UNIVERSITY OF MINHO
SCHOOL OF ENGINEERING

Master in Industrial Electronics and Computers Engineering

8051

ESRG

Authors:
Luís Coelho, PG50568
Rui Gomes, PG50735

Supervisor:
Adriano Tavares

Guimarães, December 2022

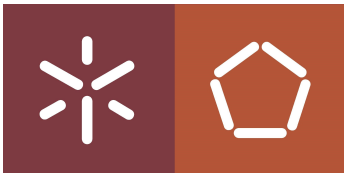
Contents

1 Introduction **8**
1.1 Problem Statement . . . 8
1.2 Market Research . . . 9

2 Analysis **12**
2.1 Requirements . . . 12
2.2 Constraints . . . 12
2.3 System Overview . . . 12
2.4 State Diagram . . . 13

3 Design **15**
3.1 Control Unit . . . 15
3.1.1 State Machine . . . 15
3.1.2 Instructions . . . 17
3.2 Datapath . . . 17
3.2.1 ROM . . . 17
3.2.2 RAM and Stack . . . 17
3.2.3 SFRs . . . 18
3.2.4 ALU . . . 18
3.3 Peripherals . . . 19
3.3.1 Clock . . . 19
3.3.2 Timer0 and Timer1 . . . 20
3.3.3 Clock Divider . . . 20
3.3.4 UART . . . 20
3.3.5 Debouncer . . . 21
3.3.6 Interrupts . . . 21

4 Implementation **24**
4.1 Diagrams . . . 24
4.1.1 ALU and the MUXs . . . 26
4.1.2 RAM . . . 27
4.1.3 ROM . . . 27
4.1.4 Timer SFRs . . . 28
4.1.5 Timers . . . 29
4.1.6 Clock Dividers . . . 30
4.1.7 Debouncer . . . 31



4.1.8 Uart 32
4.2 Behaviour Tests 33
4.2.1 PSW/ALU/Instructions Test 33
4.2.2 LCALL and RET Test 33
4.2.3 Timer Overflow Test 34
4.2.4 Interrupt Test 34
4.2.5 Uart Test 35
4.2.6 Timer 0 Overflow Test 35

5 Conclusion **37**
5.1 Work Done vs Defined Requirements 37

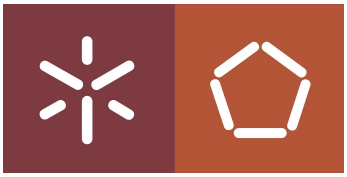
6 References **37**

List of Figures

1 ZYBO Z7-10 . . . 8
2 8051 from Intel . . . 8
3 8051 Datapath example . . . 10
4 8051 RAM example . . . 10
5 System Overview . . . 13
6 Simple State Machine . . . 13
7 State Machine . . . 15
8 Simple Flow Charts from C.U. behavior . . . 16
9 ALU Flow Chart . . . 19
10 TMOD register . . . 20
11 TCON Register . . . 20
12 Uart Protocol . . . 21
13 Debounce Example . . . 21
14 IE Register . . . 22
15 System Overview Vivado design from the code implemented . . . 24
16 Signals . . . 25
17 ALU and the MUXs . . . 26
18 RAM . . . 27
19 ROM . . . 27
20 Timer SFRs . . . 28
21 Timers Implementation . . . 29
22 Clock Dividers Implementation . . . 30
23 Clock Dividers Implementation Internally . . . 30
24 Debouncer Implementation . . . 31
25 Uart Implementation . . . 32
26 PSW/ALU/Instructions Test . . . 33
27 LCALL and RET Test . . . 33
28 Timer Overflow Test . . . 34
29 Interrupt Test . . . 34
30 Uart Test . . . 35

List of Tables

1 Instructions Implemented 17



Listings

1 SFR 18

Introduction

1 Introduction

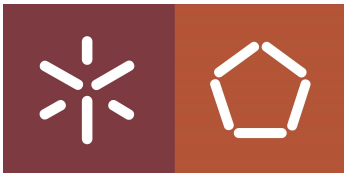
1.1 Problem Statement

Within the scope of the course of Advanced Microprocessor Architectures was sugested by the professor, the design and implementation of the microcontroller 8051 in verilog.

Wthi this in mind we have to implement it in a FPGA, for that we will be using the ZYBO Z7-10 since it was a constraint and we have been having classes about it and also other projects done with this same board like EC-1 and EC-2 during this semester.

Figure 1: ZYBO Z7-10

Figure 2: 8051 from Intel



1.2 Market Research

During research we found a lot different implementations of the 8051, but all follow the same logic so we went for the key points like:

- 8-bit ALU
- 8-bit ACC and PSW
- Program Counter and Instruction Register
- Total of 256 Instructions
- From no to 4/8 Kbytes ROM
- 128 bytes of on-chip RAM
- 128 bytes of SFRs
- 4 fast switchable register banks with eight registers each
- 5 to 6 Interrupts with 2-Level priority structure
- 2 to 3 16bits timers/event counters
- 32 IO ports
- Programmable full duplex serial ports
- Bit addressable RAM

Here is a example of a datapath(Intel 8051 MCS-51) and RAM:

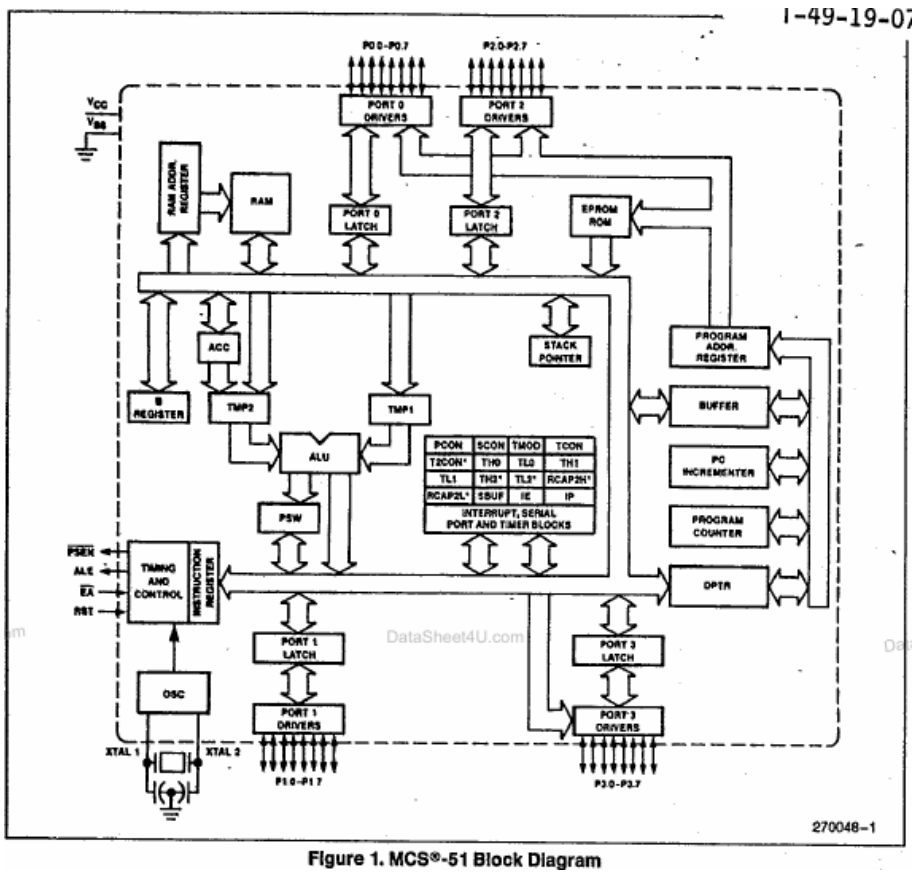


Figure 3: 8051 Datapath example

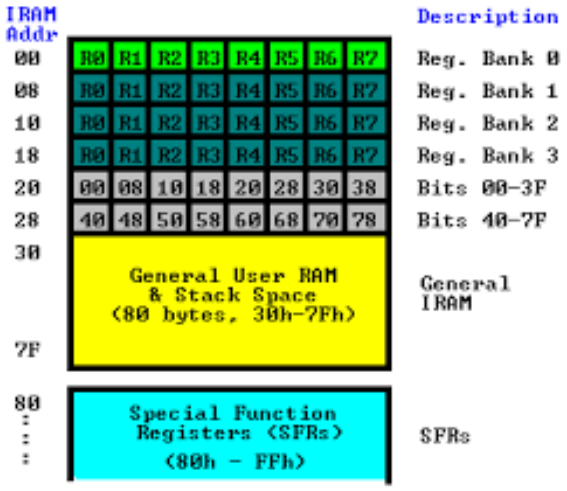
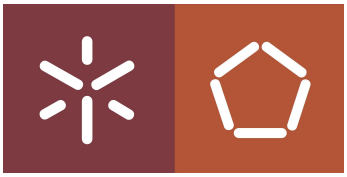


Figure 4: 8051 RAM example



Analysis

2 Analysis

2.1 Requirements

The professor set some requirements that we need to archive and we add some too, like:

- Implement interrupts
- Implement a peripheral(Timer for example)
- At leats 4 instruction from each type(arithmetic,logic,data transfer,jump in execution)
- Some type of serial port (Uart for example)
- Have at least 1MHz per instruction

2.2 Constraints

The constraints set by the professor are the following(we also set some):

- 8-bit word size
- Use Zybo 70XX
- Program in verilog on Vivado from Xilinx
- Instructions have all the same size

2.3 System Overview

The System overview that we thought that could achieve all the requirements proposed should look like this:

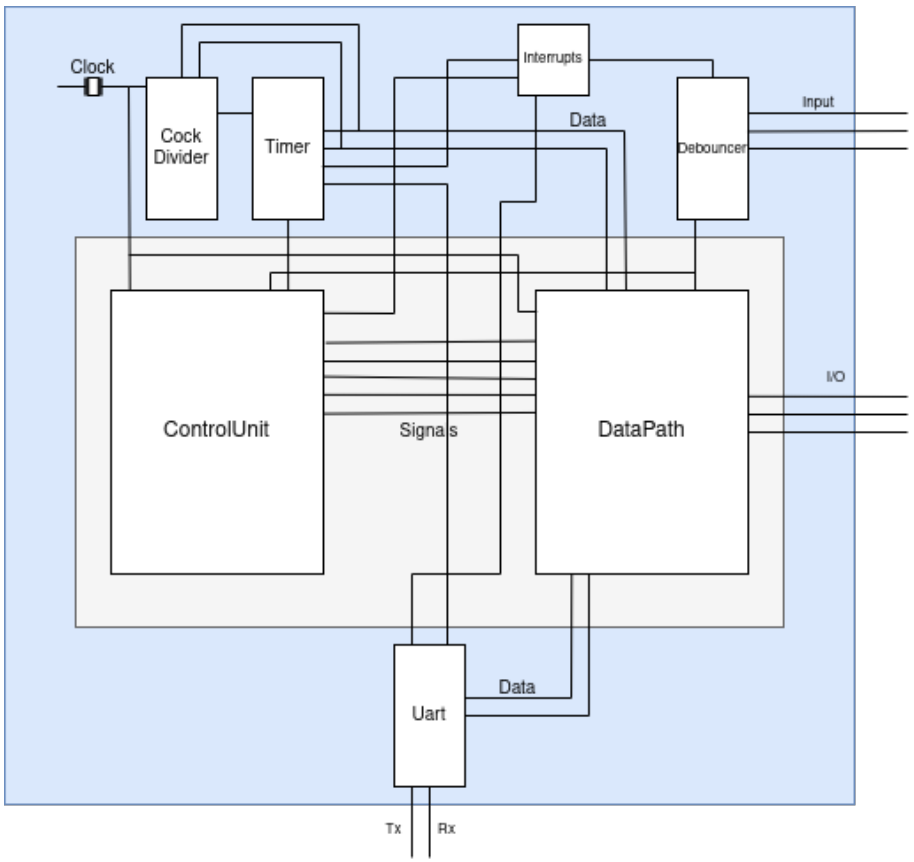


Figure 5: System Overview

2.4 State Diagram

This diagram represents the states needed to get the program executing the way we think is best and that will achieve the requirements, probably will be needed more states (Fetch1, Fetch2, Fetch3, for 3 bytes instructions for example)so we get everything in time and no multi machine cycles needed.

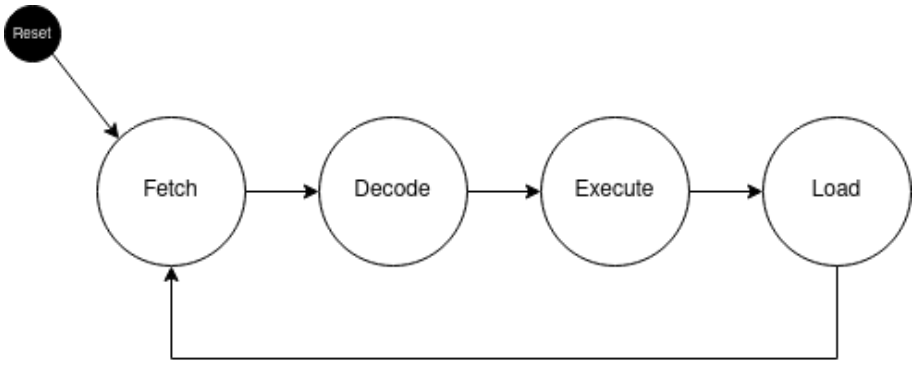


Figure 6: Simple State Machine

Design

3 Design

3.1 Control Unit

3.1.1 State Machine

In order to attend the necessities from the 3 bytes instructions and to let the ram process everything at time we decide to make a 8 states state machine, the 3 fetches are there to get the 3 bytes from that instructions that need them. The state wait is to let the RAM process enough time that is secure to get the data out of it , the state load was brought from the analysis because we found it use full for load everything that is needed at the end of each instruction.

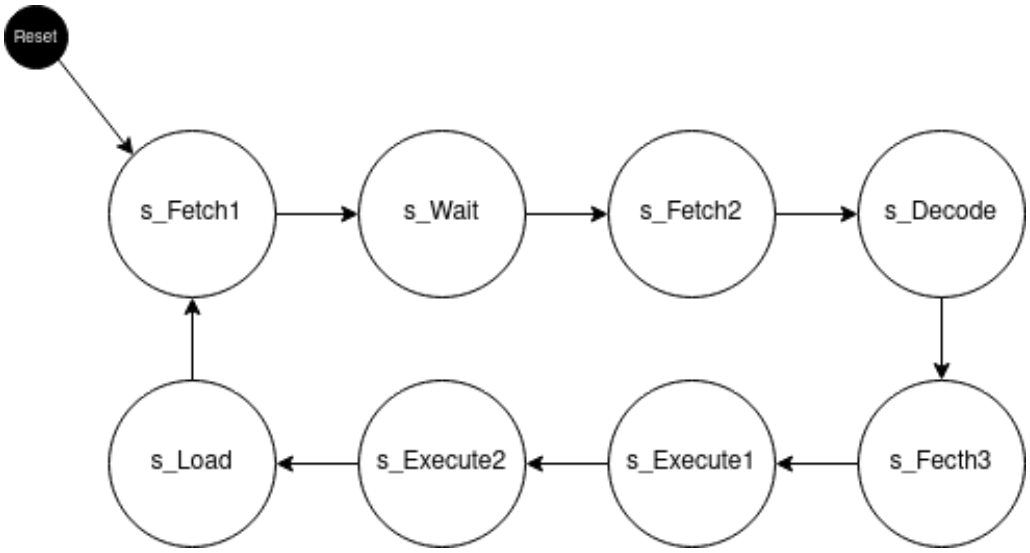


Figure 7: State Machine

The following flow charts represent the working of the control unit changing between states this state machine is a simple one since the actually one is too complex but follows the same thinking process.

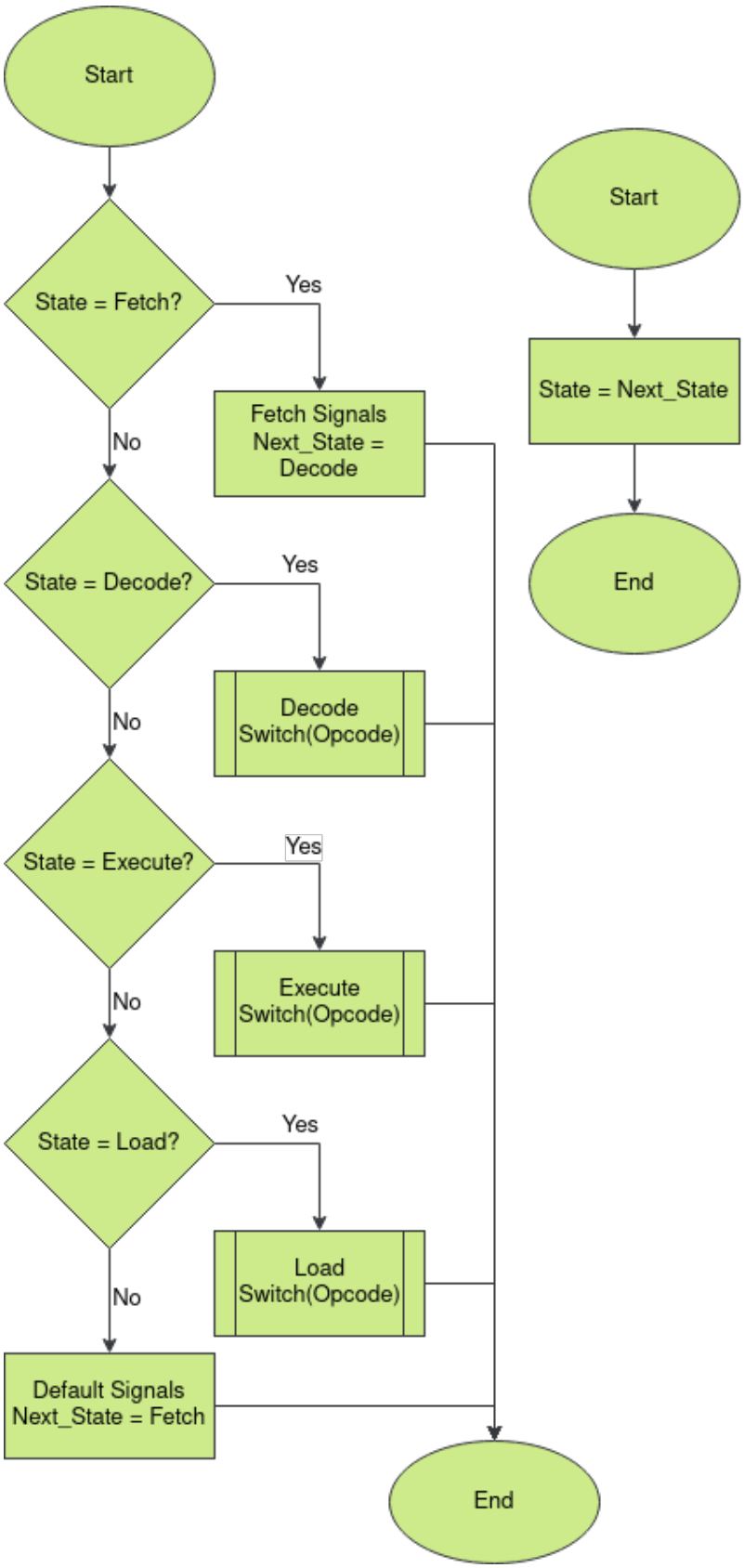


Figure 8: Simple Flow Charts from C.U. behavior

3.1.2 Instructions

X Means 0 -> 7 ins
AJMPX, INCRX, DECRX, ADDARX, ANLARX, MOVRXI, CJNERXIO

NOP	AJMPX	LJMP
RR	INCA	INCRX
LCALL	RRC	DECA
DECRX	RET	RL
ADDAI	ADDAD	ADDARX
RLC	ORLDI	ORLAI
ORLAD	ANLDI	ANLAI
ANLAD	ANLARX	JZ
XRLDI	JNZ	MOVAI
MOVDI	MOVRXI	SUBBAI
SUBBAD	CJNEAIO	CJNERXIO
CLRA	MOVAD	CPLA

Table 1: Instructions Implemented

So its a total of 85 instructions

3.2 Datapath

3.2.1 ROM

The ROM is where he save our code that the microcontroller can run the size of it is 4096 bytes, and the program counter run though all of the ROM.

3.2.2 RAM and Stack

The RAM and Stack are 2 separated things , that is different from the original 8051 that is unified, the size of the RAM is 128 bytes and the Stack as a height of 16 only since we only need to save the program counter, because the instruction push and pop will not be implemented. The registers R0 to R7 and all 4 banks need to be implemented.
We will do not have bit addressable RAM to bypass that we implement the ANLDI, ORLDI and XRLDI so the compiler can change only one bit, but it will be needed more processing, since the ALU will need to work and data will need to be transferred.

3.2.3 SFRs

The SFRs will be implemented direct in the datapath so they can be easily accessed by the the other components , but they must be addressable by the following addresses even though they aren't in the RAM stack.

```
`timescale 1ns / 1ps

//SFR Addresses
`define P0   8'H80
`define SP   8'H81
`define DPL  8'H82
`define DPH  8'H83
`define PCON 8'H87
`define TCON 8'H88
`define TMOD 8'H89
`define TL0  8'H8A
`define TL1  8'H8B
`define TH0  8'H8C
`define TH1  8'H8D
`define AUXR 8'H8E
`define P1   8'H90
`define SCON 8'H98
`define P2   8'HA0
`define IE   8'HA8
`define P3   8'HB0
`define IPH  8'HB7
`define IP   8'HB8
`define PSW  8'HD0
`define ACC  8'HE0
`define B    8'HF0
`define CKDIV 8'HF1
`define SBUF_TX 8'HF2
`define SBUF_RX 8'HF3
```

Listing 1: SFR

3.2.4 ALU

The following flow charts represent the ALU input there is only the ALU_IN1 but the other ALU input is equal and the ALU itself where the ALU_EN flag say that the inputs are ready and the the ALU can do the calculation defined by the opcode.

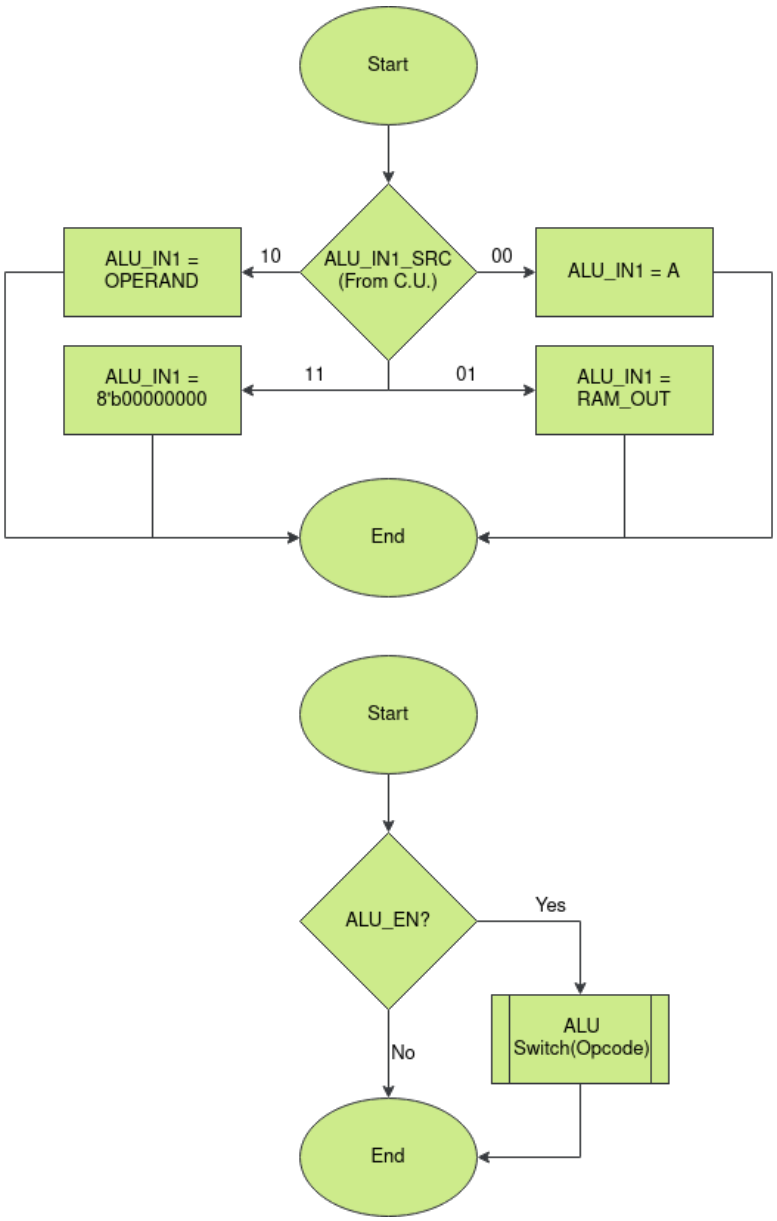


Figure 9: ALU Flow Chart

3.3 Peripherals

Here we will present the peripherals Timers , Clock Dividers, Uart and Debouncer that will be implement in the final 8051.

3.3.1 Clock

Our clock must run at 80MHz so we can get an instruction at 10MHz ,because we have 8 states per instruction, this clock will need to be divided for the timers, because some times we don't a clock that run that fast.

3.3.2 Timer0 and Timer1

We will have 2 Timers, Timer0 and Timer1 so can run UART with no problem and also have a extra timer to do what ever the user wants both timer will have only 2 mode 16 bit and 8-bit auto-reload, a flag must be turned on on every overflow just like 8051 timers, the register TCON, TMOD,TH1,TH0,TL1 and TL0 must be implemented so the timer run properly.

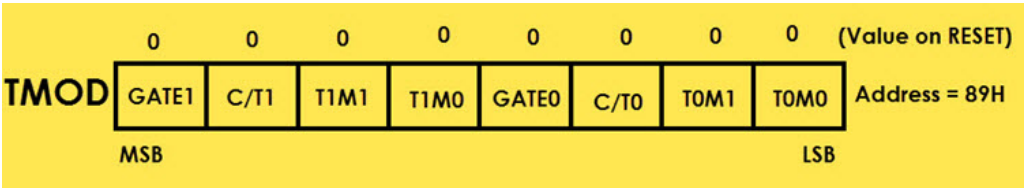


Figure 10: TMOD register

Figure 11: TCON Register

3.3.3 Clock Divider

Since our clock have 80MHz we need a clock divider to make less complex codes for a slower timer it will have at least 4 modes clock/2 ,clock/8, clock,2000 and clock/80000, this can be configured by changing the register CKDIV that is a SFR present at the address 0xF1.

3.3.4 UART

Uart will be our way of communication the universal asynchronous receiver/-transmitter it must implemented wthi a interrupt and reach at least 9600bps. The following image show the uart protocol, the interrupt is caused when the byte is received or transmited.

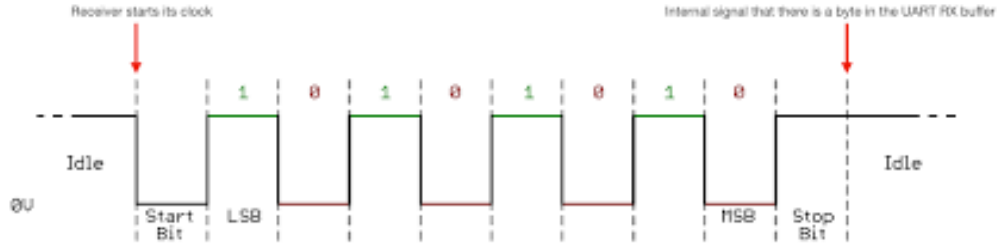


Figure 12: Uart Protocol

3.3.5 Debouncer

A debouncer must be implemented so the reset button or any other buttons that could exist have a software,this will be implemented in a sliding window method debounce so the value of it doesn't have variation in a initial stage, because we know that button can have that behavior some times like in the image.

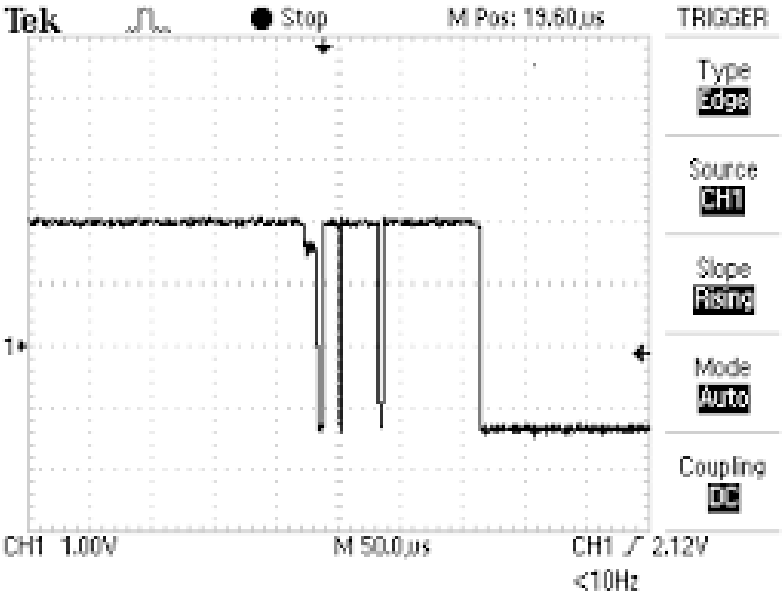


Figure 13: Debounce Example

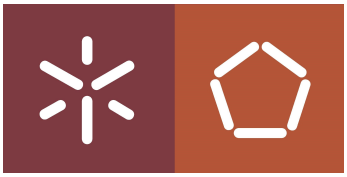
3.3.6 Interrupts

We will implement 4 Interrupts , Timer 0, Timer 1, EXT0 and UART, those interrupts will be changing the course of the program during run time, we also have the register IE(Interrutp Enable) that controls all the interrupts.

EA	–	–	ES	ET1	EX1	ET0	EX0

EA	IE.7	Disables all interrupts. If EA=0, no interrupt will be acknowledged. If EA=1, interrupt source is individually enable or disabled by setting or clearing its enable bit.
–	IE.6	Not implemented, reserved for future use*.
–	IE.5	Not implemented, reserved for future use*.
ES	IE.4	Enable or disable the Serial port interrupt.
ET1	IE.3	Enable or disable the Timer 1 overflow interrupt.
EX1	IE.2	Enable or disable External interrupt 1.
ET0	IE.1	Enable or disable the Timer 0 overflow interrupt.
EX0	IE.0	Enable or disable External interrupt 0.

Figure 14: IE Register



Implementation

4 Implementation

4.1 Diagrams

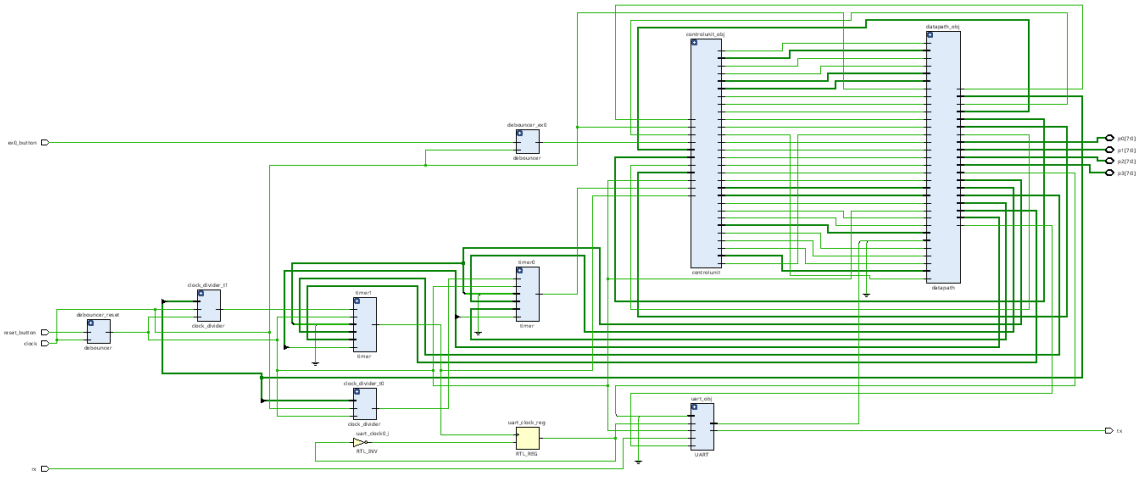


Figure 15: System Overview Vivado design from the code implemented

As we can see the design proposed by the vivado is some how identical to the system overview proposed at the analysis phase but with the obvious differences like 2 timers , 2 clock dividers ,the interrupts are inside the control unit and more signals and wires.

We saw a overview, now we will went in more detail.

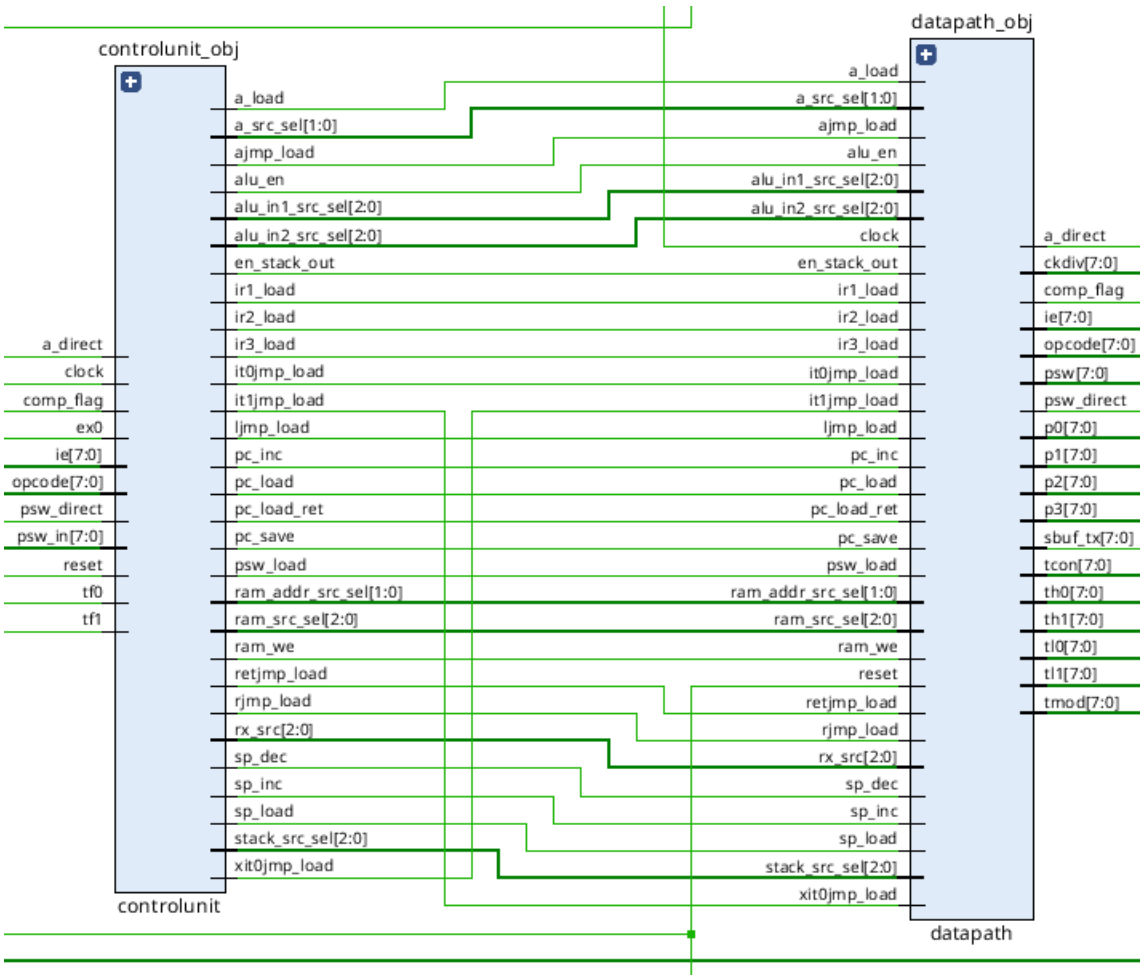


Figure 16: Signals

Here we can see the signals that the ControlUnit send to the DataPath so it works properly we can also see some flags that the DataPath send to the ControlUnit like comp_flag, that's the flag that compares register values.

Since the DataPath and ControlUnit is too complex there is no point in showing the Vivado Design.

4.1.1 ALU and the MUXs

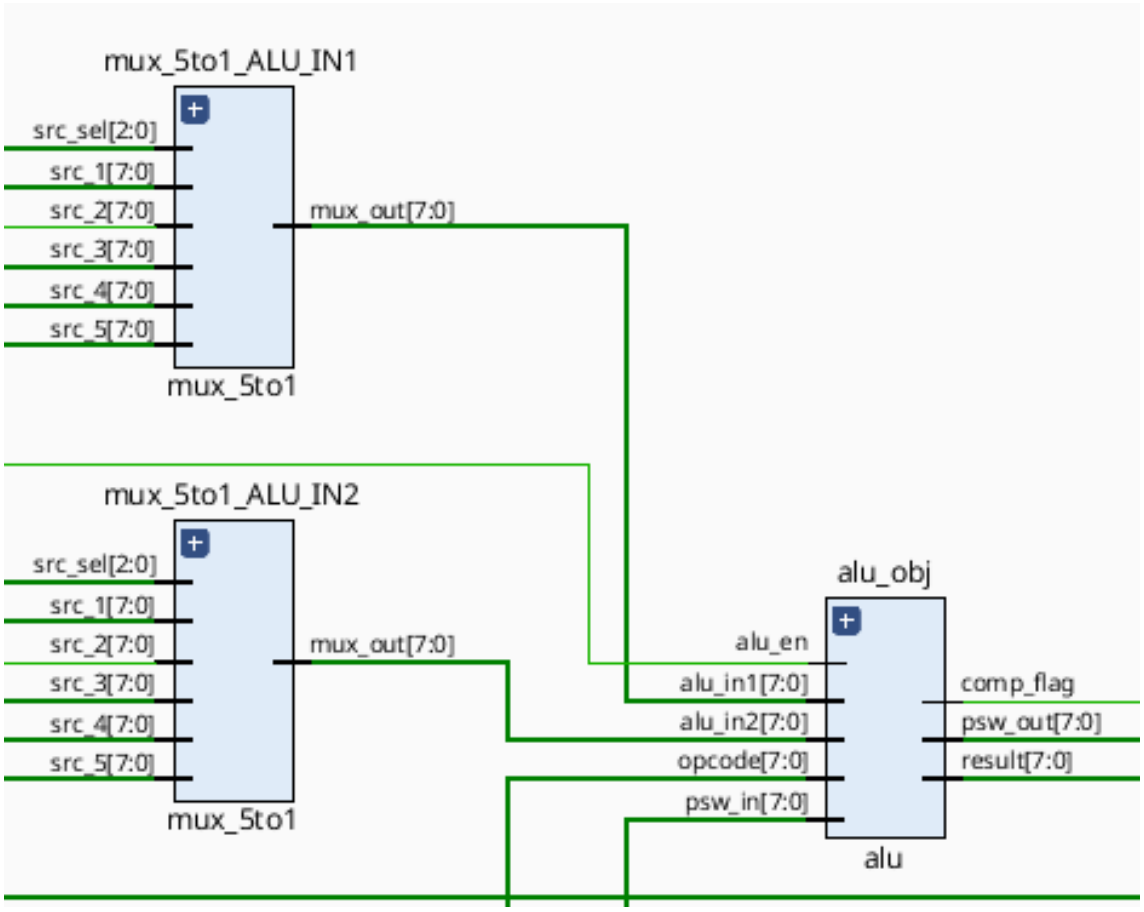


Figure 17: ALU and the MUXs

Here we have the ALU and the muxes presented in the desing but with a slightly difference because the muxes have 5 entrees.

4.1.2 RAM

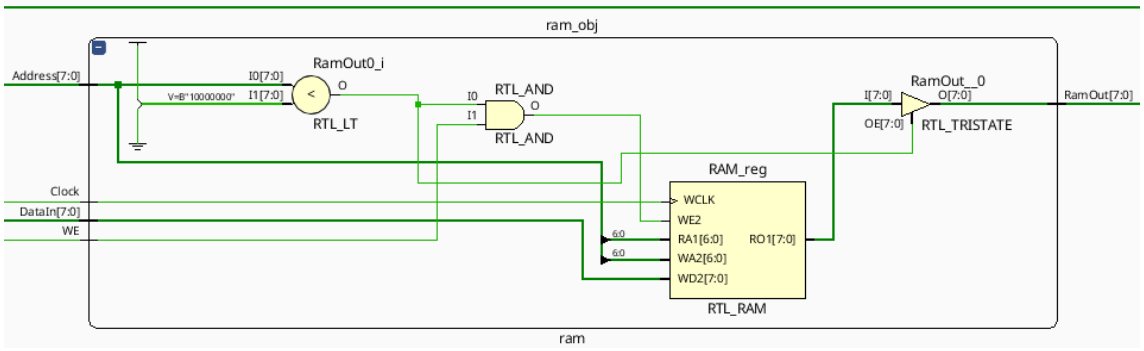


Figure 18: RAM

This image show the RAM inside with the registers and the Tristate-buffer at the way out and also some logic, because of the RX registers and all the 4 banks.

4.1.3 ROM

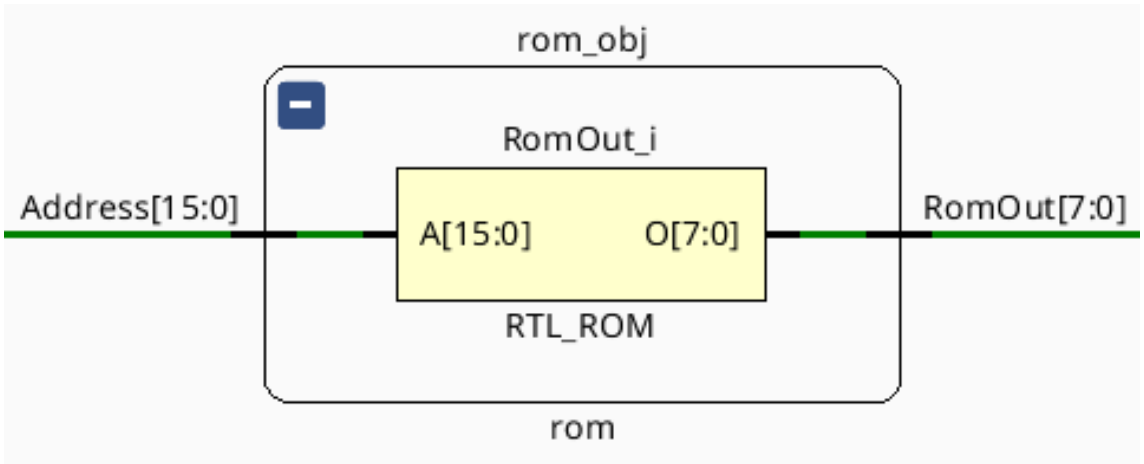


Figure 19: ROM

This is our ROM wthi only Program Counter connected and the entry and with only on exit the rom_out

4.1.4 Timer SFRs

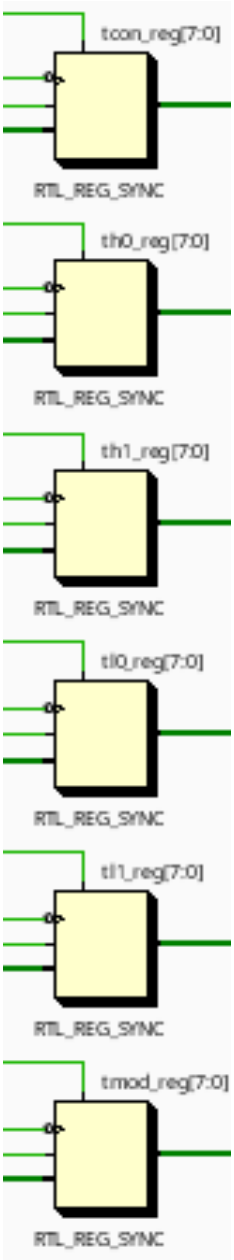


Figure 20: Timer SFRs

This are the SFRs used in the timers to configure them.

4.1.5 Timers

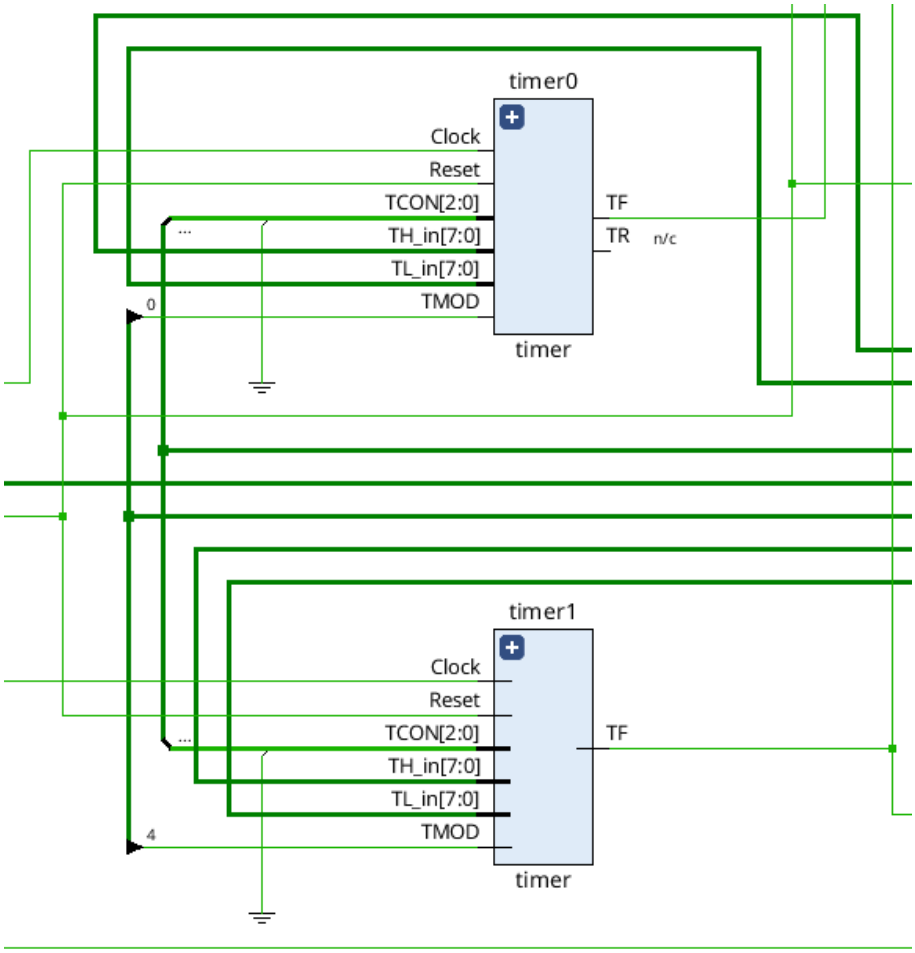


Figure 21: Timers Implementation

Here we can see the timers connected using the TMOD ,TCON, TH and TL SFRs just like in the design phase, the TF is the timer overflow that is connected to the ControlUnit so it can provoke the timer interrupt if the interrupt is enable by the IE register.

4.1.6 Clock Dividers

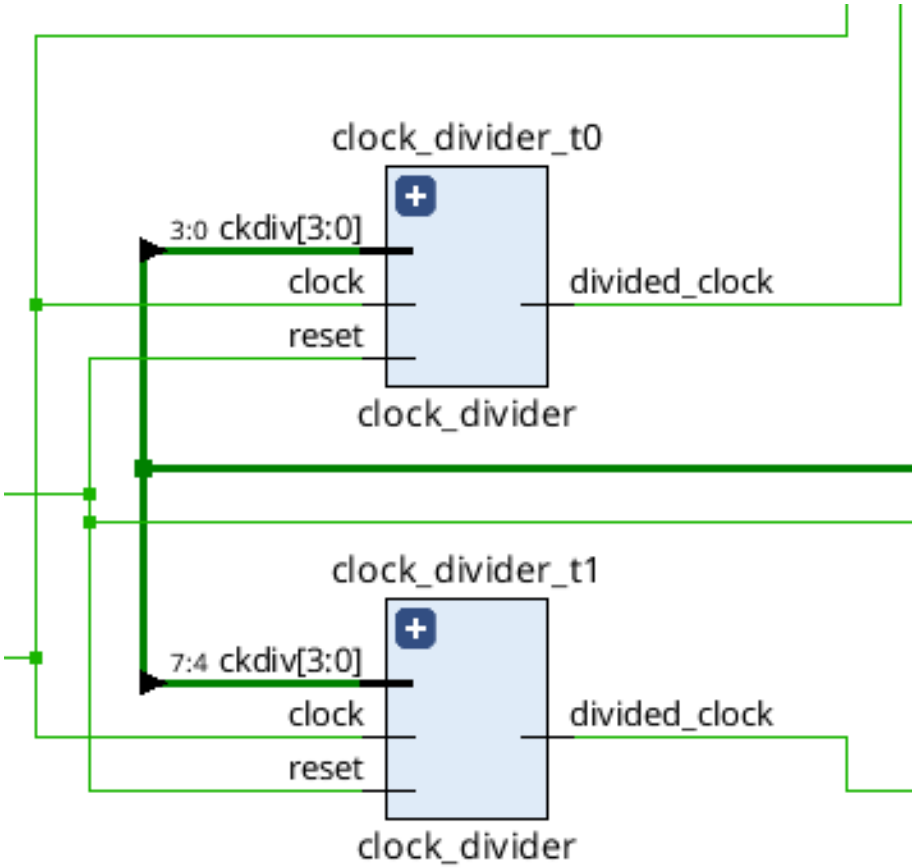


Figure 22: Clock Dividers Implementation

Here we can see the Clock Divider 0 and 1 that are conected to the timers 0 and 1 there is also the SFR CKDIV that is used to configure the clock division.

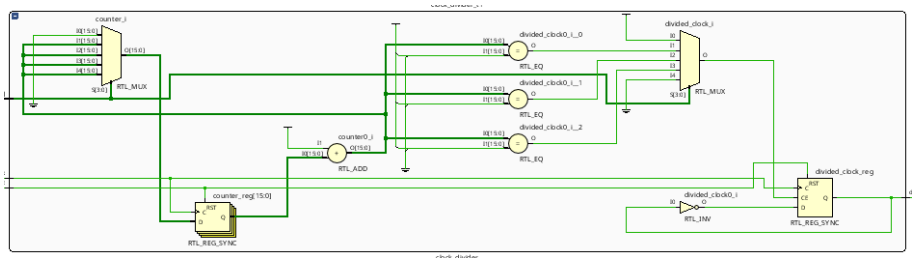


Figure 23: Clock Dividers Implementation Internally

Now we can see how the Vivado designed the clock divider internally we can

see clearly the multiplexer with the 4 modes and the counter that when it reaches a number the output is changed for the complement in the register as we can see.

4.1.7 Debouncer

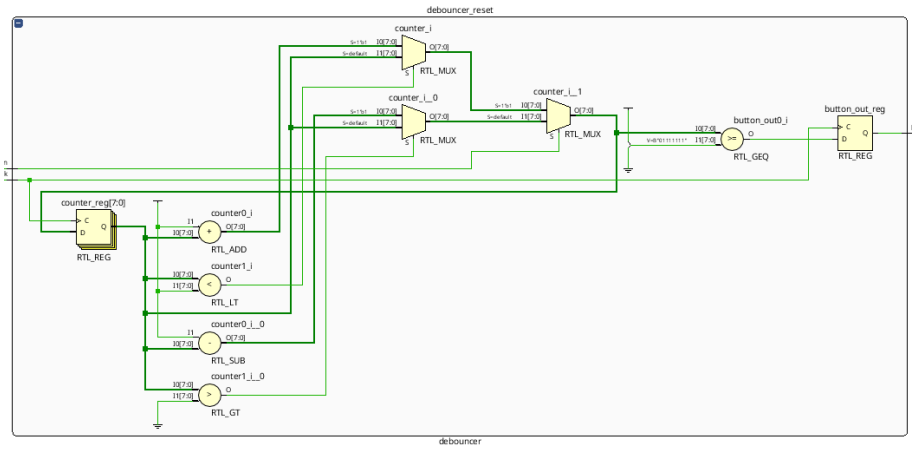


Figure 24: Debouncer Implementation

This is the debouncer that Vivado designed taking in consideration the code wrote in verilog we can see the counter in the debouncer and all the logic that makes only the output on when the counter reach a defined number.

4.1.8 Uart

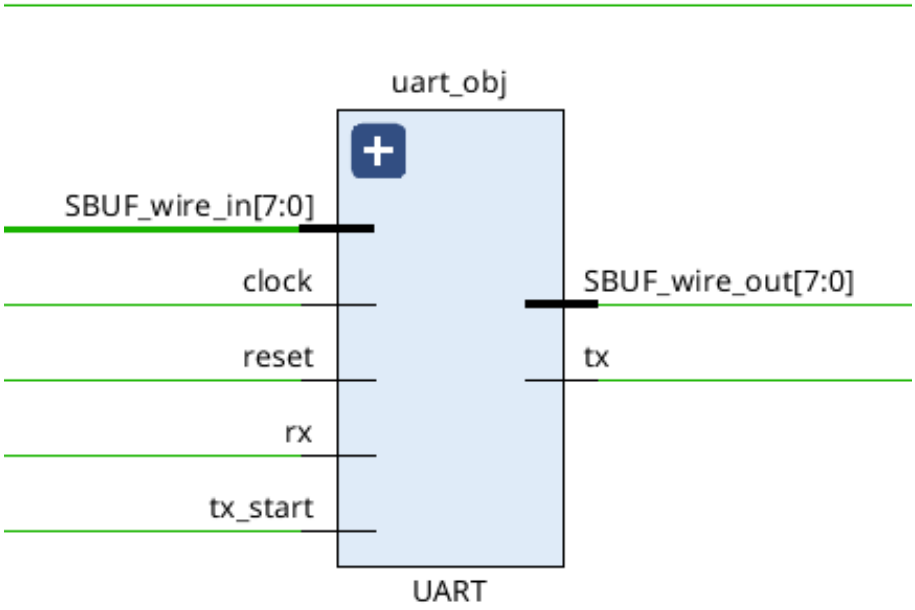


Figure 25: Uart Implementation

This is the block of the Uart design was we can see it receives a flag to start the transmission and it starts to receive when rx goes to 0 and sends the byte received in the sbuf wire.

4.2 Behaviour Tests

4.2.1 PSW/ALU/Instructions Test

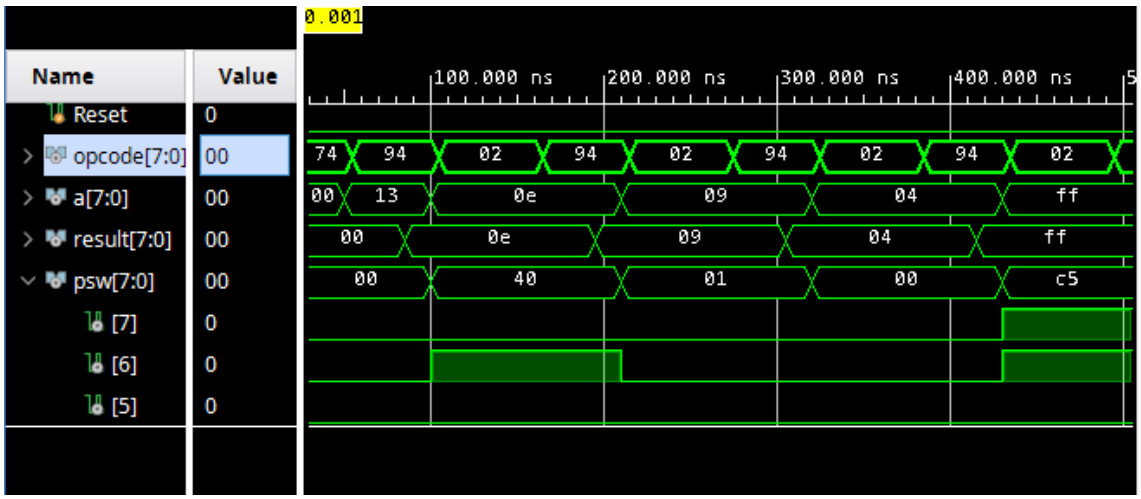


Figure 26: PSW/ALU/Instructions Test

Frist we change the A value to 0x13 with the 0x74 instruction(MOVAI) that we run instruction 0x94(SUBBAI) -5 and the flag PSW.6 changes to 1 because it "borrow" one form the upper nibble then is made a 0x02(LJMP) to the same instruction 0x94 and is subtracted again -5 and this repeat till it reaches 04H and it becomes ffH and both PSW.7 and PSW.6 are set to 1.

4.2.2 LCALL and RET Test

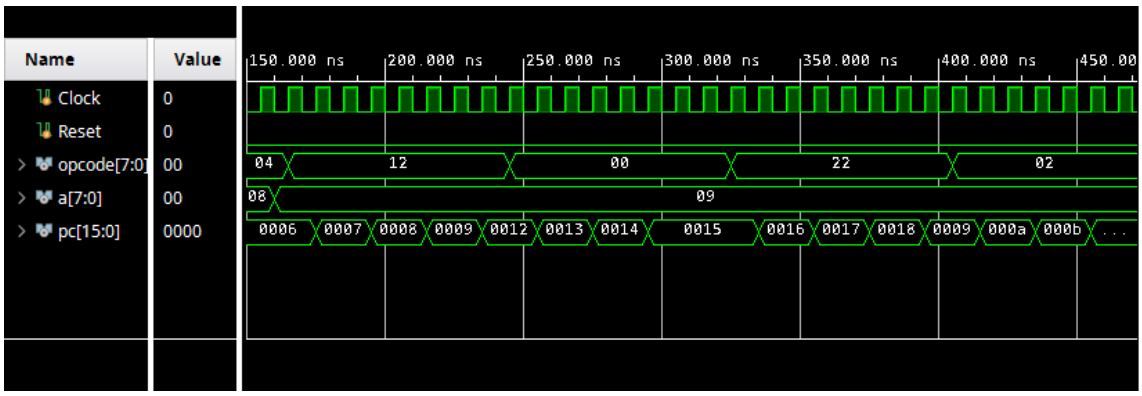


Figure 27: LCALL and RET Test

The first instruction is 0x04(INCA) and the value of a increases to 09h from 08h then the next instruction is 0x12(LCALL) and so the Program Counter jumps to the address 0x0012 and runs the instruction 0x00(NOP) then it reaches the end of the call at the 0x22(RET) instruction and comes back to the address it was in and runs the next instruction normally 0x02(LJMP)

4.2.3 Timer Overflow Test

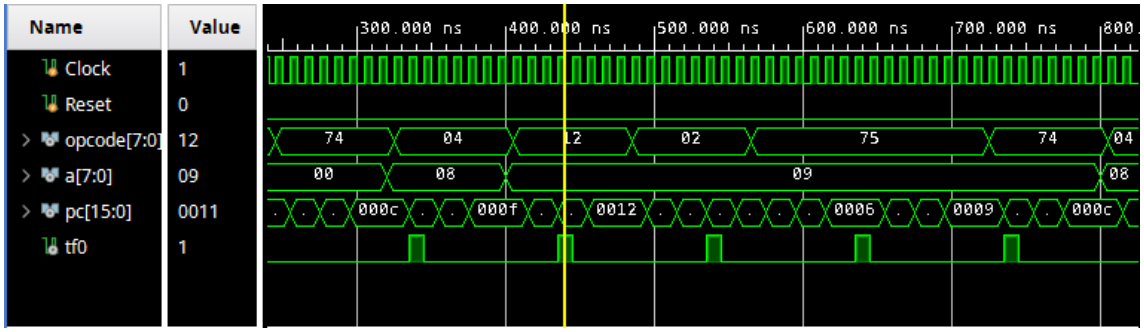


Figure 28: Timer Overflow Test

In this test all the timer is configured before so is used the MOVDI to configure the TMOD, TCON, TH, so the timer can run in the 8-bit auto-reload mode , with a defined auto-reload and it can start running as we can see the code continues running doing instruction properly like MOVAI, INCA ,LCALL, LJMP and MOVDI ,while the code is running we are getting a timer overflow every 0x0A clock cycles just like configured.

4.2.4 Interrupt Test

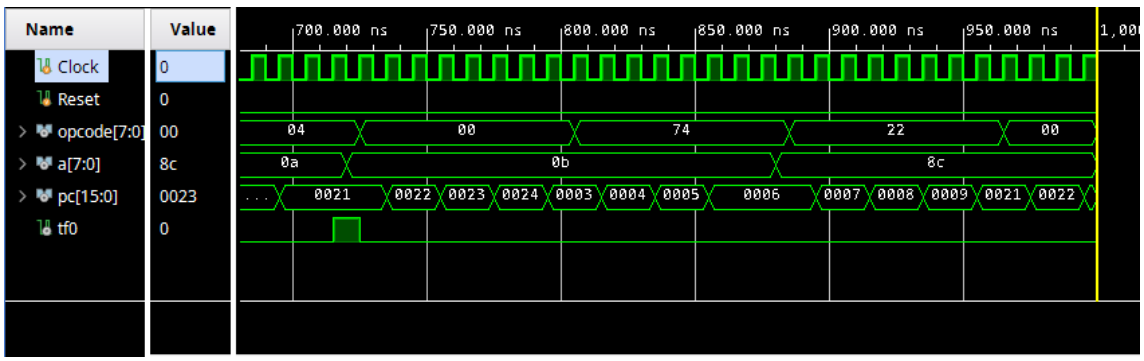


Figure 29: Interrupt Test

As we can see it starts with a 0x04(INCA) a increasing A to 0x0b then a timer overflow occurs and triggers a interrupt the interrupt jump is runs at the time of the 0x00(NOP) instruction and it jumps to the address 0x0003 and runs the instruction 0x74(MOVAI) and the A value changes to 0x8c then the 0x22(RET) instruction is there so the interrupt is over and get back to the 0x0021 address and keep running the code. Before this was needed to configure the timer so it generate a overflow and also the IE needed to be changed so the interrupts would be available and go to the address of the interrupt in the case of Timer 0 the address defined was 0x0003.

4.2.5 Uart Test

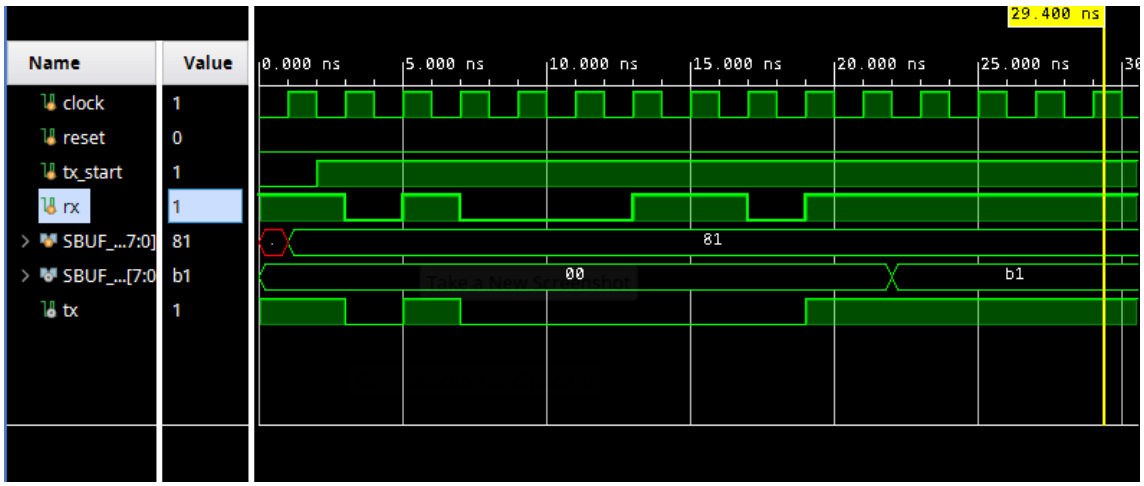
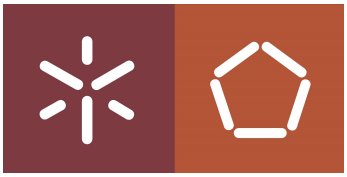


Figure 30: Uart Test

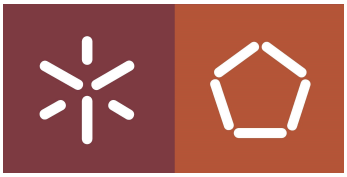
In this test we can see the working of the Uart when the flag tx_start is turned on the transmission is started in the next clock cycle so it has a start bit and then it changes the value taking in consideration the SBUF_TX register then when it ends it put the tx wire at 1, at the same times was received a start bit on rx and it start receiving as well and at the end of the byte without parity bit, this is a configuration that both the receiver and the transmitter should take in consideration, after receive the 8 bits the SBUF_RX is set as 0xB1 just like the byte receive we must pay attention that the fisrt bit is the LSB in both transmission and reception.

4.2.6 Timer 0 Overflow Test

This video is a led blinking with a period of 0.4s.
Video : Timer 0 Overflow



Conclusion



5 Conclusion

In Conclusion we can say that we achieve the requirements set at the start and we maintain inside the constraints defined in the start of the project by the professor and also ours that are extra.

This project was a bit hard at the start since we didn't had a lot to work with, but during the times it became easier, and we can say that is was a good opportunity to learn more about the 8051 itself ,how to program in verilog using Vivado, and the processor design/implementation process.

5.1 Work Done vs Defined Requirements

- Implement interrupts -> (4 Implemented)
- Implement a peripheral(Timer for example) -> (2 Timers , Debouncer for buttons, Uart, Clock Divider)
- At leats 4 instruction from each type(arithmetic, logic , data transfer, jump in execution) -> (85 instructions implemented)
- Some type of serial port (Uart for example)(defined by us) ->(Uart Implemented)
- Have at least 1MHz per instruction(defined by us) -> (10MHz per instruction)

6 References

https://developer.arm.com/documentation/101655/0961/?lang=en

https://www.win.tue.nl/~aeb/comp/8051/set8051.html

Tavares, A., Lima, C., Cabral, J., Mendes, J., Cardoso, P. (2012). Programação de Microcontroladores. LIDEL. (Obra original publicada em 2012)