHITECTURE FOR A PACEMAKER APPLICATION

By

Edgar Martí Arbona

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

ELECTRICAL ENGINEERING

UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS

May, 2009

Approved by:

Eduardo Juan, Ph.D Member, Graduate Committee	Date
Manuel Jiménez, Ph.D Member, Graduate Committee	Date
Rogelio Palomera García, Ph.D President, Graduate Committee	Date
Iván Baiges, Ph.D Representative of Graduate Studies	Date
Isidoro Couvertier Reyes, Ph.D Chairperson of the Department	Date

Abstract of Dissertation Presented to the Graduate School
of the University of Puerto Rico in Partial Fulfillment of the
Requirements for the Degree of Master of Science

DESIGN OF A CUSTOM CPU ARCHITECTURE FOR A PACEMAKER APPLICATION

By

Edgar Martí Arbona

May 2009

Chair: Rogelio Palomera García
Major Department: Electrical and Computer Engineering

Dual-chamber pacemakers have shown many advantages in the treatment of bradarrhythmia, but their power consumption is high when compared with single-chamber devices. This research presents a custom Central Process Unit (CPU) architecture optimized for dual-chamber pacemakers. It also contains an efficient way to control pacemaker peripheral management in order to decrease the power dissipation of a dual-chamber pacemaker device. This is done by using the same hardware for both chambers input and output which decrease the quantity of peripherals used. Additionally, a fast enable/disable action of the pacemaker peripherals was added. Software techniques such as the reduction of instructions and zero-optimization per instruction were used to ensure the CPU low power design. A power analysis of this CPU was done and a static power of 0.4μW was obtained.

Resumen de Disertación Presentado a Escuela Graduada
de la Universidad de Puerto Rico como requisito parcial de los
Requerimientos para el grado de Maestría en Ciencias

DISEÑO DE LA ARQUITECTURE DE UN CPU ESPECÍFICA DE PARA APLICACIONES DE MARCAPASOS

Por

Edgar Martí Arbona

Julio 2009

Consejero: Rogelio Palomera García
Departamento: Ingeniería Eléctrica y Computadoras

Al momento de tratar enfermedades del corazón, como bradiaritmias, los marcapasos de dos cámaras ofrecen varias ventajas gracias a su sincronización, aunque al mismo tiempo poseen un alto consumo de potencia en comparación con los de una sóla cámara. En esta investigación se presenta la arquitectura de una Unidad Central de Control (CPU) especializada para marcapasos de dos cámaras que optimiza el uso de los dispositivos internos. Para lograr esto se utilizó el mismo sistema de detección y estimulación para ambas cámaras del corazón controlado directamente por el CPU. Esto logra una reducción en el número de dispositivos y una forma rápida y eficiente de habilitarlos. Además, técnicas de programación para el bajo consumo de potencia fueron utilizadas tomando en cuenta la frecuencia de uso de cada instrucción. De esta forma se permite la optimización del uso de dispositivos en marcapasos de dos cámaras y por ende se incrementa la vida de la batería. Al hacer un analysis de potencia de éste procesador se obtuvo una potencia estática de 0.4μW.

To my wife, for her comprehension, support and love.

I love you darling.

To my family, for their support and trust.

Acknowledgments

Special thanks to my wife Alix Rivera Albino who always was there for me. Also to Professor Rogelio Palomera for his continuous guidance and support. Thanks also go to my committee members Dr. Eduardo Juan and Dr. Manuel Jimenez. The invaluable help of Pablo Rebollo and his crew was crucial in this work. To Victor Montaño, Venture , Melisa, Yilda, Carlitos, Ricky and Carlos Bula who were always there when I needed them. Finally, to professors Gladys O. Ducoudray, Gillermo Serrano, Nelson Sepúlveda who gave me their guidance in many difficulties.

TABLE OF CONTENTS

List of Tables

List of Figures

List of Abbreviations

FFT	Fast Fourier Transform.
DCFT	Discrete Chirp Fourier Transform.

List of Symbols

t	Time (seconds)
τ	Time between events.
μg	Micrograms.

CHAPTER 1

Introduction

Heart diseases represent the main cause of death in the United States, equaling deaths associated with cancer, accidents and diabetes combined [1]. The latest statistics from the American Heart Association (AHA) showed that one in three people suffer one or more heart diseases. A large group of these responds to the classification of bradycardia [1] and have been successfully treated using pacemaker devices. Hence, improvement of these devices for better performance and expanded operational ranges is of outmost importance.

Pacemakers are classified in single and dual-chamber, depending on their functions and the number of heart chambers they monitor and stimulate. Dual-chamber pacemakers offer more advantages in terms of patient life quality, such as reduction in atria fibrillation, pacemaker syndrome, and heart failure [2, 3]. An additional advantage is that they may operate as single chamber pacemakers if necessary. However, they suffer from higher costs and a shorter battery lifetime when compared with a single-chamber devices, drawbacks that are seriously considered when selecting a pacemaker. As a consequence, many patients for whom this could be the necessary device must limit their choice to a single-chamber pacemaker in other words, this does limit the use by many patients.

[1] Bradycardia, heart diseases that causes an abnormally slow heart rate.

Given the advantages of dual-chamber over single-chamber devices, it is desirable to develop an alternative that might keep the advantages of dual-chamber pacemakers while offering a cost and power consumption competitive with the single-chamber ones. The objective of this research is to develop a specialized CPU pursuing these goals. This design is based on low power software techniques using as an architecture model the microcontroller MSP430 from Texas Instruments [4] and an open source pacemaker software [5].

Given that the stimulation energy depends on the patients' physiology, this research does not deal with this side of the pacemaker operation but focuses only the characteristics that are patient independent. However, the patients' needs have been considered by taking into account the frequency of instructions used in programs for pacemakers, where these programs are precisely obeying to their physiology.

1.1 Motivation

A single-chamber pacemaker senses and stimulates one chamber of the heart with no synchrony with the other chamber. As a consequence, this type of stimulation can cause additional conditions to the patient since the stimulated chamber does not know what happened in the other and might make the heart to struggle more to beat.

A dual-chambers pacemaker doubles the hardware of single-chamber pacemaker for the sense/pace task as shown in Figure 1–1 [6]. This hardware addition provides the solution to the asynchrony problem. It senses and stimulates both chambers resembling more closely the normal physiology of the cardiac activation at the same time that it can work as single-chamber if needed. Although it is superior in performance compared with the single-chamber pacemaker, it also has some drawbacks that decreases it popularity like higher cost and higher power consumption. The

extra power consumption is associated to the additional hardware needed to sence and stimulate an additional chamber and to the pace energy spent in that chamber, since the atria and ventricle can be stimulated in the same heart cycle.

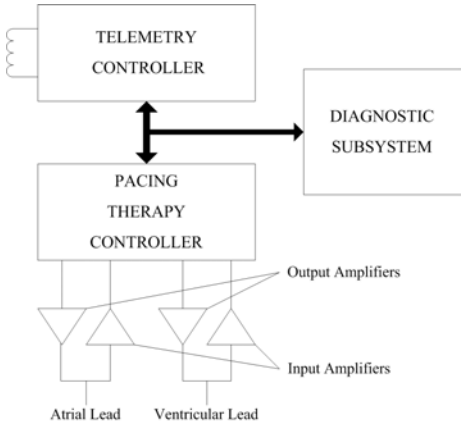


Figure 1–1: Dual-Chamber Pacemaker

The hardware in a dual-chamber pacemaker is divided into control logic, input side, output side, and housekeeping [7]. The control logic is the main part of this device since it manages all the pacemaker tasks. Among those tasks are the hardware peripherals management to ensure good functionality and synchrony. The stimulation parameters are also controlled by the control logic to ensure a good performance. Finally, it gives flexibility to use the same hardware for different patients by customizing internal parameters through external programming.

Focusing on the control logic, our purpose is in designing a custom CPU architecture to decrease the control logic extra devices and include extra features to control the pacemakers' peripherals to reduce its power dissipation. These features form part of the contibutions of this thesis and are mentioned bellow:

1. Disablement of peripherals during the period of time when they are not in use (*e.g.* when pacing, the sense part can be off).
2. Utilization of the same hardware for the input and output sides' for the two chambers by controlling some analog multiplexers.

This concept is shown in Fig. 1–2. In this way, these features have impact in cost and power because of less hardware and power dissipation. The result should be a pacemaker with dual-chamber functionalities, where cost and power are comparable to that of a single-chamber. Other contribution of this work is an CPU opensource VHDL code developed during the research.

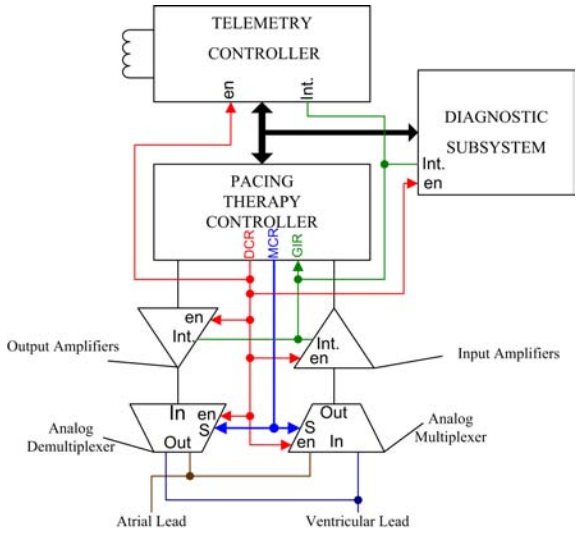


Figure 1–2: Proposed Dual-Chamber Pacemaker

1.2 Summary of Following Chapters

The next chapters begin by presenting the needed basic theory in Chapter 2. This would help understanding the literature review presented in Chapter 3. Chapter 4 states the objectives and methods to accomplish the thesis. The methods discussed in the Chapter 4 were followed and the design procedure was developed and presented in Chapter 5. The obtained results after finishing the design and implementation are presented in Chapter 6. Then elemental power estimation was done in order to compare the simulated results. Finally, the conclusions and cotributions of this work are mentioned in Chapter 8.

CHAPTER 2

Theoretical Background

This chapter deals with the basic knowledge needed to understand the work done in this thesis. The physiological background section is where the concepts about the cells electrical behavior and heart conduction properties are presented. Then, the treatment devices section emphasizes the treatment of some arrhythmias. Finally, the pacemaker control unit is discussed focusing in the Control Process Unit (CPU) architecture and design.

2.1 Physiological Background

The physiological background shows the cell electrical behavior and the heart conduction characteristics. Concepts illustrated here are of great importance in understanding the cardiac stimulation.

2.1.1 Cells Electric Behavior

Membrane Potential

Body cells have different characteristics and behavior depending its functions. Excitable cells that form nerves, muscles, and glandular tissue, produce bioelectric potentials as result of electrochemical activities in the membrane [8]. These biopotentials are classified as resting membrane potential and as action potential.

The membrane of the cell is usually charged positively at the exterior with respect to the interior. This occurs because of an unbalanced distribution of negative

and positive charged ions nears to the membrane. It is an important behavior for the ionic conduction in muscles and nerves[9].

In the cell, potassium (K+) has the greatest concentration in the inside and sodium (Na+) in the outside. Their quantity is determined by channels in the membrane. Those channels are selective, allowing only one kind of ions to pass through it and excluding the others. These channels can be opened or closed; the open and close status can be affected either by chemical or electrical factors.

Resting Potential

High concentration of sodium in the outside and high concentration of potassium in the inside is maintained by active transport[1] as well as by diffusion, maintaining the membrane polarized at a steady potential (see fig2–1) [9]. The outside/inside gradient of potassium is approximately 145mM/10mM and of sodium 4.5mM/140mM.

Resting potential lies between -50 and -100 mV in the internal medium with respect to the external. For cardiac cells, this potential is approximately 90mV [6]. The equilibrium potential for potassium E_K can be calculated by utilizing the Nertz equation (2.1) [8]. Here **n** is the value of K^+, $[K^+]_i$ and $[K^+]_o$ are the concentrations in moles per liter of K^+ inside and outside of the cell respectively, T is the temperature in K, F is the Faraday constant and R is the universal gas constant.

$$E_K = \frac{RT}{nF} ln \frac{[K^+]_o}{[K^+]_i} = 0.0615 log10 \frac{[K]_o}{[K]_i} \tag{2.1}$$

[1] Active Transport: molecules or ions are carried through membranes by carriers molecules from region of lower concentration to region of higher concentration.

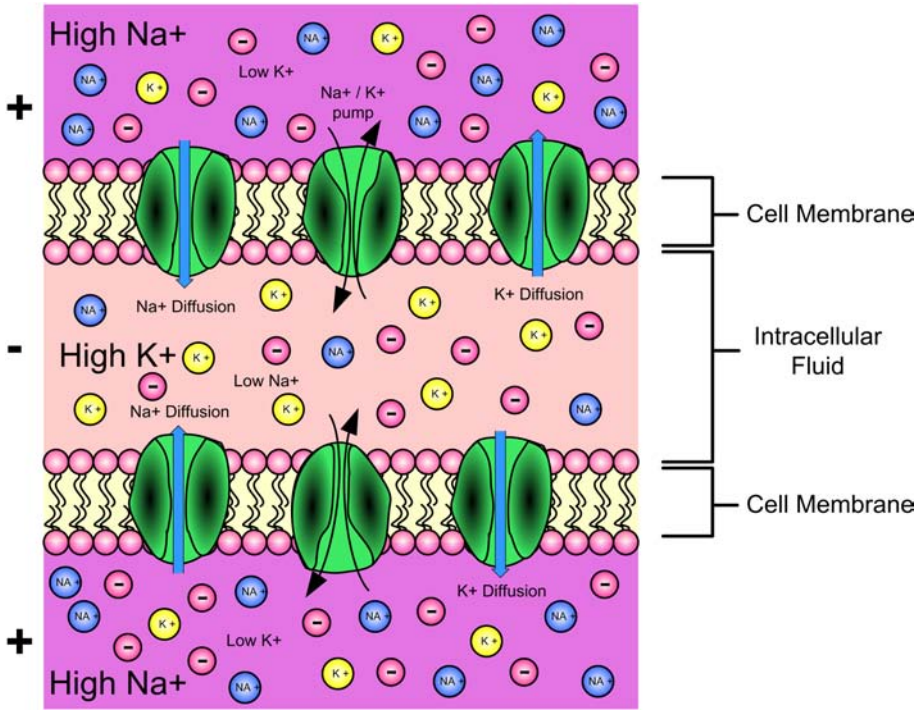


Figure 2–1: Resting Membrane Potential

Action Potential

Some ion channels are sensitive to membrane voltage, staying closed at the resting potential. If a cell is exited in a way that the potential through the membrane reaches a threshold voltage, it opens for a short period changing the membrane potential. Figure 2–2 shows this characteristic and details the phases of the action potential. The peak can reach a membrane potential of +30mV. After the action potential, the membrane is repolarized until the next stimulation. An action potential in a cell stimulates the adjacent membrane to its threshold, triggering another action potential and this action potential is transferred among adjacent membranes, forming a series of action potentials traveling like an electric current through the muscle without decrease in amplitude.

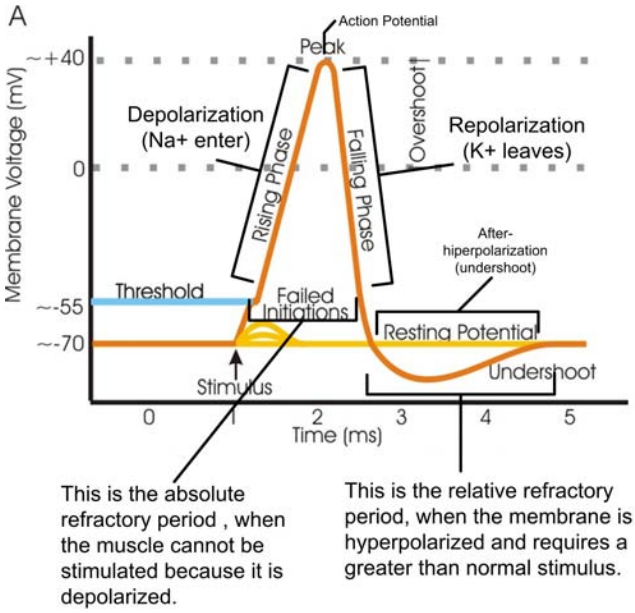


Figure 2–2: Action Potential (Original by en:User:Chris 73, updated by en:User:Diberri, converted to SVG by tiZomr at Wikipedia and edited to add action potential phases)

2.1.2 Heart

The heart is the muscle in charge to circulate the blood through the body. That is of vital importance since in this way oxygen and nutrients are provided to the body cells, at the same time that the waste is removed from them. A lack of this activity results in death.

Figure 2–3 gives an overall description of the heart [9]. It has four chambers: left atrium, left ventricle, right atrium, and right ventricle. The atria are in the upper side and the ventricles at the bottom. This muscle is also composed by veins, arteries and valves that control the blood flow. The heart chamber works in a coordinated way: when the atria contract the ventricles are relaxed, and when ventricle contract the atria are relaxed. Then, both are relaxed for a moment until the cycle starts again.

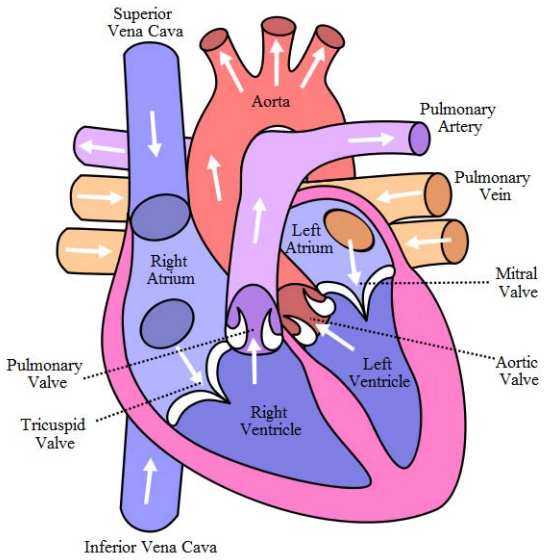


Figure 2–3: Heart Parts (Created by Wapcaplet in Sodipodi. Cropped by Yaddah to remove white space (this cropping is not the same as Wapcaplet's original crop). from Wikipedia)

Heart Cycle

The cardiac muscle is contracted and relaxed to maintain a normal blood flow in the body. When the right atria is relaxed the blood coming from the body through the vena cava fills it, then it contracts and, through the tricuspid valve, fill the right ventricle. Once the right ventricle is filled, it contracts and pumps the blood to the lungs through the pulmonary valve where it is oxygenated. After passing through the lungs, the blood reaches the left atrium. This one contracts to fill the left ventricle through the bicuspid valve. After that, the left ventricle contracts and passes the blood to the body. Analyzing this cycle, it can be inferred that the high pressure side of the heart is the left side and the low pressure is at the right side. The way in which the heart is contracted is controlled by the cells at its conduction system.

Tissues Behavior

The cardiac muscle is composed of different types of tissues such as sinuatrial (SA) and atrioventricular (AV) nodes, atria, Purkinje, and ventricular tissue. The

cells of each type of tissue differ in anatomic characteristics and are electrically excitable, but each one presents its own characteristic action potential. Each tissue can have spontaneous excitation after a certain period of time, been the natural pacemaker the SA node, which has a shorter excitation period. As can be appreciated in Figure 2–4, the conduction system starts at the top with the SA node which excites the atria muscle, followed by the AV node where a delay of approximately 120 to 200 ms occurs. That gives time to the atrium to fill the ventricles. The action potential propagates to the Bundle of His and Purkinje fivers that propagate it through the ventricles, which contract to pump the blood.

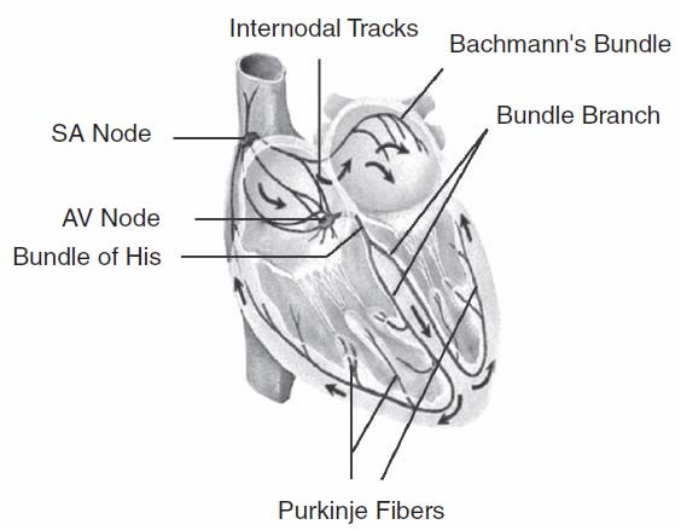


Figure 2–4: Heart Conduction System [10]

The electrogram (EGM) and electrocardiogram (EGC) are two methods used to monitor the electrical behavior of the heart. In the EGM, the measure is made directly to the heart. In the other hand, the ECG measurement is made over the body surface. Figure 2–5 shows a typical ECG. The P wave is where atria depolarization occurs. PR interval represents the AV node delay. QRS complex is where the ventricular depolarization takes place with duration of 50 to 100ms and, finally, T wave is where the ventricular repolarization occurs. The normal heart rate (normal

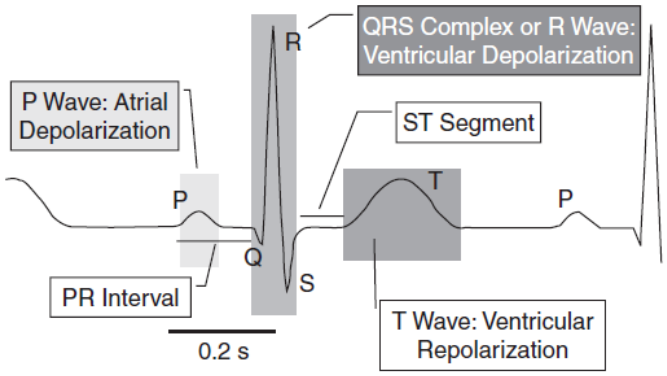


Figure 2–5: Electrocardiogram [10]

sinus rhythm) is about 70 beats/minute. ECG is a measure of the electrical behavior of the heart and mechanical activity cannot be deduced from it. For mechanical activity other methods like blood pressure measurement have been developed.

Diseases

The cardiac muscle is susceptible to many diseases that can affect its performance and alter its normal rhythm. Between them are vascular and conduction system diseases. The vascular diseases are related with the blood flow and include valves and circulation problems. In the conduction system, diseases have two divisions: one for high rate (higher than 100 beats per minutes (bpm)) and one for low rate (bradicardia (less than 60 beats per minutes (bpm))). High rate conditions have classifications such as tachycardia caused for example by SA node disease. Other type of high rate disease is known as fibrillation, where the heart has unorganized contractions that doesn't pump blood. Low rate diseases are known as bradicardias and are caused mainly by blocks in the heart's conduction system. Between these diseases is the sinus node disease -also known as sinus arrest-, which is defined as an affliction of the sinuatrial node that either prevents or delays the impulse generation [6]. Other low rate disease is the atrioventricular node system disease. In the latter, the AV node conducts slower than the heart rate (2-1 block, 3-1 block, etc) or does

not conduct at all (total AV block).

2.2 Treatment Devises

Since the conduction diseases are due to malfunctions of the electrical conduction system, the modern medicine is using some devices that try to compensate these failures to return the normal activation cycle to the heart. These devices are classified by the type of disease that they treat. For fibrillation disease, defibrillators are used. They are devices that deliver to the heart a high energy impulse to virtually reset the cardiac muscle when a fibrillation is detected. Sometimes, pacemakers are used for treating tachycardia. In this case, the heart is stimulated by the pacemaker at a rate higher than the tachycardia. Then, the pacemaker starts to decrease the rate slowly until it reaches the normal rhythm. In the case of the low rate syndromes, pacemakers are used but this time as a pulse generator that stimulates the heart if the rhythm is slower than normal.

2.2.1 Pacemaker Clasification

The pacemakers are classified depending on the chamber paced, chamber sensed and the mode of response if the simplest codification scheme is used. As can be appreciated in Table 2–1 each category has different letters that represents the option in each field. For example, DDD pacemaker sense atrium and ventricle; pacing is inhibited in atria channel by sensed ventricular or atria activity and is inhibited in ventricular channel by ventricular activity but triggered by atria activity.

This classification divides the pacemakers in two groups: the group that works in only one chamber (single chamber) and the group that works in both chambers (dual chamber).

Table 2–1: Three Letters Pacemaker Code [11]

Position	1^{st}	2^{nd}	3^{rd}
Category	Chamber(s) Paced	Chamber(s) Sensed	Mode of Operation
Letters	V = Ventricle	V = Ventricle	T = Triggered
	A = Atrium	A = Atrium	I = Inhibited
	S = Single	S = Single	O = None
	D = Double	O = None	D = Double (Inhibited & Triggered)
		D = Double (A & V)	

2.2.2 Hardware

Pacemakers are composed of four main parts: (1) Input stage, (2) output stage, (3) housekeeping, and (4) control unit.

The input stage has the task of detecting the heart activity and to notify the control unit that an event has occurred. Commonly, it is composed by an electronic amplifier that amplifies and filters the cardiac signal. In Figure 2–6 the block diagram of the sense amplifier can be observed. It shows that the input signal can be unipolar or bipolar. A unipolar connection is between the tip and the can, and the bipolar is between the tip and the ring. Next, it has the amplifier to increase the signal amplitude, a band pass filter to clean the signal, and the comparator with a programmable sensitivity threshold generator to compare the input signal with a fixed reference.

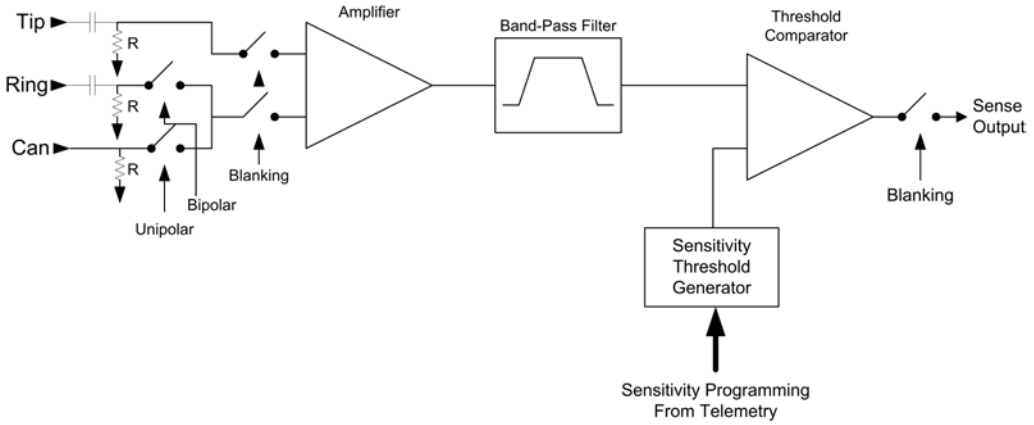


Figure 2–6: Input Sense Amplifier

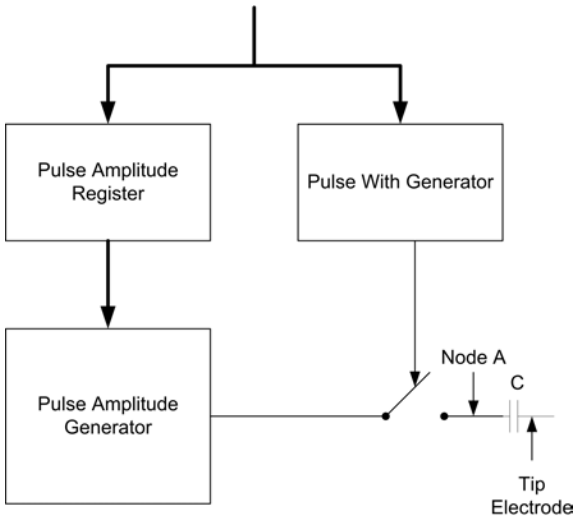


Figure 2–7: Output Stage

The output stage has as function to deliver a pacing pulse to the heart tissue through the pacing lead. This pulse has amplitude and a pulse width. All these characteristics are controlled by the control unit with the values programmed by the doctor as shown in Figure 2–7. The pulse amplitude is generated by a voltage multiplier or divider and the control unit.

Housekeeping is the part where the well functioning of the pacemaker device and any irregular change in the battery are monitored. If a voltage below a threshold is detected, this device will reset the control unit in order to ensure that the data is correct. It also has a battery charge detector, where the control unit can verify how much charge remains. It is important for its own functionality and is saved for the doctor in the follow up visits. Another function is the verification of leads impedance in order to avoid any pace failure.

Finally, the control unit is composed by the needed logic to keep time, pace parameters, develop the pacing algorithm, save data, communicate with doctor by telemetry, etc. It is the part that controls the functionalities of the pacemaker. In it, digital timers are used to keep time and pulse width while registers or memory store the pace parameters. Custom logic or a microprocessor is needed to develop the

pacing algorithm and, finally, a telemetry system is used for communication with the programmer.

2.2.3 Single Chamber Pacemakers

The single chamber pacemaker senses and/or stimulates only in one chamber. It can be asynchronous or synchronous. The asynchronous type is not used today since it can cause death. A synchronic pacemaker monitors the heart activity to decide if it stimulates the heart or not. This device is composed of one input stage, one output stage, the housekeeping and the control unit. In this way, it is used only when the sense and stimulation is in one chamber. This device can cause additional conditions to the patient since it has no synchronization with the other chamber. The synchronization helps the heart chambers to fill completely before the contraction. As a consequence, the lack of synchronization cause that when the patient does physical activity, the heart will struggle more to pump the blood.

2.2.4 Dual Chamber Pacemaker

Dual chamber pacemaker is a device utilized to sense and pace two heart chambers. In this way it knows what happens in the other chamber and the synchrony to the heart is maintained. This device is composed by two input stages (one for each chamber), two output stages, one housekeeping and one control unit. That means that it is the same scheme of the single chamber, but with twice the hardware at the input and output stage in order to support two chambers.

2.2.5 Timing Cycle of a DDD Pacemaker

The timing cycle of a DDD pacemaker has seven divisions. These are some of the parameters that can be programmed by the doctor.

Lower Rate Interval (LRI)

This is the longest time between a ventricular events (sensed or paced) and the next one, without the intervention of a sensed events. In other words, this interval fixes the lower stimulation rate when a natural event does not occur [11].

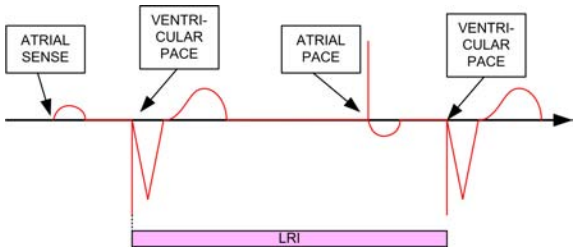


Figure 2–8: Lower Rate Interval

Ventricular Refractory Period (VRP)

This interval is initiated by a ventricular event and is the time in which the ventricle cannot be stimulated or sensed. In other words, during the VRP a new LIR cannot be initiated [11].

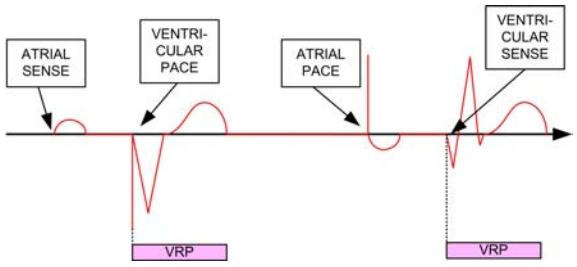


Figure 2–9: Ventricular Refractory Period

Atrioventricular Interval (AVI)

This is the time between an atrial event (sensed or paced) and the start of a ventricular stimulation. It is classified in the pAVI that initiates after a paced atrial event and the sAVI which initiates after a sensed atrial event [11].

Atrial Escape Interval (AEI)

The atrial escape interval is the time between a ventricular event and the atrial stimulus. It is given by the equation 2.2 which is used by the processor to calculate this value [11].

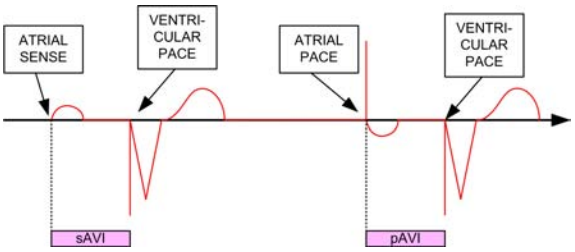


Figure 2–10: Atrioventricular Interval

$$AEI = LRI - AVI \tag{2.2}$$

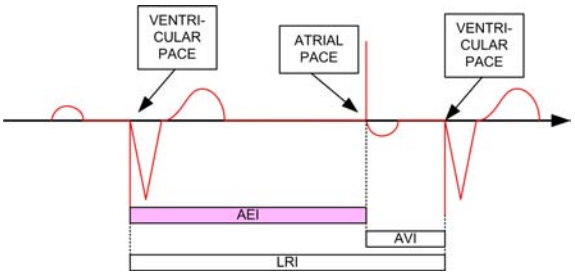


Figure 2–11: Atrial Escape Interval

Postventricular Refractory Period (PVARP)

The postvetricular refectory period is the interval after a ventricular event where an atrial event cannot be sensed. This means that the atrial sensor can be turned off [11].

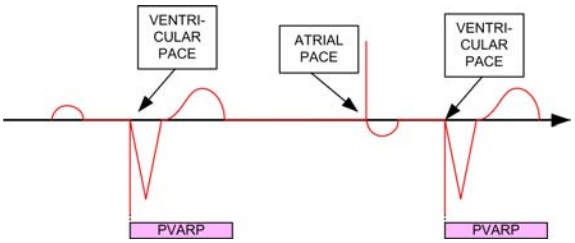


Figure 2–12: Postventricular Refractory Period

Total Atrial Refractory Period (TARP)

Is the shorter period of time between two sensed atrial event. In this way when the time between two atrial events becomes shorter than TARP the second will not be recognized. This interval is defined by the equation 2.3 [11].

$$TARP = AVI + PVARP \tag{2.3}$$

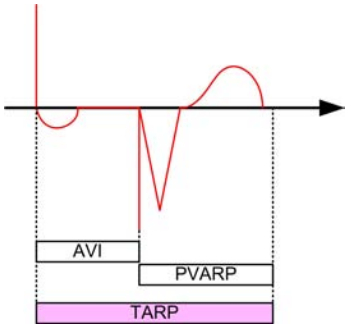


Figure 2–13: Total Atrial Refractory Period

Postatrial Ventricular Blanking (PAVB)

This is a brief period of time used to prevent the sense of the atria event by the ventricular sensor (cross talk). This is a short period of time that starts after an atrial event [11].

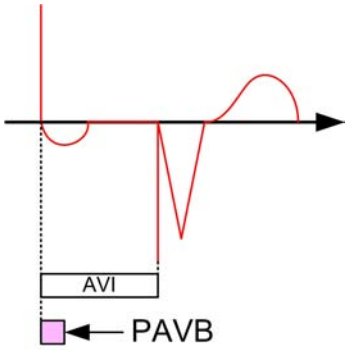


Figure 2–14: Postatrial Ventricular Blanking

Upper Rate Interval (URI)

This interval does the opposite than the LRI. It sets the fastest stimulation ratio for the ventricles. It shortest value has to be larger than the TARP [11].

2.3 Control Unit

The pacemaker's control unit main part is the logic control. This logic can be made in two ways: customized for a given pacemaker or with a microprocessor.

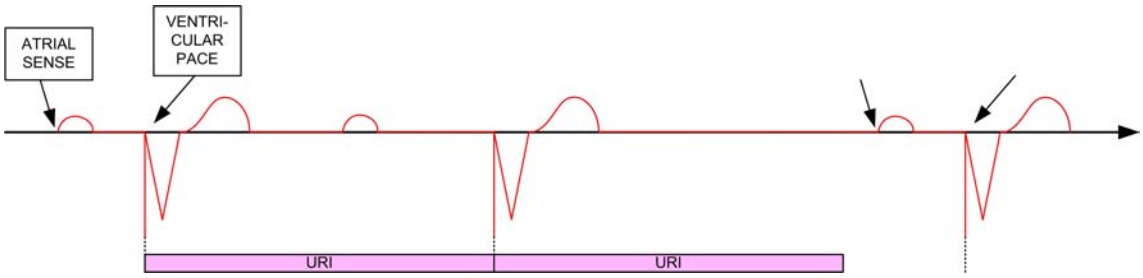


Figure 2–15: Upper Rate Interval

The custom logic has the advantage that its power consumption is low, but once it is made, further changes are not allowed. On the other side, the microprocessor consumes more power, but gives the flexibility to be used at different types of pacemakers by only changing its software.

2.3.1 Microprocessor

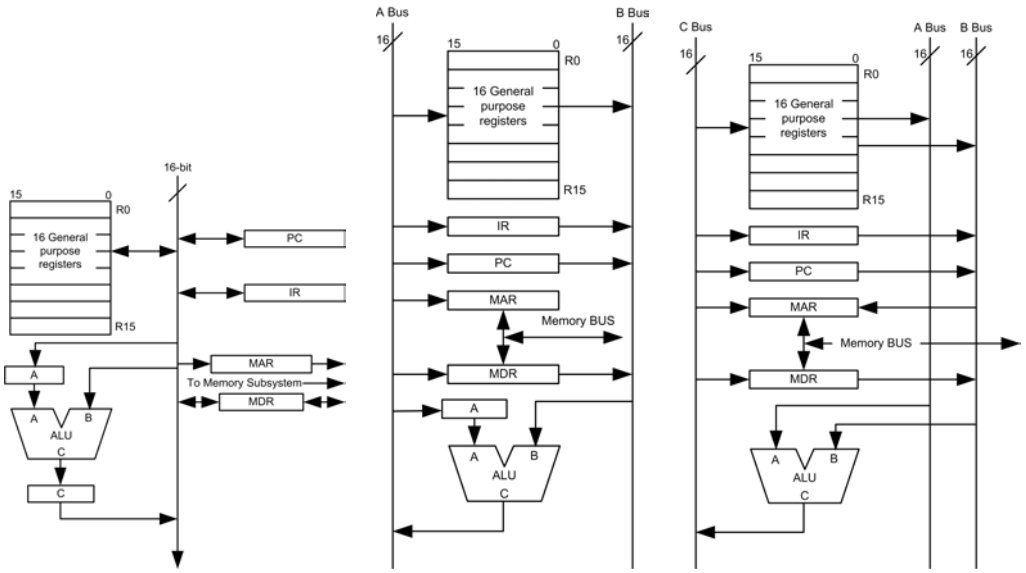
A microprocessor is a device that performs logic and arithmetic operations using temporal storage devices called registers. It is composed also by an arithmetic logic unit (ALU), and a data path and control unit that together can run the software. The software is stored in an external memory.

Architecture

Microprocessors can be designed for different applications. The two main divisions are Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computers (RISC). The first are computers for high performance since it can make different tasks with a single instruction. The second one are used more for low power and low level programming, since it has less instructions and, therefore, less hardware.

Datapath

The datapath is the part of the microprocessor that controls the flow of information. Commonly it is built with multiplexers or buffer tristates. Figure 2.3.1 shows three types of datapath. A datapath of one bus used to transport the two operands and the information back to the register destination is that of Figure 2–16(a). Figure 2–16(b) shows a datapath of two buses, one for the two source operands and one for the destination. Finally, Figure 2–16(c) presents a datapath of three buses, one for each operand and one for the destination.



(a) One Bus Datapath (b) Two Bus Datapath (c) Three Bus Datapath

Figure 2–16: Bus Confuguration

Registers

Registers are storage units capable of holding a collection of bits. In a microprocessor, they are classified as general purpose and dedicated function registers. The general purpose ones are named architectural registers, given that are available to the programmer to manipulate information. They are used to store data, accumulators, address, etc. The dedicated function registers are used for specific

functions like holding the address of the next instruction, holding the present instruction, temporal registers, etc.

Arithmetic Logic Unit

The data manipulation takes place in the Arithmetic Logic Unit. It is mainly composed by an adder, subtracter, and, or, xor, shifter, among others. As output is received, the result of the instruction and bits named flags (Zero, Overflow, Carry, Negative) which are stored in the status register. These flags are manipulated mainly by arithmetic and logic equations and are used for conditionals.

Control Unit

The control unit (CU) is where the control signals are generated. It has the instruction register (IR) where the current instruction is hold as an input. With the information provided by the IR, the CU knows the ALU operation, the source and destination registers, and the algorithm to complete the instruction. It controls all parts of the microcontroller like datapath, ALU, and registers.

CHAPTER 3

Literature Review

Throughout the following pages, a comprehensive literature review of the progression of pacemaker performance through the years and how the power decrease of dual-chamber type can help in the future is presented.

3.1 Cardiac Pacing History

The first external pacemaker was developed in 1932 by Hyman [13] prolonging patients' lives from 24 to 48 hours. Some years later, in 1958, the first pacemaker was implanted in Sweden [14], containing an inductively rechargeable battery that had to be recharged once a week. This type of pacemakers did not have synchronization with the heart, causing a competition between the device and natural heart excitation. The lack of synchronization caused irregular heart beat in patients and sometimes death.

To avoid this, the "demand pacemaker" was developed in 1965[10]. The "demand pacemaker" sensed heart activity to avoid competition, at the same time that its power consumption was decreased.

The use of microcontrollers in medical devices was discussed by Berkman and Prak [15] in 1981. They developed a 4-bit microcontroller using CMOS (complementary-symmetry metal-oxide-semiconductor) technology in the weak inversion operation area, programmable input sensors and sleep modes to save the battery. Later, in 1985, a new microcontroller design for biomedical applications was developed by Stotts and Baker [16], using the same techniques outlined by Berkman and Prak,

included modifications such as 8-bit data, I/O ports, 6 timers and an instructions parity check to avoid errors. The block diagram for Stotts and Baker's design is shown in Figure 3–1.

These two designs were great advances in the area of biomedical implantable devices however, for today's needs some extra applications must be included. Among them are the addition of hardware and software interrupts to save energy, addition of memory for electrocardiogram recording, and a detailed instructions study to make sure that only the needed instructions were included in the processor, since each extra instruction means extra hardware and consequently extra power dissipation. These features were developed as part of this thesis. Stotts's processor was used as the core on later designs.

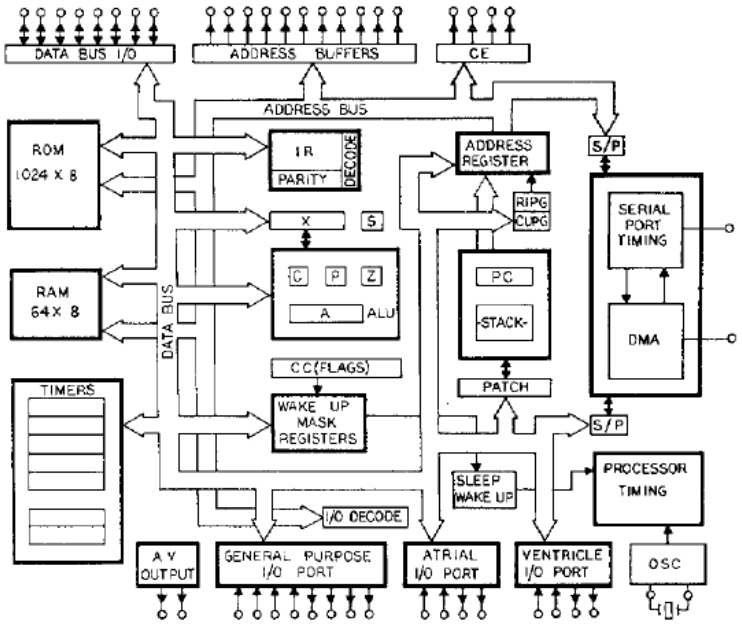


Figure 3–1: Block diagram of 8 bits microcontroller [16]

In 1989, Stotts and Baker [17, 18] proposed the implementation of a pacemaker in a single chip. Their design included the microprocessor, a sense amplifier, a voltage multiplier, and timers. The microprocessor allowed the programming and controlling of device functions. The sense amplifier was used to sense body signals,

and the voltage multiplier was used to control the amplitude of the output signal. The timers had the function of keeping the timing and control the pulse width of the output signal. During operation, a drain current of 9 microamperes at 3 V and a minimum operating voltage of 1.7 V were achieved. Although encouraging results were obtained, this pacemaker didn't include some important devices or applications in their IC design.

Some of the missing details were a battery management system, a lead impedance sensing circuit, and enough memory for data storage for follow up of the device and the patient's condition.

In 1993, David B. Shaw and Michel Horwood [19] studied task recording on the pacemaker. Electrograms and blood pressure are some of the cardiac events that can be recorded by a pacemaker for follow-up diagnostics. Shaw and Horwook calculated the memory size necessary in order to record irregular cardiac events. The sampling rates used were 400 and 100 samples per second for electrograms and blood pressure, respectively. A memory of 128 kilobytes was used in order to save 16 seconds in a circular buffer and 16 arrhythmia events.

The evolution of the different parts of the pacemaker was compiled by Sanders and Lee [20] in 1996. In terms of battery technology, the earliest pacemakers needed 4 to 5 mercury zinc batteries to operate, given that each battery was only 1.35 Volts. Nowadays, only one lithium iodine battery is required and it is used to supply 2.8 Volts. On the subject of the electronics, the research has been focused on decreasing the drain current and increasing options flexibility by including digital controllers (microprocessors). The use of software has provided the advantage of using the same hardware with various pacemaker types, changing only its configuration. The software also has the advantage that permits the modification of its parameters without surgery.

Throughout the years, the interest in a low-power pacemaker has been increasing. In 2004, the complete design of a low power dual-chamber pacemaker was developed by Wong, et. al. [7]. In their design, they used custom control logic instead of a microprocessor. They divided their design in four stages: 1) the input, 2) the output, 3) housekeeping, and 4) logic. The input stage consists of an ADC, and a series of amplifiers and filters that forms the sensing system. The output stage consists of a high voltage multiplier and a high voltage output pulse generator, which are used to control the amplitude and the pulse duration of the output signal. The housekeeping consists of the battery management system, the bias and the reference generator. Finally, the logic circuit contains algorithms for therapy control and oscillators. Their design showed an average power of 8 microwatts at 2.8 Volts, with chip dimensions of 7 x 7 mm. This design is shown in Figure 3–2. Notice that it has separate input and output leads for each chamber.

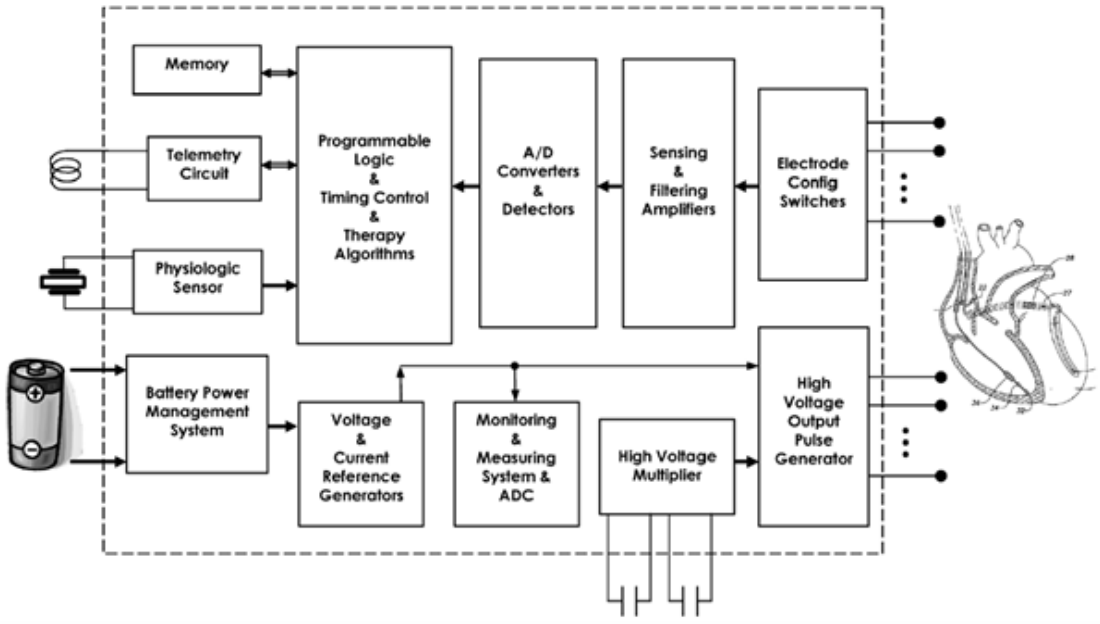


Figure 3–2: Low Power Pacemaker Block Diagram Presented in [7]

Gonzalez [5] designed a low-power software for a "demand pacemaker" in 2004. His software was designed based on measurements of the current drained by each

instruction. Other points taken into consideration were the reduction of memory access and the reduction of ones, since ones increase the use of current. This was achieved using a low power general purpose microcontroller (MSP430) and discrete components. The final design consumed approximately 9μA with a 3 Volt battery, which are reasonable values. These values could be improved by developing the pacemaker in a single IC with a custom microprocessor.

The controversy between the dual-channel and single-channel pacing was discussed in 1998 by Ovsyshcher, et. al. [21]. Here, they use some studies to validate the theory of the superiority of the Dual-Chamber pacemakers in some treatments. They concluded that dual-chamber pacing resembles the normal way of cardiac activation. However, they have some disadvantages such as high cost and high power consumption, which reduces the life of the device. At the same time, they did not discard the single-chamber pacemaker because they are still needed as treatment for some conditions.

3.2 Pacemaker Current Trends

Today's dual-chamber (DDD) pacemakers are composed by two input and two output stages as shown in Figure 1–1[6, 11], which are used to sense and stimulate each one of the two right chambers, in order to keep them synchronized. As was depicted, DDD pacemakers are currently getting more popularity, since they have many health advantages. However, their high cost and power consumption (battery life is more than 1 years shorter than single-chamber) still make them inaccessible for a great number of patients.

3.2.1 Power Consumption

Over time, pacemaker power consumption has been a great constraint in the design of these devices. There are many factors that affect power consumption such as leads impedance, patient's physiology, electronic static current dissipation, etc.

This thesis is focused on circuitry current dissipation. Tables 3–1 and 3–2 illustrate an example of a typical case of a DDD pacemaker. The first one shows the programmable parameters for the example and the second one are the current dissipation for a variety of stimulus amplitudes.

Table 3–1: Programmed Parameters [6]

Parameter	Value	Units
Lead impedance (R)	500	ohms
Rate	60	bpm
Percent Paced	100	%
Pulse With (PW)	0.5	msec
Output Capacitance (C)	10	μF
Static Current Drain	10	μA
Battery capacity	1	Ah

Table 3–2: Effect of stimulus Amplitude on Pulse Generator Longevity [6]

Stimulus Amplitude (V)	Multiplier (X)	C_{eq} (μF)	Pacing Current (μA)	Current (μA)	Longevity (yr)
1.4	0.5	20	0.7	11.4	10.04
2.8	1	10	2.7	15.3	7.44
5.6	2	5	10.2	30.3	3.76
8.4	3	3.3	21.8	53.3	2.13

From this data, the relation between the overall pacemaker current dissipation can be extracted. The current dissipated by the circuitry, the overall current and the stimulus amplitude are specified by [6] to be 5μA, 15.3μA and 2.8V respectively. This means that a reduction in the power dissipated by the electronics affect around 1/3 of the total current consumed.

3.2.2 Dual Chamber Pacemaker Timing

Figure 3–3 shows the time diagram of a DDD (Made as in [11]). From this figure, the timing constrain of a DDD can be extracted. The values of these intervals are depicted in Table 3–3.

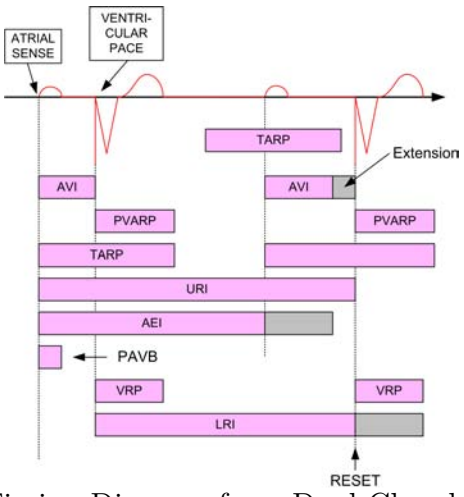


Figure 3–3: Timing Diagram for a Dual Chamber Pacemaker

Table 3–3: Dual Chamber Pacemaker Time Cycles [22]

Abbreviation	Description	Time
sAVI	Post Sensing AV Delay	53 ms to 272 ms
pAVI	Post Pacing AV Delay	53 ms to 272 ms
PVARP	Ventricular Refractory Period	195 ms to 430 ms
TARP	Total Atrial Refractory Period	248 to 702 ms
URI	Upper Rate Interval	60 to 147 min-1
AEI	Atrial Escape Interval	454 ms to 1.95 s
PAVB	Post Atrial Ventricular Blanking	21 ms to 68 ms
VSP	Ventricular Safety Pacing Window	100 to 110 ms
VRP	Ventricular Refractory Period	195 ms to 430 ms
LRI	Lower Rate Interval or Basic Pace	30 to 132 min-1
	Atrial and Ventricular Pulse Widths	0.122 ms to 1.464 ms

From the information presented, the time that limits the design speed can be obtained. It is the Post Atrial Ventricular Blanking (PAVB) which has a duration of 21 ms. The pulse width is not a constraint since it is generated by a timer programmed by the CPU.

3.3 Problem

Dual-chamber pacemakers have great advantages over the single-chamber ones, having its power consumption and cost as limiting factor. For that reason, this thesis focuses on the developing of a specialized CPU architecture for dual-chamber

pacemakers. This architecture will offer an alternative that keeps the advantages of dual chamber pacemakers while offering cost and power consumption values close to the single chamber ones. A custom CPU architecture was optimized to use the less number of hardware for the application. This architecture was focused on the peripherals optimization, as well as on reducing their use.

CHAPTER 4

Objectives and Methods

4.1 Objectives

Pacemakers are devices used worldwide to treat bradarrhythmia. Among them the dual chamber pacemaker is the most likely to be used, but it cost and power consumption presents serious obstacles to be used by many people. The solution of this problem has many aspects. As a first step, this thesis develops a CPU specialize for dual chamber pacemaker application. Specifically this thesis aims to design a custom CPU architecture optimized to contain only the hardware needed to support dual-chamber pacemaker tasks. At the same time it incorporates specific applications for pacemakers. With this objective we achieve reduction in power and size which is one of the main issues in integrated circuits.

In order to achieve this goal three objectives should be fulfilled:

1. Reduce the processor instructions set. The amount of hardware supporting instructions is less than that in a processor with all its instructions, achieving the reduction of the static power dissipation of this non used hardware in the processor. Some of the parts that will have fewer components are the arithmetic logic unit (ALU) and the control unit.

2. Incorporate a system that will be used for fast on/off swithching of the pacemaker peripherals when they are not being used. The ability to shut down these

peripherals also achieves the reduction of the general power in the pacemaker eliminating power consumption when devices are not realizing an instruction.

3. Develop a special sector of the CPU to control the operation of the multiplexers that will decide which lead will have the input and output hardware in an efficient way and reduce power consumption. This presents a solution for the high power consumption in dual chamber pacemakers. These devices sense and stimulate in the two heart chambers (atrium and ventricle) so they have an input system for each chamber (2 inputs hardware) and an output system for each chamber (2 outputs hardware) [6]. This is one of the reasons why their electronic spend more power than the electronics of those of a single chamber. The proposed solution to this problem is to use a single hardware for both inputs and outputs using only one input hardware and one output hardware. They will be controlled by analog multiplexers to decide which chamber will use them.

4.2 Methods

The design of this CPU is divided into four main parts:

1. Architecture Selection
2. Instruction Selection
3. 1's Reduction
4. Registers Increment and Classification

For this design, a Reduced Instruction Set Computer (RISC) architecture is preferred since it offers many advantages for low power operation [23]. A 16-bit address bus provides sufficient memory space to save users' histograms, electrograms (EGM), battery status and other important data. The public pacemaker software in [5] uses each bit of the registers to store the pacemaker parameters, so a large

data bus is required (in this case one of sixteen (16) bits is used). Additionally a low power sleep mode will be included.

Instruction Set Array (ISA) for this CPU starts with a detailed study of the core instructions of the MSP430 microcontroller. In order to choose the minimum number of instructions and addressing modes required, the open source software found in [5] was analyzed, since it was done precisely for the MSP430. From this data the instruction register format, Arithmetic Logic Unit (ALU) operation, CPU organization, and registers quantity and organization are selected.

In addition to the ISA size, other low power software techniques are to be used in this design. As was demonstrated by [5, 24] the amount of 1's in a given instruction affects the power it consumes by, increasing the static power consumption since this implies more leakage current. Another technique comes from the CMOS gate transition. If successive data that will go through a gate is switching from zero to one and vice versa, the gate goes many times through the short-circuit state. This means that the data must have less changes on its bits to reduce power. With this in mind, in the selection of op-codes, registers and ALU operations code we assigned more 0's to the codification of the more frequently used instructions and operands. Also there are reduced the differences in the op-codes of instructions that usually go together. Additionally, using sixteen general purpose registers allows us to reduce power and increase performance by decreasing the necessity of memory access.

To add performance using sixteen general purpose registers, an additional register set was used. It is assigned to dedicated functions such as program counter (PC), status register (SR) and stack pointer (SP) for the functionality of the CPU. Additionally, other registers in this new set are dedicated to the interrupt enable (IER), general interrupt (GIR), device control (DCR) and multiplexers control to manage the pacemaker peripherals. This result in faster manipulation and monitoring some peripherals with lower power consumption.

CHAPTER 5

Design Pocedure and Specifications

5.1 Registers

The proposed architecture has two register sets as depicted in Fig. 5–1. This allows more convenience to fulfill the need of extra registers to control external peripherals. Set zero (0) is for the general purpose registers and set one (1) for dedicated functions. Both groups of registers are accessible to the user at any time.

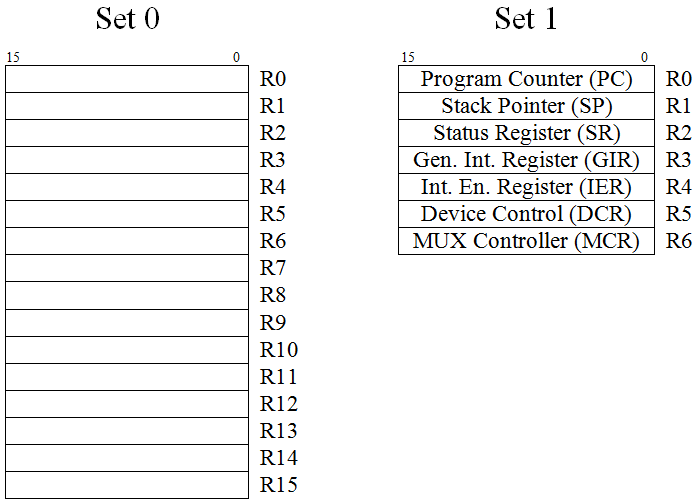


Figure 5–1: Register Sets

The special purpose register set has seven registers: Program Counter (PC), Stack Pointer (SP), Status Register (SR), General Interrupt Register (GIR), Interrupt Enable Register (IER), Device Control Register (DCR) and Multiplexer

Control Register (MCR). All registers can be in any instruction as either source or destination exept in the following restrictions:

1. No memory to memory transfers are allowed.
2. Two Registers of set 1 cannot be used in the same instruction.

The Program Counter (PC) and the Stack Pointer (SP) behave as usual; the SP must always be initialized by the user.

The Status Register (SR)(Figure 5–2) stores the flags information and some control bits of the CPU. It is register R2 of the Register Set 1. A description of the function of these bits can be found in Table 5–1. The four less significant bits are affected by arithmetic and logic operations. The Interrupt Enable (IE) bit is used to enable all the maskable interrupts. The Oscillator Off (OSC OFF) when set, turns off the crystal oscillator (it can be used to disable multiple clock dependent devices). The CPU OFF, when set, turns off the CPU.

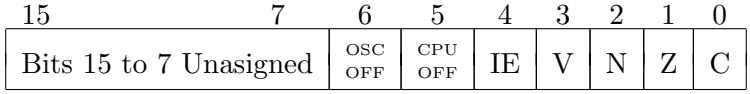


Figure 5–2: Status Register

The general interrupt register (GIR) is used to classify and identify the interrupt that is occurring. In this register each bit represents an exclusive interrupt action. When a bit is set this means that a given interrupt has occurred. The priority of the interrupts starts with the least significant bit through the most significant bit, as shown in Figure 5.1.

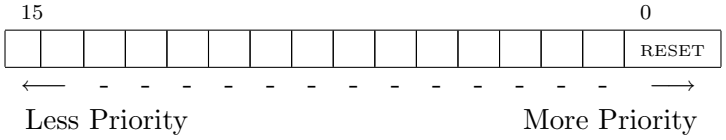


Figure 5–3: General Interrup Register

Table 5–1: Status Register

Bit	Description
C	Carry Bit: This bit is set when arithmetic or a logic operation produce a carry and reset when is not produced. ADD, SUB, CMP: Set if an arithmetic carry is produced and reset otherwise. XOR, AND, BIT: Is set if the result is not zero (not (Z)) and reset otherwise.
Z	Zero Bit: This bit is set if all the bits in the result of an operation are zero and reset otherwise.
N	Negative Bit: This bit is set when the most significant bit (bit 15) of the result in an operation is one (1) and reset if it is zero (0).
V	Overflow Bit: It is set by an overflow in an arithmetic operation of by some logic operations ADD: Set if: $Positive + Positive = Negative$ or $Negative + Negative = Positive$, reset otherwise SUB, CMP: Set if: $Positive - Negative = Negative$ or $Negative - Positive = Positive$, reset otherwise XOR: Set if both operands are negatives and reset otherwise. AND, BIT, XOR: Always Reset
IE	Interrupts Enable: This bit is used to enable all the maskable interrupts.
OSC OFF	Oscillator Off. This bit, when set, turns off the crystal oscillator.
CPU OFF	CPU OFF. This bit, when set, turns off the CPU. To leave this mode, this bit must be modified in the stack during an interrupt.

The interrupt enable register (IER) is used to enable interrupts. Each single bit of this register represents a given interrupt. The bits position in the GIR and INR are used for the same interrupt.

The device control register (DCR) is used to turn on and off the pacemaker peripherals when they are not being used. Each bit controls one device or some devices that always work at the same period of time. For example, one bit connected to the section of the battery management system which measures the battery voltage will be disabled until the software wants to verify how much charge the battery has.

This practice decreases the amount of static power dissipated by the device thereby reducing the total power consumption and processing time.

The multiplexer control register (MCR) is dedicated to the control of the multiplexers that manipulate the input and output sides of the pacemaker. For example, one bit could be for the multiplexer that connects the input stage (composed with amplifiers, filters, etc.) with the atrial or ventricular lead. In this way the programmer, using only one input stage, can sense the two chambers. With the DCR and the MCR the two main contributions of this thesis are achieved.

5.2 Interrupts

Interrupt is the best way to detect a peripheral request since it is activated when an event occurs. To control the interrupts, two registers must be configured: the bit IE in the status register and the bit corresponding to the interrupt in the IER (non-maskable interrupt cannot be disable). When an interrupt ocurrs, the processor does the following steps:

1. Move the PC to the top of the stack (TOS)
2. Move the SR to the TOS
3. Reset the SR

The interrupt priorities were selected using the timing information discussed in theory. The highest priority was given to the reset, since if any perturbation in the power source occurs, the data in the CPU can be damaged, so the first step in this situation is to reset the CPU to avoid delivering the wrong treatment to the patient.

The following in priority is the ventricle sense interrupt. This one is the only non-makeable interrupt. It is important to give it a high priority, because if the ventricle is stimulated after an event occurs, it can fibrillate and the patient will die.

Then, the next priority is for the atrial sense interrupts. It is important to avoid pace the atria after a natural event.

The follow DDD cycle interrupts were placed by the function they do. Then, the housekeeping and telemetry response interrupts were placed.

Table 5–2: Interrupts Priority

GIE Bit	System Interrupt	Vector	Description
0	RESET	00000h	Restore default values and Start software.
1	Ventricle Sense	00001h	Starts the VRP, PVARP AEI and LRI/URI periods and prevent ventriculas pace.
2	Atrium Sense	00002h	Starts the AVI, PAVB and prevent atrial pace.
3	Ventricular Pace	00004h	Stimulation to ventricle.
4	Atrial Pace	00005h	Stimulation to atrium.
5	Ventricular Refractory Period	00003h	Period were the ventricles cannot be sensed or paced.
6	MUX/Device Control	00006h	Manage the MCR and DCR.
7	AVI	00007h	Atrioventricular Interval
8	PVARP	00008h	Postventricular Refractory Period
9	AEI	00009h	Atrial Escape Interval
10	PAVB	0000Ah	Postatrial Ventricular Blanking
11	Electrogram	0000Bh	Take electrograms for histograms.
12	Battery Check	0000Ch	Verify battery charge.
13	Impedance Check	0000Dh	Verify leads impedances.
14	Telemetry	0000Eh	Programmer detection and communication.
15	Reed Switch	0000Fh	Enter in VOO mode if the reed switch is activated.

5.3 Instruction Selection Process

In the instruction selection process the MSP430 Instruction Set Array (ISA) was taken as a starting point. It has 27 core instructions. The instruction usage in the pacemaker software was counted to verify the most and less frequently used instructions. The result of this process is shown in the histogram in Fig 5–4.

From this data, the sixteen instructions shown in Table 5–3 were selected. These instructions are essential for a complete code and are the most used as . Keeping in mind the relationship between the quantity of 1's in an instruction and the power dissipation, the op-code for each instruction was assigned.

As it can be observed from Figure 5–4 and Table 5–3 data, the op-code was assigned depending on the repetition quantity of the instruction: the most used instruction gets the op-code with less 1's while less used instructions have op-codes

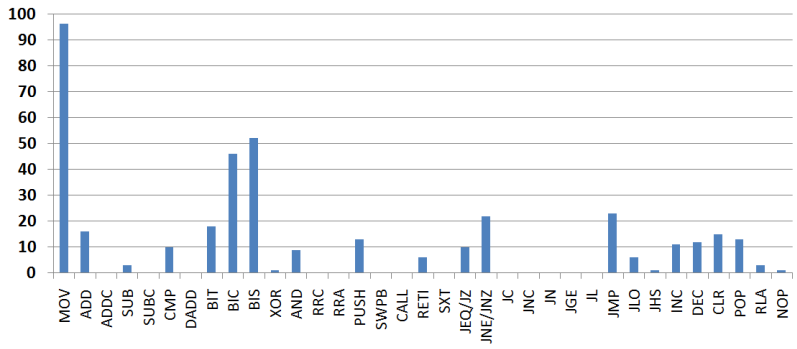


Figure 5–4: Instructions Usage Histogram

Table 5–3: Instructions

Op-Code	Mnemotic	S-Reg D-Reg	Operarion	V	N	Z	C
0000	MOV	src,dst	$src \rightarrow dst$	-	-	-	-
0001	BIS	src,dst	$src.or.dst \rightarrow dst$	-	-	-	-
0010	BIC	src,dst	$.not.src.and.dst \rightarrow dst$	-	-	-	-
0011	SUBC	src,dst	$src + .not.dst + C \rightarrow dst$	*	*	*	*
0100	BIT	src,dst	$src.and.dst$	0	*	*	*
0101	CMP	src,dst	$src - dst$	*	*	*	*
0110	AND	src,dst	$src.and.dst \rightarrow dst$	0	*	*	*
0111	JEQ/JZ	Label	Jump to label if zero bit is set	-	-	-	-
1000	ADDC	src,dst	$src + dst + C \rightarrow dst$	*	*	*	*
1001	RETI		$TOS \rightarrow SR,\ SP + 1 \rightarrow SP$ $TOS \rightarrow PC, SP + 1 \rightarrow SP$	*	*	*	*
1010	PUSH	src	$SP - 1 \rightarrow SP, src \rightarrow 0(SP)$	-	-	-	-
1011	XOR	src,dst	$src.xor.dst \rightarrow dst$	*	*	*	*
1100	POP	dst	$0(SP) \rightarrow dst, SP + 1 \rightarrow SP$	-	-	-	-
1101	JGE	Label	Jump to label if (N .XOR. V) = 0	-	-	-	-
1110	JC	Label	Jump to label if carry bit is set	-	-	-	-
1111	JN	Label	Jump to label if negative bit is set	-	-	-	-

* The status bit is affected
- The status bit is not affected
0 The status bit is cleared
1 The status bit is set

with more 1's. The source/destination format and affected flags it affects, were taken from the MSP430 specifications.

The selection of the addressing mode was done in the same way, resulting in four different modes. The format and description is shown in the Table 5–4. Only one addressing modes can be used one per instruction (only for source or destination), except for the register mode, which can be used as source and destination on the same instruction.

Table 5–4: Addressing Modes

As/Ad	Addressing Mode	Syntax	Description
00/0	Register	Rn	Register contents are operand.
01/1	Indexed	X(Rn)	$(Rn + X)$ points to the operand. X is stored in the next word.
10/1	Absolute	&ADDR	The word following the instruction contains the absolute address. X is stored in the next word. Indexed mode X(R0_0) is used for destine.
11/-	Immediate	#N	The word following the instruction contains the immediate constant N.

5.3.1 Instruction Register:

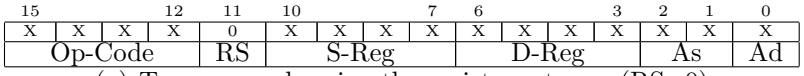
The next step in the design process is to develop the instruction register. Figures 5–5,5–6,5.3.1 show the formats for the different types of instructions. The IR format is selected by looking at the op-code of the instruction.

Two operand instructions have three different IR formats depending on the register set that is used as source or destination. Figure 5–5 show the different situations when both operands are from register set 0 and when the source or destination is from register set 1. Both operands cannot be from register set 1.

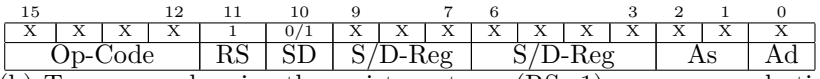
The fields are oganized as follow:

1. Op-Code refers to the operation code.
2. RS is the register set (0 for set zero and 1 for set one).

3. SD is used when one of the operands is from set one and define the bits 7 to 9 as source or destination (0 for source and 1 for destination).
4. S-Reg and D-Reg are the source and destination registers respectively.
5. As and Ad are the selector of the addressing modes as defined perviously.



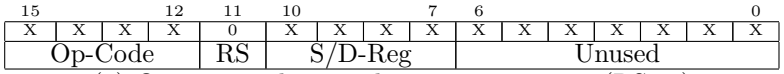
(a) Two operands using the register set zero (RS=0)



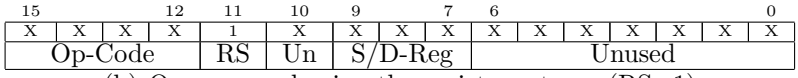
(b) Two operands using the register set one (RS=1) as source or destination (SD = 0/1)

Figure 5–5: Instruction Register Formats for Two Operand Instructions

There are two IR formats for one operand instructions as shown in Figure 5–6. These operations do not have addressing modes bits since they are only the push and pop instructions.



(a) One operand using the register set zero (RS=0)



(b) One operand using the register set one (RS=1)

Figure 5–6: IR Format for One Operand Instructions

The jump format in Figure 5.3.1 has only 12 bits offset since jumps are relative to the PC. The 12-bits PC Offset is a signed extended number that gives flexibility to jump forward and backward in the code.

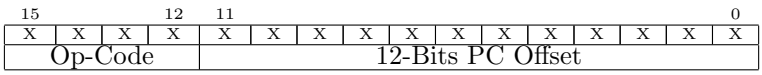


Figure 5–7: IR Format for Jumps

5.4 Zeros Optimization:

The quantity of 1's in an instruction affects its power consumption [5, 24] as mentioned previously. The trasitions in a CMOS gates take place also in the power disipation. For this reason the used instructions were optimized by referring to the frequency of instruction use, showed in Fig. 5–4. The op-code of those instructions was designed to use less 1's than instructions used less frequently as can be appreciated in Table 5–3. This decrease the static power and the switching in many gates. Another part of the CPU that directly affects the instructions operation is the ALU, where the operations were also classified depending on the frequency of use. The ALU instructions are shown in Table 5–5. The register use was also taken into consideration and for that reason the program counter is the register zero at the set one.

Table 5–5: ALU Operations

S3	S2	S1	S0		Operations
0	0	0	0	Bypass	F = A
0	0	0	1	INC	F = A + 1
0	0	1	0	OR	F = A .or. B
0	0	1	1	XOR	F = A .xor. B
0	1	0	0	AND	F = A .and. B
0	1	0	1	ADDC	F = A + B + C
0	1	1	0	SUBC	F = A - B - C
0	1	1	1	—NO—	—NO—
1	0	0	0	BIC	F = .not A .and. B
1	0	0	1	DEC	F = A - 1
1	0	1	0	CMP	F = A - B
1	0	1	1	—NO—	—NO—
1	1	0	0	ADD	F = A + B
1	1	0	1	—NO—	—NO—
1	1	1	0	—NO—	—NO—
1	1	1	1	—NO—	—NO—

5.5 Architecture

A three buss architecture is used to help the performance of the CPU, as shown in Figure 5–8. This choice was done since the additional power that can be consumed by the two additional busse's hardware is compensated by the additional cycles and temporal registers that are used by the architecture of fewer busses. It is

important to denote that more cycles implied more dynamic power consumed which is higher than static power consumption. Buses A and B are source and destination respectively, and bus C is for their response. This helps in reducing the cycles that a given instruction needs to be completed. The architecture also includes previously mentioned components such as the ALU, Register files (have inside set 0 and set 1), instruction register, memory data register (MDR) and memory address register (MAR) for communication with the memory and other important devices.

5.6 VHDL Design

VHDL and a Place&Route CAD tool were used to generate the layout for the processor.

In this tesis the processor behavior using VHDL was described and then passed to a Place and Rute Cad tool to transform it to the layout.

The process starts with the design done in VHDL where it is simulated and tested to ensure the right bahavior (VHDL in Apendix B). Then it is pased to a synthesize tool that converts the code into a schematic using a standard cells library. This schematic is saved in Verilog and loaded into the Palce & Route tool. This tool takes the verilog description and the standard cells library to place the cells into the die and then make the routing. These cells are an open source ami06 and 5V form the Oklahoma State University [25]. This layout is later exported to the layout editor to do the final rules verifications. After the layout is verified it is sent for building to finally be tested.

5.7 FPGA

An FPGA was used to phyisically verify that the design can be well synthetized by using the Cad tools. For this, the setup shown in Figure 5–9 was built using the Xilinx University Program Virtex-II Development System board [26].

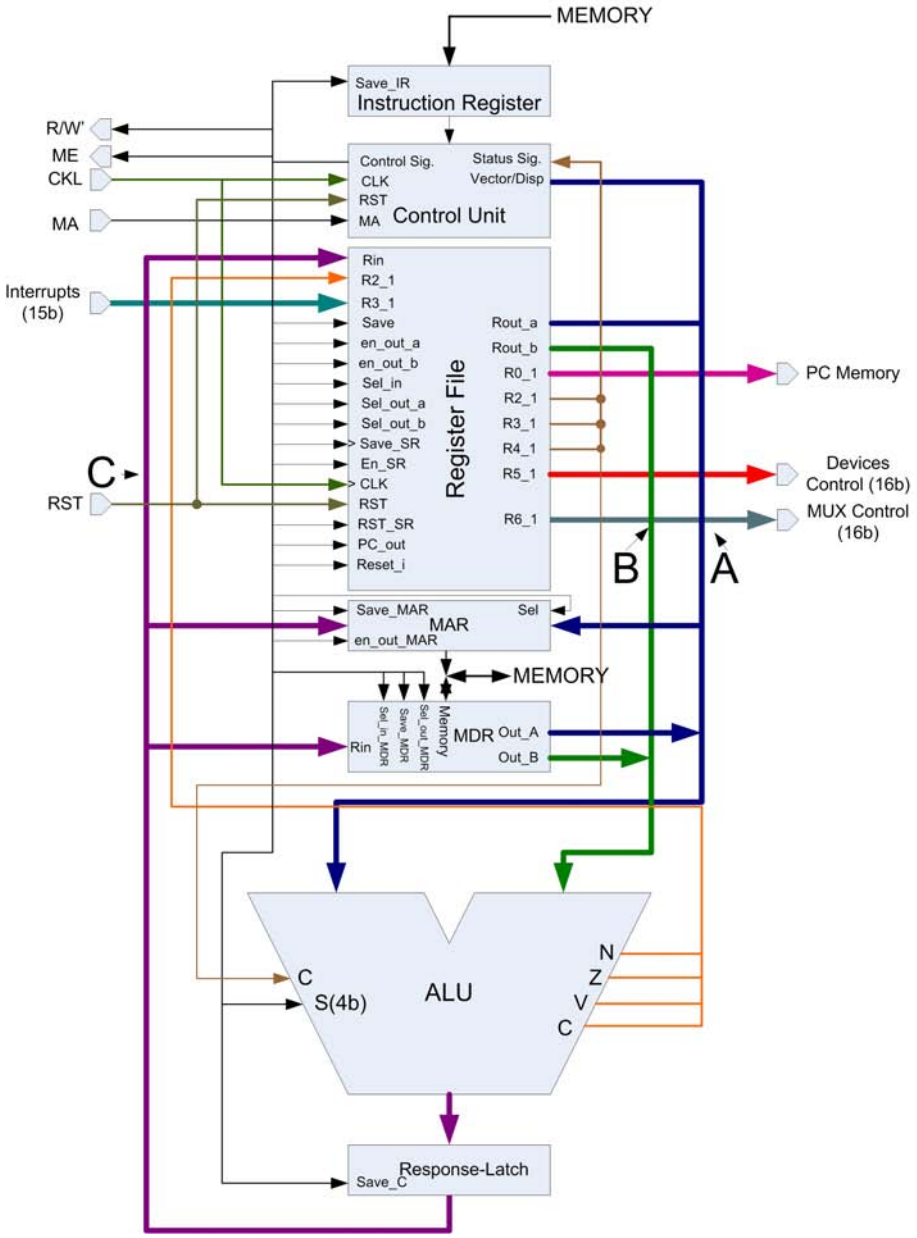


Figure 5–8: Architecture

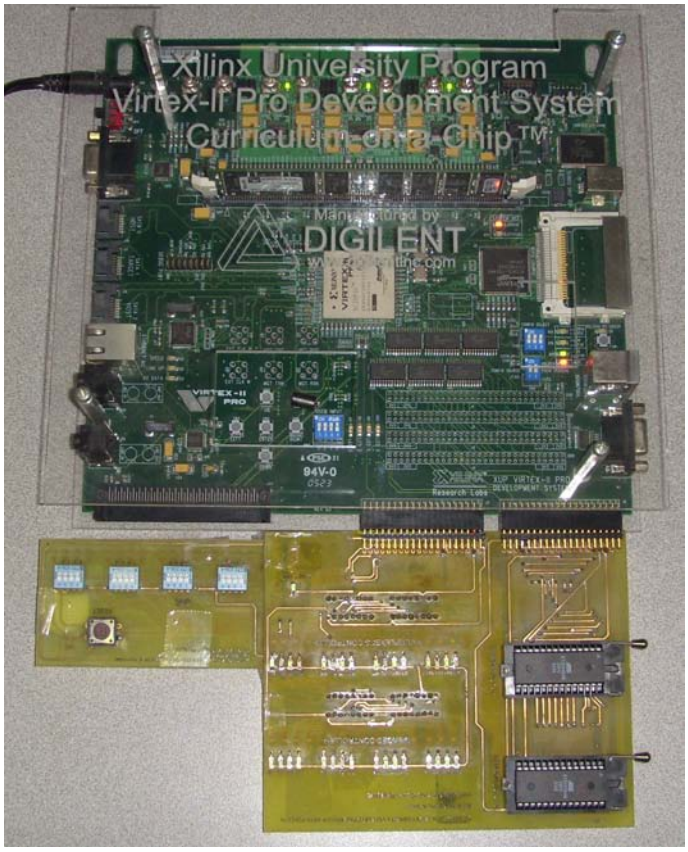


Figure 5–9: FPGA setup

The boards that contain the inputs and outputs were designed first. Two 8-bits EEPROMs were used as memory. To test the two peripherals control registers (MCR and DCR) 32 leds were placed. Finally, 16 switches were added to verify the interrupts. Additionaly, a push button for reset and an LED for the OSC_OFF signal were on the boards.

The CPU is burned in the FPGA assigning the conrserpondent I/O pins. The test programs are in the EEPROMs, and the result can be stored in the EEPROM, or one of the two device control registers. Apendix A gives a more detailed description of the boards.

CHAPTER 6

Results and Analysis

A custom CPU architecture that internally controls a dual-chamber pacemaker peripheral is studied and presented in this work. This chapter presents and discusses the obtained results. The simulations will be addressed first, fol ed by the measurements and finally the layout and power estimation.

6.1 Simulated Results

After the VHDL code was finished, a test bench was developed in order to test the final design. All the instructions (considering their variations) were tested in this test bench. Part of the test code will be presented in order to show the functionality of the VHDL code. Table 6–1 shows a memory segment with some of the instructions that were used, the memory location and the machine code in hexadecimal. Figures 6–1 to 6–4, shows the waveforms of the test bench simulation.

The first row of the table is the memory address 0000h where the address of the reset service routine (in this case is the address of the first instruction of the software) is located. This address is loaded into the program counter when the reset interrupt is triggered. As it can be observed in Figure 6–1, once the CPU is turned on it should be put in to a known state by doing reset. After that, the interrupt vector is loaded in order to get the address where the software starts. Once this is done, the code execution begins.

Table 6–1: Test bench Instructions

Address	Instruction	Machine Code
0000	Reset Vector	0010
0001	Sense Interrupt	0020
.		
.		
.		
0010	MOV #6699h, R5_1	0E86
0011		6699
0012	BIS #5555h, R0_0	1006
0013		5555
0014	ADDC R0_0, R6_1	8F00
0015	MOV #0030h, R1_1	0C86
0016		0030
0017	BIS #0002h, R4_1	1E06
0018		0002
0019	BIS #0070h, R2_1	1D06
001A		0070
.		
.		
.		
0020	MOV #F0F0h, R2_0	0016
0021		F0F0
0022	RETI	9000

In this example, the first instruction is in the 0010h address. This instruction sets the value 6699h in the DCR. Then, it sets 5555h to the MCR. This tests the instructions MOV, ADD, BIS, and the immediate addressing mode.

Now, in order to initialize the stack pointer the value 0030h in the register zero of the set zero was set, and then added to the stack pointer wich is initally zero. The reset is an important task since it guarantees that the stack is in a sector of the memory that does not affect software or data. Figure 6–2 shows that the immediate addressing mode is used for the instruction and also that the data was stored in the stack pointer register.

Then, the ventricle sense interrupt is enabled. This is done by setting the corresponding bit of the IER as shown in Figure 6–2.

Figure 6–3 shows how the processor enters a sleep mode and enables the interrupts. This instructions set the CPU_OFF, OSC_OFF and the IE bits of the status register, which were explained on Figure 5–2 and Table 5–1. In Figure 6–3 shows how the clock is turned off inside the processor.

The ventricle sense interrupt service routine is shown in Figures 6–3 and 6–4. When the interrupts occur, the PC and SR are moved to the top of the stack and the status register is reset. Then, the ventricle sense service routine vector is moved to the PC. The instruction that was done in the service routing was to set a value in the register two of set zero. Finally, the instruction to return from interrupt was used. This instruction returns the program counter and the status register values turning the CPU off again.

This example of the test benches ilustrates that all the functions of the design are are working correctly. This was done with an ideal clock of 25 MHz, which covers well the necessities of the application.

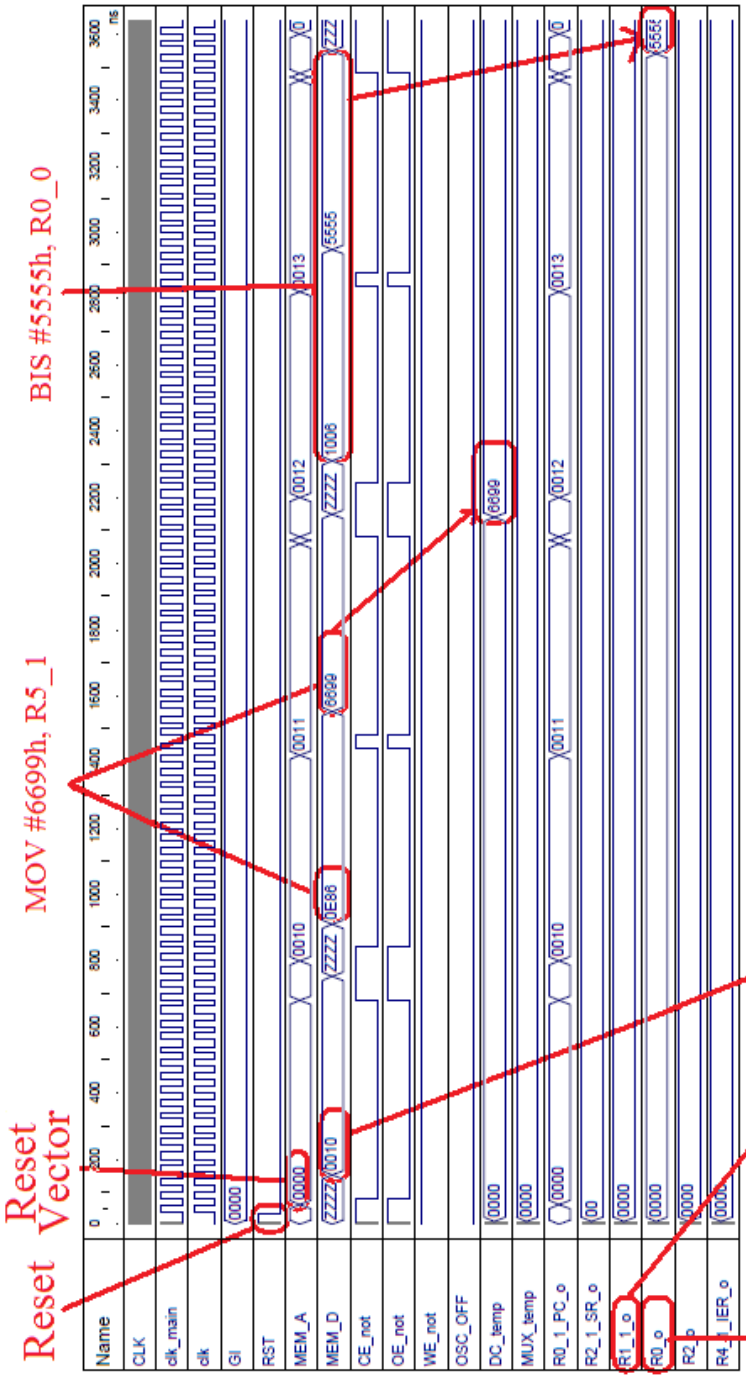


Figure 6–1: Test bench Part 1

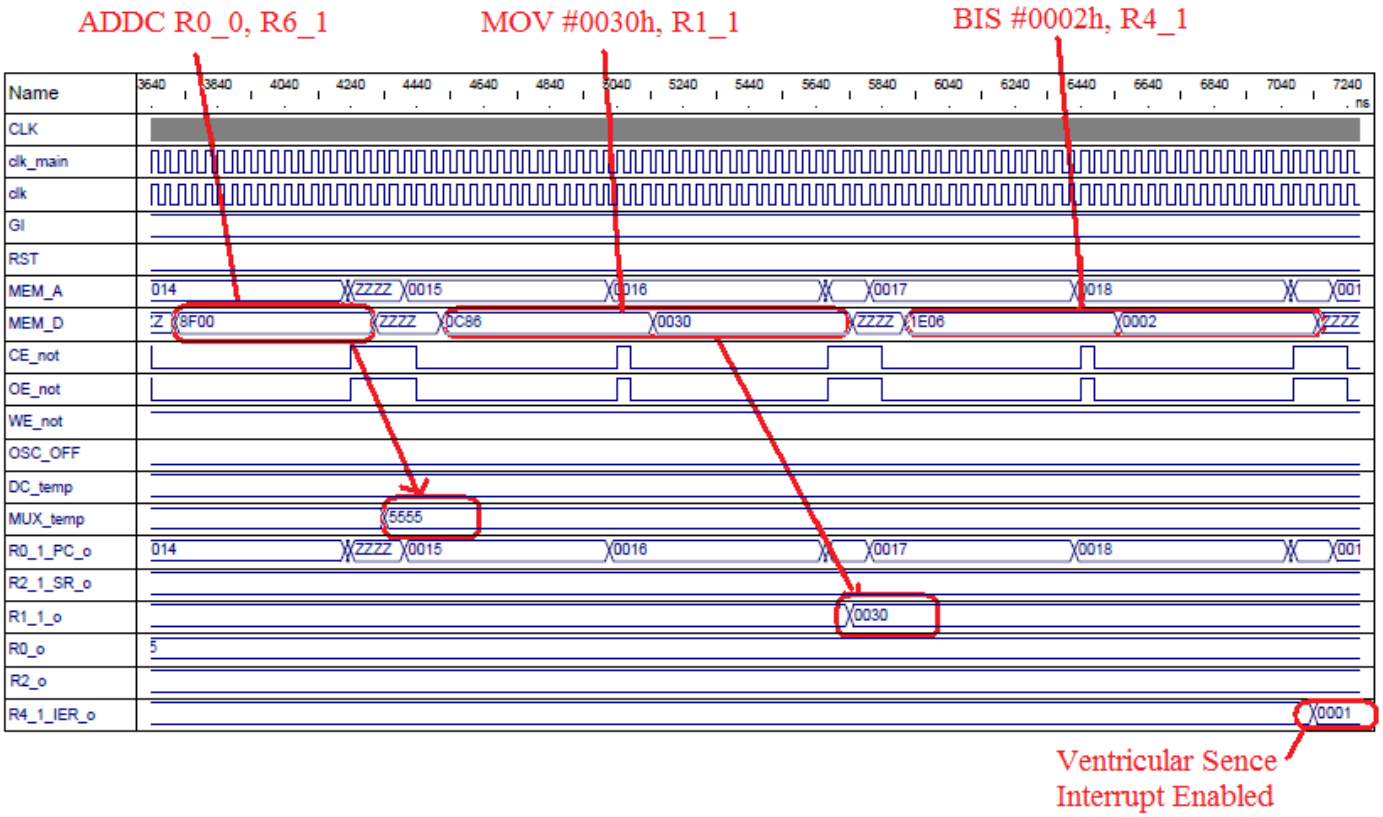


Figure 6–2: Testbench Part 2

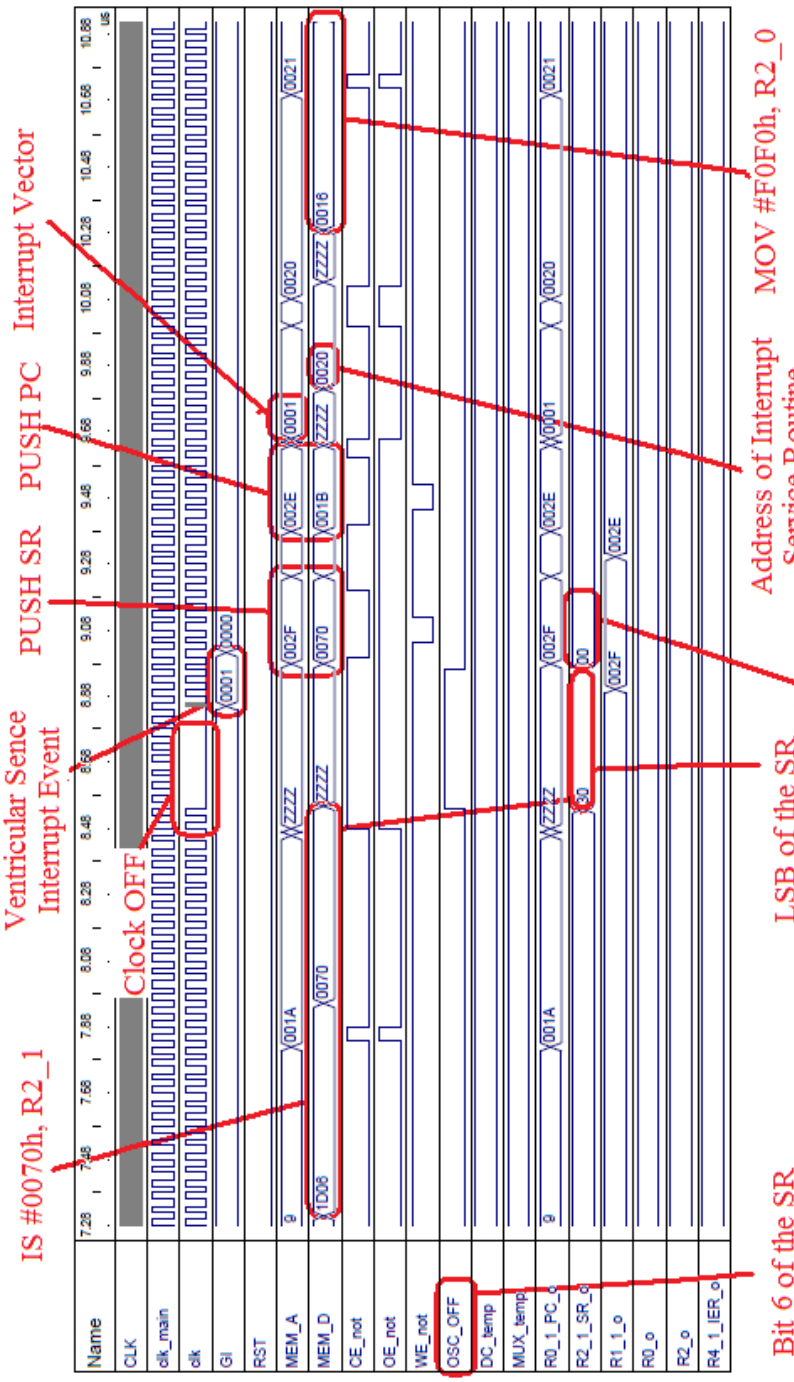


Figure 6–3: Testbench Part 3

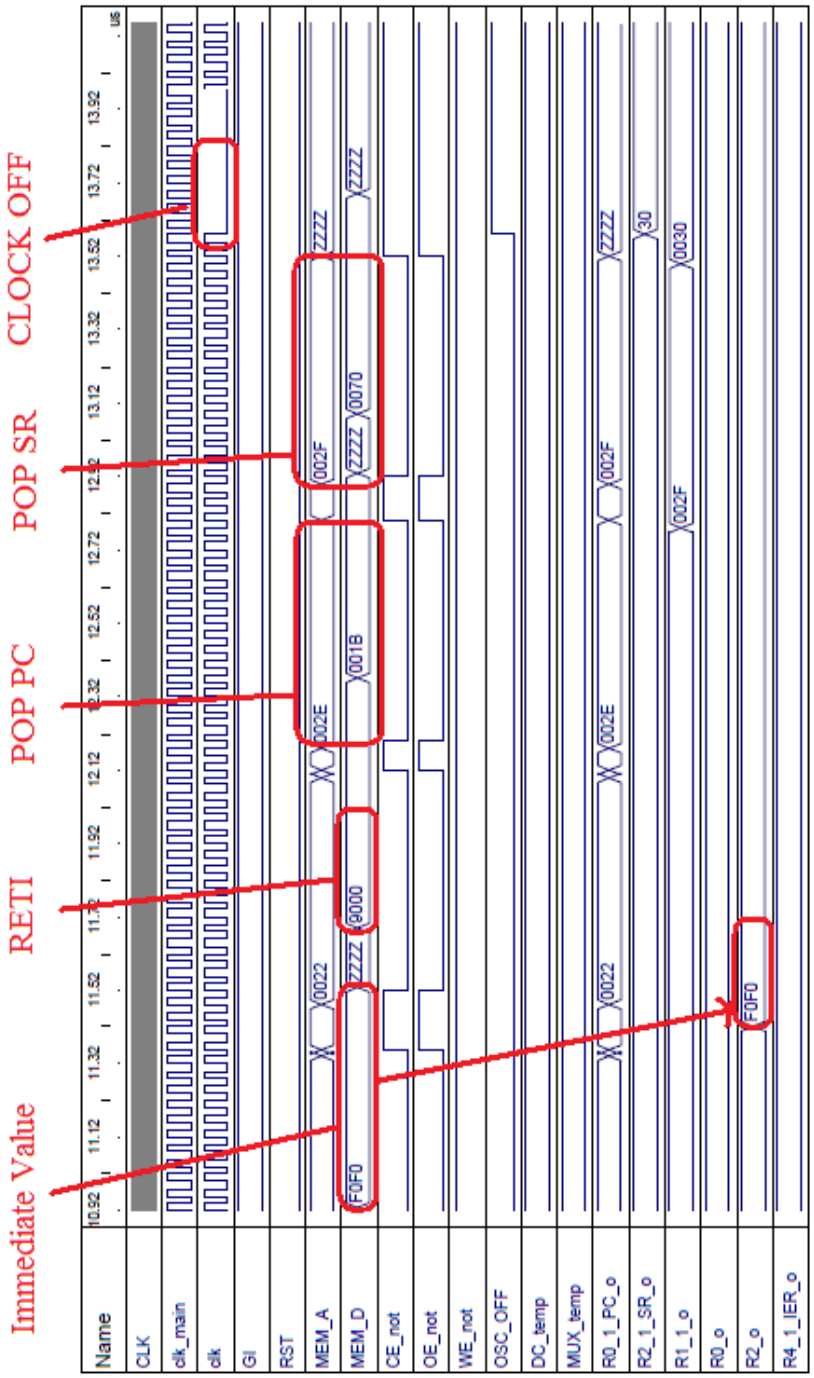


Figure 6–4: Testbench Part 4

6.2 FPGA Test

To ensure the optimum behavior after synthesis, the design was burned into an FPGA. In this process some of the CPU instructions were tested. Figure 6–5 shows the results of the MCR, MDR and OSC_OFF testing. The immediate and the register addressing modes as well as the CPU sleep mode were tested at the same time. Table 6–2 has the list of instructions that were used in this test.

Table 6–2: Board Test Instructions

Address	Instruction	Machine Code
0000	Reset Vector	0010
.		
.		
.		
0010	MOV #6699h, R5_1	0E86
0011		6699
0100	BIS #5555h, R0_0	1006
0101		5555
0110	ADDC R0_0, R6_1	0F00
0111	BIS #0070h, R2_1	1D06
1000		0070

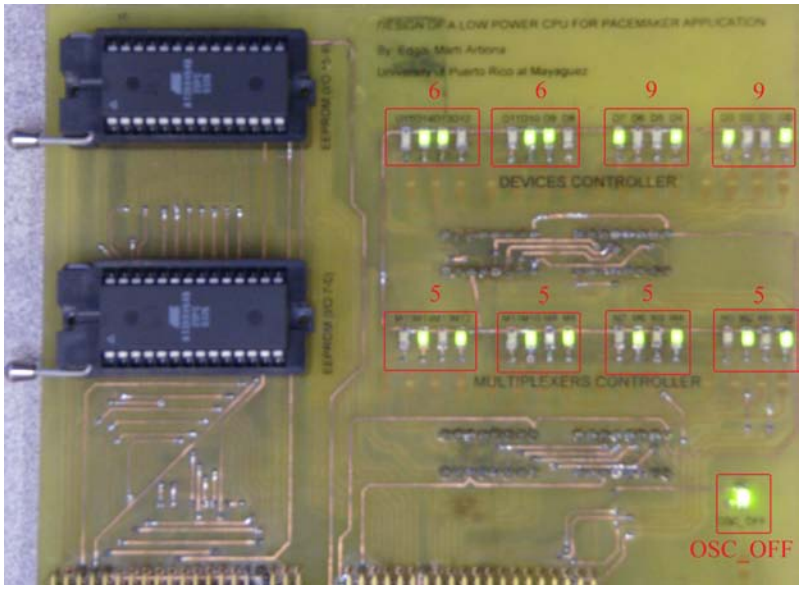


Figure 6–5: Test Board

This design was tested in a Xilinx Virtex II FPGA. The CPU and its functionality were verified in this FPGA.

6.3 Layout Design

The next step in the design process was the layout generation. This process starts with the design synthesis using RTL Compiler. This schematic was saved as a verilog file to be used as input to the Encounter Place & Route tool.

The layout is then generated using the Encounter tool. Here the verilog file is loaded with the standard cells library [25]. Figure 6–6 shows the final layout of the CPU without pads, that is, the effective die area. The design is to be used in a die with other components. The die dimensions are 1.9 by 1.7 mm. It has 2769 cells. Figure 6–7 shows the final layout. Pads are necessary since it will be tested.

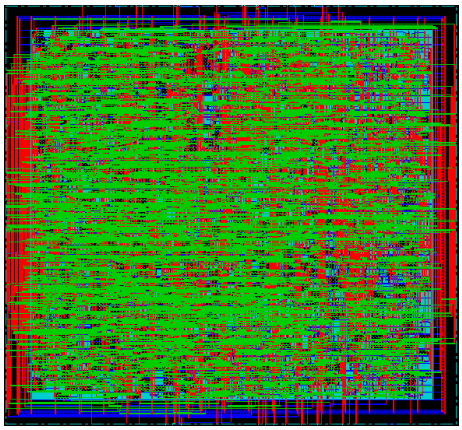


Figure 6–6: Design Layout without Pads

From this step, the final design obtained which has a low quantity of instance gates and a small dimension. Therefore, the size can be decreased since these cells are in technology of 0.6 μm and smaller technologies are available.

6.4 Power Estimation

Finally, an initial power estimation of the design was done. Figure 6–8 presents the Cadence Encounter output file of the power analysis. It indicates that the leakage power of the design is 0.40253 μW at 5V and the average power is about 1.8871 e+02 mW.

This result was obtained with an open source library form Oklahoma State University since tere are no low voltage and low power open source libraries. The

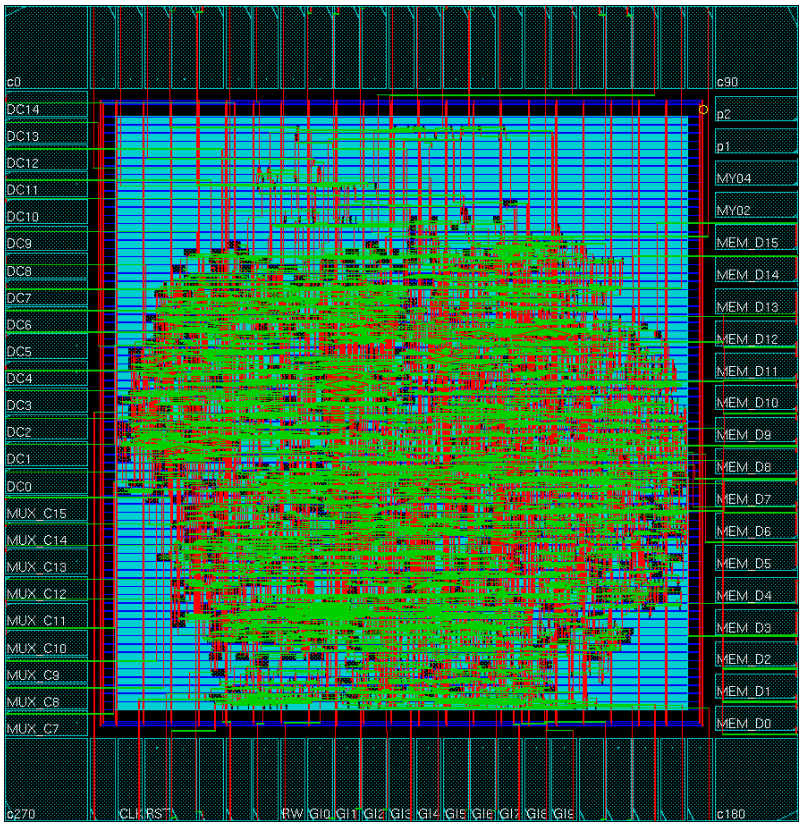


Figure 6–7: Design Layout with Pads

problem is that these cells were designed in ami06 thechnology which works at 5V. This made the cell consume high power. In order to decrease this power disipation, a specialized low voltage and low power standard cells library must be developed.

Another thing is that this estimation is assuming that the CPU is running at 100 MHz all the time, which is not true for our application since the CPU will spend most of the time in sleep mode, dissipating only static power. This are good news since the static power presented is low.

Next chapter presents an estimation of power for this design with the 2.8V and the needed frequency for the application.

```
################################################
# The Power Analysis Report for vdd net      #
################################################
power supply: 5 V
average power(default): 1.8871e+02 mW
    average switching power(default): 8.5408e+01 mW
    average internal power(default): 1.0330e+02 mW
    average leakage power(default): 4.0253e-04 mW
    average user specified power(default): 0.0000e+00 mW
average power by clock domain category:
    unclock domain(0.2) : 1.8871e+02 mW
average power by cell category:
    core:  1.8871e+02 mW
    block: 0.0000e+00 mW
    io:    0.0000e+00 mW
average power(considered in rail analysis): 1.8871e+02 mW
worst IR drop average analysis: 1.6224e-01 V
  number of nodes in rail network: 4147 nodes
worst EM:
    "M1"  4.3515e+00 mA/u
    "M2"  4.3515e+00 mA/u
    "M3"  0.0000e+00 mA/u
    "V12" 2.4658e+00 mA/cut
    "V23" 0.0000e+00 mA/cut
biggest toggled net: C[12]
    no. of terminal: 26
    total cap: 1.6135e+03 ff
```

Figure 6–8: Power Estimation Results

CHAPTER 7

Power Analysis

An elementary power estimation was done in order to get a better understanding of the advantages of the improvements on this device. This information confirmed the significance of this design.

7.1 Pacemaker Power Consumption

The dual-chamber pacemaker battery life was estimated and compared with this new design. To do this estimation, the parameters given by Furman were used [6]. These parameters are the operation voltage of the pacemaker (2.8V), the lead impedance (500Ω), the pulse duration (0.5ms), the electronic static current (5μA) and a heart rate of (60 bpm). The following calculations compare the energy used by the electronic circuitry and the total energy of the pacemaker. This computation was done assuming a pacing of 100%.

$$\begin{aligned} P_{pulse} &= \frac{V_{dd}^2}{R} \\ &= \frac{2.8^2}{500\Omega} \\ &= 15.68\mu W/pulse \end{aligned} \tag{7.1}$$

$$\begin{aligned} U_{pulse} &= P_{pulse} \times PulseDuration \\ &= 15.68\mu W/pulse \times 0.5ms \\ &= 7.84\mu J/pulse \end{aligned} \tag{7.2}$$

To get the energy per minute the pulse energy must be multiplied by 60 s, since

one pulse is delivered every second.

$$\begin{aligned} U_{pulse} &= U_{pulse} \times Secondsperminute \\ &= 7.84\mu J/min \times 60 \\ &= 470.4\mu J/min \end{aligned} \tag{7.3}$$

Now, using the remaining information, the energy consumed by the electronics can be calculated.

$$\begin{aligned} P_{elec} &= V_{dd} \times I_{elec} \\ &= 2.8V \times 5\mu A \\ &= 14\mu W \end{aligned} \tag{7.4}$$

$$\begin{aligned} U_{elec} &= P_{elec} \cdot 60s \\ &= 14\mu W \cdot 60s \\ &= 840\mu J/min \end{aligned} \tag{7.5}$$

From this the total energy of the pacemaker can be calculated.

$$\begin{aligned} U_{Total} &\approx U_{pulse} + U_{elect} \\ &\approx 470.4\mu J/min + 840\mu J/min \\ &\approx 1.31mJ/min \end{aligned} \tag{7.6}$$

The percent of the total energy that can be affected by this research can now be calculated.

$$Percent\_of\_impact = \frac{840\mu J}{1.31mJ} \times 100 = 64\% \tag{7.7}$$

Now the consumption of the new design will be calculated. For this, some assumptions about the hardware reduction in the CPU were done. The ALU functions

used to support instructions was reduced from around 16 to 8. Also, the instruction set was reduced from 27 to 11 instructions. In terms of the electronic circuitry, an improvement of about 45% is achieved, since the presented alternative eliminates around a 45% of the stimulation and sensing hardware. Therefore, an estimated 45% of power reduction on the electronics was achieved, and the new power consumption is:

$$\begin{aligned} P_{elec_{new}} &= 14\mu W \cdot (1 - 0.45) \\ &= 7.7\mu W \end{aligned} \tag{7.8}$$

Table 7–1: Calculations of the new Electronic Circuitry

Calculus	Old	New
Power	$15.68\mu W$	$7.7\mu W$
Energy/min	$7.847.7\mu J$	$462\mu J$
Energy Total	1.31mJ	$932\mu J$

From this information, the approximated improvement of our design can be calculated to be:

$$Improvement = \frac{1310 - 932}{1310} \times 100 = 29\% \tag{7.9}$$

This gives and estimated improvement of 29% of the total energy consumption with this design. Now if a battery of Lithium-Iodine of 2.8V and a capacity of 1 Ah is assumed, the battery life for each one of the cases can be calculated.

$$U_{battery} = charge(Ah) \cdot V_{dd} \cdot 3600 \tag{7.10}$$

$$BatteryLife = \frac{U_{battery}}{U_{current/new} \cdot 525,600} \tag{7.11}$$

The battery life for a current dual-chamber pacemaker is:

$$BatteryLife_{current} = \frac{1Ah \cdot 2.8V \cdot 3600}{1.31mW \cdot 525,600} = 14.6yr \tag{7.12}$$

The battery life for the new design is:

$$BatteryLife_{new} = \frac{1Ah \cdot 2.8V \cdot 3600}{932\mu W \cdot 525,600} = 20.6yr \tag{7.13}$$

This represents an estimated improvement in the battery life of 6 years when using this new design. Considering that this is a rough estimation we can asume that improvement has benn over estimated but even considering a 50% error the improvement still significant.

7.2 Frequency Calculation

The system frequency was calculated by assuming that the instruction with more cycles is repeated 50 times in each interrupt service routine. This gives a safe margin needed for a complete routine. The RETI is the slower instruction. It takes 37 clock cycles and it has two memory access. Finally, the slower cycle time is 21 ms [22]. Using the information provided the system frequency is calculated as follows:

$$\begin{aligned} frequency &= \frac{1}{\frac{CycleTime}{CPUCycles}} \\ &= \frac{1}{\frac{21ms}{37 \cdot 50}} \\ &= 88kHz \end{aligned} \tag{7.14}$$

Now giving it an extra 10% to ensure proper operation, we have:

$$frequency = 88kHz \cdot 1.1 \approx 100kHz \tag{7.15}$$

7.3 CPU Power Estimation

Power estimation is an important factor in order to have an idea about the dissipation of the design. For this design, broad power estimation will be done. In order to do the estimation, the formula to be used is the following:

$$P_{av} = \left(\sum_{i=1}^{\#ofgates} \alpha_i \cdot C_i \right) \cdot V_{dd}^2 \cdot f \tag{7.16}$$

To simplify the equation for this application it was assumed that the voltage at each gate is equal to V_{dd}. Also, it is assumed that the capacitances are uniform in all the circuit. Finally, the transition factor α will be approximated to the pacemaker duty cycle. These approximations are very broad and are not taking into account all the factors involved. However, for our application they are enough. The assumed values and the resulting equation are the following [27]:

1. Vdd = 2.8V
2. f = $100kHz$
3. C_{load} = 1pF
4. $\alpha \approx$ Duty Cycle

$$P_{av} = \alpha C_{load} V_{dd}^2 f(\#ofgates) \tag{7.17}$$

The power calculation is:

$$\begin{aligned} P_{av} &= \alpha \cdot 1pF \cdot 2.8^2 \cdot 100kHz \cdot 2769 \\ &= \alpha \cdot 2.17mW \end{aligned} \tag{7.18}$$

Now given that the duty cycle of the pacemaker will depend on the patient physiology a table that shows some duty cycles is presented(Table 7–2). This table also includes the power estimated for each one of the duty cycle values.

Table 7–2: Power Estimation with Diferent Duty Cycles

Duty Cycle	Power
1%	21.7μW
10%	217μW
20%	434μW
50%	1.1μW
100%	2.17mW

7.4 Power Results Comparation

Comparing the calculated results with the simulated provided in Figure 6–8 and referring to equation 7.17 we can conclude the following:

Changing the supply voltage from 5V to 2.8V we have:

$$P = 189mW\frac{2.8^2V}{5^2V} = 59mW \tag{7.19}$$

Now changing the frequency of 100 MHz to 100 kHz is obtained:

$$P = 189mW\frac{100kHz}{100MHz} = 59\mu W \tag{7.20}$$

This shows that the power consumption to be expected is aligned to Table 7–2.

CHAPTER 8

Conclusion and Future Work

8.1 Conclusion

The custom architecture of a CPU has been designed with the objective of optimizing dual-chamber pacemakers control logic and its hardware components. The design took into consideration the instruction frequency in programs and assigned the number of 1's per instruction in such a way that the more frequent instructions have less 1's in the machine instruction. Another step done during this research was the reduction of the CPU instuctions, eliminating ther non used by the application. This decreases the size and power consumed by the CPU.

A fully functional soft core of a CPU has been presented here. It is specialized for dual-chamber pacemakers. This CPU size and power was designed by taking into account the instructions usage and the hardware needed to operate the pacemaker. The performance of a pacemaker is also increased by this CPU, since it provides the ability of controlling internally some peripherals.

The simulated results show that the power consumed by this design is 1.8871e+2 mW. This number is unrealistic for this application since a standard cell library of 5V and a frequency of 100 MHz was used. However this provides a practical probe of concepts of the logic and gives us also a better estimated for the power estimation. As shown in the previous chapter, this power is decreased drastically by using a voltage of 2.8V and a frequency of 100 kHz.

8.2 Contribution of this Work

The contributions of this work are:

1. A custom CPU architecture specialized for dual chamber pacemaker application.
2. Two separated registers sets in a CPU (General Purpose and Dedicated Functions).
3. A special register to control and enable peripherals.
4. A special register to control some analog multiplexers to borrow the input and output stage for both chambers in a dual chamber pacemaker.
5. An open source soft core for this CPU.

8.3 Future Work

The presented design is a customized architecture that was simplified to have the less number of hardware to be used as control unit in a pacemaker. In order to achieve a low power design, many tasks need to be completed.

It is important is to design a specialized standard cells library for low voltage and low power. This will give this design a lower power consumption. It is also important to develop again the new layout with this library. In this way the design can be tested and the physical power consumption can be measured.

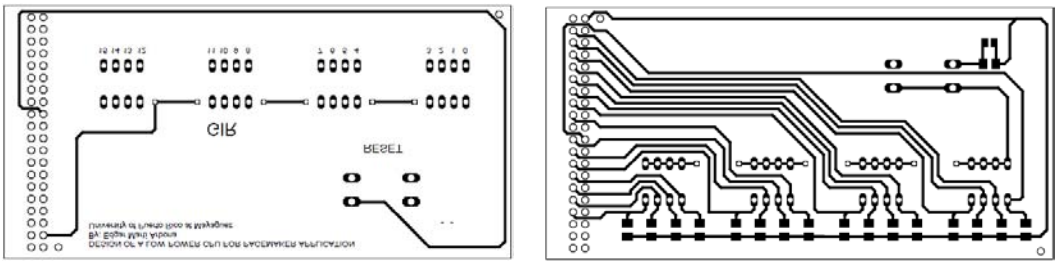
There are other key tasks that need to be treated in order to improve this design and the pacemaker technology in general. These tasks are the optimization of the other peripherals in the pacemaker. Then, assemble them to complete the design of a dual-chamber pacemaker by using this design as control unit. In this way it can be verified that the elimination of one input and one output stage truly reduces the power consumption significantly.

Appendices

APPENDIX A

FPGA Test Boards

The full functionality of the design was tested in Xilinx Spartan3 Starter Kit Board. Two printed circuit boars (PCB) were built in order to give the CPU the ability to be programmed and to achieve all inputs and outputs it needs. The smaller board is the input one, where 16 switches for the interrupts and one push button for reset are placed. Figure A–1(a) presents the top layer board and Figure A–1(b) the bottom of the layout for the input board. The switches are four groups of four bits through hole switches and one throug hole push-button.

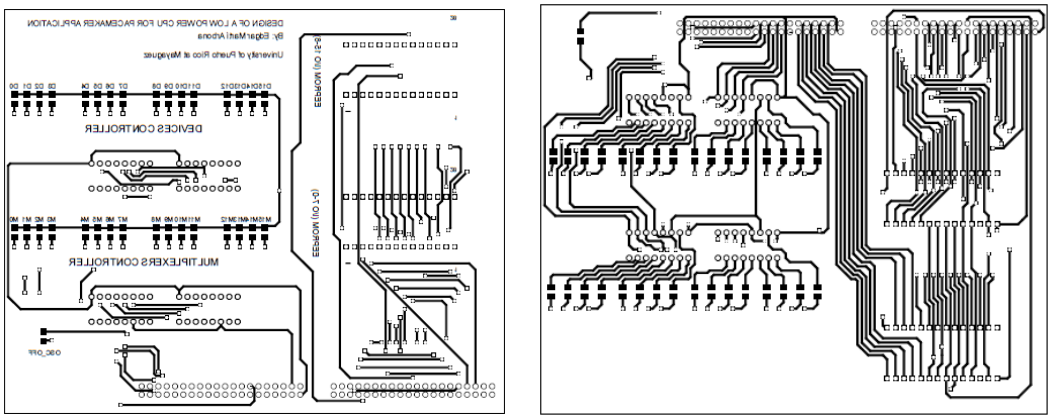


(a) Top Layer (b) Bottom Layer

Figure A–1: Input Board

The output board has the EEPROMS and 33 leds. They are divided as follows: 16 are for MCR, 16 for DCR and 1 for the OSC_OFF. Figure A–2 presents the top and bottom layouts of this stage.

A list of all the devices used in the test setup is presented in Table A–1. The part model of the more specific devices and their quantities are provided in this table. This information is important since parts of the board are designed for specific devices.



(a) Top Layer (b) Bottom Layer

Figure A–2: Output Board

Table A–1: List of Parts

No.	Part Number	Descripton	Quantity
1	XC3S200	Spartan-3 Starter Board	1
2	AT28BV64B	8k X 8 EEPROM	2
3	PGF138PC	1 of 8 Demultiplexer	4
4	SML-LX126GC-TR1	LEDS	33
5	ERJ-8ENF1000V	100 ohm Resistors	49
6	28-6554-10	28 PIN ZIF SOCKET	2
7		4-bits Binary Switch	4
8		Push Buttom	1

Figure A–3 shows a block diagram of the test board parts and their routing. In this diagram, the main part is the FPGA where the CPU is programed. The reset push button and the interrupts switches are at the top. In the bottom are the components of the other board, the EEPROM, the OSC_OFF led, and the thirty two leds for the MCR and DCR. The SRAM and the clock are is also shown but they are part of the spartan3 board.

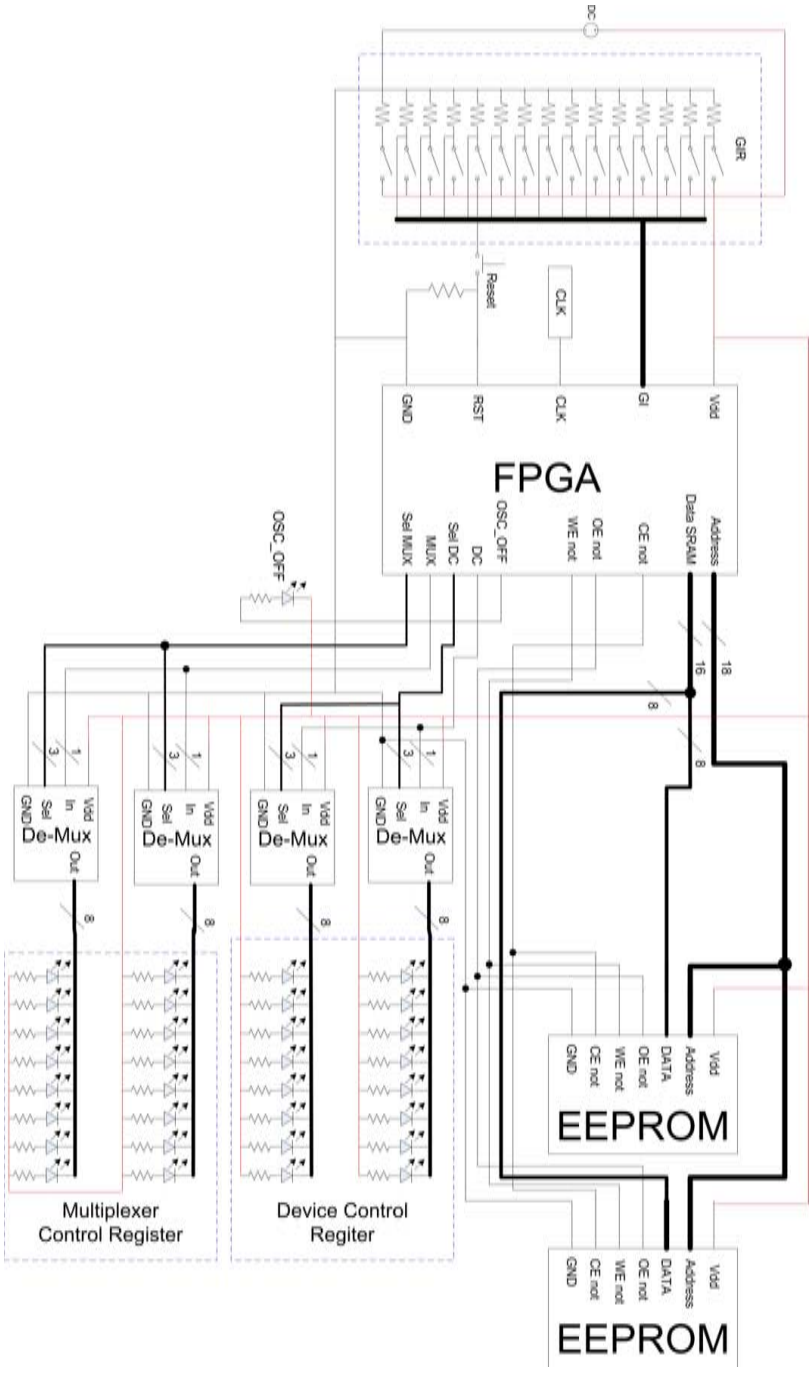


Figure A–3: Board Block Diagram

APPENDIX B

Soft Core

B.1 Interface Board

B.1.1 Interface

```
1  ----------------------------------------------------------------------------------
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:     02:03:16 03/07/2009
6  -- Design Name:
7  -- Module Name:     Interface - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 ----------------------------------------------------------------------------------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 ---- Uncomment the following library declaration if instantiating
26 ---- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity Interface is
31     Port ( GI : in  STD_LOGIC_VECTOR (14 downto 0);
32            CLK : in  STD_LOGIC;
33            RST : in  STD_LOGIC;
34            DC : out  STD_LOGIC_VECTOR (1 downto 0);
35            SEL_DC : out  STD_LOGIC_VECTOR (2 downto 0);
36            MUX : out  STD_LOGIC_VECTOR (1 downto 0);
37            SEL_MUX : out  STD_LOGIC_VECTOR (2 downto 0);
38            MEM_A : out  STD_LOGIC_VECTOR (17 downto 0);
```

```
39              MEM_D : inout  STD_LOGIC_VECTOR (15 downto 0);
40                              MEM_D_EP : inout  STD_LOGIC_VECTOR (7 downto 0);
41              CE_not_EEPROM : out  STD_LOGIC;
42                              CE_not_SRAM : out  STD_LOGIC;
43              OE_not : out  STD_LOGIC;
44              WE_not : out  STD_LOGIC;
45                              UB : out  STD_LOGIC;
46              LB : out  STD_LOGIC;
47                              OSC_OFF : out  STD_LOGIC);
48  end Interface;
49
50  architecture Behavioral of Interface is
51
52          COMPONENT CPU IS
53                  port(
54                              CLK : in STD_LOGIC;
55                              RST : in STD_LOGIC;
56                              GI : in STD_LOGIC_VECTOR(14 downto 0);
57                              DC : out STD_LOGIC_vector(15 downto 0);
58                              OSC_OFF : out STD_LOGIC;
59                              MEM_A : out STD_LOGIC_VECTOR(15 downto 0);
60                              MEM_D : inout STD_LOGIC_VECTOR(15 downto 0);
61                              MUX_C : out STD_LOGIC_VECTOR(15 downto 0);
62                              OE_not : out std_logic;
63                              WE_not : out std_logic;
64                              CE_not : out std_logic;
65                              RW : out std_logic
66                      );
67          END COMPONENT;
68
69          component clk_divider is
70                  Port ( clk : in  STD_LOGIC;
71             reset : in  STD_LOGIC;
72             output : out  STD_LOGIC);
73          end component;
74
75          component main_clk is
76                  Port ( clk : in  STD_LOGIC;
77             reset : in  STD_LOGIC;
78             output : out  STD_LOGIC);
79          end component;
80
81          component Board is
82                  Port ( clk : in  STD_LOGIC;
83             dc_in : in  STD_LOGIC_VECTOR (15 downto 0);
84                              mux_in : in  STD_LOGIC_VECTOR (15 downto 0);
85             reset : in  STD_LOGIC;
86             mux_c : out  STD_LOGIC_VECTOR (1 downto 0);
87             dc : out  STD_LOGIC_VECTOR (1 downto 0);
88                              sel_mux : out std_logic_vector (2 downto 0);
89                              sel_dc : out std_logic_vector (2 downto 0));
90          end component;
91
92          component arreglo is
93      Port ( MEM_D : inout  STD_LOGIC_VECTOR (15 downto 0);
94             MEM_D_EP : inout  STD_LOGIC_VECTOR (7 downto 0);
```

```
95              RW : in  STD_LOGIC;
96              CE_not_EE : in  STD_LOGIC;
97              CE_not_SR : in  STD_LOGIC;
98                              MEM_D_out : inout  STD_LOGIC_VECTOR (15 downto 0)
99              );
100         end component;
101
102         component DEBOUNCE is
103     Port(
104                             Clk         : IN STD_LOGIC;
105                             Key         : IN STD_LOGIC;  -- active low input
106                             pulse       : OUT STD_LOGIC);
107         end component;
108
109          component DEBOUNCE_15 is
110     Port ( GIR : in  STD_LOGIC_VECTOR (15 downto 1);
111            GIR_O : out  STD_LOGIC_VECTOR (15 downto 1);
112            CLK : in  STD_LOGIC);
113         end component;
114
115         signal osc_off_temp, clock, CE_not, CE_not_EEPROM_temp, CE_not_SRAM_temp, RW, clk_main,
                reset : std_logic;
116         signal DC_temp, MUX_temp, MEM_A_temp : std_logic_vector(15 downto 0);
117         signal mem_d_temp : std_logic_vector(15 downto 0);
118         signal GIR : std_logic_vector(14 downto 0);
119
120 begin
121
122         U1 : CPU PORT MAP (clk_main, reset, GIR, DC_temp, OSC_OFF_temp, MEM_A_temp, mem_d_temp,
                MUX_temp, OE_not, WE_not, CE_not, RW);
123
124         U2 : Board port map (clock, DC_temp, MUX_temp, reset, MUX, DC, sel_mux, sel_dc);
125
126         U3 : clk_divider port map (clk, reset, clock);
127
128         U4 : main_clk port map (clk, reset, clk_main);
129
130         U5 : arreglo port map(MEM_D, MEM_D_EP, RW, CE_not_EEPROM_temp, CE_not_SRAM_temp,
                mem_d_temp);
131
132         U6 : DEBOUNCE port map (clk, rst, reset);
133
134         U7 : DEBOUNCE_15 port map (GI, GIR, clk_main);
135
136 ------------------MEMORY ADDRESS----------------------
137
138         MEM_A <= '0' & '0' & MEM_A_temp;
139
140 ---------------------- Chip Enable-----------------------
141
142         CE_not_EEPROM_temp <= CE_not when MEM_A_temp(15 downto  13) = "00000" else '1';
143         CE_not_SRAM_temp   <= CE_not when MEM_A_temp(15 downto 13) /= "00000" else '1';
144
145         CE_not_EEPROM <= CE_not_EEPROM_temp;
146         CE_not_SRAM   <= CE_not_SRAM_temp;
147 ------------------- U byte/L byte----------------------
```

```
148
149         UB <= '0';
150         LB <= '0';
151
152         osc_off <= not reset when osc_off_temp = '1' else '0';
153
154 end Behavioral;
```

B.1.2 Memory Data Board

```
1  ----------------------------------------------------------------------------------
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6
7  entity arreglo is
8      Port ( MEM_D : inout  STD_LOGIC_VECTOR (15 downto 0);
9             MEM_D_EP : inout  STD_LOGIC_VECTOR (7 downto 0);
10            RW : in  STD_LOGIC;
11            CE_not_EE : in  STD_LOGIC;
12            CE_not_SR : in  STD_LOGIC;
13                        MEM_D_out : inout  STD_LOGIC_VECTOR (15 downto 0)
14            );
15 end arreglo;
16
17 architecture Behavioral of arreglo is
18
19         component bidirectional_buffer is
20                 Port (Input : in  STD_LOGIC_VECTOR (7 downto 0);
21                                 Output : inout  STD_LOGIC_VECTOR (7 downto 0);
22                                 En : in  STD_LOGIC
23                                 );
24         end component;
25
26         signal  ee1, ee2, sram1, sram2, meml1, meml2 : std_logic;
27
28 begin
29
30         ee1 <= '0' when CE_not_EE= '0' and RW = '1' else '1';
31         ee2 <= '0' when CE_not_EE = '0' and RW = '0' else '1';
32         sram1 <= '0' when CE_not_SR = '0' and RW = '1' else '1';
33         sram2 <= '0' when CE_not_SR = '0' and RW = '0' else '1';
34         meml1 <= '0' when (CE_not_EE = '0' and RW = '1') or (CE_not_SR = '0' and RW = '1') else
               '1';
35         meml2 <= '0' when (CE_not_EE = '0' and RW = '0') or (CE_not_SR = '0' and RW = '0') else
               '1';
36
37     U1 : bidirectional_buffer port map (MEM_D_EP, MEM_D_out(15 downto 8), ee1);  -- EEPROM
38
39         U2 : bidirectional_buffer port map (MEM_D_out(15 downto 8), MEM_D_EP, ee2);  -- EEPROM
40
41         U3 : bidirectional_buffer port map (MEM_D(15 downto 8), MEM_D_out(15 downto 8), sram1);
               -- SRAM
42
```

```
43          U4 : bidirectional_buffer port map (MEM_D_out(15 downto 8), MEM_D(15 downto 8), sram2);
                 -- SRAM
44
45          U5 : bidirectional_buffer port map (MEM_D(7 downto 0), MEM_D_out(7 downto 0), meml1);
46
47          U6 : bidirectional_buffer port map (MEM_D_out(7 downto 0), MEM_D(7 downto 0), meml2);
48
49  end Behavioral;
```

B.1.3 Bidirectional Buffer Tristate Data

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:     13:37:41 03/10/2009
6   -- Design Name:
7   -- Module Name:     bidirectional_tristate buffer - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25  ---- Uncomment the following library declaration if instantiating
26  ---- any Xilinx primitives in this code.
27  --library UNISIM;
28  --use UNISIM.VComponents.all;
29
30  entity bidirectional_buffer is
31      Port ( Input : in  STD_LOGIC_VECTOR (7 downto 0);
32                     Output : Out  STD_LOGIC_VECTOR (7 downto 0);
33             En : in  STD_LOGIC
34                    );
35  end
36  bidirectional_buffer;
37
38  architecture Behavioral of bidirectional_buffer is
39
40  begin
41          --Output <= Input;
42  --      process(en)
43  --      begin
44  --
45  --              if en = '1' then
```

```
46  --
47  ----                        Input <= "ZZZZZZZZ";
48  --                          Output <= "ZZZZZZZZ";
49  --
50  --                  else
51  --
52  ----                        Input <= Output;
53  --                          Output <= Input;
54  --
55  --                  end if;
56  --
57  --          end process;
58
59            --Input <= Output when En = '0' else "ZZZZZZZZ";
60            Output <= Input when En = '0' else "ZZZZZZZZ";
61
62  end Behavioral;
```

B.1.4 Board Demultiplexers

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:     12:33:43 03/03/2009
6   -- Design Name:
7   -- Module Name:     Board - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25  ---- Uncomment the following library declaration if instantiating
26  ---- any Xilinx primitives in this code.
27  --library UNISIM;
28  --use UNISIM.VComponents.all;
29
30  entity Board is
31            Port ( clk : in  STD_LOGIC;
32              dc_in : in  STD_LOGIC_VECTOR (15 downto 0);
33                                  mux_in : in  STD_LOGIC_VECTOR (15 downto 0);
34              reset : in  STD_LOGIC;
35              mux_c : out  STD_LOGIC_VECTOR (1 downto 0);
36              dc : out  STD_LOGIC_VECTOR (1 downto 0);
```

```
37                          sel_mux : out std_logic_vector (2 downto 0);
38                          sel_dc : out std_logic_vector (2 downto 0));
39
40  end Board;
41
42  architecture Behavioral of Board is
43          signal counter : std_logic_vector(2 downto 0);
44
45  begin
46
47          mux_c(0) <= not mux_in(0) when counter = "000" else
48                                       not mux_in(1) when counter = "001" else
49                                       not mux_in(2) when counter = "010" else
50                                       not mux_in(3) when counter = "011" else
51                                       not mux_in(4) when counter = "100" else
52                                       not mux_in(5) when counter = "101" else
53                                       not mux_in(6) when counter = "110" else
54                                       not mux_in(7) when counter = "111";
55
56
57          mux_c(1) <= not mux_in(8) when counter = "000" else
58                                       not mux_in(9) when counter = "001" else
59                                       not mux_in(10) when counter = "010" else
60                                       not mux_in(11) when counter = "011" else
61                                       not mux_in(12) when counter = "100" else
62                                       not mux_in(13) when counter = "101" else
63                                       not mux_in(14) when counter = "110" else
64                                       not mux_in(15) when counter = "111";
65
66
67          dc(0) <= not dc_in(0) when counter = "000" else
68                                  not dc_in(1) when counter = "001" else
69                                  not dc_in(2) when counter = "010" else
70                                  not dc_in(3) when counter = "011" else
71                                  not dc_in(4) when counter = "100" else
72                                  not dc_in(5) when counter = "101" else
73                                  not dc_in(6) when counter = "110" else
74                                  not dc_in(7) when counter = "111";
75
76
77          dc(1) <= not dc_in(8) when counter = "000" else
78                                  not dc_in(9) when counter = "001" else
79                                  not dc_in(10) when counter = "010" else
80                                  not dc_in(11) when counter = "011" else
81                                  not dc_in(12) when counter = "100" else
82                                  not dc_in(13) when counter = "101" else
83                                  not dc_in(14) when counter = "110" else
84                                  not dc_in(15) when counter = "111";
85
86
87
88          process(clk, reset)
89          begin
90                          if (reset = '1') then
91                                  counter <= "000";
92                          elsif (clk='1' and clk'event) then
```

```
93                                          counter <= counter + 1;
94                          end if;
95
96          end process;
97
98          sel_mux <= counter;
99          sel_dc <= counter;
100
101 end Behavioral;
```

B.1.5 Clock Divider for Board Demultiplexers

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:     12:53:22 03/03/2009
6   -- Design Name:
7   -- Module Name:     clk_divider - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25  ---- Uncomment the following library declaration if instantiating
26  ---- any Xilinx primitives in this code.
27  --library UNISIM;
28  --use UNISIM.VComponents.all;
29
30  entity clk_divider is
31      generic(max_count : integer := 59000);
32           Port ( clk : in  STD_LOGIC;
33             reset : in  STD_LOGIC;
34             output : out  STD_LOGIC);
35  end clk_divider;
36
37  architecture Behavioral of clk_divider is
38
39  begin
40
41          Process_1: process(clk,reset)
42                          variable cont: integer range 0 to max_count;
43                                          begin
44
```

```
45                          if reset = '1' then
46                                  cont := 0;
47                                  output <= '0';
48                          else
49                                  if clk='1' and clk'event then
50
51                                                  cont := cont + 1;
52                                                  output <= '0';
53                                                  if cont = max_count then
54                                                          cont := 0;
55                                                          output <= '1';
56                                                  else
57                                                          cont := cont;
58                                                  end if;
59
60                                  end if;
61                                  end if;
62
63                  end process;
64
65  end Behavioral;
```

B.2 CPU

B.2.1 CPU Mainfile

```
1  --
       ----------------------------------------------------------------------------------------------------

2  --
3  -- File         : CPU_MAIN_FILE.vhd
4  --
5  --
       ----------------------------------------------------------------------------------------------------

6  --
7  -- Description : This file combine all the internal
8  --
9  --
       ----------------------------------------------------------------------------------------------------


10
11  library IEEE;
12  use IEEE.STD_LOGIC_1164.all;
13
14  entity CPU is
15          port(
16                  CLK : in STD_LOGIC;
17                  RST : in STD_LOGIC;
18                  GI : in STD_LOGIC_VECTOR(14 downto 0);
19                  DC : out STD_LOGIC_vector(15 downto 0);
20                  OSC_OFF : out STD_LOGIC;
21                  MEM_A : out STD_LOGIC_VECTOR(15 downto 0);
22                  MEM_D : inout STD_LOGIC_VECTOR(15 downto 0);
```

```
23                  MUX_C : out STD_LOGIC_VECTOR(15 downto 0);
24                  OE_not : out std_logic;
25                  WE_not : out std_logic;
26                  CE_not : out std_logic;
27                  RW : out std_logic
28              );
29  end CPU;
30
31  architecture CPU_1 of CPU is
32
33  ------------------General Register-----------------
34
35          COMPONENT gen_register IS
36                  port(
37                          Rin         : in STD_LOGIC_VECTOR(15 downto 0);
38                          Save        : in STD_LOGIC;
39                          Rout        : out STD_LOGIC_VECTOR(15 downto 0)
40                      );
41          END COMPONENT;
42
43  ------------------Control Unit-----------------
44
45          COMPONENT CU IS
46                  port(
47                          IR        : in STD_LOGIC_VECTOR(15 downto 0);
48                          MA        : in STD_LOGIC;
49                          clk_in    : in STD_LOGIC;
50                          rst       : in STD_LOGIC;
51                          Sr        : in std_logic_vector(5 downto 0);
52                          GIR       : in std_logic_vector(15 downto 0);
53                          IER       : in std_logic_vector(15 downto 1);
54                          save_ir   : out std_logic;
55                          RW      : out std_logic;
56                          ME            : out std_logic;
57                          save      : out std_logic;
58                          en_out_a  : out std_logic;
59                          en_out_b  : out std_logic;
60                          save_sr   : out std_logic;
61                          en_SR     : out std_logic;
62                          rst_sr    : out std_logic;
63                          pc_out    : out std_logic;
64                          reset_i   : out std_logic;
65                          save_mar  : out std_logic;
66                          en_out_mar      : out std_logic;
67                          sel       : out std_logic;
68                          sel_in_mdr      : out std_logic;
69                          sel_out_mdr : out std_logic_vector(1 downto 0);
70                          save_mdr  : out std_logic;
71                          save_c    : out std_logic;
72                  S         : out std_logic_vector(3 downto 0);
73                          Bus_A     : out std_logic_vector(15 downto 0);
74                          A_reg     : out std_logic_vector(4 downto 0);
75                          B_C_reg   : out std_logic_vector(4 downto 0)
76                      );
77          END COMPONENT;
78
```

```
79  ------------------Register File-------------------
80
81          COMPONENT reg_file IS
82                  port(
83                          Rin         : in STD_LOGIC_VECTOR(15 downto 0);
84                          R2_1_SR_i   : in STD_LOGIC_VECTOR(3 downto 0);
85                          R3_1_GIR_i  : in STD_LOGIC_VECTOR(15 downto 0);
86                          en_out_a    : in STD_LOGIC;
87                          en_out_b    : in STD_LOGIC;
88                          S_in        : in STD_LOGIC_VECTOR(4 downto 0);
89                          S_out_a     : in STD_LOGIC_VECTOR(4 downto 0);
90                          S_out_b     : in STD_LOGIC_VECTOR(4 downto 0);
91                          Save        : in STD_LOGIC;
92                          Save_SR     : in STD_LOGIC;
93                          en_SR       : in STD_LOGIC;
94                          clk         : in STD_LOGIC;
95                          rst         : in STD_LOGIC;
96                          rst_sr      : in STD_LOGIC;
97                          pc_out      : in STD_LOGIC;
98                          reset_i     : in STD_LOGIC;
99                          OSC_OFF     : out std_logic;
100                         Rout_a      : out STD_LOGIC_VECTOR(15 downto 0);
101                         Rout_b      : out STD_LOGIC_VECTOR(15 downto 0);
102                         R0_1_PC_o   : out STD_LOGIC_VECTOR(15 downto 0);
103                         R2_1_SR_o   : out STD_LOGIC_VECTOR(5 downto 0);
104                         R3_1_GIR_o  : out STD_LOGIC_VECTOR(15 downto 0);
105                         R4_1_IER_o  : out STD_LOGIC_VECTOR(15 downto 1);
106                         R5_1_DCR_o  : out STD_LOGIC_VECTOR(15 downto 0);
107                         R6_1_MCR_o  : out STD_LOGIC_VECTOR(15 downto 0)
108                     );
109         END COMPONENT;
110
111 ------------------Memory Address Register-----------------
112
113         COMPONENT mar IS
114                 port(
115                         Rin         : in STD_LOGIC_VECTOR(15 downto 0);
116                         Rin_A       : in STD_LOGIC_VECTOR(15 downto 0);
117                         Save        : in STD_LOGIC;
118                         Sel                 : in STD_LOGIC;
119                         en_out      : in STD_LOGIC;
120                         Rout        : out STD_LOGIC_VECTOR(15 downto 0)
121                     );
122         END COMPONENT;
123
124 ------------------Memory Data Register-----------------
125
126         COMPONENT mdr IS
127                 port(
128                         Rin         : in STD_LOGIC_VECTOR(15 downto 0);
129                         Save        : in STD_LOGIC;
130                         Sel_out     : in STD_LOGIC_VECTOR(1 downto 0);
131                         Sel_in      : in STD_LOGIC;
132                         Rout_a      : out STD_LOGIC_VECTOR(15 downto 0);
133                         Rout_b      : out STD_LOGIC_VECTOR(15 downto 0);
134                         Mem_in_out  : inout STD_LOGIC_VECTOR(15 downto 0)
```

```
135                         );
136         END COMPONENT;
137
138 ------------------Arithmetic Logic Unit------------------
139
140         COMPONENT alu IS
141                 port(
142                         Cin : in STD_LOGIC;
143                         A : in STD_LOGIC_VECTOR(15 downto 0);
144                         B : in STD_LOGIC_VECTOR(15 downto 0);
145                         S : in STD_LOGIC_VECTOR(3 downto 0);
146                         F : out STD_LOGIC_VECTOR(15 downto 0);
147                         Co: out STD_LOGIC;
148                         N : out STD_LOGIC;
149                         V : out STD_LOGIC;
150                         Z : out STD_LOGIC
151                         );
152         END COMPONENT;
153
154
155 ------------------Arithmetic Logic Unit------------------
156
157         COMPONENT r_w IS
158                 port(
159                          clk     : in STD_LOGIC;
160                         RW      : in STD_LOGIC;
161                         ME      : in STD_LOGIC;
162                          rst     : in std_logic;
163                         CE_not          : out STD_LOGIC;
164                         OE_not          : out STD_LOGIC;
165                         WE_not          : out STD_LOGIC;
166                         MA      : out std_logic
167                         );
168         END COMPONENT;
169
170 ------------------Internal Signals------------------
171         signal flags, S : std_logic_vector(3 downto 0);
172         signal A, B, C, GIR, IR, temp_c : std_logic_vector(15 downto 0);
173         signal sr :std_logic_vector(5 downto 0);
174         signal save_ir, save, en_out_a, en_out_b, save_sr, en_sr, rst_sr, pc_out, reset_i,
                save_mar, en_out_mar, sel, sel_in_mdr, save_mdr, save_c, MA, RW_temp, ME : std_logic
                ;
175         signal sel_out_mdr : std_logic_vector(1 downto 0);
176         signal A_reg, B_C_reg : std_logic_vector(4 downto 0);
177         signal GI_temp :std_logic_vector(15 downto 0);
178         signal IER :std_logic_vector(15 downto 1);
179
180 begin
181
182 ------------------General Register------------------
183
184         U1 : gen_register PORT MAP ( MEM_D, save_ir, IR);
185
186 ------------------Control Unit------------------
187
```

```
188         U2 : CU PORT MAP(IR,MA, clk, rst, sr, GIR, IER(15 downto 1), save_ir, RW_temp, ME, save,
                 en_out_a, en_out_b, save_sr, en_sr, rst_sr, pc_out, reset_i, save_mar, en_out_mar,
                 sel, sel_in_mdr, sel_out_mdr, save_mdr, save_c, S, A, A_reg, B_C_reg);
189
190 ------------------Register File-------------------
191
192     U3 : reg_file PORT MAP (C, flags, GI_temp, en_out_a, en_out_b,B_C_reg, A_reg, B_C_reg, save,
              save_sr, en_sr, clk, rst, rst_sr, pc_out, reset_i, OSC_OFF, A, B, MEM_A, sr, GIR, IER,
             DC, MUX_C);
193
194 ------------------Memory Address Register-------------------
195
196         U4 : mar PORT MAP (C, A, save_mar, sel, en_out_mar, MEM_A);
197
198 ------------------Memory Data Register-------------------
199
200         U5 : mdr PORT MAP (C, save_mdr, sel_out_mdr, sel_in_mdr, A, B, MEM_D);
201
202 ------------------Arithmetic Logic Unit-------------------
203
204         U6 : alu PORT MAP (sr(0), A, B, S, temp_c, flags(0), flags(2), flags(3), flags(1));
205
206 ----------------------Bus C Register-----------------------
207
208         U7 : gen_register PORT MAP ( temp_c, save_c, C);
209
210 ---------------Bus C Register----------------------
211
212         U8 : r_w PORT MAP ( clk, RW_temp, ME, rst, CE_not, OE_not, WE_not, MA);
213
214 --------------------GI----------------------------
215
216         Gi_temp <= GI & '0';
217
218 -------------------RW----------------------------
219
220         RW <= RW_temp;
221
222
223 end CPU_1;
```

B.2.2 ALU

```
1  --
       ------------------------------------------------------------------------------------------------
2  --
3  -- File          : alu.vhd
4  --
5  --
       ------------------------------------------------------------------------------------------------
6  --
7  -- Description :                                        A(16 b) B(16 b)
8  --                                  --|--   --|--
9  --                                                                    \       \/       /
```

```
10 --                                                                    S(4 b) -\          /- flags(Co, N,
       Z, V)
11 --                                                                        Cin-\______/
12 --                                                                                    |
13 --                                                                                 F(16 b)
14 --    Aritmetic Logic Unit
15 --
       ------------------------------------------------------------------------------------------------------


16
17 library IEEE;
18 use IEEE.STD_LOGIC_1164.all;
19 use IEEE.STD_LOGIC_ARITH.ALL;
20 use IEEE.STD_LOGIC_UNSIGNED.ALL;
21
22 entity alu is
23         port(
24                 Cin : in STD_LOGIC;
25                 A : in STD_LOGIC_VECTOR(15 downto 0);
26                 B : in STD_LOGIC_VECTOR(15 downto 0);
27                 S : in STD_LOGIC_VECTOR(3 downto 0);
28                 F : out STD_LOGIC_VECTOR(15 downto 0);
29                 Co: out STD_LOGIC;
30                 N : out STD_LOGIC;
31                 V : out STD_LOGIC;
32                 Z : out STD_LOGIC
33                 );
34 end alu;
35
36 architecture a_alu of alu is
37
38         signal carry_in, CARRY_IN_1, v1, v2, v3, inc, sub, add, sub_co, cout, M0, M1, C_and_xor,
               Ctemp, xor_1, Z_temp, C_add_sub, V_f : std_logic;
39         signal B41, B42, I, subtract, bypass, and_not, sum, or_out, and_out, xor_out, M00, M11,
               fout : STD_LOGIC_VECTOR(15 downto 0);
40
41 begin
42
43 ------------------------------------------------------------------------
44 --                              A anb B buses control
45 ------------------------------------------------------------------------
46
47         I <= "0000000000000001";
48         sub <= '1' when S = "0110" or S = "1001" or S = "1010" else '0';
49         sub_co <= '1' when S = "0110" or S = "1001" or S = "1010" else '0';
50         subtract <= "1111111111111111" when sub = '1' else "0000000000000000";
51         bypass <= "0000000000000000" when S = "0000" else "1111111111111111";
52         and_not <= "1111111111111111" when S = "1000" else "0000000000000000";
53         xor_1 <= '1' when S = "0011" else '0';
54         B41 <= I when S = "0001" or S = "1001" else B;
55         B42 <= B41 xor subtract;
56         inc <= '1' when S = "001" else '0';
57         add <= '1' when S = "0101" else '0';
58         CARRY_IN_1 <= '0' when S = "1010" or S = "0001" or S = "1100" or S = "1001" else Cin;
59         carry_in <= (CARRY_IN_1 and not inc) xor sub ;
60
```

```
61  ------------------------------------------------------------------------------
62  --                                    ADDER
63  ------------------------------------------------------------------------------
64          PROCESS (A, B42, carry_in)
65                  VARIABLE carry : STD_LOGIC_VECTOR (16 DOWNTO 0);
66          BEGIN
67                   carry(0) := carry_in;
68                  FOR i IN 0 TO 15 LOOP
69                          sum(i) <= A(i) XOR B42(i) XOR carry(i);
70                          carry(i+1) := (A(i) AND B42(i)) OR (a(i) AND carry(i)) OR (B42(i) AND
                                carry(i));
71                  END LOOP;
72                   cout <= carry(16);
73
74          END PROCESS;
75
76  ------------------------------------------------------------------------------
77  --                                    OR
78  ------------------------------------------------------------------------------
79
80          or_out <= A or (B and bypass);
81
82  ------------------------------------------------------------------------------
83  --                                    XOR
84  ------------------------------------------------------------------------------
85
86          xor_out <= A xor B;
87
88  ------------------------------------------------------------------------------
89  --                                    AND
90  ------------------------------------------------------------------------------
91
92          and_out <= (A xor and_not) and B;
93
94  ------------------------------------------------------------------------------
95  --                                  16 b MUX 4-1
96  ------------------------------------------------------------------------------
97
98          M1 <= (not S(3) and S(0)) or (S(2) and S(1)) or (not S(2) and S(0)) or (S(3) and S(1) and
                  not S(0)) or (S(3) and S(2) and not S(0));
99          M0 <= S(2) or S(3) or (not S(1) and S(0));
100
101         M00 <= "1111111111111111" when M0 = '1' else "0000000000000000";
102         M11 <= "1111111111111111" when M1 = '1' else "0000000000000000";
103
104         fout <= (or_out AND NOT M11 AND NOT M00) OR
105                         (and_out AND NOT M11 AND M00) OR
106                         (xor_out AND M11 AND NOT M00) OR
107                         (sum AND M11 AND M00);
108         F <= fout;
109
110 ------------------------------------------------------------------------------
111 --                              Flags (Co, N, P, V)
112 ------------------------------------------------------------------------------
113
114 ---------------------------------- N - negative ----------------------------------
```

```
115
116         N <= '1' when  fout(15) = '1' else '0';
117
118 ---------------------------------- Z - zero ---------------------------------------
119
120         Z_temp <= '1' when  fout = "0000000000000000" else '0';
121         Z <= Z_temp;
122
123 ---------------------------------- C - carry --------------------------------------
124
125         Ctemp <= cout xor sub_co;
126         C_and_xor <= (not Z_temp) when S = "0011" or S = "0100" else '0';
127         C_add_sub <= Ctemp when S = "0001" or S = "0101" or S = "0110" or S = "1001" OR S ="1010"
                OR S = "1100" else '0';
128         Co <= C_add_sub or C_and_xor;
129
130 ---------------------------------- V - overflow -----------------------------------
131
132         v1 <= sub and (((not A(15)) and B(15) and  fout(15)) or (A(15) and (not B(15)) and (not
                fout(15))));
133         v2 <= add and (((not A(15)) and (not B(15)) and  fout(15)) or (A(15) and  B(15) and (not
                fout(15))));
134         v3 <= xor_1 and (A(15) and B(15));
135         V_f <= v1 or v2 or v3;
136         V <= '0' when  S = "0100" else V_f;
137
138 ------------------------------------------------------------------------------------
139 end a_alu;
```

B.2.3 Register File

```
1  --------------------------------------------------------------------------------------------------------
2  --
3  -- Title      : Register File
4  --
5  --------------------------------------------------------------------------------------------------------
6  --
       -------------------------
7  -- Description :                                  |     Register_File      |
8  --                                                                        |
                                        |
9  --                                                                  ---|Rin    (16b)     Rout_a(16
       b)|---
10 --                                                                  ---|R2_1      (4b)   Rout_b(16
       b)|---
11 --                                                                  ---|R3_1      (16b)   R0_1(16
       b)|--- PC
12 --                                                                  ---|en_out_a (1b)    R2_1(7b)
       |--- SR
13 --                                                                  ---|en_out_b (1b)   R3_1(16b)
       |--- GIR
14 --                           ---|Sel_in   (4b)   R4_1(16b)|--- IER
```

```
15 --                                                                    ---|Sel_out_a(4b)    R5_1(16b)
       |--- DCR
16 --                                                                    ---|Sel_out_b(4b)    R6_1(16b)
       |--- MCR
17 --                                                                    ---|>Save     (1b) OSC_OFF(1b)
       |---OSC_OFF
18 --                                                                    ---|>Save_SR (1b)
         |
19 --                          ---|en_SR     (1b)            |
20 --                                                                    ---|> CLK     (1b)
                     |
21 --                                                                    ---|RST       (1b)
                     |
22 --                                                                    ---|RST _SR   (1b)
                     |
23 --                                                                    ---|PC_out    (1b)
                     |
24 --                          ---|reset_i   (1b)            |
25 --                                                                       |
                                    |
26 --                                                                          |
                                       |
27 --
       ------------------------
28 --
29 --
       ------------------------------------------------------------------------------------------------------------
```

```
30
31 library IEEE;
32 use IEEE.STD_LOGIC_1164.all;
33 use IEEE.STD_LOGIC_ARITH.ALL;
34 use IEEE.STD_LOGIC_UNSIGNED.ALL;
35
36 entity reg_file is
37         port(
38                 Rin         : in STD_LOGIC_VECTOR(15 downto 0);
39                 R2_1_SR_i   : in STD_LOGIC_VECTOR(3 downto 0);
40                 R3_1_GIR_i  : in STD_LOGIC_VECTOR(15 downto 0);
41                 en_out_a    : in STD_LOGIC;
42                 en_out_b    : in STD_LOGIC;
43                 S_in        : in STD_LOGIC_VECTOR(4 downto 0);
44                 S_out_a     : in STD_LOGIC_VECTOR(4 downto 0);
45                 S_out_b     : in STD_LOGIC_VECTOR(4 downto 0);
46                 Save        : in STD_LOGIC;
47                 Save_SR     : in STD_LOGIC;
48                 en_SR       : in STD_LOGIC;
49                 clk         : in STD_LOGIC;
50                 rst         : in STD_LOGIC;
51                 rst_sr      : in STD_LOGIC;
52                 pc_out      : in STD_LOGIC;
53                 reset_i     : in STD_LOGIC;
54                 OSC_OFF     : out std_logic;
55                 Rout_a      : out STD_LOGIC_VECTOR(15 downto 0);
56                 Rout_b      : out STD_LOGIC_VECTOR(15 downto 0);
57                 R0_1_PC_o   : out STD_LOGIC_VECTOR(15 downto 0);
```

```
58                  R2_1_SR_o  : out STD_LOGIC_VECTOR(5 downto 0);
59                  R3_1_GIR_o : out STD_LOGIC_VECTOR(15 downto 0);
60                  R4_1_IER_o : out STD_LOGIC_VECTOR(15 downto 1);
61                  R5_1_DCR_o : out STD_LOGIC_VECTOR(15 downto 0);
62                  R6_1_MCR_o : out STD_LOGIC_VECTOR(15 downto 0)
63                  );
64  end reg_file;
65
66
67
68  architecture reg_file of reg_file is
69
70          signal save_0, save_1, save_2, save_3, save_4, save_5, save_6, save_7, save_8, save_9,
                save_10, save_11, save_12, save_13, save_14, save_15 : std_logic;
71          signal save_0_1, save_1_1, save_2_1, save_2a_1, save_3_1, save_3a_1, save_3b_1, save_4_1,
                 save_5_1, save_6_1 : std_logic;
72          signal Rout_out_a, Rout_out_b, Rin2a, Rin2b, Rin3a, Rin3b, R0_o, R1_o, R2_o, R3_o, R4_o,
                R5_o, R6_o, R7_o, R8_o, R9_o, R10_o, R11_o, R12_o, R13_o, R14_o, R15_o :
                STD_LOGIC_VECTOR(15 downto 0);
73          signal R0_1_o : STD_LOGIC_VECTOR(15 downto 0); -- PC
74          signal R1_1_o : STD_LOGIC_VECTOR(15 downto 0); -- SP
75          signal R2_1_o : STD_LOGIC_VECTOR(15 downto 0); -- SR
76          signal R3_1_o : STD_LOGIC_VECTOR(15 downto 0); -- GIR
77          signal R4_1_o : STD_LOGIC_VECTOR(15 downto 0); -- IER
78          signal R5_1_o : STD_LOGIC_VECTOR(15 downto 0); -- DCR
79          signal R6_1_o : STD_LOGIC_VECTOR(15 downto 0); -- DCR
80          signal int : STD_LOGIC_VECTOR(15 downto 0); -- Interrupt detector
81          signal and_1, and_2, and_3,     and_4, and_5, and_6, and_7,     and_8, and_9, and_10,
                and_11, and_12, and_13, and_14 : STD_LOGIC;
82  begin
83
84
85  ----------------------------------------------------------------------------------
86  --                              save MUX
87  ----------------------------------------------------------------------------------
88
89          save_0 <= Save WHEN S_in="00000" ELSE '0';
90          save_1 <= Save WHEN S_in="00001" ELSE '0';
91          save_2 <= Save WHEN S_in="00010" ELSE '0';
92          save_3 <= Save WHEN S_in="00011" ELSE '0';
93          save_4 <= Save WHEN S_in="00100" ELSE '0';
94          save_5 <= Save WHEN S_in="00101" ELSE '0';
95          save_6 <= Save WHEN S_in="00110" ELSE '0';
96          save_7 <= Save WHEN S_in="00111" ELSE '0';
97          save_8 <= Save WHEN S_in="01000" ELSE '0';
98          save_9 <= Save WHEN S_in="01001" ELSE '0';
99          save_10 <= Save WHEN S_in="01010" ELSE '0';
100         save_11 <= Save WHEN S_in="01011" ELSE '0';
101         save_12 <= Save WHEN S_in="01100" ELSE '0';
102         save_13 <= Save WHEN S_in="01101" ELSE '0';
103         save_14 <= Save WHEN S_in="01110" ELSE '0';
104         save_15 <= Save WHEN S_in="01111" ELSE '0';
105         save_0_1 <= Save WHEN S_in="10000" ELSE '0';
106         save_1_1 <= Save WHEN S_in="10001" ELSE '0';
107         save_2a_1 <= Save WHEN S_in="10010" ELSE '0';
108         save_3a_1 <= Save WHEN S_in="10011" ELSE '0';
```

```
109         save_4_1 <= Save WHEN S_in="10100" ELSE '0';
110         save_5_1 <= Save WHEN S_in="10101" ELSE '0';
111         save_6_1 <= Save WHEN S_in="10110" ELSE '0';
112
113
114         save_3b_1 <= '0' WHEN R3_1_GIR_i="0000000000000000" or S_in = "10011" ELSE clk;
115         save_3_1 <= save_3a_1 or save_3b_1;
116         Rin3a <= R3_1_GIR_i or R3_1_o;
117         Rin3b <= Rin WHEN R3_1_GIR_i="0000000000000000" or S_in = "10011" ELSE Rin3a;
118
119         save_2_1 <= save_2a_1 WHEN en_SR = '0' ELSE Save_SR;
120         Rin2a <= R2_1_o(15 downto 4) & R2_1_SR_i;
121         Rin2b <= Rin2a WHEN en_SR = '1' ELSE Rin;
122 ------------------------------------------------------------------------------
123 --                              Registers
124 ------------------------------------------------------------------------------
125
126         PROCESS (rst, save_0)
127         BEGIN
128                 IF      (rst = '1') THEN
129                         R0_o <= "0000000000000000";
130                 ELSIF (save_0'EVENT AND save_0='1') THEN
131                         R0_o <= Rin;
132                 END IF;
133         END PROCESS;
134
135         PROCESS (rst, save_1)
136         BEGIN
137                 IF      (rst = '1') THEN
138                         R1_o <= "0000000000000000";
139                 ELSIF   (save_1'EVENT AND save_1='1') THEN
140                         R1_o <= Rin;
141                 END IF;
142         END PROCESS;
143
144         PROCESS (rst, save_2)
145         BEGIN
146                 IF      (rst = '1') THEN
147                         R2_o <= "0000000000000000";
148                 ELSIF   (save_2'EVENT AND save_2='1') THEN
149                         R2_o <= Rin;
150                 END IF;
151         END PROCESS;
152
153         PROCESS (rst, save_3)
154         BEGIN
155                 IF      (rst = '1') THEN
156                         R3_o <= "0000000000000000";
157                 ELSIF   (save_3'EVENT AND save_3='1') THEN
158                         R3_o <= Rin;
159                 END IF;
160         END PROCESS;
161
162         PROCESS (rst, save_4)
163         BEGIN
164                 IF      (rst = '1') THEN
```

```
165                     R4_o <= "0000000000000000";
166             ELSIF   (save_4'EVENT AND save_4='1') THEN
167                     R4_o <= Rin;
168             END IF;
169     END PROCESS;
170
171     PROCESS (rst, save_5)
172     BEGIN
173             IF      (rst = '1') THEN
174                     R5_o <= "0000000000000000";
175             ELSIF   (save_5'EVENT AND save_5='1') THEN
176                     R5_o <= Rin;
177             END IF;
178     END PROCESS;
179
180     PROCESS (rst, save_6)
181     BEGIN
182             IF      (rst = '1') THEN
183                     R6_o <= "0000000000000000";
184             ELSIF   (save_6'EVENT AND save_6='1') THEN
185                     R6_o <= Rin;
186             END IF;
187     END PROCESS;
188
189     PROCESS (rst, save_7)
190     BEGIN
191             IF      (rst = '1') THEN
192                     R7_o <= "0000000000000000";
193             ELSIF   (save_7'EVENT AND save_7='1') THEN
194                     R7_o <= Rin;
195             END IF;
196     END PROCESS;
197
198     PROCESS (rst, save_8)
199     BEGIN
200             IF      (rst = '1') THEN
201                     R8_o <= "0000000000000000";
202             ELSIF   (save_8'EVENT AND save_8='1') THEN
203                     R8_o <= Rin;
204             END IF;
205     END PROCESS;
206
207     PROCESS (rst, save_9)
208     BEGIN
209             IF      (rst = '1') THEN
210                     R9_o <= "0000000000000000";
211             ELSIF   (save_9'EVENT AND save_9='1') THEN
212                     R9_o <= Rin;
213             END IF;
214     END PROCESS;
215
216     PROCESS (rst, save_10)
217     BEGIN
218             IF      (rst = '1') THEN
219                     R10_o <= "0000000000000000";
220             ELSIF   (save_10'EVENT AND save_10='1') THEN
```

```
221                         R10_o <= Rin;
222                 END IF;
223         END PROCESS;
224
225         PROCESS (rst, save_11)
226         BEGIN
227                 IF      (rst = '1') THEN
228                         R11_o <= "0000000000000000";
229                 ELSIF   (save_11'EVENT AND save_11='1') THEN
230                         R11_o <= Rin;
231                 END IF;
232         END PROCESS;
233
234         PROCESS (rst, save_12)
235         BEGIN
236                 IF      (rst = '1') THEN
237                         R12_o <= "0000000000000000";
238                 ELSIF   (save_12'EVENT AND save_12='1') THEN
239                         R12_o <= Rin;
240                 END IF;
241         END PROCESS;
242
243         PROCESS (rst, save_13)
244         BEGIN
245                 IF      (rst = '1') THEN
246                         R13_o <= "0000000000000000";
247                 ELSIF   (save_13'EVENT AND save_13='1') THEN
248                         R13_o <= Rin;
249                 END IF;
250         END PROCESS;
251
252         PROCESS (rst, save_14)
253         BEGIN
254                 IF      (rst = '1') THEN
255                         R14_o <= "0000000000000000";
256                 ELSIF   (save_14'EVENT AND save_14='1') THEN
257                         R14_o <= Rin;
258                 END IF;
259         END PROCESS;
260
261         PROCESS (rst, save_15)
262         BEGIN
263                 IF      (rst = '1') THEN
264                         R15_o <= "0000000000000000";
265                 ELSIF   (save_15'EVENT AND save_15='1') THEN
266                         R15_o <= Rin;
267                 END IF;
268         END PROCESS;
269
270         PROCESS (rst, save_0_1)
271         BEGIN
272                 IF      (rst = '1') THEN
273                         R0_1_o <= "0000000000000000";
274                 ELSIF   (save_0_1'EVENT AND save_0_1='1') THEN
275                         R0_1_o <= Rin;
276                 END IF;
```

```
277         END PROCESS;
278
279         PROCESS (rst, save_1_1)
280         BEGIN
281                 IF      (rst = '1') THEN
282                         R1_1_o <= "0000000000000000";
283                 ELSIF   (save_1_1'EVENT AND save_1_1='1') THEN
284                         R1_1_o <= Rin;
285                 END IF;
286         END PROCESS;
287
288         PROCESS (rst, rst_sr, save_2_1)
289         BEGIN
290                 IF      (rst = '1' or rst_sr = '1') THEN
291                         R2_1_o <= "0000000000000000";
292                 ELSIF   (save_2_1'EVENT AND save_2_1='1') THEN
293                         R2_1_o <= Rin2b;
294                 END IF;
295         END PROCESS;
296
297         PROCESS (rst, reset_i, save_3_1, int)
298         BEGIN
299                 IF      (rst = '1') THEN
300                         R3_1_o <= "0000000000000001";
301                 elsif(rst = '0' and reset_i = '1' and int(0) = '1')     then
302                         R3_1_o(0) <= '0';
303                 elsif(rst = '0' and reset_i = '1' and int(1) = '1')     then
304                         R3_1_o(1) <= '0';
305                 elsif(rst = '0' and reset_i = '1' and int(2) = '1')     then
306                         R3_1_o(2) <= '0';
307                 elsif(rst = '0' and reset_i = '1' and int(3) = '1')     then
308                         R3_1_o(3) <= '0';
309                 elsif(rst = '0' and reset_i = '1' and int(4) = '1')     then
310                         R3_1_o(4) <= '0';
311                 elsif(rst = '0' and reset_i = '1' and int(5) = '1')     then
312                         R3_1_o(5) <= '0';
313                 elsif(rst = '0' and reset_i = '1' and int(6) = '1')     then
314                         R3_1_o(6) <= '0';
315                 elsif(rst = '0' and reset_i = '1' and int(7) = '1')     then
316                         R3_1_o(7) <= '0';
317                 elsif(rst = '0' and reset_i = '1' and int(8) = '1')     then
318                         R3_1_o(8) <= '0';
319                 elsif(rst = '0' and reset_i = '1' and int(9) = '1')     then
320                         R3_1_o(9) <= '0';
321                 elsif(rst = '0' and reset_i = '1' and int(10) = '1') then
322                         R3_1_o(10) <= '0';
323                 elsif(rst = '0' and reset_i = '1' and int(11) = '1') then
324                         R3_1_o(11) <= '0';
325                 elsif(rst = '0' and reset_i = '1' and int(12) = '1') then
326                         R3_1_o(12) <= '0';
327                 elsif(rst = '0' and reset_i = '1' and int(13) = '1') then
328                         R3_1_o(13) <= '0';
329                 elsif(rst = '0' and reset_i = '1' and int(14) = '1') then
330                         R3_1_o(14) <= '0';
331                 elsif(rst = '0' and reset_i = '1' and int(15) = '1') then
332                         R3_1_o(15) <= '0';
```

```
333                 ELSIF   (save_3_1'EVENT AND save_3_1='1') THEN
334                         R3_1_o <= Rin3b;
335                 END IF;
336         END PROCESS;
337
338         PROCESS (rst, save_4_1)
339         BEGIN
340                 IF      (rst = '1') THEN
341                         R4_1_o <= "0000000000000000";
342                 ELSIF   (save_4_1'EVENT AND save_4_1='1') THEN
343                         R4_1_o <= Rin;
344                 END IF;
345         END PROCESS;
346
347         PROCESS (rst, save_5_1)
348         BEGIN
349                 IF      (rst = '1') THEN
350                         R5_1_o <= "0000000000000000";
351                 ELSIF   (save_5_1'EVENT AND save_5_1='1') THEN
352                         R5_1_o <= Rin;
353                 END IF;
354         END PROCESS;
355
356         PROCESS (rst, save_6_1)
357         BEGIN
358                 IF      (rst = '1') THEN
359                         R6_1_o <= "0000000000000000";
360                 ELSIF   (save_6_1'EVENT AND save_6_1='1') THEN
361                         R6_1_o <= Rin;
362                 END IF;
363
364         END PROCESS;
365
366
367         R0_1_PC_o <= R0_1_o WHEN pc_out = '1' ELSE "ZZZZZZZZZZZZZZZZ";
368         R2_1_SR_o <= R2_1_o(5 downto 0);
369         OSC_OFF <= R2_1_o(6);
370         R3_1_GIR_o <= R3_1_o;
371         R4_1_IER_o <= R4_1_o(15 downto 1);
372         R5_1_DCR_o <= R5_1_o;
373         R6_1_MCR_o <= R6_1_o; --R6_1_o;
374
375 ------------------------------------------------------------------------------
376 --                        Interrupt Detector
377 ------------------------------------------------------------------------------
378     int(0) <= R3_1_o(0) and '1';
379         int(1) <= R3_1_o(1) and R4_1_o(1) and (not int(0));
380         int(2) <= R3_1_o(2) and R4_1_o(2) and and_1;
381         int(3) <= R3_1_o(3) and R4_1_o(3) and and_2;
382         int(4) <= R3_1_o(4) and R4_1_o(4) and and_3;
383         int(5) <= R3_1_o(5) and R4_1_o(5) and and_4;
384         int(6) <= R3_1_o(6) and R4_1_o(6) and and_5;
385         int(7) <= R3_1_o(7) and R4_1_o(7) and and_6;
386         int(8) <= R3_1_o(8) and R4_1_o(8) and and_7;
387         int(9) <= R3_1_o(9) and R4_1_o(9) and and_8;
388         int(10) <= R3_1_o(10) and R4_1_o(10) and and_9;
```

```
389         int(11) <= R3_1_o(11) and R4_1_o(11) and and_10;
390         int(12) <= R3_1_o(12) and R4_1_o(12) and and_11;
391         int(13) <= R3_1_o(13) and R4_1_o(13) and and_12;
392         int(14) <= R3_1_o(14) and R4_1_o(14) and and_13;
393         int(15) <= R3_1_o(15) and R4_1_o(15) and and_14;
394
395         and_1 <= (not int(0)) and (not int(1));
396         and_2 <= and_1 and (not int(2));
397         and_3 <= and_2 and (not int(3));
398         and_4 <= and_3 and (not int(4));
399         and_5 <= and_4 and (not int(5));
400         and_6 <= and_5 and (not int(6));
401         and_7 <= and_6 and (not int(7));
402         and_8 <= and_7 and (not int(8));
403         and_9 <= and_8 and (not int(9));
404         and_10 <= and_9 and (not int(10));
405         and_11 <= and_10 and (not int(11));
406         and_12 <= and_11 and (not int(12));
407         and_13 <= and_12 and (not int(13));
408         and_14 <= and_13 and (not int(14));
409         --and_15 <= and_14 and (not int(15));
410
411 --      process(rst, reset_i)
412 --      begin
413 --              if(rst = '1') then
414 --                      R3_1_o <= "0000000000000001";
415 --              elsif(rst = '0' and reset_i = '1' and int(0) = '1')      then
416 --                      R3_1_o(0) <= '0';
417 --              elsif(rst = '0' and reset_i = '1' and int(1) = '1')      then
418 --                      R3_1_o(1) <= '0';
419 --              elsif(rst = '0' and reset_i = '1' and int(2) = '1')      then
420 --                      R3_1_o(2) <= '0';
421 --              elsif(rst = '0' and reset_i = '1' and int(3) = '1')      then
422 --                      R3_1_o(3) <= '0';
423 --              elsif(rst = '0' and reset_i = '1' and int(4) = '1')      then
424 --                      R3_1_o(4) <= '0';
425 --              elsif(rst = '0' and reset_i = '1' and int(5) = '1')      then
426 --                      R3_1_o(5) <= '0';
427 --              elsif(rst = '0' and reset_i = '1' and int(6) = '1')      then
428 --                      R3_1_o(6) <= '0';
429 --              elsif(rst = '0' and reset_i = '1' and int(7) = '1')      then
430 --                      R3_1_o(7) <= '0';
431 --              elsif(rst = '0' and reset_i = '1' and int(8) = '1')      then
432 --                      R3_1_o(8) <= '0';
433 --              elsif(rst = '0' and reset_i = '1' and int(9) = '1')      then
434 --                      R3_1_o(9) <= '0';
435 --              elsif(rst = '0' and reset_i = '1' and int(10) = '1') then
436 --                      R3_1_o(10) <= '0';
437 --              elsif(rst = '0' and reset_i = '1' and int(11) = '1') then
438 --                      R3_1_o(11) <= '0';
439 --              elsif(rst = '0' and reset_i = '1' and int(12) = '1') then
440 --                      R3_1_o(12) <= '0';
441 --              elsif(rst = '0' and reset_i = '1' and int(13) = '1') then
442 --                      R3_1_o(13) <= '0';
443 --              elsif(rst = '0' and reset_i = '1' and int(14) = '1') then
444 --                      R3_1_o(14) <= '0';
```

```
445 --                elsif(rst = '0' and reset_i = '1' and int(15) = '1') then
446 --                        R3_1_o(15) <= '0';
447 --                end if;
448 --
449 --        end process;
450 ------------------------------------------------------------------------------------
451 --                                MUX Out A
452 ------------------------------------------------------------------------------------
453
454           Rout_out_a <= R0_o WHEN S_out_a="00000" ELSE
455                         R1_o WHEN S_out_a="00001" ELSE
456                                 R2_o WHEN S_out_a="00010" ELSE
457                                                 R3_o WHEN S_out_a="00011" ELSE
458                                                 R4_o WHEN S_out_a="00100" ELSE
459                                                 R5_o WHEN S_out_a="00101" ELSE
460                                                 R6_o WHEN S_out_a="00110" ELSE
461                                                 R7_o WHEN S_out_a="00111" ELSE
462                                                 R8_o WHEN S_out_a="01000" ELSE
463                                                 R9_o WHEN S_out_a="01001" ELSE
464                                                 R10_o WHEN S_out_a="01010" ELSE
465                                                 R11_o WHEN S_out_a="01011" ELSE
466                                                 R12_o WHEN S_out_a="01100" ELSE
467                                                 R13_o WHEN S_out_a="01101" ELSE
468                                                 R14_o WHEN S_out_a="01110" ELSE
469                                                 R15_o WHEN S_out_a="01111" ELSE
470                                                 R0_1_o WHEN S_out_a="10000" ELSE
471                                                 R1_1_o WHEN S_out_a="10001" ELSE
472                                                 R2_1_o WHEN S_out_a="10010" ELSE
473                                                 R3_1_o WHEN S_out_a="10011" ELSE
474                                                 R4_1_o WHEN S_out_a="10100" ELSE
475                                                 R5_1_o WHEN S_out_a="10101" ELSE
476                                                 R6_1_o WHEN S_out_a="10110" ELSE
477                                                 "0000000000000000";
478
479           Rout_a <= Rout_out_a WHEN en_out_a = '1' ELSE "ZZZZZZZZZZZZZZZZ";
480
481 ------------------------------------------------------------------------------------
482 --                                MUX Out B
483 ------------------------------------------------------------------------------------
484
485           Rout_out_b <= R0_o WHEN S_out_b="00000" ELSE
486                                                 R1_o WHEN S_out_b="00001" ELSE
487                                                 R2_o WHEN S_out_b="00010" ELSE
488                                                 R3_o WHEN S_out_b="00011" ELSE
489                                                 R4_o WHEN S_out_b="00100" ELSE
490                                                 R5_o WHEN S_out_b="00101" ELSE
491                                                 R6_o WHEN S_out_b="00110" ELSE
492                                                 R7_o WHEN S_out_b="00111" ELSE
493                                                 R8_o WHEN S_out_b="01000" ELSE
494                                                 R9_o WHEN S_out_b="01001" ELSE
495                                                 R10_o WHEN S_out_b="01010" ELSE
496                                                 R11_o WHEN S_out_b="01011" ELSE
497                                                 R12_o WHEN S_out_b="01100" ELSE
498                                                 R13_o WHEN S_out_b="01101" ELSE
499                                                 R14_o WHEN S_out_b="01110" ELSE
500                                                 R15_o WHEN S_out_b="01111" ELSE
```

```
501                                             R0_1_o WHEN S_out_b="10000" ELSE
502                                             R1_1_o WHEN S_out_b="10001" ELSE
503                                             R2_1_o WHEN S_out_b="10010" ELSE
504                                             R3_1_o WHEN S_out_b="10011" ELSE
505                                             R4_1_o WHEN S_out_b="10100" ELSE
506                                             R5_1_o WHEN S_out_b="10101" ELSE
507                                             R6_1_o WHEN S_out_b="10110" ELSE
508                                             "0000000000000000";
509
510         Rout_b <= Rout_out_b WHEN en_out_b = '1' ELSE "ZZZZZZZZZZZZZZZZ";
511
512 ------------------------------------------------------------------------------
513 end reg_file;
```

B.2.4 Memory Access Signals

```
1  --
       ------------------------------------------------------------------------------------------------
2  --
3  -- File         : write.vhd
4  --
5  --
       ------------------------------------------------------------------------------------------------
6  --
7  -- Description : This file control the comunication with the memory
8  --
9  --
       ------------------------------------------------------------------------------------------------
10
11 library IEEE;
12 use IEEE.STD_LOGIC_1164.all;
13
14 entity r_w is
15          port(
16                  clk     : in STD_LOGIC;
17                 RW      : in STD_LOGIC;
18                  ME      : in STD_LOGIC;
19                  rst     : in std_logic;
20                  CE_not          : out STD_LOGIC;
21                  OE_not          : out STD_LOGIC;
22                  WE_not          : out STD_LOGIC;
23                  MA      : out std_logic
24              );
25 end r_w;
26
27 --}} End of automatically maintained section
28
29 architecture r_w of r_w is
30         signal state : std_logic_vector(3 downto 0);
31         signal write_1 : std_logic;
32 begin
33
34         write_1 <= '1' when (RW = '0' and ME = '1') or (ME = '0' and state = "0111") else '0';
```

```
35
36
37          Process (clk, rst)
38          begin
39                  if(rst = '1') then
40                          state <= "0000";
41                  elsif(clk'event and clk = '1') then
42                          if(state = "0000" and ME = '0') then
43                                  state <= "0000";
44                          elsif(state = "0000" and ME = '1') then
45                                  state <= "0001";
46                          elsif(state = "0001") then
47                                  state <= "0010";
48                          elsif(state = "0010") then
49                                  state <= "0011";
50                          elsif(state = "0011") then
51                                  state <= "0100";
52                          elsif(state = "0100") then
53                                  state <= "0101";
54                          elsif(state = "0101") then
55                                  state <= "0110";
56                          elsif(state = "0110" and ME = '1') then
57                                  state <= "0111";
58                          elsif(state = "0110" and ME = '0') then
59                                  state <= "0000";
60                          elsif(state = "0111" and ME = '1') then
61                                  state <= "1000";
62                          elsif(state = "0111" and ME = '0') then
63                                  state <= "0000";
64                          elsif(state = "1000" and ME = '1') then
65                                  state <= "1001";
66                          elsif(state = "1000" and ME = '0') then
67                                  state <= "0000";
68                          elsif(state = "1001" and ME = '1') then
69                                  state <= "1010";
70                          elsif(state = "1001" and ME = '0') then
71                                  state <= "0000";
72                          elsif(state = "1010") then
73                                  state <= "1011";
74                          elsif(state = "1011") then
75                                  state <= "1100";
76                          elsif(state = "1100" and ME = '1') then
77                                  state <= "1101";
78                          elsif(state = "1101") then
79                                  state <= "1110";
80                          elsif(state = "1110" and ME = '0') then
81                                  state <= "0000";
82                          elsif(state = "1110" and ME = '1') then
83                                  state <= "1111";
84                          elsif(state = "1111") then
85                                  state <= "0000";
86                          end if;
87                  end if;
88
89          end process;
90
```

```
91          CE_not <= '1' when state = "0000" else
92                          '0' when state = "0001" or state = "0010" or state = "0011" or state = "
                                0100" or state = "0101" else
93                          '0' when (state = "0110" or state = "0111" or state = "1000" or state =
                                "1001" or state = "1010" or state = "1011" or state = "1100" or
                                state = "1101" or state = "1110" or state = "1111") and write_1 =
                                '0' else
94                          '1' when (state = "0110" or state = "0111" or state = "1000" or state =
                                "1001" or state = "1010" or state = "1011" or state = "1100" or
                                state = "1101" or state = "1110" or state = "1111") and write_1 =
                                '1' else
95                          '0';
96
97          WE_not <= '0' when (state = "0010" or state = "0011") and write_1 = '1' else
98                          '1';
99
100         OE_not <= '1' when write_1 = '1' else
101                         '1' when write_1 = '0' and state = "0000" else
102                         '0';
103
104         MA <= '1' when write_1 = '0' and (state = "1110" or state = "1111") else
105                        '1' when write_1 = '1' and (state = "0110" or state = "0111" or state = "
                               1000" or state = "1001" or state = "1010" or state = "1011" or state
                               = "1100"  or state = "1101" or state = "1110" or state = "1111")
                               else
106                        '0';
107
108 end r_w;
```

B.2.5 Memory Data Register (MDR)

```
1  --
       ------------------------------------------------------------------------------------------------
2  --
3  -- Title       : Memory Data Register
4  --
5  --
       ------------------------------------------------------------------------------------------------
6  --
        -------------------------
7  -- Description :                                    |  Memory Data Register   |
8  --                                                                  |
                                          |
9  --                                                                  -->|Rin     (16b)  Rout_a(16b)
       |-->
10 --                                                                  --->|save    (1b)      Rout_b(16
       b)|-->
11 --                                                                  ---|se_in    (1b)  Memory(16b)
       |<->
12 --                                                                  ---|sel_out (1b)              |
13 --                              |                       |
14 --                              |                       |
15 --
       -------------------------
```

```
16 --
17 --  ------------------------------------------------------------------------------------------------

18
19 library IEEE;
20 use IEEE.STD_LOGIC_1164.all;
21 use IEEE.STD_LOGIC_ARITH.ALL;
22 use IEEE.STD_LOGIC_UNSIGNED.ALL;
23
24 entity mdr is
25         port(
26                 Rin         : in STD_LOGIC_VECTOR(15 downto 0);
27                 Save        : in STD_LOGIC;
28                 Sel_out     : in STD_LOGIC_VECTOR(1 downto 0);
29                 Sel_in      : in STD_LOGIC;
30                 Rout_a      : out STD_LOGIC_VECTOR(15 downto 0);
31                 Rout_b      : out STD_LOGIC_VECTOR(15 downto 0);
32                 Mem_in_out  : inout STD_LOGIC_VECTOR(15 downto 0)
33                 );
34 end mdr;
35
36
37
38 architecture mdr of mdr is
39
40         signal Rout, Rin_1 : STD_LOGIC_VECTOR(15 downto 0);
41
42 begin
43
44 ------------------------------------------------------------------------------------------------
45 --                                MUX  IN
46 ------------------------------------------------------------------------------------------------
47
48         Rin_1 <= Rin         WHEN Sel_in='0' ELSE
49                  Mem_in_out WHEN Sel_in='1' ;
50
51 ------------------------------------------------------------------------------------------------
52 --                                Register
53 ------------------------------------------------------------------------------------------------
54
55         PROCESS (Save)
56         BEGIN
57                 IF (Save'EVENT AND Save='1') THEN
58                         Rout <= Rin_1;
59                 END IF;
60         END PROCESS;
61
62 ------------------------------------------------------------------------------------------------
63 --                    DEMUX Out A, B, Memory
64 ------------------------------------------------------------------------------------------------
65
66
67         Rout_a <= Rout WHEN Sel_out = "01" ELSE "ZZZZZZZZZZZZZZZZ";
68         Rout_b <= Rout WHEN Sel_out = "10" ELSE "ZZZZZZZZZZZZZZZZ";
69     Mem_in_out <= Rout WHEN Sel_out = "11" ELSE "ZZZZZZZZZZZZZZZZ";
```

```
70
71 ------------------------------------------------------------------------------
72 end mdr;
```

B.2.6 Memory Address Register (MAR)

```
1  --
       ------------------------------------------------------------------------------------------------
2  --
3  -- File          : alu.vhd
4  --
5  --
       ------------------------------------------------------------------------------------------------
6  --
       -------------------------
7  -- Description :                                          | Memory Address Register      |
8  --                                                                    |
                                          |
9  --                                                                 -->|Rin     (16b)        Rout
       (16b)|-->
10 --                                                                 ---|>save    (1b)
                        |
11 --                                                                 ---|en_out   (1b)              |
12 --                                      ---|sel      (1b)             |
13 --                                         |                         |
14 --
       ---------------------------
15 --
16 --
       ------------------------------------------------------------------------------------------------
```

```
17
18 library IEEE;
19 use IEEE.STD_LOGIC_1164.all;
20 use IEEE.STD_LOGIC_ARITH.ALL;
21 use IEEE.STD_LOGIC_UNSIGNED.ALL;
22
23 entity mar is
24         port(
25                 Rin         : in STD_LOGIC_VECTOR(15 downto 0);
26                 Rin_A       : in STD_LOGIC_VECTOR(15 downto 0);
27                 Save        : in STD_LOGIC;
28                 Sel                 : in STD_LOGIC;
29                 en_out      : in STD_LOGIC;
30                 Rout        : out STD_LOGIC_VECTOR(15 downto 0)
31                 );
32 end mar;
33
34
35
36 architecture mar of mar is
37
38         signal Rin_1, Rout_out : STD_LOGIC_VECTOR(15 downto 0);
39
```

```
40  begin
41
42  ------------------------------------------------------------------------------
43  --                                  Mux Save
44  ------------------------------------------------------------------------------
45
46          Rin_1 <= Rin WHEN Sel ='0' ELSE Rin_A;
47
48  ------------------------------------------------------------------------------
49  --                                  Register
50  ------------------------------------------------------------------------------
51
52          PROCESS (Save)
53          BEGIN
54                  IF (Save'EVENT AND Save='1') THEN
55                          Rout_out <= Rin_1;
56                  END IF;
57          END PROCESS;
58
59  ------------------------------------------------------------------------------
60  --                                Output Enable
61  ------------------------------------------------------------------------------
62
63          Rout <= Rout_out WHEN en_out='1' ELSE "ZZZZZZZZZZZZZZZZ";
64
65  ------------------------------------------------------------------------------
66  end mar;
```

B.2.7 Sixteen bits Register

```
1  --
       ------------------------------------------------------------------------------
2  --
3  -- Title       : Register
4  --
5  --
       ------------------------------------------------------------------------------
6  --
        -------------------------
7  -- Description :                                  | Memory Address Register |
8  --                                                                  |
                                        |
9  --                                                                  -->|Rin     (16b)       Rout (16b
       )|-->
10 --                                                                  --->|save    (1b)
                           |
11 --                                  |                          |
12 --                                  |                          |
13 --
       ---------------------------
14 --
15 --
       ------------------------------------------------------------------------------
```

```
16
17 library IEEE;
18 use IEEE.STD_LOGIC_1164.all;
19 use IEEE.STD_LOGIC_ARITH.ALL;
20 use IEEE.STD_LOGIC_UNSIGNED.ALL;
21
22 entity gen_register is
23         port(
24                 Rin         : in STD_LOGIC_VECTOR(15 downto 0);
25                 Save        : in STD_LOGIC;
26                 Rout        : out STD_LOGIC_VECTOR(15 downto 0)
27                 );
28 end gen_register;
29
30
31
32 architecture gen_register of gen_register is
33
34
35 begin
36
37 ---------------------------------------------------------------------------
38 --                              Register
39 ---------------------------------------------------------------------------
40
41         PROCESS (Save)
42         BEGIN
43                 IF (Save'EVENT AND Save='1') THEN
44                         Rout <= Rin;
45                 END IF;
46         END PROCESS;
47
48 ---------------------------------------------------------------------------
49 end gen_register;
```

B.2.8 Register Outpus Selector

```
1 --
      ------------------------------------------------------------------------------------------------
2 --
3 -- File          : 4-1 MUX.vhd
4 --
5 --
      ------------------------------------------------------------------------------------------------
6 --
7 -- Description :          MUX for Interrupt vectors and diplacement from the instruction register
8 --
9 --
      ------------------------------------------------------------------------------------------------
10
11 library IEEE;
12 use IEEE.STD_LOGIC_1164.all;
13
```

```
14  entity mux_dv is
15          port(
16                  SDV : in STD_LOGIC;
17                  EDV : in STD_LOGIC;
18                  Disp : in STD_LOGIC_VECTOR(15 downto 0);
19                  Vector : in STD_LOGIC_VECTOR(15 downto 0);
20                  Q : out std_logic_vector(15 downto 0)
21              );
22  end mux_dv;
23
24  architecture mux_1 of mux_dv is
25
26          signal Q_temp : std_logic_vector(15 downto 0);
27
28  begin
29
30          Q_temp <= Disp when SDV = '0' else
31                          Vector;
32
33          Q <= Q_temp when EDV = '1' else "ZZZZZZZZZZZZZZZZ";
34
35  end mux_1;
```

B.2.9 Addressing Mode Decode

```
1
2   --
        ------------------------------------------------------------------------------------------------
3   --
4   -- File         : addressing_decode.vhd
5   --
6   --
        ------------------------------------------------------------------------------------------------
7   --
8   -- Description : Decide which addressingf mode is been used in the current instruction
9   --
10  --
        ------------------------------------------------------------------------------------------------
11
12
13  library IEEE;
14  use IEEE.STD_LOGIC_1164.all;
15
16  entity address_dec is
17           port(
18                  Ad : in STD_LOGIC;
19                  As : in STD_LOGIC_VECTOR(1 downto 0);
20                  rb : in std_logic_vector(4 downto 0);
21                  A : out STD_LOGIC_VECTOR(1 downto 0)
22                  );
23  end address_dec;
24
25
```

```
26
27 architecture address_dec_1 of address_dec is
28         signal A_1 : std_logic_vector(1 downto 0);
29 begin
30
31         A_1 <="01" When As="01" or (As="00" and Ad='1' and rb/="00000") else
32                    "10" When As="10" or (As="00" and Ad='1' and rb="00000") else
33                    "11" When As="11" else
34                         "00";
35
36
37         A <= A_1;
38 end address_dec_1;
```

B.2.10 Vector/Displacement Multiplexer

```
1  --
       ------------------------------------------------------------------------------------------------
2  --
3  -- File         : bus_mux.vhd
4  --
5  --
       ------------------------------------------------------------------------------------------------
6  --
7  -- Description : deside the register that will exit to the datapath
8  --
9  --
       ------------------------------------------------------------------------------------------------
10
11 library IEEE;
12 use IEEE.STD_LOGIC_1164.all;
13
14 entity b_mux is
15          port(
16                   r : in STD_LOGIC_VECTOR(4 downto 0);
17                   s : in STD_LOGIC_VECTOR(1 downto 0);
18                   Q : out std_logic_vector(4 downto 0)
19               );
20 end b_mux;
21
22 architecture b_mux_1 of b_mux is
23 begin
24
25         Q <= "10000" when s = "00" else
26                   "10001" when s = "01" else
27                   r  when s = "10" else
28                   "10010" when s = "11";
29
30 end b_mux_1;
```

B.2.11 Clock Contol

```
1  --
       ------------------------------------------------------------------------------------------

2  --
3  -- File         : clk_control.vhd
4  --
5  --
       ------------------------------------------------------------------------------------------

6  --
7  -- Description : Desible the clock
8  --
9  --
       ------------------------------------------------------------------------------------------

10
11 library IEEE;
12 use IEEE.STD_LOGIC_1164.all;
13
14 entity clkc is
15          port(
16                  clk : in STD_LOGIC;
17                  CLK_OFF : in STD_LOGIC;
18                  int : in STD_LOGIC;
19                  clk_out : out STD_LOGIC
20              );
21 end clkc;
22
23 architecture clkc_1 of clkc is
24 begin
25
26          clk_out <=      clk when int = '1' or CLK_OFF = '0' else '0';
27
28 end clkc_1;
```

B.2.12 Control Unit

```
1  --
       ------------------------------------------------------------------------------------------

2  --
3  -- File         : Control Unit.vhd
4  --
5  --
       ------------------------------------------------------------------------------------------

6  --
7  -- Description : This file is the main file of the contol unit which call all the singnals
8  --                generators of the different instructions.
9  --
10 --
       ------------------------------------------------------------------------------------------

11
12 library IEEE;
13 use IEEE.STD_LOGIC_1164.all;
```

```
14
15 entity CU is
16         port(
17                  IR        : in STD_LOGIC_VECTOR(15 downto 0);
18                  MA        : in STD_LOGIC;
19                  clk_in    : in STD_LOGIC;
20                  rst       : in STD_LOGIC;
21                  Sr        : in std_logic_vector(5 downto 0);
22                  GIR       : in std_logic_vector(15 downto 0);
23                  IER       : in std_logic_vector(15 downto 1);
24                  save_ir   : out std_logic;
25                  RW      : out std_logic;
26                  ME            : out std_logic;
27                  save      : out std_logic;
28                  en_out_a  : out std_logic;
29                  en_out_b  : out std_logic;
30                  save_sr   : out std_logic;
31                  en_SR     : out std_logic;
32                  rst_sr    : out std_logic;
33                  pc_out    : out std_logic;
34                  reset_i   : out std_logic;
35                  save_mar  : out std_logic;
36                  en_out_mar      : out std_logic;
37                  sel       : out std_logic;
38                  sel_in_mdr      : out std_logic;
39                  sel_out_mdr : out std_logic_vector(1 downto 0);
40                  save_mdr  : out std_logic;
41                  save_c    : out std_logic;
42           S         : out std_logic_vector(3 downto 0);
43                  Bus_A     : out std_logic_vector(15 downto 0);
44                  A_reg     : out std_logic_vector(4 downto 0);
45                  B_C_reg   : out std_logic_vector(4 downto 0)
46               );
47 end CU;
48
49 architecture structural of CU is
50
51
52         COMPONENT fetch IS
53                  port(
54                           IR        : in STD_LOGIC_VECTOR(15 downto 0);
55                           Op        : out STD_LOGIC_VECTOR(3 downto 0);
56                           As        : out STD_LOGIC_VECTOR(1 downto 0);
57                           Ad        : out STD_LOGIC;
58                           Reg_A     : out STD_LOGIC_VECTOR(4 downto 0);
59                           Reg_B_C   : out STD_LOGIC_VECTOR(4 downto 0);
60                           Disp : out STD_LOGIC_VECTOR(15 downto 0)
61                           );
62         END COMPONENT;
63
64 ----------------Addressing Decoder--------------------
65
66         COMPONENT address_dec IS
67                  port(
68                            Ad : in STD_LOGIC;
69                            As : in STD_LOGIC_VECTOR(1 downto 0);
```

```
70                          rb : in std_logic_vector(4 downto 0);
71                          A : out STD_LOGIC_VECTOR(1 downto 0)
72                      );
73          END COMPONENT;
74
75  ----------------Execute Decoder-----------------------
76
77          COMPONENT eid IS
78                  port(
79                          Op : in STD_LOGIC_VECTOR(3 downto 0);
80                          sr : in STD_LOGIC_VECTOR(3 downto 0);
81                          E  : in std_logic;
82                          E_int : in std_logic;
83                          reset : in std_logic;
84                          P     : out std_logic;
85                          R     : out std_logic;
86                          J     : out std_logic;
87                          enable_d :      out std_logic;
88                          enable_p :      out std_logic;
89                          enable_interrupt : out std_logic
90                      );
91          END COMPONENT;
92
93  ----------------Interrupt Decoder-----------------------
94
95          COMPONENT int IS
96                  port(
97                              GIR         : in STD_LOGIC_VECTOR(15 downto 0);
98                              IER         : in STD_LOGIC_VECTOR(15 downto 1);
99                              IE          :      in STD_LOGIC;
100                             vector      : out STD_LOGIC_VECTOR(15 downto 0);
101                             Interrupt : out STD_LOGIC;
102                             Reset       : out std_logic
103                     );
104         END COMPONENT;
105
106 ----------------Bus MUX-----------------------
107
108         COMPONENT b_mux IS
109                 port(
110                         r : in STD_LOGIC_VECTOR(4 downto 0);
111                         s : in STD_LOGIC_VECTOR(1 downto 0);
112                         Q : out std_logic_vector(4 downto 0)
113                     );
114         END COMPONENT;
115
116 ----------------Displacement Vector MUX-----------------------
117
118         COMPONENT mux_dv IS
119                 port(
120                         SDV : in STD_LOGIC;
121                         EDV : in STD_LOGIC;
122                         Disp : in STD_LOGIC_VECTOR(15 downto 0);
123                         Vector : in STD_LOGIC_VECTOR(15 downto 0);
124                         Q : out std_logic_vector(15 downto 0)
125                     );
```

```
126         END COMPONENT;
127
128 ----------Clock controller----------
129
130         COMPONENT clkc IS
131                 port(
132                         clk : in STD_LOGIC;
133                         CLK_OFF : in STD_LOGIC;
134                         int : in STD_LOGIC;
135                         clk_out : out STD_LOGIC
136                     );
137         END COMPONENT;
138
139 ----------Main State Mechine----------
140
141
142         COMPONENT msn IS
143                 port(
144                         clk : in STD_LOGIC;
145                         rst : in STD_LOGIC;
146                         E : out STD_LOGIC_VECTOR(3 downto 0)
147                     );
148         END COMPONENT;
149
150 ----------Fetch State Mechine----------
151
152         COMPONENT fsm IS
153                 port(
154                         F : in STD_LOGIC;
155                         MA : in STD_LOGIC;
156                         clk : in STD_LOGIC;
157                         rst : in STD_LOGIC;
158                         state : out STD_LOGIC_VECTOR(1 downto 0);
159                         state1 : out STD_LOGIC_VECTOR(1 downto 0)
160                     );
161         END COMPONENT;
162
163 ----------Interrupt State Mechine----------
164
165         COMPONENT ism IS
166                 port(
167                         I : in STD_LOGIC;
168                         MA : in STD_LOGIC;
169                         clk : in STD_LOGIC;
170                         rst : in STD_LOGIC;
171                         state : out STD_LOGIC_VECTOR(3 downto 0);
172                         nstate : out STD_LOGIC_VECTOR(3 downto 0);
173                         second : out std_logic
174               );
175         END COMPONENT;
176
177 ----------Decode state Mashine----------
178
179         COMPONENT dsm IS
180                 port(
181                         D : in STD_LOGIC;
```

```
182                         A : in STD_LOGIC_VECTOR(1 downto 0);
183                         MA : in STD_LOGIC;
184                         rst : in STD_LOGIC;
185                         clk : in STD_LOGIC;
186                         State : out STD_LOGIC_VECTOR(2 downto 0);
187                         Nstate : out STD_LOGIC_VECTOR(2 downto 0)
188             );
189         END COMPONENT;
190
191 ---------------Execute two operands state Mashine--------------------
192
193         COMPONENT edsm IS
194                 port(
195                         clk : in STD_LOGIC;
196                         rst : in STD_LOGIC;
197                         E : in STD_LOGIC;
198                         MA : in STD_LOGIC;
199                         state : out STD_LOGIC_VECTOR(1 downto 0);
200                         nstate : out STD_LOGIC_VECTOR(1 downto 0)
201             );
202         END COMPONENT;
203
204 ---------------Execute push/pop/reti state Mashine--------------------
205
206         COMPONENT epuposm IS
207                 port(
208                         E : in STD_LOGIC;
209                         MA : in STD_LOGIC;
210                         P : in STD_LOGIC;                -- PUSH/POP
211                         R : in STD_LOGIC;            -- RETI
212                         clk : in STD_LOGIC;
213                         rst : in STD_LOGIC;
214                         T1 : out std_logic;
215                         state : out STD_LOGIC_VECTOR(2 downto 0);
216                         nstate : out STD_LOGIC_VECTOR(2 downto 0)
217             );
218         END COMPONENT;
219
220 ----------------------------------------
221
222         COMPONENT sig IS
223                 port(
224                         fetch     : in STD_LOGIC_VECTOR(1 downto 0);
225                         fetch_1   : in STD_LOGIC_VECTOR(1 downto 0);
226                         decode    : in STD_LOGIC_VECTOR(2 downto 0);
227                         decode_1  : in STD_LOGIC_VECTOR(2 downto 0);
228                         execute_d    : in STD_LOGIC_VECTOR(1 downto 0);
229                         execute_d_1  : in STD_LOGIC_VECTOR(1 downto 0);
230                         execute_p    : in STD_LOGIC_VECTOR(2 downto 0);
231                         execute_p_1  : in STD_LOGIC_VECTOR(2 downto 0);
232                         interrupt : in STD_LOGIC_VECTOR(3 downto 0);
233                         interrupt_1 : in STD_LOGIC_VECTOR(3 downto 0);
234                         second    : in std_logic;
235                         as        : in std_logic_vector(1 downto 0);
236                         ad        :     in std_logic;
237                         A         : in std_logic_vector(1 downto 0);
```

```
238                             Op        : in std_logic_vector(3 downto 0);
239                             R         : in std_logic;
240                             P         : in std_logic;
241                             J         : in std_logic;
242                             reset     : in std_logic;
243                             int       : in std_logic;
244                             sa        : out std_logic_vector(1 downto 0);
245                             sb        : out std_logic_vector(1 downto 0);
246                             SDV       : out std_logic;
247                             EDV       : out std_logic;
248                             save_ir   : out std_logic;
249                             RW      : out std_logic;
250                             ME            : out std_logic;
251                             save      : out std_logic;
252                             en_out_a  : out std_logic;
253                             en_out_b  : out std_logic;
254                             save_sr   : out std_logic;
255                             en_SR     : out std_logic;
256                             rst_sr    : out std_logic;
257                             pc_out    : out std_logic;
258                             reset_i   : out std_logic;
259                             save_mar  : out std_logic;
260                             en_out_mar      : out std_logic;
261                             sel       : out std_logic;
262                             sel_in_mdr      : out std_logic;
263                             sel_out_mdr : out std_logic_vector(1 downto 0);
264                             save_mdr  : out std_logic;
265                             save_c    : out std_logic;
266                     S         : out std_logic_vector(3 downto 0);
267                             next_state      : out std_logic;
268                             escape    : out std_logic
269                         );
270         END COMPONENT;
271
272         signal enable_interrupt, clk, ad, next_st, sec_i, sec_r, sec, enable_execute, escape,
                Reset, P, R, J, enable_d, enable_p, enable_i, enable_decode, enable_f, interrupt,
                SDV, EDV : std_logic;
273         signal Pfetch, Nfetch, A, Ped, Ned, sa, sb, as : std_logic_vector(1 downto 0);
274         signal Pi, Ni : std_logic_vector(3 downto 0);
275         signal Pd, Nd, Pep, Nep : std_logic_vector(2 downto 0);
276         signal En, Op : std_logic_vector(3 downto 0);
277         signal Reg_A, Reg_b_C : std_logic_vector(4 downto 0);
278         signal Disp, vector : std_logic_vector(15 downto 0);
279
280 begin
281
282 ----------Address Decoder----------
283
284         U0: address_dec PORT MAP (ad, as, Reg_B_C,A);
285
286 ----------Execute and Interrupt Decoder------
287
288         U1: eid PORT MAP (Op, sr(3 downto 0), enable_execute, enable_i, Reset, P, R, J, enable_d,
                enable_p, enable_interrupt);
289
290 ----------Interrupt Decoder----------------
```

```
291
292         U2: int PORT MAP (GIR, IER(15 downto 1), sr(4), vector, interrupt, Reset);
293
294 --------Register Selector Bus A----------
295
296         U3: b_mux PORT MAP (Reg_A, sa, A_reg);
297
298 --------Register Selector Bus B----------
299
300         U4: b_mux PORT MAP (Reg_B_C, sb, B_C_reg);
301
302 --------Vector Displacement Controller----------
303
304         U5: mux_dv PORT MAP (SDV, EDV, Disp, vector, Bus_A);
305
306 --------Clock Controller----------
307
308         U6: clkc PORT MAP (clk_in, sr(5), interrupt, clk);
309
310 ----------------Fetch-------------------------
311
312         U7: fetch PORT MAP (IR, Op, as, ad, Reg_A, Reg_B_C, Disp);
313
314 -------------Main State Machine-------------
315         U8: msn PORT MAP (next_st, rst, En);
316
317 -------------Interrupt State Machine--------
318
319         U9: ism PORT MAP (enable_interrupt, MA, clk, rst, Pi, Ni, sec_i);
320
321 ---------------Fetch State Machine----------
322
323         U10: fsm        PORT MAP (enable_f, MA, clk, rst, Pfetch, Nfetch);
324
325 ------Decode State Machine---------
326
327         U11: dsm        PORT MAP (enable_decode, A, MA, rst, clk, Pd, Nd);
328
329 ------ Execute instuction with two operands (state machine)
330
331         U12: edsm PORT MAP ( clk, rst, enable_d, MA, Ped, Ned);
332
333 ------PUSH and POP state machine---------
334
335         U13: epuposm PORT MAP (enable_p, MA,P, R, clk, rst, sec_r, Pep, Nep);
336
337 --------Signals generator----------------
338
339         U14:    sig     PORT MAP (Pfetch, Nfetch, Pd, Nd, Ped, Ned, Pep, Nep, Pi, Ni, sec, as, ad
                , A, Op, R, P, J, reset, interrupt, sa, sb, SDV, EDV, save_ir, RW, ME, save,
                en_out_a, en_out_b, save_sr, en_sr, rst_sr, pc_out, reset_i, save_mar, en_out_mar,
                sel, sel_in_mdr, sel_out_mdr, save_mdr, save_c, S, next_st, escape);
340
341
342 ---------------Escape from state machines--------------------
343         enable_i <=  En(0) and not escape;
```

```
344     enable_f <=  En(1) and not escape;
345         enable_decode <=  En(2) and not escape;
346         enable_execute <=  En(3) and not escape;
347
348         sec <= sec_i or sec_r;
349
350
351 end structural;
```

B.2.13 Decode Signals

```
1  --
       ------------------------------------------------------------------------------------------------

2  --
3  -- File         : decode_signals.vhd
4  --
5  --
       ------------------------------------------------------------------------------------------------

6  --
7  -- Description :         this file has as imput the decode state machine and generate it signals
8  --
9  --
       ------------------------------------------------------------------------------------------------

10
11 library IEEE;
12 use IEEE.STD_LOGIC_1164.all;
13
14 entity ds is
15         port(
16                 decode    : in STD_LOGIC_VECTOR(2 downto 0);
17                 decode_1  : in STD_LOGIC_VECTOR(2 downto 0);
18                 as        : in std_logic_vector(1 downto 0);
19                 ad        :     in std_logic;
20                 A         : in std_logic_vector(1 downto 0);
21                 Op        :     in std_logic_vector(3 downto 0);
22                 sa        : out std_logic_vector(1 downto 0);
23                 sb        : out std_logic_vector(1 downto 0);
24                 SDV       : out std_logic;
25                 EDV       : out std_logic;
26                 save_ir   : out std_logic;
27                 RW        : out std_logic;
28                 ME            : out std_logic;
29                 save      : out std_logic;
30                 en_out_a  : out std_logic;
31                 en_out_b  : out std_logic;
32                 save_sr   : out std_logic;
33                 en_SR     : out std_logic;
34                 rst_sr    : out std_logic;
35                 pc_out    : out std_logic;
36                 reset_i   : out std_logic;
37                 save_mar  : out std_logic;
38                 en_out_mar      : out std_logic;
39                 sel       : out std_logic;
```

```
40                  sel_in_mdr      : out std_logic;
41                  sel_out_mdr : out std_logic_vector(1 downto 0);
42                  save_mdr : out std_logic;
43                  save_c    : out std_logic;
44          S        : out std_logic_vector(3 downto 0);
45                  next_state      : out std_logic;
46                  escape : out std_logic
47              );
48  end ds;
49
50  architecture ds_1 of ds is
51  begin
52
53  ------------------------------------------------------------
54  --                      Decode                            --
55  ------------------------------------------------------------
56                  process(decode, decode_1, A, Op, as, ad)
57                  begin
58  ---------------------State 000-----------------------------
59                  if(decode = "000" and decode_1 = "000") then
60                          sa          <= "00";
61                          sb          <= "00";
62                          S                   <= "0000";
63                          sel_out_mdr <= "00";
64                          SDV         <= '0';
65                          EDV         <= '0';
66                          save_ir     <= '0';
67                          RW          <= '0';
68                          ME          <= '0';
69                          save        <= '0';
70                          en_out_a    <= '0';
71                          en_out_b    <= '0';
72                          save_sr         <= '0';
73                          en_SR       <= '0';
74                          rst_sr      <= '0';
75                          pc_out          <= '0';
76                          save_mar        <= '0';
77                          en_out_mar  <= '0';
78                          sel             <= '0';
79                          sel_in_mdr  <= '0';
80                          save_mdr        <= '0';
81                          save_c      <= '0';
82                          next_state  <= '0';
83                          escape      <= '0';
84
85  --------------------State 001 000---------------------------
86                  elsif(decode = "001" and decode_1 = "000" and A/="00") then
87                          sa          <= "00";
88                          sb          <= "00";
89                          S               <= "0001";
90                          sel_out_mdr <= "00";
91                          SDV         <= '0';
92                          EDV         <= '0';
93                          save_ir     <= '0';
94                          RW          <= '1';
95                          ME          <= '1';
```

```
96                          save        <= '0';
97                          en_out_a    <= '1';
98                          en_out_b    <= '0';
99                          save_sr         <= '0';
100                         en_SR       <= '0';
101                         rst_sr      <= '0';
102                         pc_out          <= '1';
103                         save_mar        <= '0';
104                         en_out_mar  <= '0';
105                         sel             <= '0';
106                         sel_in_mdr  <= '0';
107                         save_mdr        <= '0';
108                         save_c      <= '0';
109                         next_state  <= '0';
110                         escape      <= '0';
111
112 ----------------------State 001 000----Register-----------------------
113                 elsif(decode = "001" and decode_1 = "000" and (A="00" or Op = "0111" or Op = "
                        1011" or Op = "1101" or Op = "1110" or Op = "1010" or Op = "1100" or Op = "
                        1001")) then
114                         sa          <= "00";
115                         sb          <= "00";
116                         S               <= "0000";
117                         sel_out_mdr <= "00";
118                         SDV         <= '0';
119                         EDV         <= '0';
120                         save_ir     <= '0';
121                         RW          <= '0';
122                         ME          <= '0';
123                         save        <= '0';
124                         en_out_a    <= '0';
125                         en_out_b    <= '0';
126                         save_sr         <= '0';
127                         en_SR       <= '0';
128                         rst_sr      <= '0';
129                         pc_out          <= '0';
130                         save_mar        <= '0';
131                         en_out_mar  <= '0';
132                         sel             <= '0';
133                         sel_in_mdr  <= '0';
134                         save_mdr        <= '0';
135                         save_c      <= '0';
136                         next_state  <= '1';
137                         escape      <= '0';
138
139
140 ----------------------State 001 001----------------------------
141                 elsif(decode = "001" and decode_1 = "001" and A/="00") then
142                         sa          <= "00";
143                         sb          <= "00";
144                         S               <= "0001";
145                         sel_out_mdr <= "00";
146                         SDV         <= '0';
147                         EDV         <= '0';
148                         save_ir     <= '0';
149                         RW          <= '1';
```

```
150                         ME          <= '1';
151                         save        <= '0';
152                         en_out_a    <= '1';
153                         en_out_b    <= '0';
154                         save_sr         <= '0';
155                         en_SR       <= '0';
156                         rst_sr      <= '0';
157                         pc_out          <= '1';
158                         save_mar        <= '0';
159                         en_out_mar  <= '0';
160                         sel             <= '0';
161                         sel_in_mdr  <= '1';
162                         save_mdr        <= '0';
163                         save_c      <= '1';
164                         next_state  <= '0';
165                         escape      <= '0';
166
167 ----------------------State 001 001------Register----------------------
168                     elsif(decode = "001" and decode_1 = "001" and (A="00" or Op = "0111" or Op = "
                            1011" or Op = "1101" or Op = "1110" or Op = "1010" or Op = "1100" or Op = "
                            1001")) then
169                         sa          <= "00";
170                         sb          <= "00";
171                         S               <= "0000";
172                         sel_out_mdr <= "00";
173                         SDV         <= '0';
174                         EDV         <= '0';
175                         save_ir     <= '0';
176                         RW          <= '0';
177                         ME          <= '0';
178                         save        <= '0';
179                         en_out_a    <= '0';
180                         en_out_b    <= '0';
181                         save_sr         <= '0';
182                         en_SR       <= '0';
183                         rst_sr      <= '0';
184                         pc_out          <= '0';
185                         save_mar        <= '0';
186                         en_out_mar  <= '0';
187                         sel             <= '0';
188                         sel_in_mdr  <= '0';
189                         save_mdr        <= '0';
190                         save_c      <= '0';
191                         next_state  <= '1';
192                         escape      <= '0';
193
194 ----------------------State 011 001-not Immediate------------------
195                     elsif(decode = "011" and decode_1 = "001" and A/="11") then
196                         sa          <= "00";
197                         sb          <= "00";
198                         S               <= "0000";
199                         sel_out_mdr <= "00";
200                         SDV         <= '0';
201                         EDV         <= '0';
202                         save_ir     <= '0';
203                         RW          <= '1';
```

```
204                         ME          <= '1';
205                         save        <= '0';
206                         en_out_a    <= '0';
207                         en_out_b    <= '0';
208                         save_sr         <= '0';
209                         en_SR       <= '0';
210                         rst_sr      <= '0';
211                         pc_out          <= '1';
212                         save_mar        <= '0';
213                         en_out_mar  <= '0';
214                         sel             <= '0';
215                         sel_in_mdr  <= '1';
216                         save_mdr        <= '1';
217                         save_c      <= '0';
218                         next_state  <= '0';
219                         escape      <= '0';
220
221 ----------------------State 011 011---not Immediate---------------
222                     elsif(decode = "011" and decode_1 = "011" and A/="11") then
223                         sa          <= "00";
224                         sb          <= "00";
225                         S               <= "0000";
226                         sel_out_mdr <= "00";
227                         SDV         <= '0';
228                         EDV         <= '0';
229                         save_ir     <= '0';
230                         RW          <= '1';
231                         ME          <= '1';
232                         save        <= '1';
233                         en_out_a    <= '0';
234                         en_out_b    <= '0';
235                         save_sr         <= '0';
236                         en_SR       <= '0';
237                         rst_sr      <= '0';
238                         pc_out          <= '1';
239                         save_mar        <= '0';
240                         en_out_mar  <= '0';
241                         sel             <= '0';
242                         sel_in_mdr  <= '1';
243                         save_mdr        <= '1';
244                         save_c      <= '0';
245                         next_state  <= '0';
246                         escape      <= '0';
247
248 ----------------------State 011 001---Immediate----------------
249                     elsif(decode = "011" and decode_1 = "001" and A="11") then
250                         sa          <= "00";
251                         sb          <= "00";
252                         S               <= "0000";
253                         sel_out_mdr <= "00";
254                         SDV         <= '0';
255                         EDV         <= '0';
256                         save_ir     <= '0';
257                         RW          <= '1';
258                         ME          <= '1';
259                         save        <= '0';
```

```
260                         en_out_a    <= '0';
261                         en_out_b    <= '0';
262                         save_sr         <= '0';
263                         en_SR       <= '0';
264                         rst_sr      <= '0';
265                         pc_out          <= '1';
266                         save_mar        <= '0';
267                         en_out_mar  <= '0';
268                         sel             <= '0';
269                         sel_in_mdr  <= '1';
270                         save_mdr        <= '1';
271                         save_c      <= '0';
272                         next_state  <= '0';
273                         escape      <= '0';
274
275 ----------State 011 011---Immediate----------
276                 elsif(decode = "011" and decode_1 = "011" and A="11") then
277                         sa          <= "00";
278                         sb          <= "00";
279                         S               <= "0000";
280                         sel_out_mdr <= "00";
281                         SDV         <= '0';
282                         EDV         <= '0';
283                         save_ir     <= '0';
284                         RW          <= '1';
285                         ME          <= '1';
286                         save        <= '1';
287                         en_out_a    <= '0';
288                         en_out_b    <= '0';
289                         save_sr         <= '0';
290                         en_SR       <= '0';
291                         rst_sr      <= '0';
292                         pc_out          <= '1';
293                         save_mar        <= '0';
294                         en_out_mar  <= '0';
295                         sel             <= '0';
296                         sel_in_mdr  <= '1';
297                         save_mdr        <= '1';
298                         save_c      <= '0';
299                         next_state  <= '1';
300                         escape      <= '0';
301
302 ----------State 010 011 -Indexed source----------
303                 elsif(decode = "010" and decode_1 = "011" and As="01") then
304                         sa          <= "10";
305                         sb          <= "00";
306                         S                   <= "1100";
307                         sel_out_mdr <= "10";
308                         SDV         <= '0';
309                         EDV         <= '0';
310                         save_ir     <= '0';
311                         RW          <= '0';
312                         ME          <= '0';
313                         save        <= '0';
314                         en_out_a    <= '1';
315                         en_out_b    <= '0';
```

```
316                         save_sr         <= '0';
317                         en_SR       <= '0';
318                         rst_sr      <= '0';
319                         pc_out          <= '0';
320                         save_mar        <= '0';
321                         en_out_mar  <= '0';
322                         sel             <= '0';
323                         sel_in_mdr  <= '0';
324                         save_mdr        <= '0';
325                         save_c      <= '0';
326                         next_state  <= '0';
327                         escape      <= '0';
328
329 ----------------------State 010 010 Indexed Source----------------------
330                     elsif(decode = "010" and decode_1 = "010" and As="01") then
331                         sa          <= "10";
332                         sb          <= "00";
333                         S                   <= "1100";
334                         sel_out_mdr <= "10";
335                         SDV         <= '0';
336                         EDV         <= '0';
337                         save_ir     <= '0';
338                         RW          <= '0';
339                         ME          <= '0';
340                         save        <= '0';
341                         en_out_a    <= '1';
342                         en_out_b    <= '0';
343                         save_sr         <= '0';
344                         en_SR       <= '0';
345                         rst_sr      <= '0';
346                         pc_out          <= '0';
347                         save_mar        <= '0';
348                         en_out_mar  <= '0';
349                         sel             <= '0';
350                         sel_in_mdr  <= '0';
351                         save_mdr        <= '0';
352                         save_c      <= '1';
353                         next_state  <= '0';
354                         escape      <= '0';
355
356 ----------------------State 010 011 Indexed Destination-----------------
357                     elsif(decode = "010" and decode_1 = "011" and As="00" and Ad='1' and A = "01")
                            then
358                         sa          <= "00";
359                         sb          <= "10";
360                         S                   <= "1100";
361                         sel_out_mdr <= "01";
362                         SDV         <= '0';
363                         EDV         <= '0';
364                         save_ir     <= '0';
365                         RW          <= '0';
366                         ME          <= '0';
367                         save        <= '0';
368                         en_out_a    <= '0';
369                         en_out_b    <= '1';
370                         save_sr         <= '0';
```

```
371                             en_SR       <= '0';
372                             rst_sr      <= '0';
373                             pc_out          <= '0';
374                             save_mar        <= '0';
375                             en_out_mar  <= '0';
376                             sel             <= '0';
377                             sel_in_mdr  <= '0';
378                             save_mdr        <= '0';
379                             save_c      <= '0';
380                             next_state  <= '0';
381                             escape      <= '0';
382
383 ----------------------State 010 010 Indexed Destination--------------
384                     elsif(decode = "010" and decode_1 = "010" and As="00" and Ad = '1') then
385                             sa          <= "00";
386                             sb          <= "10";
387                             S                   <= "1100";
388                             sel_out_mdr <= "01";
389                             SDV         <= '0';
390                             EDV         <= '0';
391                             save_ir     <= '0';
392                             RW          <= '0';
393                             ME          <= '0';
394                             save        <= '0';
395                             en_out_a    <= '0';
396                             en_out_b    <= '1';
397                             save_sr         <= '0';
398                             en_SR       <= '0';
399                             rst_sr      <= '0';
400                             pc_out          <= '0';
401                             save_mar        <= '0';
402                             en_out_mar  <= '0';
403                             sel             <= '0';
404                             sel_in_mdr  <= '0';
405                             save_mdr        <= '0';
406                             save_c      <= '1';
407                             next_state  <= '0';
408                             escape      <= '0';
409
410 ----------------State 110 010 Indexed--------------------
411                     elsif(decode = "110" and decode_1 = "010" and A="01") then
412                             sa          <= "00";
413                             sb          <= "00";
414                             S                   <= "0000";
415                             sel_out_mdr <= "00";
416                             SDV         <= '0';
417                             EDV         <= '0';
418                             save_ir     <= '0';
419                             RW          <= '0';
420                             ME          <= '0';
421                             save        <= '0';
422                             en_out_a    <= '0';
423                             en_out_b    <= '0';
424                             save_sr         <= '0';
425                             en_SR       <= '0';
426                             rst_sr      <= '0';
```

```
427                         pc_out          <= '0';
428                         save_mar        <= '0';
429                         en_out_mar  <= '0';
430                         sel             <= '0';
431                         sel_in_mdr  <= '0';
432                         save_mdr        <= '0';
433                         save_c      <= '0';
434                         next_state  <= '0';
435                         escape      <= '0';
436
437 ----------------State 110 110 Indexed-not MOV----------------------
438                 elsif(decode = "110" and decode_1 = "110" and A="01" and Op /= "0000") then
439                         sa          <= "00";
440                         sb          <= "00";
441                         S                   <= "0000";
442                         sel_out_mdr <= "00";
443                         SDV         <= '0';
444                         EDV         <= '0';
445                         save_ir     <= '0';
446                         RW          <= '1';
447                         ME          <= '1';
448                         save        <= '0';
449                         en_out_a    <= '0';
450                         en_out_b    <= '0';
451                         save_sr         <= '0';
452                         en_SR       <= '0';
453                         rst_sr      <= '0';
454                         pc_out          <= '0';
455                         save_mar        <= '1';
456                         en_out_mar  <= '1';
457                         sel             <= '0';
458                         sel_in_mdr  <= '1';
459                         save_mdr        <= '0';
460                         save_c      <= '0';
461                         next_state  <= '0';
462                         escape      <= '0';
463
464 ----------------State 110 110 Indexed MOV-Source----------------------
465                 elsif(decode = "110" and decode_1 = "110" and A="01" and Op = "0000" and As = "01
                        ") then
466                         sa          <= "00";
467                         sb          <= "00";
468                         S                   <= "0000";
469                         sel_out_mdr <= "00";
470                         SDV         <= '0';
471                         EDV         <= '0';
472                         save_ir     <= '0';
473                         RW          <= '1';
474                         ME          <= '1';
475                         save        <= '0';
476                         en_out_a    <= '0';
477                         en_out_b    <= '0';
478                         save_sr         <= '0';
479                         en_SR       <= '0';
480                         rst_sr      <= '0';
481                         pc_out          <= '0';
```

```
482                         save_mar        <= '1';
483                         en_out_mar  <= '1';
484                         sel             <= '0';
485                         sel_in_mdr  <= '1';
486                         save_mdr        <= '0';
487                         save_c      <= '0';
488                         next_state  <= '0';
489                         escape      <= '0';
490
491 ----------State 110 110 Indexed MOV-Destination----------
492                     elsif(decode = "110" and decode_1 = "110" and A="01" and Op = "0000" and As = "00
                            ") then
493                         sa          <= "00";
494                         sb          <= "00";
495                         S                   <= "0000";
496                         sel_out_mdr <= "00";
497                         SDV         <= '0';
498                         EDV         <= '0';
499                         save_ir     <= '0';
500                         RW          <= '0';
501                         ME          <= '0';
502                         save        <= '0';
503                         en_out_a    <= '0';
504                         en_out_b    <= '0';
505                         save_sr         <= '0';
506                         en_SR       <= '0';
507                         rst_sr      <= '0';
508                         pc_out          <= '0';
509                         save_mar        <= '1';
510                         en_out_mar  <= '0';
511                         sel             <= '0';
512                         sel_in_mdr  <= '0';
513                         save_mdr        <= '0';
514                         save_c      <= '0';
515                         next_state  <= '1';
516                         escape      <= '0';
517
518 ----------State 110 010 Absolute----------
519                     elsif(decode = "110" and decode_1 = "011" and A="10") then
520                         sa          <= "00";
521                         sb          <= "00";
522                         S                   <= "0000";
523                         sel_out_mdr <= "01";
524                         SDV         <= '0';
525                         EDV         <= '0';
526                         save_ir     <= '0';
527                         RW          <= '0';
528                         ME          <= '0';
529                         save        <= '0';
530                         en_out_a    <= '0';
531                         en_out_b    <= '0';
532                         save_sr         <= '0';
533                         en_SR       <= '0';
534                         rst_sr      <= '0';
535                         pc_out          <= '0';
536                         save_mar        <= '0';
```

```
537                         en_out_mar  <= '0';
538                         sel             <= '1';
539                         sel_in_mdr  <= '0';
540                         save_mdr        <= '0';
541                         save_c      <= '0';
542                         next_state  <= '0';
543                         escape      <= '0';
544
545 ------------State 110 110 Absolute---Not MOV-------------
546                     elsif(decode = "110" and decode_1 = "110" and A="10" and Op /= "0000") then
547                          sa          <= "00";
548                         sb          <= "00";
549                         S                   <= "0000";
550                         sel_out_mdr <= "01";
551                         SDV         <= '0';
552                         EDV         <= '0';
553                         save_ir     <= '0';
554                         RW          <= '1';
555                         ME          <= '1';
556                         save        <= '0';
557                         en_out_a    <= '0';
558                         en_out_b    <= '0';
559                         save_sr         <= '0';
560                         en_SR       <= '0';
561                         rst_sr      <= '0';
562                         pc_out          <= '0';
563                         save_mar        <= '1';
564                         en_out_mar  <= '1';
565                         sel             <= '1';
566                         sel_in_mdr  <= '0';
567                         save_mdr        <= '0';
568                         save_c      <= '0';
569                         next_state  <= '0';
570                         escape      <= '0';
571
572 ------------State 110 110 Absolute Source---MOV-------------
573                     elsif(decode = "110" and decode_1 = "110" and A="10" and Op = "0000" and As = "10
                            ") then
574                         sa          <= "00";
575                         sb          <= "00";
576                         S                   <= "0000";
577                         sel_out_mdr <= "01";
578                         SDV         <= '0';
579                         EDV         <= '0';
580                         save_ir     <= '0';
581                         RW          <= '1';
582                         ME          <= '1';
583                         save        <= '0';
584                         en_out_a    <= '0';
585                         en_out_b    <= '0';
586                         save_sr         <= '0';
587                         en_SR       <= '0';
588                         rst_sr      <= '0';
589                         pc_out          <= '0';
590                         save_mar        <= '1';
591                         en_out_mar  <= '1';
```

```
592                             sel             <= '1';
593                             sel_in_mdr  <= '0';
594                             save_mdr        <= '0';
595                             save_c      <= '0';
596                             next_state  <= '0';
597                             escape      <= '0';
598
599     ----------State 110 110 Absolute Destination--MOV----------
600                         elsif(decode = "110" and decode_1 = "110" and A="10" and Op = "0000" and As = "00
                                ") then
601                             sa          <= "00";
602                             sb          <= "00";
603                             S                   <= "0000";
604                             sel_out_mdr <= "01";
605                             SDV         <= '0';
606                             EDV         <= '0';
607                             save_ir     <= '0';
608                             RW          <= '0';
609                             ME          <= '0';
610                             save        <= '0';
611                             en_out_a    <= '0';
612                             en_out_b    <= '0';
613                             save_sr         <= '0';
614                             en_SR       <= '0';
615                             rst_sr      <= '0';
616                             pc_out          <= '0';
617                             save_mar        <= '1';
618                             en_out_mar  <= '0';
619                             sel             <= '1';
620                             sel_in_mdr  <= '0';
621                             save_mdr        <= '0';
622                             save_c      <= '0';
623                             next_state  <= '1';
624                             escape      <= '0';
625
626     ----------State 100 110----------
627                         elsif(decode = "100" and decode_1 = "110") then
628                             sa          <= "00";
629                             sb          <= "00";
630                             S                   <= "0000";
631                             sel_out_mdr <= "00";
632                             SDV         <= '0';
633                             EDV         <= '0';
634                             save_ir     <= '0';
635                             RW          <= '1';
636                             ME          <= '1';
637                             save        <= '0';
638                             en_out_a    <= '0';
639                             en_out_b    <= '0';
640                             save_sr         <= '0';
641                             en_SR       <= '0';
642                             rst_sr      <= '0';
643                             pc_out          <= '0';
644                             save_mar        <= '0';
645                             en_out_mar  <= '1';
646                             sel             <= '0';
```

```
647                         sel_in_mdr  <= '1';
648                         save_mdr        <= '0';
649                         save_c      <= '0';
650                         next_state  <= '0';
651                         escape      <= '0';
652
653 ----------------------State 100 100----------------------------
654                     elsif(decode = "100" and decode_1 = "100") then
655                         sa          <= "00";
656                         sb          <= "00";
657                         S                   <= "0000";
658                         sel_out_mdr <= "00";
659                         SDV         <= '0';
660                         EDV         <= '0';
661                         save_ir     <= '0';
662                         RW          <= '1';
663                         ME          <= '1';
664                         save        <= '0';
665                         en_out_a    <= '0';
666                         en_out_b    <= '0';
667                         save_sr         <= '0';
668                         en_SR       <= '0';
669                         rst_sr      <= '0';
670                         pc_out          <= '0';
671                         save_mar        <= '0';
672                         en_out_mar  <= '1';
673                         sel             <= '0';
674                         sel_in_mdr  <= '1';
675                         save_mdr        <= '1';
676                         save_c      <= '0';
677                         next_state  <= '1';
678                         escape      <= '0';
679
680 ----------------------State 101 101----------------------------
681                     elsif(decode = "101" and decode_1 = "101") then
682                         sa          <= "00";
683                         sb          <= "00";
684                         S                   <= "0000";
685                         sel_out_mdr <= "00";
686                         SDV         <= '0';
687                         EDV         <= '0';
688                         save_ir     <= '0';
689                         RW          <= '0';
690                         ME          <= '0';
691                         save        <= '0';
692                         en_out_a    <= '0';
693                         en_out_b    <= '0';
694                         save_sr         <= '0';
695                         en_SR       <= '0';
696                         rst_sr      <= '0';
697                         pc_out          <= '0';
698                         save_mar        <= '0';
699                         en_out_mar  <= '0';
700                         sel             <= '0';
701                         sel_in_mdr  <= '0';
702                         save_mdr        <= '0';
```

```
703                         save_c      <= '0';
704                         next_state  <= '1';
705                         escape      <= '0';
706
707 ----------------------State 111 111----------------------------
708                     elsif(decode = "111" and decode_1 = "111") then
709                         sa          <= "00";
710                         sb          <= "00";
711                         S                   <= "0000";
712                         sel_out_mdr <= "00";
713                         SDV         <= '0';
714                         EDV         <= '0';
715                         save_ir     <= '0';
716                         RW          <= '0';
717                         ME          <= '0';
718                         save        <= '0';
719                         en_out_a    <= '0';
720                         en_out_b    <= '0';
721                         save_sr         <= '0';
722                         en_SR       <= '0';
723                         rst_sr      <= '0';
724                         pc_out          <= '0';
725                         save_mar        <= '0';
726                         en_out_mar  <= '0';
727                         sel             <= '0';
728                         sel_in_mdr  <= '0';
729                         save_mdr        <= '0';
730                         save_c      <= '0';
731                         next_state  <= '1';
732                         escape      <= '0';
733
734 ----------------------State 000 111----------------------------
735                     elsif(decode = "000" and decode_1 = "111") then
736                         sa          <= "00";
737                         sb          <= "00";
738                         S                   <= "0000";
739                         sel_out_mdr <= "00";
740                         SDV         <= '0';
741                         EDV         <= '0';
742                         save_ir     <= '0';
743                         RW          <= '0';
744                         ME          <= '0';
745                         save        <= '0';
746                         en_out_a    <= '0';
747                         en_out_b    <= '0';
748                         save_sr         <= '0';
749                         en_SR       <= '0';
750                         rst_sr      <= '0';
751                         pc_out          <= '0';
752                         save_mar        <= '0';
753                         en_out_mar  <= '0';
754                         sel             <= '0';
755                         sel_in_mdr  <= '0';
756                         save_mdr        <= '0';
757                         save_c      <= '0';
758                         next_state  <= '1';
```

```
759                             escape      <= '0';
760
761   ----------------------State 000 101----------------------------
762                     elsif(decode = "000" and decode_1 = "101") then
763                             sa          <= "00";
764                             sb          <= "00";
765                             S                   <= "0000";
766                             sel_out_mdr <= "00";
767                             SDV         <= '0';
768                             EDV         <= '0';
769                             save_ir     <= '0';
770                             RW          <= '0';
771                             ME          <= '0';
772                             save        <= '0';
773                             en_out_a    <= '0';
774                             en_out_b    <= '0';
775                             save_sr         <= '0';
776                             en_SR       <= '0';
777                             rst_sr      <= '0';
778                             pc_out          <= '0';
779                             save_mar        <= '0';
780                             en_out_mar  <= '0';
781                             sel             <= '0';
782                             sel_in_mdr  <= '0';
783                             save_mdr        <= '0';
784                             save_c      <= '0';
785                             next_state  <= '1';
786                             escape      <= '0';
787
788   ----------------------State 000 001----------------------------
789                     elsif(decode = "000" and decode_1 = "001") then
790                             sa          <= "00";
791                             sb          <= "00";
792                             S                   <= "0000";
793                             sel_out_mdr <= "00";
794                             SDV         <= '0';
795                             EDV         <= '0';
796                             save_ir     <= '0';
797                             RW          <= '0';
798                             ME          <= '0';
799                             save        <= '0';
800                             en_out_a    <= '0';
801                             en_out_b    <= '0';
802                             save_sr         <= '0';
803                             en_SR       <= '0';
804                             rst_sr      <= '0';
805                             pc_out          <= '0';
806                             save_mar        <= '0';
807                             en_out_mar  <= '0';
808                             sel             <= '0';
809                             sel_in_mdr  <= '0';
810                             save_mdr        <= '0';
811                             save_c      <= '0';
812                             next_state  <= '1';
813                             escape      <= '0';
814
```

```
815 ----------State 000 011----------
816                 elsif(decode = "000" and decode_1 = "011") then
817                         sa          <= "00";
818                         sb          <= "00";
819                         S                   <= "0000";
820                         sel_out_mdr <= "00";
821                         SDV         <= '0';
822                         EDV         <= '0';
823                         save_ir     <= '0';
824                         RW          <= '0';
825                         ME          <= '0';
826                         save        <= '0';
827                         en_out_a    <= '0';
828                         en_out_b    <= '0';
829                         save_sr     <= '0';
830                         en_SR       <= '0';
831                         rst_sr      <= '0';
832                         pc_out          <= '0';
833                         save_mar        <= '0';
834                         en_out_mar  <= '0';
835                         sel             <= '0';
836                         sel_in_mdr  <= '0';
837                         save_mdr        <= '0';
838                         save_c      <= '0';
839                         next_state  <= '1';
840                         escape      <= '0';
841
842 ----------State 000 010----------
843                 elsif(decode = "000" and decode_1 = "010") then
844                         sa          <= "00";
845                         sb          <= "00";
846                         S                 <= "0000";
847                         sel_out_mdr <= "00";
848                         SDV         <= '0';
849                         EDV         <= '0';
850                         save_ir     <= '0';
851                         RW          <= '0';
852                         ME          <= '0';
853                         save        <= '0';
854                         en_out_a    <= '0';
855                         en_out_b    <= '0';
856                         save_sr         <= '0';
857                         en_SR       <= '0';
858                         rst_sr      <= '0';
859                         pc_out          <= '0';
860                         save_mar        <= '0';
861                         en_out_mar  <= '0';
862                         sel             <= '0';
863                         sel_in_mdr  <= '0';
864                         save_mdr        <= '0';
865                         save_c      <= '0';
866                         next_state  <= '1';
867                         escape      <= '0';
868
869 ----------State 000 110----------
870                 elsif(decode = "000" and decode_1 = "110") then
```

```
871                         sa          <= "00";
872                         sb          <= "00";
873                         S                   <= "0000";
874                         sel_out_mdr <= "00";
875                         SDV         <= '0';
876                         EDV         <= '0';
877                         save_ir     <= '0';
878                         RW          <= '0';
879                         ME          <= '0';
880                         save        <= '0';
881                         en_out_a    <= '0';
882                         en_out_b    <= '0';
883                         save_sr         <= '0';
884                         en_SR       <= '0';
885                         rst_sr          <= '0';
886                         pc_out          <= '0';
887                         save_mar        <= '0';
888                         en_out_mar  <= '0';
889                         sel             <= '0';
890                         sel_in_mdr  <= '0';
891                         save_mdr        <= '0';
892                         save_c      <= '0';
893                         next_state  <= '1';
894                         escape      <= '0';
895
896 ----------State 000 100----------
897                     elsif(decode = "000" and decode_1 = "100") then
898                         sa          <= "00";
899                         sb          <= "00";
900                         S                   <= "0000";
901                         sel_out_mdr <= "00";
902                         SDV         <= '0';
903                         EDV         <= '0';
904                         save_ir     <= '0';
905                         RW          <= '0';
906                         ME          <= '0';
907                         save        <= '0';
908                         en_out_a    <= '0';
909                         en_out_b    <= '0';
910                         save_sr     <= '0';
911                         en_SR       <= '0';
912                         rst_sr      <= '0';
913                         pc_out          <= '0';
914                         save_mar        <= '0';
915                         en_out_mar  <= '0';
916                         sel             <= '0';
917                         sel_in_mdr  <= '0';
918                         save_mdr        <= '0';
919                         save_c      <= '0';
920                         next_state  <= '1';
921                         escape      <= '0';
922
923 ----------State ----------
924                     else
925                         sa          <= "00";
926                         sb          <= "00";
```

```
927                             S                   <= "0000";
928                             sel_out_mdr <= "00";
929                             SDV         <= '0';
930                             EDV         <= '0';
931                             save_ir     <= '0';
932                             RW          <= '0';
933                             ME          <= '0';
934                             save        <= '0';
935                             en_out_a    <= '0';
936                             en_out_b    <= '0';
937                             save_sr     <= '0';
938                             en_SR       <= '0';
939                             rst_sr      <= '0';
940                             pc_out          <= '0';
941                             save_mar        <= '0';
942                             en_out_mar  <= '0';
943                             sel             <= '0';
944                             sel_in_mdr  <= '0';
945                             save_mdr        <= '0';
946                             save_c      <= '0';
947                             next_state  <= '1';
948                             escape      <= '0';
949
950
951                     end if;
952 ------------------------------------------------------------
953             end process;
954             reset_i <= '0';
955 end ds_1;
```

B.2.14 Decode State Machine

```
1  --
       ------------------------------------------------------------------------------------------------------------------------

2  --
3  -- File         : decode_sm.vhd
4  --
5  --
       ------------------------------------------------------------------------------------------------------------------------

6  --
7  -- Description :         Decode State Machine
8  --
9  --
       ------------------------------------------------------------------------------------------------------------------------

10
11 library IEEE;
12 use IEEE.STD_LOGIC_1164.all;
13
14 entity dsm is
15             port(
16                     D : in STD_LOGIC;
17                     A : in STD_LOGIC_VECTOR(1 downto 0);
18                     MA : in STD_LOGIC;
```

```
19                  rst : in STD_LOGIC;
20                  clk : in STD_LOGIC;
21                  State : out STD_LOGIC_VECTOR(2 downto 0);
22                  Nstate : out STD_LOGIC_VECTOR(2 downto 0)
23              );
24 end dsm;
25
26 --}} End of automatically maintained section
27
28 architecture dsm_1 of dsm is
29
30         signal state_temp : std_logic_vector(2 downto 0);
31
32 begin
33
34          process (clk, rst)
35         begin
36                 if (rst = '1') then
37                         state_temp <= "000";
38                 elsif (clk'EVENT AND clk='1') then
39                         IF(state_temp="000" and D = '0') then
40                                 state_temp <= "000";
41                         elsif (state_temp="000" and D = '1') then
42                                 state_temp <= "001";
43                         elsif (state_temp="001" and D = '0' ) then
44                                 state_temp <= "000";
45                         elsif (state_temp="001" and D = '1' and MA = '0' ) then
46                                 state_temp <= "001";
47                         elsif (state_temp="001" and D = '1' and MA = '1' ) then
48                                 state_temp <= "011";
49                         elsif (state_temp="011" and D = '0') then
50                                 state_temp <= "000";
51                         elsif (state_temp="011" and D = '1' and A = "11" ) then
52                                 state_temp <= "000";
53                         elsif (state_temp="011" and D = '1' and A = "01" ) then
54                                 state_temp <= "010";
55                         elsif (state_temp="011" and D = '1' and A = "10" ) then
56                                 state_temp <= "110";
57                         elsif (state_temp="010" and D = '0') then
58                                 state_temp <= "000";
59                         elsif (state_temp="010" and D = '1' AND A/= "01") then
60                                 state_temp <= "000";
61                         elsif (state_temp="010" and D = '1' and A = "01" ) then
62                                 state_temp <= "110";
63                         elsif (state_temp="110" and D = '0') then
64                                 state_temp <= "000";
65                         elsif (state_temp="110" and D = '1'and not (A /= "10" OR A /= "01")) then
66                                 state_temp <= "000";
67                         elsif (state_temp="110" and D = '1' and MA = '0' ) then
68                                 state_temp <= "110";
69                         elsif (state_temp="110" and D = '1' and MA = '1' ) then
70                                 state_temp <= "100";
71                         elsif (state_temp="100" and (D = '0' OR D = '1')) then
72                                 state_temp <= "000";
73                         elsif (state_temp="101" and (D = '0' OR D = '1')) then
74                                 state_temp <= "000";
```

```
75                          elsif (state_temp="111" and (D = '0' OR D = '1')) then
76                                  state_temp <= "000";
77                          END IF;
78                  END IF;
79          end process;
80
81          process (clk, rst)
82          begin
83                  if (rst = '1') then
84                          Nstate <= "000";
85                  elsif (clk'EVENT AND clk='0') then
86                          Nstate <= state_temp;
87                  end if;
88
89          end process;
90                  state <= state_temp;
91
92  end dsm_1;
```

B.2.15 Execute Two Operands State Machine

```
1   --
        ------------------------------------------------------------------------------------------

2   --
3   -- File         : execute_d_sm.vhd
4   --
5   --
        ------------------------------------------------------------------------------------------

6   --
7   -- Description : Two Operands Execute State Machine
8   --
9   --
        ------------------------------------------------------------------------------------------

10
11  library IEEE;
12  use IEEE.STD_LOGIC_1164.all;
13
14  entity edsm is
15           port(
16                   clk : in STD_LOGIC;
17                   rst : in STD_LOGIC;
18                   E : in STD_LOGIC;
19                   MA : in STD_LOGIC;
20                   state : out STD_LOGIC_VECTOR(1 downto 0);
21                   nstate : out STD_LOGIC_VECTOR(1 downto 0)
22               );
23  end edsm;
24
25  architecture edsm_1 of edsm is
26          signal state_temp : std_logic_vector(1 downto 0);
27  begin
28
29          process (clk, rst)
```

```
30          begin
31                  if (rst = '1') then
32                          state_temp <= "00";
33                  elsif (clk'EVENT AND clk='1') THEN
34                          IF(state_temp="00" and E = '0' and MA = '0' ) then
35                                  state_temp <= "00";
36                          elsif (state_temp="00" and E = '0' and MA = '1' ) then
37                                  state_temp <= "00";
38                          elsif (state_temp="00" and E = '1' and MA = '0' ) then
39                                  state_temp <= "01";
40                          elsif (state_temp="00" and E = '1' and MA = '1' ) then
41                                  state_temp <= "01";
42                          elsif (state_temp="01" and E = '0' and MA = '0' ) then
43                                  state_temp <= "00";
44                          elsif (state_temp="01" and E = '0' and MA = '1' ) then
45                                  state_temp <= "00";
46                          elsif (state_temp="01" and E = '1' and MA = '0' ) then
47                                  state_temp <= "11";
48                          elsif (state_temp="01" and E = '1' and MA = '1' ) then
49                                  state_temp <= "11";
50                          elsif (state_temp="11" and E = '0' and MA = '0' ) then
51                                  state_temp <= "00";
52                          elsif (state_temp="11" and E = '0' and MA = '1' ) then
53                                  state_temp <= "00";
54                          elsif (state_temp="11" and E = '1' and MA = '0' ) then
55                                  state_temp <= "11";
56                          elsif (state_temp="11" and E = '1'and MA = '1' ) then
57                                  state_temp <= "00";
58                          elsif (state_temp="10" and E = '0'and MA = '0' ) then
59                                  state_temp <= "00";
60                          elsif (state_temp="10" and E = '0' and MA = '1' ) then
61                                  state_temp <= "00";
62                          elsif (state_temp="10" and E = '1' and MA = '0' ) then
63                                  state_temp <= "00";
64                          elsif (state_temp="10" and E = '1'and MA = '1' ) then
65                                  state_temp <= "00";
66                          END IF;
67                  END IF;
68          end process;
69
70          process (clk, rst)
71          begin
72                  if (rst = '1') then
73                          nstate <= "00";
74                  elsif (clk'EVENT AND clk='0') then
75                          nstate <= state_temp;
76                  end if;
77
78          end process;
79                  state <= state_temp;
80
81  end edsm_1;
```

B.2.16 Execute Decode

```
1  --
       ------------------------------------------------------------------------------------------------
2  --
3  -- File         : Execute_decoder.vhd
4  --
5  --
       ------------------------------------------------------------------------------------------------
6  --
7  -- Description : This file decide which istruction is in the IR and if the instruction is jump
8  --               decide if will ocure the jump
9  --
10 --
       ------------------------------------------------------------------------------------------------

11
12 library IEEE;
13 use IEEE.STD_LOGIC_1164.all;
14
15 entity eid is
16          port(
17                  Op : in STD_LOGIC_VECTOR(3 downto 0);
18                  sr : in STD_LOGIC_VECTOR(3 downto 0);
19                  E  : in std_logic;
20                  E_int : in std_logic;
21                  reset : in std_logic;
22                  P     : out std_logic;
23                  R     : out std_logic;
24                  J     : out std_logic;
25                  enable_d :      out std_logic;
26                  enable_p :      out std_logic;
27                  enable_interrupt : out std_logic
28              );
29 end eid;
30
31 architecture ed of eid is
32         signal P_1, R_1, J_1, enable_d_1 : std_logic;
33 begin
34
35         P_1 <= '1' when Op = "1010" else '0';
36         R_1 <= '1' when Op = "1001" else '0';
37      J_1 <= '1' when (Op = "1110" and sr(0)='1') or (Op = "0111" and sr(1)='1') or (Op = "1111"
            and sr(2)='1') or (Op = "1101" and (sr(3) xor sr(2))='1') else '0';
38
39          enable_d_1 <= '1' when (E ='1' and (Op="0000" or Op="0001" or Op="0010" or Op="0011" or
                Op="0100" or Op="0101" or Op="0110" or Op="1000" or Op="1011" or Op = "0111" or Op =
                 "1011" or Op = "1101" or Op = "1110")) and not (E_int = '1' and reset = '1') else
                '0';
40
41          enable_p <= '1' when (E = '1' and (Op = "1001" or Op = "1010" or Op = "1100")) or (E_int
                = '1' and reset = '1') else '0';
42
43          enable_interrupt <=     '1' when E_int = '1' and reset = '0' else '0';
44
45          P <= P_1;
```

```
46          R <= R_1;
47          J <= J_1;
48          enable_d <= enable_d_1;
49
50  end ed;
```

B.2.17 Execute Push/Pop State Machine

```
1   --
        ------------------------------------------------------------------------------------------

2   --
3   -- File         : execute_pupo_sm.vhd
4   --
5   --
        ------------------------------------------------------------------------------------------

6   --
7   -- Description : Execute push, pop state machine
8   --
9   --
        ------------------------------------------------------------------------------------------

10
11  library IEEE;
12  use IEEE.STD_LOGIC_1164.all;
13
14  entity epuposm is
15          port(
16                  E : in STD_LOGIC;
17                  MA : in STD_LOGIC;
18                  P : in STD_LOGIC;                -- PUSH/POP
19                  R : in STD_LOGIC;           -- RETI
20                  clk : in STD_LOGIC;
21                  rst : in STD_LOGIC;
22                  T1 : out std_logic;
23                  state : out STD_LOGIC_VECTOR(2 downto 0);
24                  nstate : out STD_LOGIC_VECTOR(2 downto 0)
25              );
26  end epuposm;
27
28  architecture epuposm_1 of epuposm is
29          signal state_temp, nstate_temp : std_logic_vector(2 downto 0);
30          signal T, tempin, tempout, clk_1 : std_logic;
31  begin
32
33          tempin <= tempout xor R;
34          T <= tempout;
35          T1 <= T;
36          clk_1 <= '1' When state_temp = "010" and nstate_temp = "010" else '0';
37
38          process (clk_1, rst)
39          begin
40                  if (rst = '1') then
41                          tempout <= '0';
42                  elsif (clk_1'EVENT AND clk_1 = '1') then
```

```
43                      tempout <= tempin;
44              end if;
45      end process;
46
47      process (clk, rst)
48      begin
49              if (rst = '1') then
50                      state_temp <= "000";
51              elsif (clk'EVENT AND clk='1') THEN
52                      IF(state_temp="000" and E = '0') then
53                              state_temp <= "000";
54                      elsif (state_temp="000" and E = '1') then
55                              state_temp <= "001";
56                      elsif (state_temp="001" and E = '1' and MA = '0' AND P = '0') then
57                              state_temp <= "001";
58                      elsif (state_temp="001" and E = '0') then
59                              state_temp <= "000";
60                      elsif (state_temp="001" and E = '1' and MA = '1' AND P = '0') then
61                              state_temp <= "011";
62                      elsif (state_temp="001" and E = '1' and P = '1') then
63                              state_temp <= "011";
64                      elsif (state_temp="011" and E = '0') then
65                              state_temp <= "000";
66                      elsif (state_temp="011" and E = '1') then
67                              state_temp <= "010";
68                      elsif (state_temp="010" and E = '0') then
69                              state_temp <= "000";
70                      elsif (state_temp="010" and E = '1') then
71                              state_temp <= "110";
72                      elsif (state_temp="110" and E = '1' and MA = '0' AND P = '1') then
73                              state_temp <= "110";
74                      elsif (state_temp="110" and E = '1' AND P = '0' and T = '1') then
75                              state_temp <= "001";
76                      elsif (state_temp="110" and E = '1' and MA = '1' AND P = '1') then
77                              state_temp <= "000";
78                      elsif (state_temp="110" and E = '1' AND P = '0' and T = '0') then
79                              state_temp <= "000";
80                      elsif (state_temp="110" and E = '0') then
81                              state_temp <= "000";
82                      elsif (state_temp="100") then
83                              state_temp <= "000";
84                      elsif (state_temp="101") then
85                              state_temp <= "000";
86                      elsif (state_temp="111") then
87                              state_temp <= "000";
88                      END IF;
89              END IF;
90      end process;
91
92      process (clk, rst)
93      begin
94              if (rst = '1') then
95                      nstate_temp <= "000";
96              elsif (clk'EVENT AND clk='0') then
97                      nstate_temp <= state_temp;
98              end if;
```

```
99
100         end process;
101                 state <= state_temp;
102                 nstate<= nstate_temp;
103 end epuposm_1;
```

B.2.18 Executer Signals

```
1  --
       ------------------------------------------------------------------------------------------------
2  --
3  -- File         : execute_two_j_signals.vhd
4  --
5  --
       ------------------------------------------------------------------------------------------------
6  --
7  -- Description : this file generate the signals of the two operand instructions and the jumps
8  --
9  --
       ------------------------------------------------------------------------------------------------
10
11 library IEEE;
12 use IEEE.STD_LOGIC_1164.all;
13
14 entity etjs is
15         port(
16                 execute_d   : in STD_LOGIC_VECTOR(1 downto 0);
17                 execute_d_1 : in STD_LOGIC_VECTOR(1 downto 0);
18                 execute_p   : in STD_LOGIC_VECTOR(2 downto 0);
19                 execute_p_1 : in STD_LOGIC_VECTOR(2 downto 0);
20                 as       : in std_logic_vector(1 downto 0);
21                 ad       :    in std_logic;
22                 Op       : in std_logic_vector(3 downto 0);
23                 R        : in std_logic;
24                 P        : in std_logic;
25                 J        : in std_logic;
26                 T1       : in std_logic;
27                 reset    : in std_logic;
28                 sa       : out std_logic_vector(1 downto 0);
29                 sb       : out std_logic_vector(1 downto 0);
30                 SDV      : out std_logic;
31                 EDV      : out std_logic;
32                 save_ir  : out std_logic;
33                 RW       : out std_logic;
34                 ME          : out std_logic;
35                 save     : out std_logic;
36                 en_out_a : out std_logic;
37                 en_out_b : out std_logic;
38                 save_sr  : out std_logic;
39                 en_SR    : out std_logic;
40                 rst_sr   : out std_logic;
41                 pc_out   : out std_logic;
42                 reset_i  : out std_logic;
```

```
43                  save_mar : out std_logic;
44                  en_out_mar      : out std_logic;
45                  sel       : out std_logic;
46                  sel_in_mdr      : out std_logic;
47                  sel_out_mdr : out std_logic_vector(1 downto 0);
48                  save_mdr : out std_logic;
49                  save_c    : out std_logic;
50           S         : out std_logic_vector(3 downto 0);
51                  next_state      : out std_logic;
52                  escape : out std_logic
53              );
54  end etjs;
55
56  architecture etjs of etjs is
57          signal sa_d, sb_d, sel_out_mdr_d : std_logic_vector(1 downto 0);
58          signal SDV_d, EDV_d, save_ir_d, RW_d, ME_d, save_d, en_out_a_d, en_out_b_d, save_sr_d,
                en_SR_d, rst_sr_d, pc_out_d,  reset_i_d, save_mar_d, en_out_mar_d, sel_d,
                sel_in_mdr_d, save_mdr_d, save_c_d, next_state_d, escape_d : std_logic;
59          signal S_d : std_logic_vector(3 downto 0);
60          signal sa_p, sb_p, sel_out_mdr_p : std_logic_vector(1 downto 0);
61          signal SDV_p, EDV_p, save_ir_p, RW_p, ME_p, save_p, en_out_a_p, en_out_b_p, save_sr_p,
                en_SR_p, rst_sr_p, pc_out_p,  reset_i_p, save_mar_p, en_out_mar_p, sel_p,
                sel_in_mdr_p, save_mdr_p, save_c_p, next_state_p, escape_p : std_logic;
62          signal S_p : std_logic_vector(3 downto 0);
63
64  begin
65
66          process(execute_d, execute_d_1, Op, J, as, ad)
67          begin
68
69  ------------------------------------------------------------
70  --          Two Operands and Jump Instructions            --
71  ------------------------------------------------------------
72
73  ----------------------State 00------------------------------
74
75                  if(execute_d = "00" and execute_d_1 = "00")     then
76                          sa_d            <= "00";
77                          sb_d            <= "00";
78                          S_d                <= "0000";
79                          sel_out_mdr_d   <= "00";
80                          SDV_d           <= '0';
81                          EDV_d           <= '0';
82                          save_ir_d       <= '0';
83                          RW_d            <= '0';
84                          ME_d            <= '0';
85                          save_d              <= '0';
86                          en_out_a_d      <= '0';
87                          en_out_b_d      <= '0';
88                          save_sr_d           <= '0';
89                          en_SR_d         <= '0';
90                          rst_sr_d            <= '0';
91                          pc_out_d            <= '0';
92                          reset_i_d       <= '0';
93                          save_mar_d      <= '0';
94                          en_out_mar_d    <= '0';
```

```
95                              sel_d               <= '0';
96                              sel_in_mdr_d    <= '0';
97                              save_mdr_d      <= '0';
98                              save_c_d            <= '0';
99                              next_state_d    <= '0';
100                             escape_d        <= '0';
101
102 ----------State 01 00----------
103                     elsif(execute_d = "01" and execute_d_1 = "00") then
104
105 -- Alu operation selection
106                             if(Op = "0000") then -- MOV
107                                     S_d         <= "0000";
108                                     en_SR_d         <= '0';
109                             elsif(Op = "0001") then -- BIS
110                                     S_d         <= "0010";
111                                     en_SR_d         <= '0';
112                             elsif(Op = "0010") then -- BIC
113                                     S_d         <= "1000";
114                                     en_SR_d         <= '0';
115                             elsif(Op = "0011") then -- SUBC
116                                     S_d         <= "0110";
117                                     en_SR_d         <= '1';
118                             elsif(Op = "0100") then -- BIT
119                                     S_d         <= "0100";
120                                     en_SR_d         <= '1';
121                             elsif(Op = "0101") then -- CMP
122                                     S_d         <= "1010";
123                                     en_SR_d         <= '1';
124                             elsif(Op = "0110") then -- AND
125                                     S_d         <= "0100";
126                                     en_SR_d         <= '1';
127                             elsif(Op = "1000") then -- ADDC
128                                     S_d         <= "0101";
129                                     en_SR_d         <= '1';
130                             elsif(Op = "1111") then -- XOR
131                                     S_d         <= "0011";
132                                     en_SR_d         <= '1';
133                             elsif((Op = "0111" or Op = "1111" or Op = "1101" or Op = "1110") and J =
                                    '1') then -- Jumps taken
134                                     S_d         <= "1100";
135                                     en_SR_d         <= '0';
136                             elsif((Op = "0111" or Op = "1111" or Op = "1101" or Op = "1110") and J =
                                    '0') then -- Jumps not taken
137                                     S_d         <= "0000";
138                                     en_SR_d         <= '0';
139                             else
140                                     S_d         <= "0000";
141                                     en_SR_d         <= '0';
142                             end if;
143 ----------
144                             save_ir_d       <= '0';
145                             RW_d            <= '0';
146                             ME_d            <= '0';
147                             save_d              <= '0';
148 ----addressing opperands-----
```

```
149                 if(Op = "0000" and As = "00") then -- MOV -- register
150                         sa_d            <= "10";
151                         sb_d            <= "00";
152                         en_out_a_d      <= '1';
153                         en_out_b_d      <= '0';
154                         SDV_d           <= '0';
155                         EDV_d           <= '0';
156                         sel_out_mdr_d   <= "00";
157                 elsif(Op = "0000" and As/="00") then  --MOV --register
158                         sa_d            <= "00";
159                         sb_d            <= "00";
160                         en_out_a_d      <= '0';
161                         en_out_b_d      <= '0';
162                         SDV_d           <= '0';
163                         EDV_d           <= '0';
164                         sel_out_mdr_d   <= "01";
165             elsif(Op = "0111" or Op = "1111" or Op = "1101" or Op = "1110") then -- Jumps
166                         sa_d            <= "00";
167                         sb_d            <= "00";
168                         en_out_a_d      <= '0';
169                         en_out_b_d      <= '1';
170                         SDV_d           <= '0';
171                         EDV_d           <= '1';
172                         sel_out_mdr_d   <= "00";
173                 elsif((Op="0001" or Op="0010" or Op="0011" or Op="0100" or Op="0101" or
                        Op="0110" or Op="1000" or Op="1011") and (As="00" and Ad='0')) then
174                         sa_d            <= "10";
175                         sb_d            <= "10";
176                         en_out_a_d      <= '1';
177                         en_out_b_d      <= '1';
178                         SDV_d           <= '0';
179                         EDV_d           <= '0';
180                         sel_out_mdr_d   <= "00";
181                 elsif((Op="0001" or Op="0010" or Op="0011" or Op="0100" or Op="0101" or
                        Op="0110" or Op="1000" or Op="1011") and (As/="00" and Ad='0')) then
182                         sa_d            <= "00";
183                         sb_d            <= "10";
184                         en_out_a_d      <= '0';
185                         en_out_b_d      <= '1';
186                         SDV_d           <= '0';
187                         EDV_d           <= '0';
188                         sel_out_mdr_d   <= "01";
189                 elsif((Op="0001" or Op="0010" or Op="0011" or Op="0100" or Op="0101" or
                        Op="0110" or Op="1000" or Op="1011") and (As="00" and Ad='1')) then
190                         sa_d            <= "10";
191                         sb_d            <= "00";
192                         en_out_a_d      <= '1';
193                         en_out_b_d      <= '0';
194                         SDV_d           <= '0';
195                         EDV_d           <= '0';
196                         sel_out_mdr_d   <= "10";
197                 else
198                         sa_d            <= "00";
199                         sb_d            <= "00";
200                         en_out_a_d      <= '0';
201                         en_out_b_d      <= '0';
```

```
202                                 SDV_d           <= '0';
203                                 EDV_d           <= '0';
204                                 sel_out_mdr_d   <= "00";
205                             end if;
206 ------------------
207                             save_sr_d           <= '0';
208                             rst_sr_d            <= '0';
209                             pc_out_d            <= '0';
210                             reset_i_d       <= '0';
211                             save_mar_d      <= '0';
212                             en_out_mar_d      <= '0';
213                             sel_d               <= '0';
214                             sel_in_mdr_d    <= '0';
215                             save_mdr_d      <= '0';
216                             save_c_d            <= '0';
217                             next_state_d    <= '0';
218                             escape_d        <= '0';
219
220 --------------------State 01 01-----------------------------
221                     elsif(execute_d = "01" and execute_d_1 = "01") then
222                             sa_d            <= "10";
223                             sb_d            <= "10";
224 -- Alu operation selection
225                             if(Op = "0000") then -- MOV
226                                     S_d         <= "0000";
227                                     en_SR_d         <= '0';
228                                     save_sr_d           <= '0';
229                                     next_state_d    <= '0';
230                                     escape_d        <= '0';
231                             elsif(Op = "0001") then -- BIS
232                                     S_d         <= "0010";
233                                     en_SR_d         <= '0';
234                                     save_sr_d           <= '0';
235                                     next_state_d    <= '0';
236                                     escape_d        <= '0';
237                             elsif(Op = "0010") then -- BIC
238                                     S_d         <= "1000";
239                                     en_SR_d         <= '0';
240                                     save_sr_d           <= '0';
241                                     next_state_d    <= '0';
242                                     escape_d        <= '0';
243                             elsif(Op = "0011") then -- SUBC
244                                     S_d         <= "0110";
245                                     en_SR_d         <= '1';
246                                     save_sr_d           <= '1';
247                                     next_state_d    <= '0';
248                                     escape_d        <= '0';
249                             elsif(Op = "0100") then -- BIT
250                                     S_d         <= "0100";
251                                     en_SR_d         <= '1';
252                                     save_sr_d           <= '1';
253                                     next_state_d    <= '1';
254                                     escape_d        <= '1';
255                             elsif(Op = "0101") then -- CMP
256                                     S_d         <= "1010";
257                                     en_SR_d         <= '1';
```

```
258                                 save_sr_d          <= '1';
259                                 next_state_d   <= '1';
260                                 escape_d       <= '1';
261                             elsif(Op = "0110") then -- AND
262                                 S_d            <= "0100";
263                                 en_SR_d        <= '1';
264                                 save_sr_d          <= '1';
265                                 next_state_d   <= '0';
266                                 escape_d       <= '0';
267                             elsif(Op = "1000") then -- ADDC
268                                 S_d            <= "0101";
269                                 en_SR_d        <= '1';
270                                 save_sr_d          <= '1';
271                                 next_state_d   <= '0';
272                                 escape_d       <= '0';
273                             elsif(Op = "1011") then -- XOR
274                                 S_d            <= "0011";
275                                 en_SR_d        <= '1';
276                                 save_sr_d          <= '1';
277                                 next_state_d   <= '0';
278                                 escape_d       <= '0';
279                             elsif((Op = "0111" or Op = "1111" or Op = "1101" or Op = "1110") and J =
                                    '1') then -- Jumps taken
280                                 S_d            <= "1100";
281                                 en_SR_d        <= '0';
282                                 save_sr_d          <= '0';
283                                 next_state_d   <= '0';
284                                 escape_d       <= '0';
285                             elsif((Op = "0111" or Op = "1111" or Op = "1101" or Op = "1110") and J =
                                    '0') then -- Jumps not taken
286                                 S_d            <= "0000";
287                                 en_SR_d        <= '0';
288                                 save_sr_d          <= '0';
289                                 next_state_d   <= '1';
290                                 escape_d       <= '1';
291                             else
292                                 S_d            <= "0000";
293                                 en_SR_d        <= '0';
294                                 save_sr_d          <= '0';
295                                 next_state_d   <= '0';
296                                 escape_d       <= '0';
297                             end if;
298 ----------------------
299                             sel_out_mdr_d  <= "00";
300                             save_ir_d      <= '0';
301                             RW_d           <= '0';
302                             ME_d           <= '0';
303                             save_d             <= '0';
304
305 ----addressing opperands-----
306                             if(Op = "0000" and As = "00") then -- MOV -- register
307                                 sa_d           <= "10";
308                                 sb_d           <= "00";
309                                 en_out_a_d     <= '1';
310                                 en_out_b_d     <= '0';
311                                 SDV_d          <= '0';
```

```
312                             EDV_d           <= '0';
313                             sel_out_mdr_d   <= "00";
314                         elsif(Op = "0000" and As/="00") then  --MOV --register
315                             sa_d            <= "00";
316                             sb_d            <= "00";
317                             en_out_a_d      <= '0';
318                             en_out_b_d      <= '0';
319                             SDV_d           <= '0';
320                             EDV_d           <= '0';
321                             sel_out_mdr_d   <= "01";
322                     elsif(Op = "0111" or Op = "1111" or Op = "1101" or Op = "1110") then -- Jumps
323                             sa_d            <= "00";
324                             sb_d            <= "00";
325                             en_out_a_d      <= '0';
326                             en_out_b_d      <= '1';
327                             SDV_d           <= '0';
328                             EDV_d           <= '1';
329                             sel_out_mdr_d   <= "00";
330                         elsif((Op="0001" or Op="0010" or Op="0011" or Op="0100" or Op="0101" or
                                Op="0110" or Op="1000" or Op="1011") and (As="00" and Ad='0')) then
331                             sa_d            <= "10";
332                             sb_d            <= "10";
333                             en_out_a_d      <= '1';
334                             en_out_b_d      <= '1';
335                             SDV_d           <= '0';
336                             EDV_d           <= '0';
337                             sel_out_mdr_d   <= "00";
338                         elsif((Op="0001" or Op="0010" or Op="0011" or Op="0100" or Op="0101" or
                                Op="0110" or Op="1000" or Op="1011") and (As/="00" and Ad='0')) then
339                             sa_d            <= "00";
340                             sb_d            <= "10";
341                             en_out_a_d      <= '0';
342                             en_out_b_d      <= '1';
343                             SDV_d           <= '0';
344                             EDV_d           <= '0';
345                             sel_out_mdr_d   <= "01";
346                         elsif((Op="0001" or Op="0010" or Op="0011" or Op="0100" or Op="0101" or
                                Op="0110" or Op="1000" or Op="1011") and (As="00" and Ad='1')) then
347                             sa_d            <= "10";
348                             sb_d            <= "00";
349                             en_out_a_d      <= '1';
350                             en_out_b_d      <= '0';
351                             SDV_d           <= '0';
352                             EDV_d           <= '0';
353                             sel_out_mdr_d   <= "10";
354                         else
355                             sa_d            <= "00";
356                             sb_d            <= "00";
357                             en_out_a_d      <= '0';
358                             en_out_b_d      <= '0';
359                             SDV_d           <= '0';
360                             EDV_d           <= '0';
361                             sel_out_mdr_d   <= "00";
362                         end if;
363 ------------------
364
```

```
365                         rst_sr_d          <= '0';
366                         pc_out_d          <= '0';
367                         reset_i_d       <= '0';
368                         save_mar_d    <= '0';
369                         en_out_mar_d    <= '0';
370                         sel_d             <= '0';
371                         sel_in_mdr_d    <= '0';
372                         save_mdr_d        <= '0';
373                         save_c_d          <= '1';
374
375 ----------State 11 01----------
376                 elsif(execute_d = "11" and execute_d_1 = "01") then
377                         sa_d              <= "00";
378                         S_d                 <= "0000";
379                         sel_out_mdr_d   <= "00";
380                         SDV_d             <= '0';
381                         EDV_d             <= '0';
382                         save_ir_d         <= '0';
383                         RW_d              <= '0';
384                         ME_d              <= '0';
385                         save_d                  <= '0';
386                         en_out_a_d      <= '0';
387                         en_out_b_d      <= '0';
388                         save_sr_d         <= '0';
389                         en_SR_d         <= '0';
390                         rst_sr_d          <= '0';
391                         pc_out_d          <= '0';
392                         reset_i_d       <= '0';
393                         save_mar_d        <= '0';
394                         en_out_mar_d    <= '0';
395                         sel_d             <= '0';
396                         save_mdr_d        <= '0';
397                         save_c_d          <= '0';
398                         next_state_d    <= '0';
399                         escape_d        <= '0';
400                         if((Op = "0000" or Op="0001" or Op="0010" or Op="0011" or Op="0110" or
                                Op="1000" or Op="1011") and Ad = '0') then
401                             sb_d            <= "10";
402                             sel_in_mdr_d    <= '0';
403                         elsif((Op = "0000" or Op="0001" or Op="0010" or Op="0011" or Op="0110"
                                or Op="1000" or Op="1011") and Ad = '1') then
404                             sb_d            <= "00";
405                             sel_in_mdr_d    <= '0';
406                         elsif(Op = "0111" or Op = "1111" or Op = "1101" or Op = "1110") then
407                             sb_d            <= "00";
408                             sel_in_mdr_d    <= '0';
409                         else
410                             sb_d            <= "00";
411                             sel_in_mdr_d    <= '0';
412                         end if;
413
414 ----------State 11 11----------
415                 elsif(execute_d = "11" and execute_d_1 = "11") then
416                         sa_d              <= "00";
417                         S_d                 <= "0000";
418                         SDV_d             <= '0';
```

```
419                     EDV_d           <= '0';
420                     save_ir_d       <= '0';
421                     en_out_a_d      <= '0';
422                     en_out_b_d      <= '0';
423                     rst_sr_d          <= '0';
424                     sel_d             <= '0';
425                     save_sr_d         <= '0';
426                     en_SR_d         <= '0';
427                     pc_out_d          <= '0';
428                     reset_i_d       <= '0';
429                     save_mar_d        <= '0';
430                     sel_in_mdr_d    <= '0';
431                     save_c_d          <= '0';
432                     escape_d        <= '0';
433                     if((Op = "0000" or Op="0001" or Op="0010" or Op="0011" or Op="0110" or
                            Op="1000" or Op="1011") and Ad='0') then
434                         sb_d            <= "10";
435                         sel_out_mdr_d   <= "00";
436                         RW_d            <= '0';
437                         ME_d            <= '0';
438                         save_d              <= '1';
439                         en_out_mar_d    <= '0';
440                         save_mdr_d        <= '0';
441                         next_state_d    <= '1';
442
443                     elsif((Op = "0000" or Op="0001" or Op="0010" or Op="0011" or Op="0110"
                            or Op="1000" or Op="1011") and Ad='1') then
444                         sb_d            <= "00";
445                         sel_out_mdr_d   <= "11";
446                         RW_d            <= '0';
447                         ME_d            <= '1';
448                         save_d              <= '0';
449                         en_out_mar_d    <= '1';
450                         save_mdr_d        <= '1';
451                         next_state_d    <= '0';
452
453                     elsif(Op = "0111" or Op = "1111" or Op = "1101" or Op = "1110") then
454                         sb_d            <= "00";
455                         sel_out_mdr_d   <= "00";
456                         RW_d            <= '0';
457                         ME_d            <= '0';
458                         save_d              <= '1';
459                         en_out_mar_d    <= '0';
460                         save_mdr_d        <= '0';
461                         next_state_d    <= '1';
462                     else
463                         sb_d            <= "00";
464                         sel_out_mdr_d   <= "00";
465                         RW_d            <= '0';
466                         ME_d            <= '0';
467                         save_d              <= '0';
468                         en_out_mar_d    <= '0';
469                         save_mdr_d        <= '0';
470                         next_state_d    <= '0';
471                     end if;
472
```

```
473 ----------State 10 10----------
474                 elsif(execute_d = "10" and execute_d_1 = "10") then
475                         sa_d            <= "00";
476                         sb_d            <= "00";
477                         S_d               <= "0000";
478                         sel_out_mdr_d   <= "00";
479                         SDV_d           <= '0';
480                         EDV_d           <= '0';
481                         save_ir_d       <= '0';
482                         RW_d            <= '0';
483                         ME_d            <= '0';
484                         save_d                <= '0';
485                         en_out_a_d      <= '0';
486                         en_out_b_d      <= '0';
487                         save_sr_d         <= '0';
488                         en_SR_d         <= '0';
489                         rst_sr_d          <= '0';
490                         pc_out_d          <= '0';
491                         reset_i_d       <= '0';
492                         save_mar_d        <= '0';
493                         en_out_mar_d    <= '0';
494                         sel_d             <= '0';
495                         sel_in_mdr_d    <= '0';
496                         save_mdr_d        <= '0';
497                         save_c_d          <= '0';
498                         next_state_d    <= '1';
499                         escape_d        <= '0';
500
501 ----------State 00 01----------
502                 elsif(execute_d = "00" and execute_d_1 = "01") then
503                         sa_d            <= "00";
504                         sb_d            <= "00";
505                         S_d               <= "0000";
506                         sel_out_mdr_d   <= "00";
507                         SDV_d           <= '0';
508                         EDV_d           <= '0';
509                         save_ir_d       <= '0';
510                         RW_d            <= '0';
511                         ME_d            <= '0';
512                         save_d                <= '0';
513                         en_out_a_d      <= '0';
514                         en_out_b_d      <= '0';
515                         save_sr_d         <= '0';
516                         en_SR_d         <= '0';
517                         rst_sr_d          <= '0';
518                         pc_out_d          <= '0';
519                         reset_i_d       <= '0';
520                         save_mar_d        <= '0';
521                         en_out_mar_d    <= '0';
522                         sel_d             <= '0';
523                         sel_in_mdr_d    <= '0';
524                         save_mdr_d        <= '0';
525                         save_c_d          <= '0';
526                         next_state_d    <= '1';
527                         escape_d        <= '0';
528
```

```
529 ----------State 00 11----------
530                 elsif(execute_d = "00" and execute_d_1 = "11") then
531                         sa_d            <= "00";
532                         sb_d            <= "00";
533                         S_d               <= "0000";
534                         sel_out_mdr_d   <= "00";
535                         SDV_d           <= '0';
536                         EDV_d           <= '0';
537                         save_ir_d       <= '0';
538                         RW_d            <= '0';
539                         ME_d            <= '0';
540                         save_d                <= '0';
541                         en_out_a_d      <= '0';
542                         en_out_b_d      <= '0';
543                         save_sr_d         <= '0';
544                         en_SR_d         <= '0';
545                         rst_sr_d          <= '0';
546                         pc_out_d          <= '0';
547                         reset_i_d       <= '0';
548                         save_mar_d        <= '0';
549                         en_out_mar_d    <= '0';
550                         sel_d             <= '0';
551                         sel_in_mdr_d    <= '0';
552                         save_mdr_d        <= '0';
553                         save_c_d          <= '0';
554                         next_state_d    <= '1';
555                         escape_d        <= '0';
556
557 ----------State 00 10----------
558                 elsif(execute_d = "00" and execute_d_1 = "10") then
559                         sa_d            <= "00";
560                         sb_d            <= "00";
561                         S_d               <= "0000";
562                         sel_out_mdr_d   <= "00";
563                         SDV_d           <= '0';
564                         EDV_d           <= '0';
565                         save_ir_d       <= '0';
566                         RW_d            <= '0';
567                         ME_d            <= '0';
568                         save_d                <= '0';
569                         en_out_a_d      <= '0';
570                         en_out_b_d      <= '0';
571                         save_sr_d         <= '0';
572                         en_SR_d         <= '0';
573                         rst_sr_d          <= '0';
574                         pc_out_d          <= '0';
575                         reset_i_d       <= '0';
576                         save_mar_d        <= '0';
577                         en_out_mar_d    <= '0';
578                         sel_d             <= '0';
579                         sel_in_mdr_d    <= '0';
580                         save_mdr_d        <= '0';
581                         save_c_d          <= '0';
582                         next_state_d    <= '1';
583                         escape_d        <= '0';
584
```

```
585 ----------State ----------
586                 else
587                         sa_d            <= "00";
588                         sb_d            <= "00";
589                         S_d               <= "0000";
590                         sel_out_mdr_d   <= "00";
591                         SDV_d           <= '0';
592                         EDV_d           <= '0';
593                         save_ir_d       <= '0';
594                         RW_d            <= '0';
595                         ME_d            <= '0';
596                         save_d                <= '0';
597                         en_out_a_d      <= '0';
598                         en_out_b_d      <= '0';
599                         save_sr_d         <= '0';
600                         en_SR_d         <= '0';
601                         rst_sr_d          <= '0';
602                         pc_out_d          <= '0';
603                         reset_i_d       <= '0';
604                         save_mar_d        <= '0';
605                         en_out_mar_d    <= '0';
606                         sel_d             <= '0';
607                         sel_in_mdr_d    <= '0';
608                         save_mdr_d        <= '0';
609                         save_c_d          <= '0';
610                         next_state_d    <= '1';
611                         escape_d        <= '0';
612
613                 end if;
614                 end process;
615
616 ------------------------------------------------------------
617 --          push, pop, reti, reset signals                --
618 ------------------------------------------------------------
619
620                 process(execute_p, execute_p_1, P, R, T1, reset)
621                 begin
622
623 ----------State 000----------
624                 if(execute_p = "000" and execute_p_1 = "000") then
625                         sa_p            <= "00";
626                         sb_p            <= "00";
627                         S_p                   <= "0000";
628                         sel_out_mdr_p <= "00";
629                         SDV_p           <= '0';
630                         EDV_p           <= '0';
631                         save_ir_p       <= '0';
632                         RW_p            <= '0';
633                         ME_p            <= '0';
634                         save_p       <= '0';
635                         en_out_a_p      <= '0';
636                         en_out_b_p      <= '0';
637                         save_sr_p         <= '0';
638                         en_SR_p         <= '0';
639                         rst_sr_p              <= '0';
640                         pc_out_p          <= '0';
```

```
641                         reset_i_p       <= '0';
642                         save_mar_p      <= '0';
643                         en_out_mar_p    <= '0';
644                         sel_p           <= '0';
645                         sel_in_mdr_p    <= '0';
646                         save_mdr_p      <= '0';
647                         save_c_p        <= '0';
648                         next_state_p    <= '0';
649                         escape_p        <= '0';
650
651 ----------State 001 000----------
652                 elsif(execute_p = "001" and execute_p_1 = "000") then
653                         sa_p            <= "01";
654                         sb_p            <= "00";
655                         sel_out_mdr_p   <= "00";
656                         save_ir_p       <= '0';
657                         RW_p            <= '0';
658                         ME_p            <= '0';
659                         save_p      <= '0';
660                         en_out_b_p      <= '0';
661                         save_sr_p       <= '0';
662                         en_SR_p         <= '0';
663                         rst_sr_p        <= '0';
664                         pc_out_p        <= '0';
665                         reset_i_p       <= '0';
666                         save_mar_p      <= '0';
667                         en_out_mar_p    <= '0';
668                         sel_in_mdr_p    <= '0';
669                         save_mdr_p      <= '0';
670                         save_c_p        <= '0';
671                         next_state_p    <= '0';
672                         escape_p        <= '0';
673                         if(reset='1') then      --RESET
674                                 en_out_a_p      <= '0';
675                                 S_p             <= "0000";
676                                 sel_p           <= '1';
677                                 SDV_p           <= '1';
678                                 EDV_p           <= '1';
679                         elsif(P='1' and R='0' and reset='0') then       -- PUSH
680                                 en_out_a_p      <= '1';
681                                 S_p             <= "1001";
682                                 sel_p           <= '0';
683                                 SDV_p           <= '0';
684                                 EDV_p           <= '0';
685                         elsif(P='0' and reset='0') then         --POP or RETI
686                                 en_out_a_p      <= '1';
687                                 S_p             <= "0001";
688                                 sel_p           <= '1';
689                                 SDV_p           <= '0';
690                                 EDV_p           <= '0';
691                         else
692                                 en_out_a_p      <= '0';
693                                 S_p             <= "0000";
694                                 sel_p           <= '0';
695                                 SDV_p           <= '0';
696                                 EDV_p           <= '0';
```

```
697
698                         end if;
699
700 ----------------------State 001 001----------------------------
701                     elsif(execute_p = "001" and execute_p_1 = "001") then
702                         sa_p          <= "01";
703                         sb_p          <= "00";
704                         sel_out_mdr_p <= "00";
705                         save_ir_p     <= '0';
706                         save_p       <= '0';
707                         en_out_b_p    <= '0';
708                         save_sr_p       <= '0';
709                         en_SR_p       <= '0';
710                         rst_sr_p            <= '0';
711                         pc_out_p        <= '0';
712                         reset_i_p      <= '0';
713                         sel_in_mdr_p  <= '0';
714                         save_mdr_p      <= '0';
715                         next_state_p  <= '0';
716                         escape_p      <= '0';
717                         if(reset='1') then        --Reset
718                                 en_out_a_p    <= '0';
719                                 S_p             <= "0000";
720                                 sel_p           <= '1';
721                                 SDV_p         <= '1';
722                                 EDV_p         <= '1';
723                                 RW_p          <= '1';
724                                 ME_p          <= '1';
725                                 en_out_mar_p  <= '1';
726                                 save_mar_p      <= '1';
727                                 save_c_p            <= '0';
728                         elsif(P='1' and R='0' and reset='0') then        --PUSH
729                                 en_out_a_p    <= '1';
730                                 S_p             <= "1001";
731                                 sel_p           <= '0';
732                                 SDV_p         <= '0';
733                                 EDV_p         <= '0';
734                                 RW_p          <= '0';
735                                 ME_p          <= '0';
736                                 en_out_mar_p  <= '0';
737                                 save_mar_p      <= '0';
738                                 save_c_p            <= '1';
739                         elsif(P='0' and reset='0') then          --POP or RETI
740                                 en_out_a_p    <= '1';
741                                 S_p             <= "0001";
742                                 sel_p           <= '1';
743                                 SDV_p         <= '0';
744                                 EDV_p         <= '0';
745                                 RW_p          <= '1';
746                                 ME_p          <= '1';
747                                 en_out_mar_p  <= '1';
748                                 save_mar_p      <= '1';
749                                 save_c_p            <= '1';
750                         else
751                                 en_out_a_p    <= '0';
752                                 S_p             <= "0000";
```

```
753                                 sel_p           <= '0';
754                                 SDV_p           <= '0';
755                                 EDV_p           <= '0';
756                                 RW_p            <= '0';
757                                 ME_p            <= '0';
758                                 en_out_mar_p    <= '0';
759                                 save_mar_p      <= '0';
760                                 save_c_p            <= '0';
761
762                             end if;
763
764 ----------------------State 011 001----------------------------
765                     elsif(execute_p = "011" and execute_p_1 = "001") then
766                             sa_p            <= "00";
767                             sb_p            <= "01";
768                             S_p               <= "0000";
769                             sel_out_mdr_p <= "00";
770                             SDV_p           <= '0';
771                             EDV_p           <= '0';
772                             save_ir_p       <= '0';
773                             save_p        <= '0';
774                             en_out_a_p      <= '0';
775                             en_out_b_p      <= '0';
776                             save_sr_p         <= '0';
777                             en_SR_p         <= '0';
778                             rst_sr_p            <= '0';
779                             pc_out_p          <= '0';
780                             reset_i_p       <= '0';
781                             save_mar_p        <= '0';
782                             sel_p             <= '0';
783                             save_mdr_p        <= '0';
784                             save_c_p            <= '0';
785                             next_state_p  <= '0';
786                             escape_p        <= '0';
787
788                             if(reset='1') then          --RESET
789                                     sel_in_mdr_p  <= '1';
790                                     RW_p          <= '1';
791                                     ME_p          <= '1';
792                                     en_out_mar_p  <= '1';
793                             elsif(P='1' and R='0' and reset='0') then --PUSH
794                                     sel_in_mdr_p  <= '0';
795                                     RW_p          <= '0';
796                                     ME_p          <= '0';
797                                     en_out_mar_p  <= '0';
798                             elsif(P='0' and reset='0') then --POP
799                                     sel_in_mdr_p  <= '1';
800                                     RW_p          <= '1';
801                                     ME_p          <= '1';
802                                     en_out_mar_p  <= '1';
803                             else
804                                     sel_in_mdr_p  <= '0';
805                                     RW_p          <= '0';
806                                     ME_p          <= '0';
807                                     en_out_mar_p  <= '0';
808
```

```
809                             end if;
810
811 ----------------------State 011 011----------------------------
812                     elsif(execute_p = "011" and execute_p_1 = "011") then
813                             sa_p            <= "00";
814                             sb_p            <= "01";
815                             S_p               <= "0000";
816                             sel_out_mdr_p <= "00";
817                             SDV_p           <= '0';
818                             EDV_p           <= '0';
819                             save_ir_p       <= '0';
820                             en_out_a_p      <= '0';
821                             en_out_b_p      <= '0';
822                             save_sr_p         <= '0';
823                             en_SR_p         <= '0';
824                             rst_sr_p              <= '0';
825                             pc_out_p          <= '0';
826                             reset_i_p        <= '0';
827                             save_c_p              <= '0';
828                             next_state_p  <= '0';
829                             escape_p        <= '0';
830                             if(reset='1') then          --RESET
831                                     sel_in_mdr_p  <= '1';
832                                     RW_p            <= '1';
833                                     ME_p            <= '1';
834                                     en_out_mar_p  <= '1';
835                                     save_p        <= '0';
836                                     save_mdr_p        <= '1';
837                                     save_mar_p        <= '0';
838                                sel_p                  <= '0';
839                             elsif(P='1' and R='0' and reset='0') then --PUSH
840                                     sel_in_mdr_p  <= '0';
841                                     RW_p            <= '0';
842                                     ME_p            <= '0';
843                                     en_out_mar_p  <= '0';
844                                     save_p        <= '1';
845                                     save_mdr_p        <= '0';
846                                   save_mar_p    <= '1';
847                                   sel_p                  <= '0';
848                             elsif(P='0' and reset='0') then --POP
849                                     sel_in_mdr_p  <= '1';
850                                     RW_p            <= '1';
851                                     ME_p            <= '1';
852                                     en_out_mar_p  <= '1';
853                                     save_p        <= '1';
854                                     save_mdr_p        <= '1';
855                                 save_mar_p        <= '0';
856                                   sel_p                        <= '0';
857                             else
858                                     sel_in_mdr_p  <= '0';
859                                     RW_p            <= '0';
860                                     ME_p            <= '0';
861                                     en_out_mar_p  <= '0';
862                                     save_p        <= '0';
863                                     save_mdr_p        <= '0';
864                                 save_mar_p        <= '0';
```

```
865                             sel_p                       <= '0';
866
867                         end if;
868 ----------------------State 010 011 --------------------
869                     elsif(execute_p = "010" and execute_p_1 = "011") then
870                         sa_p            <= "10";
871                         sb_p            <= "00";
872                         S_p                   <= "0000";
873                         SDV_p           <= '0';
874                         EDV_p           <= '0';
875                         save_ir_p       <= '0';
876                         RW_p            <= '0';
877                         ME_p            <= '0';
878                         save_p        <= '0';
879                         en_out_b_p      <= '0';
880                         save_sr_p         <= '0';
881                         en_SR_p         <= '0';
882                         rst_sr_p              <= '0';
883                         pc_out_p          <= '0';
884                         reset_i_p        <= '0';
885                         save_mar_p        <= '0';
886                         en_out_mar_p    <= '0';
887                         sel_p             <= '0';
888                         sel_in_mdr_p    <= '0';
889                         save_mdr_p        <= '0';
890                         save_c_p              <= '0';
891                         next_state_p    <= '0';
892                         escape_p        <= '0';
893                         if(reset='1') then          --RESET
894                                 sel_out_mdr_p <= "01";
895                                 en_out_a_p      <= '0';
896                         elsif(P='1' and reset='0') then --PUSH
897                                 sel_out_mdr_p <= "00";
898                                  en_out_a_p      <= '1';
899                         elsif(P='0' and reset='0') then --POP
900                                 sel_out_mdr_p <= "01";
901                                 en_out_a_p      <= '0';
902                         else
903                                 sel_out_mdr_p <= "00";
904                                 en_out_a_p      <= '0';
905
906                         end if;
907 ----------------------State 010 010--------------------
908                     elsif(execute_p = "010" and execute_p_1 = "010") then
909                         sa_p            <= "10";
910                         sb_p            <= "00";
911                         S_p                   <= "0000";
912                         SDV_p           <= '0';
913                         EDV_p           <= '0';
914                         save_ir_p       <= '0';
915                         RW_p            <= '0';
916                         ME_p            <= '0';
917                         save_p        <= '0';
918                         en_out_b_p      <= '0';
919                         save_sr_p         <= '0';
920                         en_SR_p         <= '0';
```

```
921                             rst_sr_p            <= '0';
922                             pc_out_p         <= '0';
923                             reset_i_p       <= '0';
924                             save_mar_p       <= '0';
925                             en_out_mar_p  <= '0';
926                             sel_p            <= '0';
927                             sel_in_mdr_p  <= '0';
928                             save_mdr_p       <= '0';
929                             save_c_p            <= '1';
930                             next_state_p  <= '0';
931                             escape_p       <= '0';
932                             if(reset='1') then          --RESET
933                                     sel_out_mdr_p <= "01";
934                                     en_out_a_p     <= '0';
935                             elsif(P='1'and reset='0') then --PUSH
936                                     sel_out_mdr_p <= "00";
937                                      en_out_a_p     <= '1';
938                             elsif(P='0'and reset='0') then --POP
939                                     sel_out_mdr_p <= "01";
940                                     en_out_a_p     <= '0';
941                             else
942                                     sel_out_mdr_p <= "00";
943                                     en_out_a_p     <= '0';
944
945                             end if;
946
947 ----------------State 110 010--------------------
948                     elsif(execute_p = "110" and execute_p_1 = "010") then
949                             sa_p            <= "00";
950                             S_p                     <= "0000";
951                             sel_out_mdr_p <= "00";
952                             SDV_p           <= '0';
953                             EDV_p           <= '0';
954                             save_ir_p       <= '0';
955                             RW_p            <= '0';
956                             ME_p            <= '0';
957                             save_p       <= '0';
958                             en_out_a_p     <= '0';
959                             en_out_b_p     <= '0';
960                             save_sr_p        <= '0';
961                             en_SR_p         <= '0';
962                             rst_sr_p            <= '0';
963                             pc_out_p         <= '0';
964                             reset_i_p       <= '0';
965                             save_mar_p       <= '0';
966                             en_out_mar_p  <= '0';
967                             sel_p            <= '0';
968                             sel_in_mdr_p  <= '0';
969                             save_mdr_p       <= '0';
970                             save_c_p            <= '0';
971                             next_state_p  <= '0';
972                             escape_p       <= '0';
973                             if(reset='1') then          --RESET
974                                     sb_p            <= "00";
975                             elsif(P='1' and R='0' and reset='0') then --PUSH
976                                     sb_p            <= "00";
```

```
977                             elsif(P='0' and R='0' and reset='0') then --POP
978                                     sb_p            <= "10";
979                             elsif(P='0' and R='1' and reset='0' and T1 = '1') then  --RETI first
                                    tiME_p
980                                     sb_p            <= "00";
981                             elsif(P='0' and R='1' and reset='0' and T1 = '0') then  --RETI second
                                    tiME_p
982                                     sb_p            <= "11";
983                             else
984                                     sb_p            <= "00";
985                             end if;
986
987
988 ----------------State 110 110--------------------
989                     elsif(execute_p = "110" and execute_p_1 = "110") then
990                             sa_p            <= "00";
991                             sb_p            <= "00";
992                             S_p                     <= "0000";
993                             SDV_p           <= '0';
994                             EDV_p           <= '0';
995                             save_ir_p       <= '0';
996                             RW_p            <= '0';
997                             en_out_a_p      <= '0';
998                             en_out_b_p      <= '0';
999                             save_sr_p         <= '0';
1000                            en_SR_p         <= '0';
1001                            rst_sr_p                <= '0';
1002                            pc_out_p          <= '0';
1003                            save_mar_p        <= '0';
1004                            sel_p             <= '0';
1005                            sel_in_mdr_p    <= '0';
1006                            save_c_p                <= '0';
1007                            escape_p        <= '0';
1008                            if(reset='1') then          --RESET
1009                                    sb_p            <= "00";
1010                                    save_p          <= '1';
1011                                    en_out_mar_p    <= '0';
1012                                    ME_p            <= '0';
1013                                    save_mdr_p          <= '0';
1014                                    sel_out_mdr_p <= "00";
1015                                    next_state_p    <= '1';
1016                                    reset_i_p        <= '1';
1017                                    escape_p        <= '0';
1018                            elsif(P='1' and R='0' and reset='0') then --PUSH
1019                                    sb_p            <= "00";
1020                                    save_p          <= '0';
1021                                    en_out_mar_p    <= '1';
1022                                    ME_p            <= '1';
1023                                    save_mdr_p          <= '1';
1024                                    sel_out_mdr_p <= "11";
1025                                    next_state_p    <= '0';
1026                               reset_i_p       <= '0';
1027                                    escape_p        <= '0';
1028                            elsif(P='0' and R='0' and reset='0') then --POP
1029                                    sb_p            <= "10";
1030                                    save_p          <= '1';
```

```
1031                                en_out_mar_p  <= '0';
1032                                ME_p          <= '0';   -- puede fallar
1033                                save_mdr_p       <= '0';
1034                                sel_out_mdr_p <= "00";
1035                                next_state_p  <= '1';
1036                              reset_i_p      <= '0';
1037                                escape_p      <= '1';
1038                            elsif(P='0' and R='1' and reset='0' and T1 = '1') then  --RETI first
                                    tiME_p
1039                                sb_p          <= "00";
1040                                save_p       <= '1';
1041                                en_out_mar_p  <= '0';
1042                                ME_p          <= '0';
1043                                save_mdr_p       <= '0';
1044                                sel_out_mdr_p <= "00";
1045                                next_state_p  <= '0';
1046                                reset_i_p      <= '0';
1047                                 escape_p      <= '0';
1048                            elsif(P='0' and R='1' and reset='0' and T1 = '0') then  --RETI second
                                    tiME_p
1049                                sb_p          <= "11";
1050                                save_p       <= '1';
1051                                en_out_mar_p  <= '0';
1052                                ME_p          <= '0';
1053                                save_mdr_p       <= '0';
1054                                sel_out_mdr_p <= "00";
1055                                next_state_p  <= '1';
1056                                reset_i_p      <= '1';
1057                                escape_p      <= '0';
1058                            else
1059                                sb_p          <= "00";
1060                                save_p       <= '0';
1061                                en_out_mar_p  <= '0';
1062                                ME_p          <= '0';
1063                                save_mdr_p       <= '0';
1064                                sel_out_mdr_p <= "00";
1065                                next_state_p  <= '0';
1066                                reset_i_p      <= '0';
1067                                escape_p      <= '0';
1068                            end if;
1069
1070
1071  ----------------------State 001 110----------------------------
1072                        elsif(execute_p = "001" and execute_p_1 = "110") then
1073                                sa_p          <= "01";
1074                                sb_p          <= "00";
1075                                sel_out_mdr_p <= "00";
1076                                save_ir_p     <= '0';
1077                                RW_p          <= '0';
1078                                ME_p          <= '0';
1079                                save_p       <= '0';
1080                                en_out_b_p    <= '0';
1081                                save_sr_p       <= '0';
1082                                en_SR_p       <= '0';
1083                                rst_sr_p           <= '0';
1084                                pc_out_p       <= '0';
```

```
1085                        reset_i_p       <= '0';
1086                        save_mar_p      <= '0';
1087                        en_out_mar_p    <= '0';
1088                        sel_in_mdr_p    <= '0';
1089                        save_mdr_p      <= '0';
1090                        save_c_p        <= '0';
1091                        next_state_p    <= '0';
1092                        escape_p        <= '0';
1093                        if(reset='1') then        --RESET
1094                                en_out_a_p      <= '0';
1095                                S_p             <= "0000";
1096                                sel_p           <= '1';
1097                                SDV_p           <= '1';
1098                                EDV_p           <= '1';
1099                        elsif(P='1' and R='0' and reset='0') then       -- PUSH
1100                                en_out_a_p      <= '1';
1101                                S_p             <= "1001";
1102                                sel_p           <= '0';
1103                                SDV_p           <= '0';
1104                                EDV_p           <= '0';
1105                        elsif(P='0' and reset='0') then          --POP or RETI
1106                                en_out_a_p      <= '1';
1107                                S_p             <= "0001";
1108                                sel_p           <= '1';
1109                                SDV_p           <= '0';
1110                                EDV_p           <= '0';
1111                        else
1112                                en_out_a_p      <= '0';
1113                                S_p             <= "0000";
1114                                sel_p           <= '0';
1115                                SDV_p           <= '0';
1116                                EDV_p           <= '0';
1117
1118                        end if;
1119
1120
1121 ----------------------State 100 100----------------------------
1122                 elsif(execute_p = "100" and execute_p_1 = "100") then
1123                        sa_p            <= "00";
1124                        sb_p            <= "00";
1125                        S_p                     <= "0000";
1126                        sel_out_mdr_p <= "00";
1127                        SDV_p           <= '0';
1128                        EDV_p           <= '0';
1129                        save_ir_p       <= '0';
1130                        RW_p            <= '0';
1131                        ME_p            <= '0';
1132                        save_p        <= '0';
1133                        en_out_a_p      <= '0';
1134                        en_out_b_p      <= '0';
1135                        save_sr_p         <= '0';
1136                        en_SR_p         <= '0';
1137                        rst_sr_p                <= '0';
1138                        pc_out_p          <= '0';
1139                        reset_i_p       <= '0';
1140                        save_mar_p        <= '0';
```

```
1141                        en_out_mar_p  <= '0';
1142                        sel_p           <= '0';
1143                        sel_in_mdr_p  <= '0';
1144                        save_mdr_p      <= '0';
1145                        save_c_p            <= '0';
1146                        next_state_p  <= '1';
1147                        escape_p      <= '0';
1148
1149  ----------------------State 101 100----------------------------
1150                    elsif(execute_p = "101" and execute_p_1 = "100") then
1151                        sa_p          <= "00";
1152                        sb_p          <= "00";
1153                        S_p                 <= "0000";
1154                        sel_out_mdr_p <= "00";
1155                        SDV_p         <= '0';
1156                        EDV_p         <= '0';
1157                        save_ir_p     <= '0';
1158                        RW_p          <= '0';
1159                        ME_p          <= '0';
1160                        save_p      <= '0';
1161                        en_out_a_p    <= '0';
1162                        en_out_b_p    <= '0';
1163                        save_sr_p       <= '0';
1164                        en_SR_p       <= '0';
1165                        rst_sr_p            <= '0';
1166                        pc_out_p        <= '0';
1167                        reset_i_p      <= '0';
1168                        save_mar_p      <= '0';
1169                        en_out_mar_p  <= '0';
1170                        sel_p           <= '0';
1171                        sel_in_mdr_p  <= '0';
1172                        save_mdr_p      <= '0';
1173                        save_c_p            <= '0';
1174                        next_state_p  <= '0';
1175                        escape_p      <= '0';
1176
1177  ----------------------State 101 101----------------------------
1178                    elsif(execute_p = "101" and execute_p_1 = "101") then
1179                        sa_p          <= "00";
1180                        sb_p          <= "00";
1181                        S_p                 <= "0000";
1182                        sel_out_mdr_p <= "00";
1183                        SDV_p         <= '0';
1184                        EDV_p         <= '0';
1185                        save_ir_p     <= '0';
1186                        RW_p          <= '0';
1187                        ME_p          <= '0';
1188                        save_p      <= '0';
1189                        en_out_a_p    <= '0';
1190                        en_out_b_p    <= '0';
1191                        save_sr_p       <= '0';
1192                        en_SR_p       <= '0';
1193                        rst_sr_p            <= '0';
1194                        pc_out_p        <= '0';
1195                        reset_i_p      <= '0';
1196                        save_mar_p      <= '0';
```

```
1197                        en_out_mar_p  <= '0';
1198                        sel_p           <= '0';
1199                        sel_in_mdr_p  <= '0';
1200                        save_mdr_p      <= '0';
1201                        save_c_p            <= '0';
1202                        next_state_p  <= '1';
1203                        escape_p      <= '0';
1204
1205  ----------------------State 111 111---------------------------
1206                    elsif(execute_p = "111" and execute_p_1 = "111") then
1207                        sa_p          <= "00";
1208                        sb_p          <= "00";
1209                        S_p                 <= "0000";
1210                        sel_out_mdr_p <= "00";
1211                        SDV_p         <= '0';
1212                        EDV_p         <= '0';
1213                        save_ir_p     <= '0';
1214                        RW_p          <= '0';
1215                        ME_p          <= '0';
1216                        save_p      <= '0';
1217                        en_out_a_p    <= '0';
1218                        en_out_b_p    <= '0';
1219                        save_sr_p       <= '0';
1220                        en_SR_p       <= '0';
1221                        rst_sr_p            <= '0';
1222                        pc_out_p        <= '0';
1223                        reset_i_p      <= '0';
1224                        save_mar_p      <= '0';
1225                        en_out_mar_p  <= '0';
1226                        sel_p           <= '0';
1227                        sel_in_mdr_p  <= '0';
1228                        save_mdr_p      <= '0';
1229                        save_c_p            <= '0';
1230                        next_state_p  <= '1';
1231                        escape_p      <= '0';
1232
1233  ----------------------State 000 111---------------------------
1234                    elsif(execute_p = "000" and execute_p_1 = "111") then
1235                        sa_p          <= "00";
1236                        sb_p          <= "00";
1237                        S_p                 <= "0000";
1238                        sel_out_mdr_p <= "00";
1239                        SDV_p         <= '0';
1240                        EDV_p         <= '0';
1241                        save_ir_p     <= '0';
1242                        RW_p          <= '0';
1243                        ME_p          <= '0';
1244                        save_p      <= '0';
1245                        en_out_a_p    <= '0';
1246                        en_out_b_p    <= '0';
1247                        save_sr_p       <= '0';
1248                        en_SR_p       <= '0';
1249                        rst_sr_p            <= '0';
1250                        pc_out_p        <= '0';
1251                        reset_i_p      <= '0';
1252                        save_mar_p      <= '0';
```

```
1253                        en_out_mar_p  <= '0';
1254                        sel_p           <= '0';
1255                        sel_in_mdr_p  <= '0';
1256                        save_mdr_p      <= '0';
1257                        save_c_p            <= '0';
1258                        next_state_p  <= '1';
1259                        escape_p      <= '0';
1260
1261  ----------------------State 000 101----------------------------
1262                    elsif(execute_p = "000" and execute_p_1 = "101") then
1263                        sa_p          <= "00";
1264                        sb_p          <= "00";
1265                        S_p                 <= "0000";
1266                        sel_out_mdr_p <= "00";
1267                        SDV_p         <= '0';
1268                        EDV_p         <= '0';
1269                        save_ir_p     <= '0';
1270                        RW_p          <= '0';
1271                        ME_p          <= '0';
1272                        save_p      <= '0';
1273                        en_out_a_p    <= '0';
1274                        en_out_b_p    <= '0';
1275                        save_sr_p       <= '0';
1276                        en_SR_p       <= '0';
1277                        rst_sr_p            <= '0';
1278                        pc_out_p        <= '0';
1279                        reset_i_p      <= '0';
1280                        save_mar_p      <= '0';
1281                        en_out_mar_p  <= '0';
1282                        sel_p           <= '0';
1283                        sel_in_mdr_p  <= '0';
1284                        save_mdr_p      <= '0';
1285                        save_c_p            <= '0';
1286                        next_state_p  <= '1';
1287                        escape_p      <= '0';
1288
1289  ----------------------State 000 001----------------------------
1290                    elsif(execute_p = "000" and execute_p_1 = "001") then
1291                        sa_p          <= "00";
1292                        sb_p          <= "00";
1293                        S_p                 <= "0000";
1294                        sel_out_mdr_p <= "00";
1295                        SDV_p         <= '0';
1296                        EDV_p         <= '0';
1297                        save_ir_p     <= '0';
1298                        RW_p          <= '0';
1299                        ME_p          <= '0';
1300                        save_p      <= '0';
1301                        en_out_a_p    <= '0';
1302                        en_out_b_p    <= '0';
1303                        save_sr_p       <= '0';
1304                        en_SR_p       <= '0';
1305                        rst_sr_p            <= '0';
1306                        pc_out_p        <= '0';
1307                        reset_i_p      <= '0';
1308                        save_mar_p      <= '0';
```

```
1309                        en_out_mar_p  <= '0';
1310                        sel_p           <= '0';
1311                        sel_in_mdr_p  <= '0';
1312                        save_mdr_p      <= '0';
1313                        save_c_p            <= '0';
1314                        next_state_p  <= '1';
1315                        escape_p      <= '0';
1316
1317  ----------------------State 000 011----------------------------
1318                    elsif(execute_p = "000" and execute_p_1 = "011") then
1319                        sa_p          <= "00";
1320                        sb_p          <= "00";
1321                        S_p                 <= "0000";
1322                        sel_out_mdr_p <= "00";
1323                        SDV_p         <= '0';
1324                        EDV_p         <= '0';
1325                        save_ir_p     <= '0';
1326                        RW_p          <= '0';
1327                        ME_p          <= '0';
1328                        save_p      <= '0';
1329                        en_out_a_p    <= '0';
1330                        en_out_b_p    <= '0';
1331                        save_sr_p       <= '0';
1332                        en_SR_p       <= '0';
1333                        rst_sr_p            <= '0';
1334                        pc_out_p        <= '0';
1335                        reset_i_p      <= '0';
1336                        save_mar_p      <= '0';
1337                        en_out_mar_p  <= '0';
1338                        sel_p           <= '0';
1339                        sel_in_mdr_p  <= '0';
1340                        save_mdr_p      <= '0';
1341                        save_c_p            <= '0';
1342                        next_state_p  <= '1';
1343                        escape_p      <= '0';
1344
1345  ----------------------State 000 010----------------------------
1346                    elsif(execute_p = "000" and execute_p_1 = "010") then
1347                        sa_p          <= "00";
1348                        sb_p          <= "00";
1349                        S_p                 <= "0000";
1350                        sel_out_mdr_p <= "00";
1351                        SDV_p         <= '0';
1352                        EDV_p         <= '0';
1353                        save_ir_p     <= '0';
1354                        RW_p          <= '0';
1355                        ME_p          <= '0';
1356                        save_p      <= '0';
1357                        en_out_a_p    <= '0';
1358                        en_out_b_p    <= '0';
1359                        save_sr_p       <= '0';
1360                        en_SR_p       <= '0';
1361                        rst_sr_p            <= '0';
1362                        pc_out_p        <= '0';
1363                        reset_i_p      <= '0';
1364                        save_mar_p      <= '0';
```

```
1365                        en_out_mar_p  <= '0';
1366                        sel_p           <= '0';
1367                        sel_in_mdr_p  <= '0';
1368                        save_mdr_p      <= '0';
1369                        save_c_p            <= '0';
1370                        next_state_p  <= '1';
1371                        escape_p      <= '0';
1372
1373  ----------------------State 000 110----------------------------
1374                    elsif(execute_p = "000" and execute_p_1 = "110") then
1375                        sa_p          <= "00";
1376                        sb_p          <= "00";
1377                        S_p                 <= "0000";
1378                        sel_out_mdr_p <= "00";
1379                        SDV_p         <= '0';
1380                        EDV_p         <= '0';
1381                        save_ir_p     <= '0';
1382                        RW_p          <= '0';
1383                        ME_p          <= '0';
1384                        save_p      <= '0';
1385                        en_out_a_p    <= '0';
1386                        en_out_b_p    <= '0';
1387                        save_sr_p       <= '0';
1388                        en_SR_p       <= '0';
1389                        rst_sr_p            <= '0';
1390                        pc_out_p        <= '0';
1391                        reset_i_p      <= '0';
1392                        save_mar_p      <= '0';
1393                        en_out_mar_p  <= '0';
1394                        sel_p           <= '0';
1395                        sel_in_mdr_p  <= '0';
1396                        save_mdr_p      <= '0';
1397                        save_c_p            <= '0';
1398                        next_state_p  <= '1';
1399                        escape_p      <= '0';
1400
1401  ----------------------State 000 100----------------------------
1402                    elsif(execute_p = "000" and execute_p_1 = "100") then
1403                        sa_p          <= "00";
1404                        sb_p          <= "00";
1405                        S_p                 <= "0000";
1406                        sel_out_mdr_p <= "00";
1407                        SDV_p         <= '0';
1408                        EDV_p         <= '0';
1409                        save_ir_p     <= '0';
1410                        RW_p          <= '0';
1411                        ME_p          <= '0';
1412                        save_p      <= '0';
1413                        en_out_a_p    <= '0';
1414                        en_out_b_p    <= '0';
1415                        save_sr_p       <= '0';
1416                        en_SR_p       <= '0';
1417                        rst_sr_p            <= '0';
1418                        pc_out_p        <= '0';
1419                        reset_i_p      <= '0';
1420                        save_mar_p      <= '0';
```

```
1421                        en_out_mar_p  <= '0';
1422                        sel_p           <= '0';
1423                        sel_in_mdr_p  <= '0';
1424                        save_mdr_p      <= '0';
1425                        save_c_p            <= '0';
1426                        next_state_p  <= '1';
1427                        escape_p      <= '0';
1428
1429 ------------------State-------------------------
1430                 else
1431                        sa_p          <= "00";
1432                        sb_p          <= "00";
1433                        S_p                 <= "0000";
1434                        sel_out_mdr_p <= "00";
1435                        SDV_p         <= '0';
1436                        EDV_p         <= '0';
1437                        save_ir_p     <= '0';
1438                        RW_p          <= '0';
1439                        ME_p          <= '0';
1440                        save_p      <= '0';
1441                        en_out_a_p    <= '0';
1442                        en_out_b_p    <= '0';
1443                        save_sr_p       <= '0';
1444                        en_SR_p       <= '0';
1445                        rst_sr_p            <= '0';
1446                        pc_out_p        <= '0';
1447                        reset_i_p      <= '0';
1448                        save_mar_p      <= '0';
1449                        en_out_mar_p  <= '0';
1450                        sel_p           <= '0';
1451                        sel_in_mdr_p  <= '0';
1452                        save_mdr_p      <= '0';
1453                        save_c_p            <= '0';
1454                        next_state_p  <= '1';
1455                        escape_p      <= '0';
1456
1457
1458                 end if;
1459 ------------------------------------------------------
1460
1461
1462                 end process;
1463
1464 ------------------------------------------------------
1465 --                      Outputs                     --
1466 ------------------------------------------------------
1467
1468                  sa          <= sa_d or sa_p;
1469                  sb          <= sb_d or sb_p;
1470                  SDV         <= SDV_d or SDV_p;
1471                  EDV         <= EDV_d or EDV_p;
1472                  save_ir     <= save_ir_d or save_ir_p;
1473                  RW          <= RW_d or RW_p;
1474                  ME              <= ME_d or ME_p;
1475                  save        <= save_d or save_p;
1476                  en_out_a    <= en_out_a_d or en_out_a_p;
```

```
1477                    en_out_b   <= en_out_b_d or en_out_b_p;
1478                    save_sr    <= save_sr_d or save_sr_p;
1479                    en_SR      <= en_SR_d or en_SR_p;
1480                    rst_sr     <= rst_sr_d or rst_sr_p;
1481                    pc_out     <= pc_out_d or pc_out_p;
1482                    reset_i    <= reset_i_d or reset_i_p;
1483                    save_mar   <= save_mar_d or save_mar_p;
1484                    en_out_mar     <= en_out_mar_d or en_out_mar_p;
1485                    sel        <= sel_d or sel_p;
1486                    sel_in_mdr     <= sel_in_mdr_d or sel_in_mdr_p;
1487                    sel_out_mdr<= sel_out_mdr_d or sel_out_mdr_p;
1488                    save_mdr   <= save_mdr_d or save_mdr_p;
1489                    save_c     <= save_c_d or save_c_p;
1490            S          <= S_d or S_p;
1491                    next_state     <= next_state_d or next_state_p;
1492                    escape     <= escape_d or escape_p;
1493    end etjs;
```

B.2.19 Signals

```
1   --
        ------------------------------------------------------------------------------------------------

2   --
3   -- Title       : sig
4   --
5   --
        ------------------------------------------------------------------------------------------------

6   --
7   -- Description :        This file call the signal generator files
8   --
9   --
        ------------------------------------------------------------------------------------------------

10
11  library IEEE;
12  use IEEE.STD_LOGIC_1164.all;
13
14  entity sig is
15          port(
16                  fetch     : in STD_LOGIC_VECTOR(1 downto 0);
17                  fetch_1   : in STD_LOGIC_VECTOR(1 downto 0);
18                  decode    : in STD_LOGIC_VECTOR(2 downto 0);
19                  decode_1  : in STD_LOGIC_VECTOR(2 downto 0);
20                  interrupt : in STD_LOGIC_VECTOR(3 downto 0);
21                  interrupt_1 : in STD_LOGIC_VECTOR(3 downto 0);
22                  execute_d   : in STD_LOGIC_VECTOR(1 downto 0);
23                  execute_d_1 : in STD_LOGIC_VECTOR(1 downto 0);
24                  execute_p   : in STD_LOGIC_VECTOR(2 downto 0);
25                  execute_p_1 : in STD_LOGIC_VECTOR(2 downto 0);
26                  second    : in std_logic;
27                  as        : in std_logic_vector(1 downto 0);
28                  ad        :     in std_logic;
29                  A         : in std_logic_vector(1 downto 0);
30                  Op        : in std_logic_vector(3 downto 0);
```

```
31                  R         : in std_logic;
32                  P         : in std_logic;
33                  J         : in std_logic;
34                  reset     : in std_logic;
35                  int       : in std_logic;
36                  sa        : out std_logic_vector(1 downto 0);
37                  sb        : out std_logic_vector(1 downto 0);
38                  SDV       : out std_logic;
39                  EDV       : out std_logic;
40                  save_ir   : out std_logic;
41                  RW      : out std_logic;
42                  ME            : out std_logic;
43                  save      : out std_logic;
44                  en_out_a  : out std_logic;
45                  en_out_b  : out std_logic;
46                  save_sr   : out std_logic;
47                  en_SR     : out std_logic;
48                  rst_sr    : out std_logic;
49                  pc_out    : out std_logic;
50                  reset_i   : out std_logic;
51                  save_mar  : out std_logic;
52                  en_out_mar      : out std_logic;
53                  sel       : out std_logic;
54                  sel_in_mdr      : out std_logic;
55                  sel_out_mdr : out std_logic_vector(1 downto 0);
56                  save_mdr  : out std_logic;
57                  save_c    : out std_logic;
58          S         : out std_logic_vector(3 downto 0);
59                  next_state      : out std_logic;
60                  escape    : out std_logic
61              );
62  end sig;
63
64  architecture sig_1 of sig is
65          COMPONENT fs IS
66                  port(
67                          fetch : in STD_LOGIC_VECTOR(1 downto 0);
68                          fetch_1 : in STD_LOGIC_VECTOR(1 downto 0);
69                          sa        : out std_logic_vector(1 downto 0);
70                          sb        : out std_logic_vector(1 downto 0);
71                          SDV       : out std_logic;
72                          EDV       : out std_logic;
73                          save_ir   : out std_logic;
74                          RW      : out std_logic;
75                          ME            : out std_logic;
76                          save      : out std_logic;
77                          en_out_a  : out std_logic;
78                          en_out_b  : out std_logic;
79                          save_sr   : out std_logic;
80                          en_SR     : out std_logic;
81                          rst_sr    : out std_logic;
82                          pc_out    : out std_logic;
83                          reset_i   : out std_logic;
84                          save_mar  : out std_logic;
85                          en_out_mar      : out std_logic;
86                          sel       : out std_logic;
```

```
87                          sel_in_mdr      : out std_logic;
88                          sel_out_mdr : out std_logic_vector(1 downto 0);
89                          save_mdr : out std_logic;
90                          save_c    : out std_logic;
91                  S        : out std_logic_vector(3 downto 0);
92                          next_state      : out std_logic;
93                          escape : out std_logic
94                      );
95          END COMPONENT;
96
97          COMPONENT ins IS
98                  port(
99                          interrupt : in STD_LOGIC_VECTOR(3 downto 0);
100                         interrupt_1 : in STD_LOGIC_VECTOR(3 downto 0);
101                         second    : in std_logic;
102                         int       : in std_logic;
103                         reset     : in std_logic;
104                         sa        : out std_logic_vector(1 downto 0);
105                         sb        : out std_logic_vector(1 downto 0);
106                         SDV       : out std_logic;
107                         EDV       : out std_logic;
108                         save_ir   : out std_logic;
109                         RW      : out std_logic;
110                         ME            : out std_logic;
111                         save      : out std_logic;
112                         en_out_a  : out std_logic;
113                         en_out_b  : out std_logic;
114                         save_sr   : out std_logic;
115                         en_SR     : out std_logic;
116                         rst_sr    : out std_logic;
117                         pc_out    : out std_logic;
118                         reset_i   : out std_logic;
119                         save_mar  : out std_logic;
120                         en_out_mar      : out std_logic;
121                         sel       : out std_logic;
122                         sel_in_mdr      : out std_logic;
123                         sel_out_mdr : out std_logic_vector(1 downto 0);
124                         save_mdr : out std_logic;
125                         save_c    : out std_logic;
126                 S        : out std_logic_vector(3 downto 0);
127                         next_state      : out std_logic;
128                         escape : out std_logic
129                     );
130         END COMPONENT;
131
132         COMPONENT ds IS
133                 port(
134                         decode    : in STD_LOGIC_VECTOR(2 downto 0);
135                         decode_1  : in STD_LOGIC_VECTOR(2 downto 0);
136                         as        : in std_logic_vector(1 downto 0);
137                         ad        :     in std_logic;
138                         A         : in std_logic_vector(1 downto 0);
139                         Op        :     in std_logic_vector(3 downto 0);
140                         sa        : out std_logic_vector(1 downto 0);
141                         sb        : out std_logic_vector(1 downto 0);
142                         SDV       : out std_logic;
```

```
143                         EDV      : out std_logic;
144                         save_ir  : out std_logic;
145                         RW       : out std_logic;
146                         ME          : out std_logic;
147                         save     : out std_logic;
148                         en_out_a : out std_logic;
149                         en_out_b : out std_logic;
150                         save_sr  : out std_logic;
151                         en_SR    : out std_logic;
152                         rst_sr   : out std_logic;
153                         pc_out   : out std_logic;
154                         reset_i  : out std_logic;
155                         save_mar : out std_logic;
156                         en_out_mar     : out std_logic;
157                         sel      : out std_logic;
158                         sel_in_mdr     : out std_logic;
159                         sel_out_mdr : out std_logic_vector(1 downto 0);
160                         save_mdr : out std_logic;
161                         save_c   : out std_logic;
162                  S       : out std_logic_vector(3 downto 0);
163                         next_state     : out std_logic;
164                         escape : out std_logic
165                     );
166          END COMPONENT;
167
168          COMPONENT etjs IS
169                  port(
170                         execute_d   : in STD_LOGIC_VECTOR(1 downto 0);
171                         execute_d_1 : in STD_LOGIC_VECTOR(1 downto 0);
172                         execute_p   : in STD_LOGIC_VECTOR(2 downto 0);
173                         execute_p_1 : in STD_LOGIC_VECTOR(2 downto 0);
174                         as       : in std_logic_vector(1 downto 0);
175                         ad       :    in std_logic;
176                         Op       : in std_logic_vector(3 downto 0);
177                         R        : in std_logic;
178                         P        : in std_logic;
179                         J        : in std_logic;
180                         T1       : in std_logic;
181                         reset    : in std_logic;
182                         sa       : out std_logic_vector(1 downto 0);
183                         sb       : out std_logic_vector(1 downto 0);
184                         SDV      : out std_logic;
185                         EDV      : out std_logic;
186                         save_ir  : out std_logic;
187                         RW       : out std_logic;
188                         ME          : out std_logic;
189                         save     : out std_logic;
190                         en_out_a : out std_logic;
191                         en_out_b : out std_logic;
192                         save_sr  : out std_logic;
193                         en_SR    : out std_logic;
194                         rst_sr   : out std_logic;
195                         pc_out   : out std_logic;
196                         reset_i  : out std_logic;
197                         save_mar : out std_logic;
198                         en_out_mar     : out std_logic;
```

```
199                         sel      : out std_logic;
200                         sel_in_mdr     : out std_logic;
201                         sel_out_mdr : out std_logic_vector(1 downto 0);
202                         save_mdr : out std_logic;
203                         save_c   : out std_logic;
204                 S        : out std_logic_vector(3 downto 0);
205                         next_state     : out std_logic;
206                         escape : out std_logic
207                     );
208         END COMPONENT;
209
210         signal sa_fetch, sb_fetch, sel_out_mdr_fetch : std_logic_vector(1 downto 0);
211         signal SDV_fetch, EDV_fetch, save_ir_fetch, RW_fetch, ME_fetch, save_fetch,
                en_out_a_fetch, en_out_b_fetch, save_sr_fetch, en_SR_fetch, rst_sr_fetch,
                pc_out_fetch, reset_i_fetch, save_mar_fetch, en_out_mar_fetch, sel_fetch,
                sel_in_mdr_fetch, save_mdr_fetch, save_c_fetch, next_state_fetch, escape_fetch  :
                std_logic;
212         signal S_fetch : std_logic_vector(3 downto 0);
213         signal sa_i, sb_i, sel_out_mdr_i  : std_logic_vector(1 downto 0);
214         signal SDV_i, EDV_i, save_ir_i, RW_i, ME_i, save_i, en_out_a_i, en_out_b_i, save_sr_i,
                en_SR_i, rst_sr_i, pc_out_i, reset_i_i, save_mar_i, en_out_mar_i, sel_i,
                sel_in_mdr_i, save_mdr_i, save_c_i, next_state_i, escape_i : std_logic;
215         signal S_i : std_logic_vector(3 downto 0);
216         signal sa_d, sb_d, sel_out_mdr_d  : std_logic_vector(1 downto 0);
217         signal SDV_d, EDV_d, save_ir_d, RW_d, ME_d, save_d, en_out_a_d, en_out_b_d, save_sr_d,
                en_SR_d, rst_sr_d, pc_out_d, reset_i_d, save_mar_d, en_out_mar_d, sel_d,
                sel_in_mdr_d, save_mdr_d, save_c_d, next_state_d, escape_d : std_logic;
218         signal S_d : std_logic_vector(3 downto 0);
219         signal sa_e, sb_e, sel_out_mdr_e  : std_logic_vector(1 downto 0);
220         signal SDV_e, EDV_e, save_ir_e, RW_e, ME_e, save_e, en_out_a_e, en_out_b_e, save_sr_e,
                en_SR_e, rst_sr_e, pc_out_e, reset_i_e, save_mar_e, en_out_mar_e, sel_e,
                sel_in_mdr_e, save_mdr_e, save_c_e, next_state_e, escape_e : std_logic;
221         signal S_e : std_logic_vector(3 downto 0);
222
223 begin
224
225 ------------------------------------------------------------
226 --                        Fetch                           --
227 ------------------------------------------------------------
228         U1:     fs      PORT MAP (fetch, fetch_1, sa_fetch, sb_fetch, SDV_fetch, EDV_fetch,
                save_ir_fetch, RW_fetch, ME_fetch, save_fetch, en_out_a_fetch, en_out_b_fetch,
                save_sr_fetch, en_sr_fetch, rst_sr_fetch, pc_out_fetch, reset_i_fetch,
                save_mar_fetch, en_out_mar_fetch, sel_fetch, sel_in_mdr_fetch, sel_out_mdr_fetch,
                save_mdr_fetch, save_c_fetch, S_fetch, next_state_fetch, escape_fetch);
229
230 ------------------------------------------------------------
231 --                      Interrupt                         --
232 ------------------------------------------------------------
233         U2:     ins PORT MAP (interrupt, interrupt_1, second, int, reset, sa_i, sb_i, SDV_i,
                EDV_i, save_ir_i, RW_i, ME_i, save_i, en_out_a_i, en_out_b_i, save_sr_i, en_sr_i,
                rst_sr_i, pc_out_i, reset_i_i, save_mar_i, en_out_mar_i, sel_i, sel_in_mdr_i,
                sel_out_mdr_i, save_mdr_i, save_c_i, S_i, next_state_i, escape_i);
234
235 ------------------------------------------------------------
236 --                      Decode                            --
237 ------------------------------------------------------------
```

```
238         U3:     ds PORT MAP (decode, decode_1, as, ad, A, Op, sa_d, sb_d, SDV_d, EDV_d, save_ir_d
                    , RW_d, ME_d, save_d, en_out_a_d, en_out_b_d, save_sr_d, en_sr_d, rst_sr_d, pc_out_d
                    , reset_i_d, save_mar_d, en_out_mar_d, sel_d, sel_in_mdr_d, sel_out_mdr_d,
                    save_mdr_d, save_c_d, S_d, next_state_d, escape_d);
239
240 ------------------------------------------------------------
241 --                      Execute                           --
242 ------------------------------------------------------------
243         U4:     etjs PORT MAP (execute_d, execute_d_1, execute_p, execute_p_1, as, ad, Op, R, P,
                    J, second, reset, sa_e, sb_e, SDV_e, EDV_e, save_ir_e, RW_e, ME_e, save_e,
                    en_out_a_e, en_out_b_e, save_sr_e, en_sr_e, rst_sr_e, pc_out_e, reset_i_e,
                    save_mar_e, en_out_mar_e, sel_e, sel_in_mdr_e, sel_out_mdr_e, save_mdr_e, save_c_e,
                    S_e, next_state_e, escape_e);
244
245 ----------------------Next state exeption-------------------
246
247         next_state <= (next_state_fetch and not next_state_i and not next_state_d and not
                    next_state_e) or (not next_state_fetch and next_state_i and not next_state_d and not
                     next_state_e) or (not next_state_fetch and not next_state_i and next_state_d and
                    not next_state_e) or (not next_state_fetch and not next_state_i and not next_state_d
                     and next_state_e);
248
249 ------------------------------------------------------------
250 --                    Signals Output                      --
251 ------------------------------------------------------------
252
253     sa          <= sa_fetch or sa_i or sa_d or sa_e;
254         sb          <= sb_fetch or sb_i or sb_d or sb_e;
255         S           <= S_fetch or S_i or S_d or S_e;
256         sel_out_mdr <= sel_out_mdr_fetch or sel_out_mdr_i or sel_out_mdr_d or sel_out_mdr_e;
257         SDV         <= SDV_fetch or SDV_i  or SDV_d or SDV_e;
258         EDV         <= EDV_fetch or EDV_i  or EDV_d or EDV_e;
259         save_ir     <= save_ir_fetch or save_ir_i  or save_ir_d or save_ir_e;
260         RW          <= RW_fetch or RW_i  or RW_d or RW_e;
261         ME          <= ME_fetch or ME_i  or ME_d or ME_e;
262         save        <= save_fetch or save_i  or save_d or save_e;
263         en_out_a    <= en_out_a_fetch or en_out_a_i or en_out_a_d or en_out_a_e;
264         en_out_b    <= en_out_b_fetch or en_out_b_i or en_out_b_d or en_out_b_e;
265         save_sr         <= save_sr_fetch or save_sr_i  or save_sr_d or save_sr_e;
266         en_SR       <= en_SR_fetch or en_SR_i  or en_SR_d or en_SR_e;
267         rst_sr      <= rst_sr_fetch or rst_sr_i or rst_sr_d or rst_sr_e;
268         pc_out          <= pc_out_fetch or pc_out_i or pc_out_d or pc_out_e;
269         reset_i     <= reset_i_fetch or reset_i_i or reset_i_d or reset_i_e;
270         save_mar        <= save_mar_fetch or save_mar_i or save_mar_d  or save_mar_e;
271         en_out_mar  <= en_out_mar_fetch or en_out_mar_i or en_out_mar_d or en_out_mar_e;
272         sel             <= sel_fetch or sel_i or sel_d  or sel_e;
273         sel_in_mdr  <= sel_in_mdr_fetch or sel_in_mdr_i or sel_in_mdr_d or sel_in_mdr_e;
274         save_mdr        <= save_mdr_fetch or save_mdr_i or save_mdr_d  or save_mdr_e;
275         save_c      <= save_c_fetch or save_c_i or save_c_d  or save_c_e;
276         escape      <= escape_fetch or escape_i or escape_d  or escape_e;
277
278
279
280 end sig_1;
```

B.2.20 IR Decode

```
--
    ------------------------------------------------------------------------------------------------
--
-- File          : alu.vhd
--
--
    ------------------------------------------------------------------------------------------------
--
     ------------------------
-- Description :                          |     Instruction  Fetch        |
--                                                                         |
                                    |
--                                                         -->|IR     (16b)             Op
     (16b)|---
--                                                                  |
        As(4b)|---
--                       |                 Ad(3b)|---
--                       |              Reg_A(5b)|---
--                       |            Reg_B_C(5b)|---
--                       |              Disp(16b)|---
--                       |                       |
--                       |                       |
--
    ------------------------
--
--
    ------------------------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity fetch is
        port(
                IR      : in STD_LOGIC_VECTOR(15 downto 0);
                Op      : out STD_LOGIC_VECTOR(3 downto 0);
                As      : out STD_LOGIC_VECTOR(1 downto 0);
                Ad      : out STD_LOGIC;
                Reg_A   : out STD_LOGIC_VECTOR(4 downto 0);
                Reg_B_C : out STD_LOGIC_VECTOR(4 downto 0);
                Disp : out STD_LOGIC_VECTOR(15 downto 0)
                );
end fetch;



architecture fetch of fetch is


begin
```

```
44
45  ------------------------------------------------------------------------------
46  --                                Op Code
47  ------------------------------------------------------------------------------
48
49          OP <= IR(15 downto 12);
50
51  ------------------------------------------------------------------------------
52  --                          Addressing Mode Source
53  ------------------------------------------------------------------------------
54
55          As <= IR(2 downto 1);
56
57  ------------------------------------------------------------------------------
58  --                          Addressing Mode Destination
59  ------------------------------------------------------------------------------
60
61          Ad <= IR(0);
62
63  ------------------------------------------------------------------------------
64  --                             Source Register
65  ------------------------------------------------------------------------------
66
67          Reg_A <= '0' & IR(10 downto 7)  WHEN IR(11)='0' ELSE
68                               '0' & IR(6 downto 3)  WHEN IR(11)='1' and IR(10)='1' and IR(15 downto
                                     12) /= "1010" ELSE
69                               "10" & IR(9 downto 7)  WHEN IR(11)='1' and (IR(10)='0' or IR(15 downto
                                    12) = "1010") ELSE
70                               '0' & IR(6 downto 3);
71
72  ------------------------------------------------------------------------------
73  --                             Source Register
74  ------------------------------------------------------------------------------
75
76          Reg_B_C <= '0' & IR(6 downto 3)  WHEN IR(11)='0' and IR(15 downto 12) /= "1100" ELSE
77                               '0' & IR(6 downto 3)  WHEN IR(11)='1' and IR(10)='0' and IR(15 downto
                                     12) /= "1100" ELSE
78                               "10" & IR(9 downto 7)  WHEN IR(11)='1' and (IR(10)='1' or IR(15 downto
                                      12) = "1100") ELSE
79                               "0" & IR(10 downto 7) WHEN IR(11)='0' and IR(15 downto 12) = "1100"
                                     ELSE
80                                  '0' & IR(6 downto 3);
81
82  ------------------------------------------------------------------------------
83  --                             Source Register
84  ------------------------------------------------------------------------------
85
86          Disp <= IR(11) & IR(11) & IR(11) & IR(11) & IR(11 downto 0);
87
88  ------------------------------------------------------------------------------
89  end fetch;
```

B.2.21 Fetch Signals

```
1  --
       ------------------------------------------------------------------------------------------------
2  --
3  -- File         : fetch_signals.vhd
4  --
5  --
       ------------------------------------------------------------------------------------------------
6  --
7  -- Description : This file generate the fetch signals
8  --
9  --
       ------------------------------------------------------------------------------------------------

10
11 library IEEE;
12 use IEEE.STD_LOGIC_1164.all;
13
14 entity fs is
15          port(
16                  fetch : in STD_LOGIC_VECTOR(1 downto 0);
17                  fetch_1 : in STD_LOGIC_VECTOR(1 downto 0);
18                  sa        : out std_logic_vector(1 downto 0);
19                  sb        : out std_logic_vector(1 downto 0);
20                  SDV       : out std_logic;
21                  EDV       : out std_logic;
22                  save_ir   : out std_logic;
23                  RW      : out std_logic;
24                  ME            : out std_logic;
25                  save      : out std_logic;
26                  en_out_a : out std_logic;
27                  en_out_b : out std_logic;
28                  save_sr   : out std_logic;
29                  en_SR     : out std_logic;
30                  rst_sr    : out std_logic;
31                  pc_out    : out std_logic;
32                  reset_i   : out std_logic;
33                  save_mar : out std_logic;
34                  en_out_mar      : out std_logic;
35                  sel       : out std_logic;
36                  sel_in_mdr      : out std_logic;
37                  sel_out_mdr : out std_logic_vector(1 downto 0);
38                  save_mdr : out std_logic;
39                  save_c    : out std_logic;
40          S         : out std_logic_vector(3 downto 0);
41                  next_state      : out std_logic;
42                  escape : out std_logic
43              );
44 end fs;
45
46 architecture fs_1 of fs is
47
48 begin
49
50          sa            <= "00";
```

```
51          sb              <= "00";
52          sel_out_mdr     <= "00";
53          SDV             <= '0';
54          EDV             <= '0';
55          en_out_b        <= '0';
56          save_sr             <= '0';
57          en_SR           <= '0';
58          rst_sr      <= '0';
59          save_mar            <= '0';
60          en_out_mar      <= '0';
61          sel                 <= '0';
62          sel_in_mdr      <= '0';
63          save_mdr            <= '0';
64
65          process(fetch, fetch_1)
66          begin
67
68  ------------------------------------------------------------
69  --                    Fetch Signals                       --
70  ------------------------------------------------------------
71
72  ----------------------State 00------------------------------
73
74                  if(fetch = "00" and fetch_1 = "00")      then
75                          S                   <= "0000";
76                          save_ir         <= '0';
77                          RW              <= '0';
78                          ME              <= '0';
79                          save             <= '0';
80                          en_out_a        <= '0';
81                          pc_out              <= '0';
82                          save_c              <= '0';
83                          next_state      <= '0';
84                          escape          <= '0';
85
86  ----------------------State 01 00---------------------------
87                  elsif(fetch = "01" and fetch_1 = "00") then
88                          S                   <= "0001";
89                          save_ir         <= '0';
90                          RW              <= '1';
91                          ME              <= '1';
92                          save             <= '0';
93                          en_out_a        <= '1';
94                          pc_out              <= '1';
95                          save_c              <= '0';
96                          next_state      <= '0';
97                          escape          <= '0';
98
99  ----------------------State 01 01---------------------------
100                 elsif(fetch = "01" and fetch_1 = "01") then
101                         S                   <= "0001";
102                         save_ir         <= '0';
103                         RW              <= '1';
104                         ME              <= '1';
105                         save             <= '0';
106                         en_out_a        <= '1';
```

```
107                         pc_out              <= '1';
108                         save_c              <= '1';
109                         next_state    <= '0';
110                         escape       <= '0';
111
112 ----------------------State 11 01-------------------------------
113                 elsif(fetch = "11" and fetch_1 = "01") then
114                         S                   <= "0000";
115                         save_ir       <= '1';
116                         RW            <= '1';
117                         ME            <= '1';
118                         save           <= '0';
119                         en_out_a      <= '0';
120                         pc_out              <= '1';
121                         save_c              <= '0';
122                         next_state    <= '0';
123                         escape       <= '0';
124
125 ----------------------State 11 11-------------------------------
126                 elsif(fetch = "11" and fetch_1 = "11") then
127                         S                   <= "0000";
128                         save_ir       <= '1';
129                         RW            <= '1';
130                         ME            <= '1';
131                         save           <= '1';
132                         en_out_a      <= '0';
133                         pc_out              <= '1';
134                         save_c              <= '0';
135                         next_state    <= '1';
136                         escape       <= '0';
137
138 ----------------------State 10 10-------------------------------
139                 elsif(fetch = "10" and fetch_1 = "10") then
140                         S                   <= "0000";
141                         save_ir       <= '0';
142                         RW            <= '0';
143                         ME            <= '0';
144                         save           <= '0';
145                         en_out_a      <= '0';
146                         pc_out              <= '0';
147                         save_c              <= '0';
148                         next_state    <= '1';
149                         escape       <= '0';
150
151 ----------------------State 00 01-------------------------------
152                 elsif(fetch = "00" and fetch_1 = "01") then
153                         S                   <= "0000";
154                         save_ir       <= '0';
155                         RW            <= '0';
156                         ME            <= '0';
157                         save           <= '0';
158                         en_out_a      <= '0';
159                         pc_out              <= '0';
160                         save_c              <= '0';
161                         next_state    <= '1';
162                         escape       <= '0';
```

```
163
164 ----------State 00 11----------
165                 elsif(fetch = "00" and fetch_1 = "11") then
166                         S               <= "0000";
167                         save_ir     <= '0';
168                         RW          <= '0';
169                         ME          <= '0';
170                         save         <= '0';
171                         en_out_a    <= '0';
172                         pc_out          <= '0';
173                         save_c          <= '0';
174                         next_state  <= '1';
175                         escape     <= '0';
176
177 ----------State 00 10----------
178                 elsif(fetch = "00" and fetch_1 = "10") then
179                         S               <= "0000";
180                         save_ir     <= '0';
181                         RW          <= '0';
182                         ME          <= '0';
183                         save         <= '0';
184                         en_out_a    <= '0';
185                         pc_out          <= '0';
186                         save_c          <= '0';
187                         next_state  <= '1';
188                         escape     <= '0';
189
190 ----------State others----------
191                 else
192                         S               <= "0000";
193                         save_ir     <= '0';
194                         RW          <= '0';
195                         ME          <= '0';
196                         save         <= '0';
197                         en_out_a    <= '0';
198                         pc_out          <= '0';
199                         save_c          <= '0';
200                         next_state  <= '1';
201                         escape     <= '0';
202                 end if;
203                 end process;
204                 reset_i  <= '0';
205 end fs_1;
```

B.2.22 Fetch State Machine

```
1 --
      ------------------------------------------------------------------------------------------------
2 --
3 -- File          : fetch_sm.vhd
4 --
5 --
      ------------------------------------------------------------------------------------------------
6 --
```

```
7  -- Description : Fetch state machine
8  --
9  --
       ------------------------------------------------------------------------------------------------------------------

10
11 library IEEE;
12 use IEEE.STD_LOGIC_1164.all;
13
14 entity fsm is
15          port(
16                  F : in STD_LOGIC;
17                  MA : in STD_LOGIC;
18                  clk : in STD_LOGIC;
19                  rst : in STD_LOGIC;
20                  state : out STD_LOGIC_VECTOR(1 downto 0);
21                  state1 : out STD_LOGIC_VECTOR(1 downto 0)
22              );
23 end fsm;
24
25 architecture fsm_1 of fsm is
26            signal state_temp : std_logic_vector(1 downto 0);
27 begin
28
29         process (clk, rst)
30         begin
31                 if (rst = '1') then
32                         state_temp <= "00";
33                 elsif (clk'EVENT AND clk='1') then
34                         if (state_temp="00" and F = '0' and MA = '0' ) then
35                                 state_temp <= "00";
36                         elsif (state_temp="00" and F = '0' and MA = '1' ) then
37                                 state_temp <= "00";
38                         elsif (state_temp="00" and F = '1' and MA = '0' ) then
39                                 state_temp <= "01";
40                         elsif (state_temp="00" and F = '1' and MA = '1' ) then
41                                 state_temp <= "01";
42                         elsif (state_temp="01" and F = '0' and MA = '0' ) then
43                                 state_temp <= "00";
44                         elsif (state_temp="01" and F = '0' and MA = '1' ) then
45                                 state_temp <= "00";
46                         elsif (state_temp="01" and F = '1' and MA = '0' ) then
47                                 state_temp <= "01";
48                         elsif (state_temp="01" and F = '1' and MA = '1' ) then
49                                 state_temp <= "11";
50                         elsif (state_temp="11" and F = '0' and MA = '0' ) then
51                                 state_temp <= "00";
52                         elsif (state_temp="11" and F = '0' and MA = '1' ) then
53                                 state_temp <= "00";
54                         elsif (state_temp="11" and F = '1' and MA = '0' ) then
55                                 state_temp <= "00";
56                         elsif (state_temp="11" and F = '1'and MA = '1' ) then
57                                 state_temp <= "00";
58                         elsif (state_temp="10" and F = '0'and MA = '0' ) then
59                                 state_temp <= "00";
60                         elsif (state_temp="10" and F = '0' and MA = '1' ) then
```

```
61                                  state_temp <= "00";
62                          elsif (state_temp="10" and F = '1' and MA = '0' ) then
63                                  state_temp <= "00";
64                          elsif (state_temp="10" and F = '1'and MA = '1' ) then
65                                  state_temp <= "00";
66                          end if;
67                  end if;
68          end process;
69
70          process (clk, rst)
71          begin
72                  if (rst = '1') then
73                          state1 <= "00";
74                  elsif (clk'EVENT AND clk='0') then
75                          state1 <= state_temp;
76                  end if;
77
78          end process;
79                  state <= state_temp;
80  end fsm_1;
```

B.2.23 Main State Machine

```
1   --
        ------------------------------------------------------------------------------------------------

2   --
3   -- File         : main_sm.vhd
4   --
5   --
        ------------------------------------------------------------------------------------------------

6   --
7   -- Description :         This state machine control the order of the other state machines
8   --
9   --
        ------------------------------------------------------------------------------------------------

10
11  library IEEE;
12  use IEEE.STD_LOGIC_1164.all;
13
14  entity msn is
15           port(
16                   clk : in STD_LOGIC;
17                   rst : in STD_LOGIC;
18                   E : out STD_LOGIC_VECTOR(3 downto 0)
19               );
20  end msn;
21
22  architecture msn_1 of msn is
23          signal temp0, temp00, reset, temp1, temp2, temp3 : std_logic;
24  begin
25
26          reset <= temp0 or temp1 or temp2 or temp3;
27          temp00 <= temp0 xnor reset;
```

```
28
29          process(clk, rst)
30          begin
31
32                  IF(rst = '1') then
33                          temp0 <= '0';
34                          temp1 <= '0';
35                          temp2 <= '0';
36                          temp3 <= '0';
37                  elsif (clk'EVENT AND clk='1') THEN
38                          temp0 <= temp3;
39                          temp1 <= temp00;
40                          temp2 <= temp1;
41                          temp3 <= temp2;
42                  END IF;
43
44          END PROCESS;
45
46          E(0) <= temp00;
47          E(1) <= temp1;
48          E(2) <= temp2;
49          E(3) <= temp3;
50
51  end msn_1;
```

B.2.24 Read/Write Signals Generator

```
1  --
       ------------------------------------------------------------------------------------------------
2  --
3  -- File         : write.vhd
4  --
5  --
       ------------------------------------------------------------------------------------------------
6  --
7  -- Description : This file control the comunication with the memory
8  --
9  --
       ------------------------------------------------------------------------------------------------
10
11 library IEEE;
12 use IEEE.STD_LOGIC_1164.all;
13
14 entity r_w is
15          port(
16                  clk     : in STD_LOGIC;
17                  RW      : in STD_LOGIC;
18                  ME      : in STD_LOGIC;
19                  rst     : in std_logic;
20                  CE_not          : out STD_LOGIC;
21                  OE_not          : out STD_LOGIC;
22                  WE_not          : out STD_LOGIC;
23                  MA      : out std_logic
```

```
24              );
25  end r_w;
26
27  --}} End of automatically maintained section
28
29  architecture r_w of r_w is
30          signal state : std_logic_vector(3 downto 0);
31          signal write_1 : std_logic;
32  begin
33
34          write_1 <= '1' when (RW = '0' and ME = '1') or (ME = '0' and state = "0111") else '0';
35
36
37          Process (clk, rst)
38          begin
39                  if(rst = '1') then
40                          state <= "0000";
41                  elsif(clk'event and clk = '1') then
42                          if(state = "0000" and ME = '0') then
43                                  state <= "0000";
44                          elsif(state = "0000" and ME = '1') then
45                                  state <= "0001";
46                          elsif(state = "0001") then
47                                  state <= "0010";
48                          elsif(state = "0010") then
49                                  state <= "0011";
50                          elsif(state = "0011") then
51                                  state <= "0100";
52                          elsif(state = "0100") then
53                                  state <= "0101";
54                          elsif(state = "0101") then
55                                  state <= "0110";
56                          elsif(state = "0110" and ME = '1') then
57                                  state <= "0111";
58                          elsif(state = "0110" and ME = '0') then
59                                  state <= "0000";
60                          elsif(state = "0111" and ME = '1') then
61                                  state <= "1000";
62                          elsif(state = "0111" and ME = '0') then
63                                  state <= "0000";
64                          elsif(state = "1000" and ME = '1') then
65                                  state <= "1001";
66                          elsif(state = "1000" and ME = '0') then
67                                  state <= "0000";
68                          elsif(state = "1001" and ME = '1') then
69                                  state <= "1010";
70                          elsif(state = "1001" and ME = '0') then
71                                  state <= "0000";
72                          elsif(state = "1010") then
73                                  state <= "1011";
74                          elsif(state = "1011") then
75                                  state <= "1100";
76                          elsif(state = "1100" and ME = '1') then
77                                  state <= "1101";
78                          elsif(state = "1101") then
79                                  state <= "1110";
```

```
80                          elsif(state = "1110" and ME = '0') then
81                                  state <= "0000";
82                          elsif(state = "1110" and ME = '1') then
83                                  state <= "1111";
84                          elsif(state = "1111") then
85                                  state <= "0000";
86                          end if;
87                  end if;
88
89          end process;
90
91          CE_not <= '1' when state = "0000" else
92                          '0' when state = "0001" or state = "0010" or state = "0011" or state = "
                                0100" or state = "0101" else
93                          '0' when (state = "0110" or state = "0111" or state = "1000" or state =
                                "1001" or state = "1010" or state = "1011" or state = "1100" or
                                state = "1101" or state = "1110" or state = "1111") and write_1 =
                                '0' else
94                          '1' when (state = "0110" or state = "0111" or state = "1000" or state =
                                "1001" or state = "1010" or state = "1011" or state = "1100" or
                                state = "1101" or state = "1110" or state = "1111") and write_1 =
                                '1' else
95                          '0';
96
97          WE_not <= '0' when (state = "0010" or state = "0011") and write_1 = '1' else
98                          '1';
99
100         OE_not <= '1' when write_1 = '1' else
101                         '1' when write_1 = '0' and state = "0000" else
102                         '0';
103
104         MA <= '1' when write_1 = '0' and (state = "1110" or state = "1111") else
105                         '1' when write_1 = '1' and (state = "0110" or state = "0111" or state = "
                                1000" or state = "1001" or state = "1010" or state = "1011" or state
                                = "1100"  or state = "1101" or state = "1110" or state = "1111")
                                else
106                         '0';
107
108 end r_w;
```

B.2.25 Interrupt Decoder

```
1  --
       ------------------------------------------------------------------------------------------
2  --
3  -- File         : alu.vhd
4  --
5  --
       ------------------------------------------------------------------------------------------
6  --
        -------------------------
7  -- Description :                                  |         Interrupt          |
8  --                                                                          |
                                        |
```

```
9  --                                                                    -->|GIR   (16b)    Vector(16b)|---
10 --                                                                    -->|IER   (16b)
       Int(1b)|---
11 --                          -->|IE    (1b)          Reset(1b)|---
12 --                             |                          |
13 --                             |                          |
14 --
    ---------------------------
15 --
16 --
    ------------------------------------------------------------------------------------------------------------
```

```
17
18 library IEEE;
19 use IEEE.STD_LOGIC_1164.all;
20 use IEEE.STD_LOGIC_ARITH.ALL;
21 use IEEE.STD_LOGIC_UNSIGNED.ALL;
22
23 entity int is
24         port(
25                 GIR        : in STD_LOGIC_VECTOR(15 downto 0);
26                 IER        : in STD_LOGIC_VECTOR(15 downto 1);
27                 IE         :     in STD_LOGIC;
28                 vector     : out STD_LOGIC_VECTOR(15 downto 0);
29                 Interrupt : out STD_LOGIC;
30                 Reset      : out std_logic
31                 );
32 end int;
33
34
35
36 architecture int_1 of int is
37
38         signal and_1, and_2, and_3,     and_4, and_5, and_6, and_7,     and_8, and_9, and_10,
               and_11, and_12, and_13, and_14, and_15 : STD_LOGIC;
39         signal int : STD_LOGIC_VECTOR(15 downto 0);
40
41 begin
42
43 ---------------------------------------------------------------------------
44 --                         Interrupt decoding
45 ---------------------------------------------------------------------------
46
47     int(0) <= GIR(0) and '1';
48         int(1) <= GIR(1) and IER(1) and (not int(0));
49         int(2) <= GIR(2) and IER(2) and and_1;
50         int(3) <= GIR(3) and IER(3) and  and_2;
51         int(4) <= GIR(4) and IER(4) and  and_3;
52         int(5) <= GIR(5) and IER(5) and  and_4;
53         int(6) <= GIR(6) and IER(6) and  and_5;
54         int(7) <= GIR(7) and IER(7) and  and_6;
55         int(8) <= GIR(8) and IER(8) and  and_7;
56         int(9) <= GIR(9) and IER(9) and  and_8;
57         int(10) <= GIR(10) and IER(10) and  and_9;
58         int(11) <= GIR(11) and IER(11) and  and_10;
59         int(12) <= GIR(12) and IER(12) and  and_11;
```

```
60          int(13) <= GIR(13) and IER(13) and  and_12;
61          int(14) <= GIR(14) and IER(14) and  and_13;
62          int(15) <= GIR(15) and IER(15) and  and_14;
63
64
65          and_1 <= (not int(0)) and (not int(1));
66          and_2 <= and_1 and (not int(2));
67          and_3 <= and_2 and (not int(3));
68          and_4 <= and_3 and (not int(4));
69          and_5 <= and_4 and (not int(5));
70          and_6 <= and_5 and (not int(6));
71          and_7 <= and_6 and (not int(7));
72          and_8 <= and_7 and (not int(8));
73          and_9 <= and_8 and (not int(9));
74          and_10 <= and_9 and (not int(10));
75          and_11 <= and_10 and (not int(11));
76          and_12 <= and_11 and (not int(12));
77          and_13 <= and_12 and (not int(13));
78          and_14 <= and_13 and (not int(14));
79          and_15 <= and_14 and (not int(15));
80
81           Interrupt <= not and_15 and IE;
82
83  ------------------------------------------------------------------------------
84  --                          Vector
85  ------------------------------------------------------------------------------
86
87          vector <= "0000000000000000" WHEN int = "0000000000000001" ELSE
88                              "0000000000000001" WHEN int = "0000000000000010" ELSE
89                              "0000000000000010" WHEN int = "0000000000000100" ELSE
90                              "0000000000000011" WHEN int = "0000000000001000" ELSE
91                              "0000000000000100" WHEN int = "0000000000010000" ELSE
92                              "0000000000000101" WHEN int = "0000000000100000" ELSE
93                              "0000000000000110" WHEN int = "0000000001000000" ELSE
94                              "0000000000000111" WHEN int = "0000000010000000" ELSE
95                              "0000000000001000" WHEN int = "0000000100000000" ELSE
96                              "0000000000001001" WHEN int = "0000001000000000" ELSE
97                              "0000000000001010" WHEN int = "0000010000000000" ELSE
98                              "0000000000001011" WHEN int = "0000100000000000" ELSE
99                              "0000000000001100" WHEN int = "0001000000000000" ELSE
100                             "0000000000001101" WHEN int = "0010000000000000" ELSE
101                             "0000000000001110" WHEN int = "0100000000000000" ELSE
102                             "0000000000001111" WHEN int = "1000000000000000" ELSE
103                             "0000000000000000";
104
105          Reset <= int(0);
106
107 ------------------------------------------------------------------------------
108 end int_1;
```

B.2.26 Interrupt Signals

```
1  --
       ------------------------------------------------------------------------------
2  --
```

```
3  -- File         : interrupt_signals.vhd
4  --
5  --
       ------------------------------------------------------------------------------------------------------------

6  --
7  -- Description :         Interrupt State Machine
8  --
9  --
       ------------------------------------------------------------------------------------------------------------

10
11 library IEEE;
12 use IEEE.STD_LOGIC_1164.all;
13
14 entity ins is
15          port(
16                  interrupt : in STD_LOGIC_VECTOR(3 downto 0);
17                  interrupt_1 : in STD_LOGIC_VECTOR(3 downto 0);
18                  second    : in std_logic;
19                  int       : in std_logic;
20                  reset     : in std_logic;
21                  sa        : out std_logic_vector(1 downto 0);
22                  sb        : out std_logic_vector(1 downto 0);
23                  SDV       : out std_logic;
24                  EDV       : out std_logic;
25                  save_ir   : out std_logic;
26                  RW        : out std_logic;
27                  ME            : out std_logic;
28                  save      : out std_logic;
29                  en_out_a  : out std_logic;
30                  en_out_b  : out std_logic;
31                  save_sr   : out std_logic;
32                  en_SR     : out std_logic;
33                  rst_sr    : out std_logic;
34                  pc_out    : out std_logic;
35                  reset_i   : out std_logic;
36                  save_mar  : out std_logic;
37                  en_out_mar      : out std_logic;
38                  sel       : out std_logic;
39                  sel_in_mdr      : out std_logic;
40                  sel_out_mdr : out std_logic_vector(1 downto 0);
41                  save_mdr  : out std_logic;
42                  save_c    : out std_logic;
43          S         : out std_logic_vector(3 downto 0);
44                  next_state      : out std_logic;
45                  escape : out std_logic
46              );
47 end ins;
48
49 architecture is_1 of ins is
50         signal i, int_temp, int_temp_1, next_state_temp: std_logic;
51
52
53 begin
54
```

```
55  ------------------------------------------------------------
56  --                      Interrupt                           --
57  ------------------------------------------------------------
58                  process(interrupt, interrupt_1, reset, int_temp, second)
59                  begin
60  ------------------------State 0000-------------------------------
61                  if(interrupt = "0000" and interrupt_1 = "0000") then
62                          sa          <= "00";
63                          sb          <= "00";
64                          S                <= "0000";
65                          sel_out_mdr <= "00";
66                          SDV         <= '0';
67                          EDV         <= '0';
68                          save_ir     <= '0';
69                          RW          <= '0';
70                          ME          <= '0';
71                          save        <= '0';
72                          en_out_a    <= '0';
73                          en_out_b    <= '0';
74                          save_sr         <= '0';
75                          en_SR       <= '0';
76                          rst_sr      <= '0';
77                          pc_out          <= '0';
78                          reset_i     <= '0';
79                          save_mar        <= '0';
80                          en_out_mar  <= '0';
81                          sel             <= '0';
82                          sel_in_mdr  <= '0';
83                          save_mdr        <= '0';
84                          save_c      <= '0';
85                          next_state_temp  <= '0';
86                          escape      <= '0';
87
88  ---------------------State 0001 0000----Interrupt-----------------------
89                  elsif(interrupt = "0001" and interrupt_1 = "0000" and (int_temp = '1' and reset =
                          '0')) then
90                          sa          <= "01";
91                          sb          <= "00";
92                          S               <= "1001";
93                          sel_out_mdr <= "00";
94                          SDV         <= '0';
95                          EDV         <= '0';
96                          save_ir     <= '0';
97                          RW          <= '0';
98                          ME          <= '0';
99                          save        <= '0';
100                         en_out_a    <= '1';
101                         en_out_b    <= '0';
102                         save_sr         <= '0';
103                         en_SR       <= '0';
104                         rst_sr      <= '0';
105                         pc_out          <= '0';
106                         reset_i     <= '0';
107                         save_mar        <= '0';
108                         en_out_mar  <= '0';
109                         sel             <= '0';
```

```
110                         sel_in_mdr  <= '0';
111                         save_mdr        <= '0';
112                         save_c      <= '0';
113                         next_state_temp  <= '0';
114                         escape      <= '0';
115
116 ----------------------State 0001 0000---No-Interrupt-----------------------
117                 elsif(interrupt = "0001" and interrupt_1 = "0000" and int_temp = '0') then
118                         sa          <= "00";
119                         sb          <= "00";
120                         S               <= "0000";
121                         sel_out_mdr <= "00";
122                         SDV         <= '0';
123                         EDV         <= '0';
124                         save_ir     <= '0';
125                         RW          <= '0';
126                         ME          <= '0';
127                         save        <= '0';
128                         en_out_a    <= '1';
129                         en_out_b    <= '0';
130                         save_sr         <= '0';
131                         en_SR       <= '0';
132                         rst_sr      <= '0';
133                         pc_out          <= '0';
134                         reset_i     <= '0';
135                         save_mar        <= '0';
136                         en_out_mar  <= '0';
137                         sel             <= '0';
138                         sel_in_mdr  <= '0';
139                         save_mdr        <= '0';
140                         save_c      <= '0';
141                         next_state_temp  <= '1';
142                         escape      <= '0';
143
144 ----------------------State 0001 0001----Interrupt-----------------------
145                 elsif(interrupt = "0001" and interrupt_1 = "0001" and (int_temp = '1' and reset =
                          '0')) then
146                         sa          <= "01";
147                         sb          <= "00";
148                         S               <= "1001";
149                         sel_out_mdr <= "00";
150                         SDV         <= '0';
151                         EDV         <= '0';
152                         save_ir     <= '0';
153                         RW          <= '0';
154                         ME          <= '0';
155                         save        <= '0';
156                         en_out_a    <= '1';
157                         en_out_b    <= '0';
158                         save_sr         <= '0';
159                         en_SR       <= '0';
160                         rst_sr      <= '0';
161                         pc_out          <= '0';
162                         reset_i     <= '0';
163                         save_mar        <= '0';
164                         en_out_mar  <= '0';
```

```
165                         sel             <= '0';
166                         sel_in_mdr  <= '0';
167                         save_mdr        <= '0';
168                         save_c      <= '1';
169                         next_state_temp  <= '0';
170                         escape      <= '0';
171
172
173 ---------------------State 001 001----No interrupt---------------------
174                 elsif(interrupt = "0001" and interrupt_1 = "0001" and int_temp = '0') then
175                         sa          <= "00";
176                         sb          <= "00";
177                         S               <= "0000";
178                         sel_out_mdr <= "00";
179                         SDV         <= '0';
180                         EDV         <= '0';
181                         save_ir     <= '0';
182                         RW          <= '0';
183                         ME          <= '0';
184                         save        <= '0';
185                         en_out_a    <= '0';
186                         en_out_b    <= '0';
187                         save_sr         <= '0';
188                         en_SR       <= '0';
189                         rst_sr      <= '0';
190                         pc_out          <= '0';
191                         reset_i     <= '0';
192                         save_mar        <= '0';
193                         en_out_mar  <= '0';
194                         sel             <= '0';
195                         sel_in_mdr  <= '0';
196                         save_mdr        <= '0';
197                         save_c      <= '0';
198                         next_state_temp  <= '1';
199                         escape      <= '0';
200
201 ---------------------State 011 001---------------------------
202                 elsif(interrupt = "0011" and interrupt_1 = "0001") then
203                         sa          <= "00";
204                         sb          <= "01";
205                         S               <= "0000";
206                         sel_out_mdr <= "00";
207                         SDV         <= '0';
208                         EDV         <= '0';
209                         save_ir     <= '0';
210                         RW          <= '0';
211                         ME          <= '0';
212                         save        <= '0';
213                         en_out_a    <= '0';
214                         en_out_b    <= '0';
215                         save_sr         <= '0';
216                         en_SR       <= '0';
217                         rst_sr      <= '0';
218                         pc_out          <= '0';
219                         reset_i     <= '0';
220                         save_mar        <= '0';
```

```
221                         en_out_mar  <= '0';
222                         sel             <= '0';
223                         sel_in_mdr  <= '0';
224                         save_mdr        <= '0';
225                         save_c      <= '0';
226                         next_state_temp  <= '0';
227                         escape      <= '0';
228
229 ----------------------State 0011 0011----------------------------
230                     elsif(interrupt = "0011" and interrupt_1 = "0011") then
231                         sa          <= "00";
232                         sb          <= "01";
233                         S               <= "0000";
234                         sel_out_mdr <= "00";
235                         SDV         <= '0';
236                         EDV         <= '0';
237                         save_ir     <= '0';
238                         RW          <= '0';
239                         ME          <= '0';
240                         save        <= '1';
241                         en_out_a    <= '0';
242                         en_out_b    <= '0';
243                         save_sr         <= '0';
244                         en_SR       <= '0';
245                         rst_sr      <= '0';
246                         pc_out          <= '0';
247                         reset_i     <= '0';
248                         save_mar        <= '1';
249                         en_out_mar  <= '0';
250                         sel             <= '0';
251                         sel_in_mdr  <= '0';
252                         save_mdr        <= '0';
253                         save_c      <= '0';
254                         next_state_temp  <= '0';
255                         escape      <= '0';
256
257 ----------------------State 0010 0011 first time--------------------
258                     elsif(interrupt = "0010" and interrupt_1 = "0011" and second = '1') then
259                         sa          <= "11";
260                         sb          <= "00";
261                         S                   <= "0000";
262                         sel_out_mdr <= "00";
263                         SDV         <= '0';
264                         EDV         <= '0';
265                         save_ir     <= '0';
266                         RW          <= '0';
267                         ME          <= '0';
268                         save        <= '0';
269                         en_out_a    <= '1';
270                         en_out_b    <= '0';
271                         save_sr         <= '0';
272                         en_SR       <= '0';
273                         rst_sr      <= '0';
274                         pc_out          <= '0';
275                         reset_i     <= '0';
276                         save_mar        <= '0';
```

```
277                         en_out_mar  <= '0';
278                         sel             <= '0';
279                         sel_in_mdr  <= '0';
280                         save_mdr        <= '0';
281                         save_c      <= '0';
282                         next_state_temp  <= '0';
283                         escape      <= '0';
284
285 ----------State 0010 0010 first time----------
286                     elsif(interrupt = "0010" and interrupt_1 = "0010" and second = '1') then
287                         sa          <= "11";
288                         sb          <= "00";
289                         S                   <= "0000";
290                         sel_out_mdr <= "00";
291                         SDV         <= '0';
292                         EDV         <= '0';
293                         save_ir     <= '0';
294                         RW          <= '0';
295                         ME          <= '0';
296                         save        <= '0';
297                         en_out_a    <= '1';
298                         en_out_b    <= '0';
299                         save_sr         <= '0';
300                         en_SR       <= '0';
301                         rst_sr      <= '0';
302                         pc_out          <= '0';
303                         reset_i     <= '0';
304                         save_mar        <= '0';
305                         en_out_mar  <= '0';
306                         sel             <= '0';
307                         sel_in_mdr  <= '0';
308                         save_mdr        <= '0';
309                         save_c      <= '1';
310                         next_state_temp  <= '0';
311                         escape      <= '0';
312
313 ----------State 0010 0011 second time----------
314                     elsif(interrupt = "0010" and interrupt_1 = "0011" and second = '0') then
315                         sa          <= "00";
316                         sb          <= "00";
317                         S                   <= "0000";
318                         sel_out_mdr <= "00";
319                         SDV         <= '0';
320                         EDV         <= '0';
321                         save_ir     <= '0';
322                         RW          <= '0';
323                         ME          <= '0';
324                         save        <= '0';
325                         en_out_a    <= '1';
326                         en_out_b    <= '0';
327                         save_sr         <= '0';
328                         en_SR       <= '0';
329                         rst_sr      <= '0';
330                         pc_out          <= '0';
331                         reset_i     <= '0';
332                         save_mar        <= '0';
```

```
333                         en_out_mar  <= '0';
334                         sel             <= '0';
335                         sel_in_mdr  <= '0';
336                         save_mdr        <= '0';
337                         save_c      <= '0';
338                         next_state_temp  <= '0';
339                         escape      <= '0';
340
341 ---------------------State 0010 0010 second time-------------
342                     elsif(interrupt = "0010" and interrupt_1 = "0010" and second = '0') then
343                         sa          <= "00";
344                         sb          <= "00";
345                         S                   <= "0000";
346                         sel_out_mdr <= "00";
347                         SDV         <= '0';
348                         EDV         <= '0';
349                         save_ir     <= '0';
350                         RW          <= '0';
351                         ME          <= '0';
352                         save        <= '0';
353                         en_out_a    <= '1';
354                         en_out_b    <= '0';
355                         save_sr         <= '0';
356                         en_SR       <= '0';
357                         rst_sr      <= '0';
358                         pc_out          <= '0';
359                         reset_i     <= '0';
360                         save_mar        <= '0';
361                         en_out_mar  <= '0';
362                         sel             <= '0';
363                         sel_in_mdr  <= '0';
364                         save_mdr        <= '0';
365                         save_c      <= '1';
366                         next_state_temp  <= '0';
367                         escape      <= '0';
368
369 ---------------State 0110 0010 first time--------------------
370                     elsif(interrupt = "0110" and interrupt_1 = "0010" and second = '1') then
371                         sa          <= "00";
372                         sb          <= "00";
373                         S                   <= "0000";
374                         sel_out_mdr <= "00";
375                         SDV         <= '0';
376                         EDV         <= '0';
377                         save_ir     <= '0';
378                         RW          <= '0';
379                         ME          <= '0';
380                         save        <= '0';
381                         en_out_a    <= '0';
382                         en_out_b    <= '0';
383                         save_sr         <= '0';
384                         en_SR       <= '0';
385                         rst_sr      <= '1';
386                         pc_out          <= '0';
387                         reset_i     <= '0';
388                         save_mar        <= '0';
```

```
389                         en_out_mar  <= '0';
390                         sel             <= '0';
391                         sel_in_mdr  <= '0';
392                         save_mdr        <= '0';
393                         save_c      <= '0';
394                         next_state_temp  <= '0';
395                         escape      <= '0';
396
397 ---------------State 0110 0110 first time--------------------
398                 elsif(interrupt = "0110" and interrupt_1 = "0110" and second = '1') then
399                         sa          <= "00";
400                         sb          <= "00";
401                         S                   <= "0000";
402                         sel_out_mdr <= "11";
403                         SDV         <= '0';
404                         EDV         <= '0';
405                         save_ir     <= '0';
406                         RW          <= '0';
407                         ME          <= '1';
408                         save        <= '0';
409                         en_out_a    <= '0';
410                         en_out_b    <= '0';
411                         save_sr         <= '0';
412                         en_SR       <= '0';
413                         rst_sr      <= '1';
414                         pc_out          <= '0';
415                         reset_i     <= '0';
416                         save_mar        <= '0';
417                         en_out_mar  <= '1';
418                         sel             <= '0';
419                         sel_in_mdr  <= '0';
420                         save_mdr        <= '1';
421                         save_c      <= '0';
422                         next_state_temp  <= '0';
423                         escape      <= '0';
424
425 -------------State 0110 0010----second time-------------
426                 elsif(interrupt = "0110" and interrupt_1 = "0010" and second = '0') then
427                         sa          <= "00";
428                         sb          <= "00";
429                         S                   <= "0000";
430                         sel_out_mdr <= "00";
431                         SDV         <= '0';
432                         EDV         <= '0';
433                         save_ir     <= '0';
434                         RW          <= '0';
435                         ME          <= '0';
436                         save        <= '0';
437                         en_out_a    <= '0';
438                         en_out_b    <= '0';
439                         save_sr         <= '0';
440                         en_SR       <= '0';
441                         rst_sr      <= '0';
442                         pc_out          <= '0';
443                         reset_i     <= '0';
444                         save_mar        <= '0';
```

```
445                         en_out_mar  <= '0';
446                         sel             <= '0';
447                         sel_in_mdr  <= '0';
448                         save_mdr        <= '0';
449                         save_c      <= '0';
450                         next_state_temp  <= '0';
451                         escape      <= '0';
452
453 ------------State 0110 0110----second time------------
454                 elsif(interrupt = "0110" and interrupt_1 = "0110" and second = '0') then
455                         sa          <= "00";
456                         sb          <= "00";
457                         S                   <= "0000";
458                         sel_out_mdr <= "11";
459                         SDV         <= '0';
460                         EDV         <= '0';
461                         save_ir     <= '0';
462                         RW          <= '0';
463                         ME          <= '1';
464                         save        <= '0';
465                         en_out_a    <= '0';
466                         en_out_b    <= '0';
467                         save_sr         <= '0';
468                         en_SR       <= '0';
469                         rst_sr      <= '0';
470                         pc_out          <= '0';
471                         reset_i     <= '0';
472                         save_mar        <= '0';
473                         en_out_mar  <= '1';
474                         sel             <= '0';
475                         sel_in_mdr  <= '0';
476                         save_mdr        <= '1';
477                         save_c      <= '0';
478                         next_state_temp  <= '0';
479                         escape      <= '0';
480
481 ----------------State 0001 0110--------------------------
482                 elsif(interrupt = "0001" and interrupt_1 = "0110") then
483                         sa          <= "01";
484                         sb          <= "00";
485                         S                   <= "1001";
486                         sel_out_mdr <= "00";
487                         SDV         <= '0';
488                         EDV         <= '0';
489                         save_ir     <= '0';
490                         RW          <= '0';
491                         ME          <= '0';
492                         save        <= '0';
493                         en_out_a    <= '1';
494                         en_out_b    <= '0';
495                         save_sr         <= '0';
496                         en_SR       <= '0';
497                         rst_sr      <= '0';
498                         pc_out          <= '0';
499                         reset_i     <= '0';
500                         save_mar        <= '0';
```

```
501                         en_out_mar  <= '0';
502                         sel             <= '0';
503                         sel_in_mdr  <= '0';
504                         save_mdr        <= '0';
505                         save_c      <= '0';
506                         next_state_temp  <= '0';
507                         escape      <= '0';
508
509 ----------State 0100 0110----------
510                     elsif(interrupt = "0100" and interrupt_1 = "0110") then
511                         sa          <= "00";
512                         sb          <= "00";
513                         S                   <= "0000";
514                         sel_out_mdr <= "00";
515                         SDV         <= '1';
516                         EDV         <= '1';
517                         save_ir     <= '0';
518                         RW          <= '0';
519                         ME          <= '0';
520                         save        <= '0';
521                         en_out_a    <= '0';
522                         en_out_b    <= '0';
523                         save_sr         <= '0';
524                         en_SR       <= '0';
525                         rst_sr      <= '0';
526                         pc_out          <= '0';
527                         reset_i     <= '0';
528                         save_mar        <= '0';
529                         en_out_mar  <= '0';
530                         sel             <= '1';
531                         sel_in_mdr  <= '0';
532                         save_mdr        <= '0';
533                         save_c      <= '0';
534                         next_state_temp  <= '0';
535                         escape      <= '0';
536
537 ----------State 0100 0100----------
538                     elsif(interrupt = "0100" and interrupt_1 = "0100") then
539                         sa          <= "00";
540                         sb          <= "00";
541                         S                   <= "0000";
542                         sel_out_mdr <= "00";
543                         SDV         <= '1';
544                         EDV         <= '1';
545                         save_ir     <= '0';
546                         RW          <= '1';
547                         ME          <= '1';
548                         save        <= '0';
549                         en_out_a    <= '0';
550                         en_out_b    <= '0';
551                         save_sr         <= '0';
552                         en_SR       <= '0';
553                         rst_sr      <= '0';
554                         pc_out          <= '0';
555                         reset_i     <= '0';
556                         save_mar        <= '1';
```

```
557                         en_out_mar  <= '1';
558                         sel             <= '1';
559                         sel_in_mdr  <= '1';
560                         save_mdr        <= '0';
561                         save_c      <= '0';
562                         next_state_temp  <= '0';
563                         escape      <= '0';
564
565 ----------State 0101 0100----------
566                 elsif(interrupt = "0101" and interrupt_1 = "0100") then
567                         sa          <= "00";
568                         sb          <= "00";
569                         S                   <= "0000";
570                         sel_out_mdr <= "00";
571                         SDV         <= '0';
572                         EDV         <= '0';
573                         save_ir     <= '0';
574                         RW          <= '1';
575                         ME          <= '1';
576                         save        <= '0';
577                         en_out_a    <= '0';
578                         en_out_b    <= '0';
579                         save_sr         <= '0';
580                         en_SR       <= '0';
581                         rst_sr      <= '0';
582                         pc_out          <= '0';
583                         reset_i     <= '0';
584                         save_mar        <= '0';
585                         en_out_mar  <= '1';
586                         sel             <= '0';
587                         sel_in_mdr  <= '1';
588                         save_mdr        <= '0';
589                         save_c      <= '0';
590                         next_state_temp  <= '0';
591                         escape      <= '0';
592
593 ----------State 0101 0101----------
594                 elsif(interrupt = "0101" and interrupt_1 = "0101") then
595                         sa          <= "00";
596                         sb          <= "00";
597                         S                   <= "0000";
598                         sel_out_mdr <= "01";
599                         SDV         <= '0';
600                         EDV         <= '0';
601                         save_ir     <= '0';
602                         RW          <= '1';
603                         ME          <= '1';
604                         save        <= '0';
605                         en_out_a    <= '0';
606                         en_out_b    <= '0';
607                         save_sr         <= '0';
608                         en_SR       <= '0';
609                         rst_sr      <= '0';
610                         pc_out          <= '0';
611                         reset_i     <= '0';
612                         save_mar        <= '0';
```

```
613                         en_out_mar  <= '1';
614                         sel             <= '0';
615                         sel_in_mdr  <= '1';
616                         save_mdr        <= '1';
617                         save_c      <= '0';
618                         next_state_temp  <= '0';
619                         escape      <= '0';
620
621 ----------State 0111 0101----------
622                     elsif(interrupt = "0111" and interrupt_1 = "0101") then
623                         sa          <= "00";
624                         sb          <= "00";
625                         S                   <= "0000";
626                         sel_out_mdr <= "01";
627                         SDV         <= '0';
628                         EDV         <= '0';
629                         save_ir     <= '0';
630                         RW          <= '0';
631                         ME          <= '0';
632                         save        <= '0';
633                         en_out_a    <= '0';
634                         en_out_b    <= '0';
635                         save_sr         <= '0';
636                         en_SR       <= '0';
637                         rst_sr      <= '0';
638                         pc_out          <= '0';
639                         reset_i     <= '0';
640                         save_mar        <= '0';
641                         en_out_mar  <= '0';
642                         sel             <= '0';
643                         sel_in_mdr  <= '0';
644                         save_mdr        <= '0';
645                         save_c      <= '0';
646                         next_state_temp  <= '0';
647                         escape      <= '0';
648
649 ----------State 0111 0111----------
650                     elsif(interrupt = "0111" and interrupt_1 = "0111") then
651                         sa          <= "00";
652                         sb          <= "00";
653                         S                   <= "0000";
654                         sel_out_mdr <= "01";
655                         SDV         <= '0';
656                         EDV         <= '0';
657                         save_ir     <= '0';
658                         RW          <= '0';
659                         ME          <= '0';
660                         save        <= '0';
661                         en_out_a    <= '0';
662                         en_out_b    <= '0';
663                         save_sr         <= '0';
664                         en_SR       <= '0';
665                         rst_sr      <= '0';
666                         pc_out          <= '0';
667                         reset_i     <= '0';
668                         save_mar        <= '0';
```

```
669                         en_out_mar  <= '0';
670                         sel             <= '0';
671                         sel_in_mdr  <= '0';
672                         save_mdr        <= '0';
673                         save_c      <= '1';
674                         next_state_temp  <= '0';
675                         escape      <= '0';
676
677 ----------State 1111 0111----------
678                 elsif(interrupt = "1111" and interrupt_1 = "0111") then
679                         sa          <= "00";
680                         sb          <= "00";
681                         S                   <= "0000";
682                         sel_out_mdr <= "00";
683                         SDV         <= '0';
684                         EDV         <= '0';
685                         save_ir     <= '0';
686                         RW          <= '0';
687                         ME          <= '0';
688                         save        <= '0';
689                         en_out_a    <= '0';
690                         en_out_b    <= '0';
691                         save_sr         <= '0';
692                         en_SR       <= '0';
693                         rst_sr      <= '0';
694                         pc_out          <= '0';
695                         reset_i     <= '0';
696                         save_mar        <= '0';
697                         en_out_mar  <= '0';
698                         sel             <= '0';
699                         sel_in_mdr  <= '0';
700                         save_mdr        <= '0';
701                         save_c      <= '0';
702                         next_state_temp  <= '0';
703                         escape      <= '0';
704
705 ----------State 1111 1111----------
706                 elsif(interrupt = "1111" and interrupt_1 = "1111") then
707                         sa          <= "00";
708                         sb          <= "00";
709                         S                   <= "0000";
710                         sel_out_mdr <= "00";
711                         SDV         <= '0';
712                         EDV         <= '0';
713                         save_ir     <= '0';
714                         RW          <= '0';
715                         ME          <= '0';
716                         save        <= '1';
717                         en_out_a    <= '0';
718                         en_out_b    <= '0';
719                         save_sr         <= '0';
720                         en_SR       <= '0';
721                         rst_sr      <= '0';
722                         pc_out          <= '0';
723                         reset_i     <= '0';
724                         save_mar        <= '0';
```

```
725                         en_out_mar  <= '0';
726                         sel             <= '0';
727                         sel_in_mdr  <= '0';
728                         save_mdr        <= '0';
729                         save_c      <= '0';
730                         next_state_temp  <= '1';
731                         escape      <= '0';
732
733 ---------------------State 0000 1111----------------------------
734                     elsif(interrupt = "0000" and interrupt_1 = "1111") then
735                         sa          <= "00";
736                         sb          <= "00";
737                         S                   <= "0000";
738                         sel_out_mdr <= "00";
739                         SDV         <= '0';
740                         EDV         <= '0';
741                         save_ir     <= '0';
742                         RW          <= '0';
743                         ME          <= '0';
744                         save        <= '0';
745                         en_out_a    <= '0';
746                         en_out_b    <= '0';
747                         save_sr         <= '0';
748                         en_SR       <= '0';
749                         rst_sr      <= '0';
750                         pc_out          <= '0';
751                         reset_i     <= '0';
752                         save_mar        <= '0';
753                         en_out_mar  <= '0';
754                         sel             <= '0';
755                         sel_in_mdr  <= '0';
756                         save_mdr        <= '0';
757                         save_c      <= '0';
758                         next_state_temp  <= '1';
759                         escape      <= '0';
760
761 ---------------------State 0000 0111----------------------------
762                     elsif(interrupt = "0000" and interrupt_1 = "0111") then
763                         sa          <= "00";
764                         sb          <= "00";
765                         S                   <= "0000";
766                         sel_out_mdr <= "00";
767                         SDV         <= '0';
768                         EDV         <= '0';
769                         save_ir     <= '0';
770                         RW          <= '0';
771                         ME          <= '0';
772                         save        <= '0';
773                         en_out_a    <= '0';
774                         en_out_b    <= '0';
775                         save_sr         <= '0';
776                         en_SR       <= '0';
777                         rst_sr      <= '0';
778                         pc_out          <= '0';
779                         reset_i     <= '0';
780                         save_mar        <= '0';
```

```
781                         en_out_mar  <= '0';
782                         sel             <= '0';
783                         sel_in_mdr  <= '0';
784                         save_mdr        <= '0';
785                         save_c      <= '0';
786                         next_state_temp  <= '1';
787                         escape      <= '0';
788
789 ----------State 0000 0101----------
790                     elsif(interrupt = "0000" and interrupt_1 = "0101") then
791                         sa          <= "00";
792                         sb          <= "00";
793                         S                   <= "0000";
794                         sel_out_mdr <= "00";
795                         SDV         <= '0';
796                         EDV         <= '0';
797                         save_ir     <= '0';
798                         RW          <= '0';
799                         ME          <= '0';
800                         save        <= '0';
801                         en_out_a    <= '0';
802                         en_out_b    <= '0';
803                         save_sr         <= '0';
804                         en_SR       <= '0';
805                         rst_sr      <= '0';
806                         pc_out          <= '0';
807                         reset_i     <= '0';
808                         save_mar        <= '0';
809                         en_out_mar  <= '0';
810                         sel             <= '0';
811                         sel_in_mdr  <= '0';
812                         save_mdr        <= '0';
813                         save_c      <= '0';
814                         next_state_temp  <= '1';
815                         escape      <= '0';
816
817 ----------State 0000 0001----------
818                     elsif(interrupt = "0000" and interrupt_1 = "0001") then
819                         sa          <= "00";
820                         sb          <= "00";
821                         S                   <= "0000";
822                         sel_out_mdr <= "00";
823                         SDV         <= '0';
824                         EDV         <= '0';
825                         save_ir     <= '0';
826                         RW          <= '0';
827                         ME          <= '0';
828                         save        <= '0';
829                         en_out_a    <= '0';
830                         en_out_b    <= '0';
831                         save_sr         <= '0';
832                         en_SR       <= '0';
833                         rst_sr      <= '0';
834                         pc_out          <= '0';
835                         reset_i     <= '0';
836                         save_mar        <= '0';
```

```
837                         en_out_mar  <= '0';
838                         sel             <= '0';
839                         sel_in_mdr  <= '0';
840                         save_mdr        <= '0';
841                         save_c      <= '0';
842                         next_state_temp  <= '1';
843                         escape      <= '0';
844
845 ----------------------State 0000 0011----------------------------
846                 elsif(interrupt = "0000" and interrupt_1 = "0011") then
847                         sa          <= "00";
848                         sb          <= "00";
849                         S                   <= "0000";
850                         sel_out_mdr <= "00";
851                         SDV         <= '0';
852                         EDV         <= '0';
853                         save_ir     <= '0';
854                         RW          <= '0';
855                         ME          <= '0';
856                         save        <= '0';
857                         en_out_a    <= '0';
858                         en_out_b    <= '0';
859                         save_sr         <= '0';
860                         en_SR       <= '0';
861                         rst_sr      <= '0';
862                         pc_out          <= '0';
863                         reset_i     <= '0';
864                         save_mar        <= '0';
865                         en_out_mar  <= '0';
866                         sel             <= '0';
867                         sel_in_mdr  <= '0';
868                         save_mdr        <= '0';
869                         save_c      <= '0';
870                         next_state_temp  <= '1';
871                         escape      <= '0';
872
873 ----------------------State 0000 0010----------------------------
874                 elsif(interrupt = "0000" and interrupt_1 = "0010") then
875                         sa          <= "00";
876                         sb          <= "00";
877                         S                   <= "0000";
878                         sel_out_mdr <= "00";
879                         SDV         <= '0';
880                         EDV         <= '0';
881                         save_ir     <= '0';
882                         RW          <= '0';
883                         ME          <= '0';
884                         save        <= '0';
885                         en_out_a    <= '0';
886                         en_out_b    <= '0';
887                         save_sr         <= '0';
888                         en_SR       <= '0';
889                         rst_sr      <= '0';
890                         pc_out          <= '0';
891                         reset_i     <= '0';
892                         save_mar        <= '0';
```

```
893                         en_out_mar  <= '0';
894                         sel             <= '0';
895                         sel_in_mdr  <= '0';
896                         save_mdr        <= '0';
897                         save_c      <= '0';
898                         next_state_temp  <= '1';
899                         escape      <= '0';
900
901 ----------------------State 0000 0110----------------------------
902                 elsif(interrupt = "0000" and interrupt_1 = "0110") then
903                         sa          <= "00";
904                         sb          <= "00";
905                         S                   <= "0000";
906                         sel_out_mdr <= "00";
907                         SDV         <= '0';
908                         EDV         <= '0';
909                         save_ir     <= '0';
910                         RW          <= '0';
911                         ME          <= '0';
912                         save        <= '0';
913                         en_out_a    <= '0';
914                         en_out_b    <= '0';
915                         save_sr         <= '0';
916                         en_SR       <= '0';
917                         rst_sr      <= '0';
918                         pc_out          <= '0';
919                         reset_i     <= '0';
920                         save_mar        <= '0';
921                         en_out_mar  <= '0';
922                         sel             <= '0';
923                         sel_in_mdr  <= '0';
924                         save_mdr        <= '0';
925                         save_c      <= '0';
926                         next_state_temp  <= '1';
927                         escape      <= '0';
928
929 ----------------------State 0000 0100----------------------------
930                 elsif(interrupt = "0000" and interrupt_1 = "0100") then
931                         sa          <= "00";
932                         sb          <= "00";
933                         S                   <= "0000";
934                         sel_out_mdr <= "00";
935                         SDV         <= '0';
936                         EDV         <= '0';
937                         save_ir     <= '0';
938                         RW          <= '0';
939                         ME          <= '0';
940                         save        <= '0';
941                         en_out_a    <= '0';
942                         en_out_b    <= '0';
943                         save_sr         <= '0';
944                         en_SR       <= '0';
945                         rst_sr      <= '0';
946                         pc_out          <= '0';
947                         reset_i     <= '0';
948                         save_mar        <= '0';
```

```
949                         en_out_mar  <= '0';
950                         sel             <= '0';
951                         sel_in_mdr  <= '0';
952                         save_mdr        <= '0';
953                         save_c      <= '0';
954                         next_state_temp  <= '1';
955                         escape      <= '0';
956
957  ----------State 0000 1000----------
958                 elsif(interrupt = "0000" and interrupt_1 = "1000") then
959                         sa          <= "00";
960                         sb          <= "00";
961                         S                   <= "0000";
962                         sel_out_mdr <= "00";
963                         SDV         <= '0';
964                         EDV         <= '0';
965                         save_ir     <= '0';
966                         RW          <= '0';
967                         ME          <= '0';
968                         save        <= '0';
969                         en_out_a    <= '0';
970                         en_out_b    <= '0';
971                         save_sr         <= '0';
972                         en_SR       <= '0';
973                         rst_sr      <= '0';
974                         pc_out          <= '0';
975                         reset_i     <= '0';
976                         save_mar        <= '0';
977                         en_out_mar  <= '0';
978                         sel             <= '0';
979                         sel_in_mdr  <= '0';
980                         save_mdr        <= '0';
981                         save_c      <= '0';
982                         next_state_temp  <= '1';
983                         escape      <= '0';
984
985  ----------State 0000 1001----------
986                 elsif(interrupt = "0000" and interrupt_1 = "1001") then
987                         sa          <= "00";
988                         sb          <= "00";
989                         S                   <= "0000";
990                         sel_out_mdr <= "00";
991                         SDV         <= '0';
992                         EDV         <= '0';
993                         save_ir     <= '0';
994                         RW          <= '0';
995                         ME          <= '0';
996                         save        <= '0';
997                         en_out_a    <= '0';
998                         en_out_b    <= '0';
999                         save_sr         <= '0';
1000                        en_SR       <= '0';
1001                        rst_sr      <= '0';
1002                        pc_out          <= '0';
1003                        reset_i     <= '0';
1004                        save_mar        <= '0';
```

```
1005                            en_out_mar  <= '0';
1006                            sel             <= '0';
1007                            sel_in_mdr  <= '0';
1008                            save_mdr        <= '0';
1009                            save_c      <= '0';
1010                            next_state_temp  <= '1';
1011                            escape      <= '0';
1012
1013  ----------------------State 0000 1010----------------------------
1014                    elsif(interrupt = "0000" and interrupt_1 = "1010") then
1015                            sa          <= "00";
1016                            sb          <= "00";
1017                            S                   <= "0000";
1018                            sel_out_mdr <= "00";
1019                            SDV         <= '0';
1020                            EDV         <= '0';
1021                            save_ir     <= '0';
1022                            RW          <= '0';
1023                            ME          <= '0';
1024                            save        <= '0';
1025                            en_out_a    <= '0';
1026                            en_out_b    <= '0';
1027                            save_sr         <= '0';
1028                            en_SR       <= '0';
1029                            rst_sr      <= '0';
1030                            pc_out          <= '0';
1031                            reset_i     <= '0';
1032                            save_mar        <= '0';
1033                            en_out_mar  <= '0';
1034                            sel             <= '0';
1035                            sel_in_mdr  <= '0';
1036                            save_mdr        <= '0';
1037                            save_c      <= '0';
1038                            next_state_temp  <= '1';
1039                            escape      <= '0';
1040
1041  ----------------------State 0000 1011----------------------------
1042                    elsif(interrupt = "0000" and interrupt_1 = "1011") then
1043                            sa          <= "00";
1044                            sb          <= "00";
1045                            S                   <= "0000";
1046                            sel_out_mdr <= "00";
1047                            SDV         <= '0';
1048                            EDV         <= '0';
1049                            save_ir     <= '0';
1050                            RW          <= '0';
1051                            ME          <= '0';
1052                            save        <= '0';
1053                            en_out_a    <= '0';
1054                            en_out_b    <= '0';
1055                            save_sr         <= '0';
1056                            en_SR       <= '0';
1057                            rst_sr      <= '0';
1058                            pc_out          <= '0';
1059                            reset_i     <= '0';
1060                            save_mar        <= '0';
```

```
1061                            en_out_mar  <= '0';
1062                            sel             <= '0';
1063                            sel_in_mdr  <= '0';
1064                            save_mdr        <= '0';
1065                            save_c      <= '0';
1066                            next_state_temp  <= '1';
1067                            escape      <= '0';
1068
1069  ----------------------State 0000 1100----------------------------
1070                    elsif(interrupt = "0000" and interrupt_1 = "1100") then
1071                            sa          <= "00";
1072                            sb          <= "00";
1073                            S                   <= "0000";
1074                            sel_out_mdr <= "00";
1075                            SDV         <= '0';
1076                            EDV         <= '0';
1077                            save_ir     <= '0';
1078                            RW          <= '0';
1079                            ME          <= '0';
1080                            save        <= '0';
1081                            en_out_a    <= '0';
1082                            en_out_b    <= '0';
1083                            save_sr         <= '0';
1084                            en_SR       <= '0';
1085                            rst_sr      <= '0';
1086                            pc_out          <= '0';
1087                            reset_i     <= '0';
1088                            save_mar        <= '0';
1089                            en_out_mar  <= '0';
1090                            sel             <= '0';
1091                            sel_in_mdr  <= '0';
1092                            save_mdr        <= '0';
1093                            save_c      <= '0';
1094                            next_state_temp  <= '1';
1095                            escape      <= '0';
1096
1097  ----------------------State 0000 1101----------------------------
1098                    elsif(interrupt = "0000" and interrupt_1 = "1101") then
1099                            sa          <= "00";
1100                            sb          <= "00";
1101                            S                   <= "0000";
1102                            sel_out_mdr <= "00";
1103                            SDV         <= '0';
1104                            EDV         <= '0';
1105                            save_ir     <= '0';
1106                            RW          <= '0';
1107                            ME          <= '0';
1108                            save        <= '0';
1109                            en_out_a    <= '0';
1110                            en_out_b    <= '0';
1111                            save_sr         <= '0';
1112                            en_SR       <= '0';
1113                            rst_sr      <= '0';
1114                            pc_out          <= '0';
1115                            reset_i     <= '0';
1116                            save_mar        <= '0';
```

```
1117                        en_out_mar  <= '0';
1118                        sel             <= '0';
1119                        sel_in_mdr  <= '0';
1120                        save_mdr        <= '0';
1121                        save_c      <= '0';
1122                        next_state_temp  <= '1';
1123                        escape      <= '0';
1124
1125  ----------------------State 0000 1110----------------------------
1126                  elsif(interrupt = "0000" and interrupt_1 = "1110") then
1127                        sa          <= "00";
1128                        sb          <= "00";
1129                        S                   <= "0000";
1130                        sel_out_mdr <= "00";
1131                        SDV         <= '0';
1132                        EDV         <= '0';
1133                        save_ir     <= '0';
1134                        RW          <= '0';
1135                        ME          <= '0';
1136                        save        <= '0';
1137                        en_out_a    <= '0';
1138                        en_out_b    <= '0';
1139                        save_sr         <= '0';
1140                        en_SR       <= '0';
1141                        rst_sr      <= '0';
1142                        pc_out          <= '0';
1143                        reset_i     <= '0';
1144                        save_mar        <= '0';
1145                        en_out_mar  <= '0';
1146                        sel             <= '0';
1147                        sel_in_mdr  <= '0';
1148                        save_mdr        <= '0';
1149                        save_c      <= '0';
1150                        next_state_temp  <= '1';
1151                        escape      <= '0';
1152
1153  ----------------------State 0000 1110----------------------------
1154                  else
1155                        sa          <= "00";
1156                        sb          <= "00";
1157                        S                   <= "0000";
1158                        sel_out_mdr <= "00";
1159                        SDV         <= '0';
1160                        EDV         <= '0';
1161                        save_ir     <= '0';
1162                        RW          <= '0';
1163                        ME          <= '0';
1164                        save        <= '0';
1165                        en_out_a    <= '0';
1166                        en_out_b    <= '0';
1167                        save_sr         <= '0';
1168                        en_SR       <= '0';
1169                        rst_sr      <= '0';
1170                        pc_out          <= '0';
1171                        reset_i     <= '0';
1172                        save_mar        <= '0';
```

```
1173                            en_out_mar  <= '0';
1174                            sel             <= '0';
1175                            sel_in_mdr  <= '0';
1176                            save_mdr        <= '0';
1177                            save_c      <= '0';
1178                            next_state_temp  <= '1';
1179                            escape      <= '0';
1180
1181                    end if;
1182    ------------------------------------------------------------
1183            end process;
1184
1185    ----------------------interrupt signnal preservation---------
1186
1187
1188            i <= '1' when interrupt = "0001" and interrupt_1 = "0000" and int_temp = '1' and reset =
                        '0' else '0';
1189
1190            process(i, next_state_temp, interrupt_1, interrupt)
1191            begin
1192                    if(next_state_temp = '1' or (interrupt = "0000" and interrupt_1 = "0000")) then
1193                            int_temp_1 <= '0';
1194                    elsif(i'event and i = '1') then
1195                            int_temp_1 <= '1';
1196                    end if;
1197
1198            end process;
1199
1200            int_temp <= int or int_temp_1;
1201            next_state <= next_state_temp;
1202    end is_1;
```

B.2.27 Interrupt State Machine

```
1   --
        ------------------------------------------------------------------------------------------------------------------------

2   --
3   -- Title        : ism
4   -- Design       : ALU
5   -- Author       : Notebook
6   -- Company      : Home
7   --
8   --
        ------------------------------------------------------------------------------------------------------------------------

9   --
10  -- File         : Interrupt_sm.vhd
11  -- Generated    : Sun Feb  1 23:16:07 2009
12  -- From         : interface description file
13  -- By           : Itf2Vhdl ver. 1.20
14  --
15  --
        ------------------------------------------------------------------------------------------------------------------------

16  --
```

```
17 -- Description :
18 --
19 --
     ------------------------------------------------------------------------------------------------------------------

20
21 --{{ Section below this comment is automatically maintained
22 --   and may be overwritten
23 --{entity {ism} architecture {ism_1}}
24
25 library IEEE;
26 use IEEE.STD_LOGIC_1164.all;
27
28 entity ism is
29          port(
30                  I : in STD_LOGIC;
31                 MA : in STD_LOGIC;
32                  clk : in STD_LOGIC;
33                  rst : in STD_LOGIC;
34                  state : out STD_LOGIC_VECTOR(3 downto 0);
35                  nstate : out STD_LOGIC_VECTOR(3 downto 0);
36                  second : out std_logic
37              );
38 end ism;
39
40 --}} End of automatically maintained section
41
42 architecture ism_1 of ism is
43         signal state_temp, nstate_temp : std_logic_vector(3 downto 0);
44         signal T, tempin, tempout, clk_1 : std_logic;
45 begin
46
47         tempin <= tempout xor '1';
48         T <= tempout;
49         clk_1 <= '1' When state_temp = "0011" and nstate_temp = "0011" else '0';
50
51         process (clk_1, rst)
52         begin
53                 if (rst = '1') then
54                         tempout <= '0';
55                 elsif (clk_1'EVENT AND clk_1 = '1') then
56                         tempout <= tempin;
57                 end if;
58         end process;
59
60         process (clk, rst)
61         begin
62                 if (rst = '1') then
63                         state_temp <= "0000";
64                 elsif (clk'EVENT AND clk='1') then
65                         if (state_temp="0000" and I = '0') then
66                                 state_temp <= "0000";
67                         elsif (state_temp="0000" and I = '1') then
68                                 state_temp <= "0001";
69                         elsif (state_temp="0001" and I = '0') then
70                                 state_temp <= "0000";
```

```
71                          elsif (state_temp="0001" and I = '1') then
72                                  state_temp <= "0011";
73                          elsif (state_temp="0011" and I = '0') then
74                                  state_temp <= "0000";
75                          elsif (state_temp="0011" and I = '1') then
76                                  state_temp <= "0010";
77                          elsif (state_temp="0010" and I = '0') then
78                                  state_temp <= "0000";
79                          elsif (state_temp="0010" and I = '1') then
80                                  state_temp <= "0110";
81                          elsif (state_temp="0110" and I = '0') then
82                                  state_temp <= "0000";
83                          elsif (state_temp="0110" and I = '1' and MA = '0') then
84                                  state_temp <= "0110";
85                          elsif (state_temp="0110" and I = '1' AND MA = '1' and T = '1') then
86                                  state_temp <= "0001";
87                          elsif (state_temp="0110" and I = '1' and MA = '1' AND T = '0') then
88                                  state_temp <= "0100";
89                          elsif (state_temp="0100" and I = '0') then
90                                  state_temp <= "0000";
91                          elsif (state_temp="0100" and I = '1' and MA = '0') then
92                                  state_temp <= "0100";
93                          elsif (state_temp="0100" and I = '1' and MA = '1') then
94                                  state_temp <= "0101";
95                          elsif (state_temp="0101" and I = '0' ) then
96                                  state_temp <= "0000";
97                          elsif (state_temp="0101" and I = '1') then
98                                  state_temp <= "0111";
99                          elsif (state_temp="0111" and I = '0') then
100                                 state_temp <= "0000";
101                         elsif (state_temp="0111" and I = '1') then
102                                 state_temp <= "1111";
103                         elsif (state_temp="1111") then
104                                 state_temp <= "0000";
105                         elsif (state_temp="1000") then
106                                 state_temp <= "0000";
107                         elsif (state_temp="1001") then
108                                 state_temp <= "0000";
109                         elsif (state_temp="1010") then
110                                 state_temp <= "0000";
111                         elsif (state_temp="1011") then
112                                 state_temp <= "0000";
113                         elsif (state_temp="1100") then
114                                 state_temp <= "0000";
115                         elsif (state_temp="1101") then
116                                 state_temp <= "0000";
117                         elsif (state_temp="1110") then
118                                 state_temp <= "0000";
119                         end if;
120                 end if;
121         end process;
122
123         process (clk, rst)
124         begin
125                 if (rst = '1') then
126                         nstate_temp <= "0000";
```

```
127                 elsif (clk'EVENT AND clk='0') then
128                         nstate_temp <= state_temp;
129                 end if;
130
131         end process;
132                 state <= state_temp;
133                 nstate<= nstate_temp;
134                 second <= T;
135
136 end ism_1;
```

Reference List

[1] American Hearth Association. Heart Disease and Stroke Statistics-2008 Update, 2008.

[2] Stphane Rinfret MD. MSc; David J. Cohen MD. MSc; Gervasio A. Lamas MD.; Kirsten E. Fleischmann MD. MPH; Milton C. Weinstein PhD; John Orav PhD; Eleanor Schron MS RN; Kerry L. Lee PhD; Lee Goldman MD. Cost-Effectiveness of Dual-Chamber Pacing Compared With Ventricular Pacing for Sinus Node Dysfunction. American Hearth Association, 1996. Circulation 111: 165-172.

[3] FRCP Stuart J. Connolly MD. FRCPC; Charles Kerr MD. FRCPC; Michael Gent DSc; Salim Yusuf, DPhil. Dual-Chamber Versus Ventricular Pacing: Critical Appraisal of Current Data. American Hearth Association, 2005. Circulation 94: 578-583.

[4] MSP430x1xx Family Users Guide. Texas Instruments Inc., 2006.

[5] Sigfredo Gonzalez-Diaz. Methodology, Design, and Implementation of a Cardiac Pacemaker Prototype Using a Commercial Low Power Microcontroller. Master's thesis, Electrical Engineering Department, University Of Puerto Rico Mayaguez Campus, 2006.

[6] Seymour Furman. *Clinical Cardiac Pacing*. Saunders, pages 112–125, 1995.

[7] L.S.Y. Wong, S. Hossain, A. Ta, L. Weaver, G. Shaquer, A. Walker, J. Edvinsson, D. Rivas, H. Naas, A. Fawzi, A. Uhrenius, J. Lindberg, J. Johansson, and P. Arvidsson. A Very Low Power CMOS Mixed-Signal IC for Implantable Pacemaker Applications. *Solid-State Circuits Conference, 2004. Digest of Technical Papers. ISSCC. 2004 IEEE International*, pages 318–530 Vol.1, 15-19 Feb. 2004.

[8] John G. Webster. *Medical Instrumentation Application and Design*. Wiley, third edition, 2002.

[9] David Shier; Jackie Bulter; Ricki Lewis. *Hole's human Anathomy & Physiology*. McGrawHill, eighth edition, 1999.

[10] S.A.P. Haddad, R.P.M. Houben, and W.A. Serdijin. The Evolution of Pacemakers. *Engineering in Medicine and Biology Magazine, IEEE*, 25(3):38–48, May-June 2006.

[11] S. Serge Barold; Roland X. Stroobandt; Alfons F. Sinnaeve. *Cardiac Pacemaker Step by Step, An ilustrated Guiide*. Blackwell, 2004.

[12] Vincent P. Heuring; Harry F. Jordan. *Computer System Design and Architecture*. Pearson Prentice Hall, second edition, 2004.

[13] D.J. Woollons. To beat or not to beat: the history and development of heart pacemakers. *Engineering Science and Education Journal*, 4(6):259–268, Dec 1995.

[14] Gutirrez Fuster Efrén. Evolución de los Marcapasos y de la Estimulación Eléctrica del Corazón, Arch. Cardiol. Mx. [periódico en la Internet]. Available in: http://scielo.unam.mx/scielo.php?script=sci_arttext&pid=S1405-99402005000300001&lng=es&nrm=iso, Sept. 2005. [visited June 17,2008].

[15] J. Berkman and J. Prak. Biomedical Microprocessor with Analog I/O. *Solid-State Circuits Conference. Digest of Technical Papers. 1981 IEEE International*, XXIV:168–169, Feb 1981.

[16] L. Stotts, Jr. Baker, R., and J. Miner. An 8b Microcomputer for Implantable Biomedical Applications. *Solid-State Circuits Conference. Digest of Technical Papers. 1985 IEEE International*, XXVIII:14–15, Feb 1985.

[17] L.J. Stotts and K.R. Infinger. An 8-bit Microcomputer with Analog Subsystems for Implantable Biomedical Applications. *Custom Integrated Circuits Conference, 1988., Proceedings of the IEEE 1988*, pages 4.6/1–4.6/3, 16-19 May 1988.

[18] L.J. Stotts, K.R. Infinger, J. Babka, and D. Genzer. An 8-bit Microcomputer With Analog Subsystems for Implantable Biomedical Application. *IEEE Journal of Solid-State Circuits*, 24(2):292–300, Apr 1989.

[19] D.B. Shaw and M. Horwood. Recording Pacemakers Memory Size-What Should be Recorded. *Intelligent Cardiac Implants, IEE Colloquium on*, pages 3/1–3/4, Jan 1993.

[20] R.S. Sanders and M.T. Lee. Implantable Pacemakers. *Proceedings of the IEEE*, 84(3):480–486, Mar 1996.

[21] Eli Ovsyshcher MD PhD; David L. Hayes MD; S. Furman MD. Dual-Chamber Pacing Is Superior to Ventricular Pacing. American Hearth Association, 1998. http://circ.ahajournals.org/cgi/content/full/97/23/2368.

[22] CCC. TEROS SSI 503 Product Specifications. Centro de Construcción de Cardioestimuladores de Uruguay S.A., 2008. http://www.ccc.com.uy/pacemaker/ddd4000.htm.

[23] T. Jamil. RISC versus CISC. *Potentials, IEEE*, 14(3):13–16, Aug/Sep 1995.

[24] S. Nikolaidis; N. Kavvadias; and P.Neofotistos. Instruction level power measurements and analysis. *Dekiverable, Aristotle University of Thessaloniki,*, 2002. http://electronics.physics.auth.gr/easy/.

[25] VLSI and Standard Cell Links. Oklahoma State University VLSI Computer Architecture Research, 2009. http://avatar.ecen.okstate.edu/projects/scells/links.php.

[26] Xilinx University Program Virtex-II Pro Development System Hardware Reference Manual v1.0. Xilinx, March 8, 2005.

[27] A.P. Chandrakasan, R. Allmon, A. Stratakos, and R.W. Brodersen. Design of portable systems. pages 259–266, May 1994.