ABSTRACT

This project is designing an 8-bit microprocessor in VHDL. All of the functionality will come from the VHDL that is written. The processor will have most all of the arithmetic and logic functions that are on an 8051, a standard 8-bit microprocessor that is readily used in lab experiments. To give the project some user interaction, I set up the instruction input to be sent in one instruction at a time via a set of dipswitches and a pushbutton on the UP2 board. The contents of the accumulator will be displayed, in hexadecimal, on the seven-segment displays at all times. All parts that were implemented work correctly and can be demonstrated on the UP2 board. Future work would include the implementation of JUMP instructions, interrupts, and adding more addressing modes.

Designing an 8-bit Microprocessor Using VHDL

Matthew Robert Headley

Advisor: Dr. Vinod Prasad

Final Report

Submitted to:
Dr. Vinod Prasad

Bradley University

Department of Electrical and Computer Engineering and Technology

May 5, 2006

Table of Contents

Introduction

Microprocessors are an important part of the field of electrical engineering. The goal of my project was to design an 8-bit microprocessor using VHDL. This is a very interesting project because processors are not as flexible as programmable logic. The ability to emulate a microprocessor on a programmable chip can lead to cheaper, more efficient and more flexible performance. The research that I did shows this well (see APPENDIX A). The processor that I tried to emulate is an 8051 microprocessor. This is a simple 8-bit microprocessor that is complex enough to handle many tasks; however it is simple enough to try to emulate. As I found during the course of this project it was a lot bigger task than I had initially thought. I was able to design and program a significant amount of the core however.

Functional Description

-

The scope of this project was to design an 8 bit microprocessor using VHDL. The design was implemented by programming it onto an FPGA. The design was first completed and simulated. Once the simulation proved successful, as in APPENDIX C, the VHDL was put onto the FPGA. The Altera UP2 Development Board includes the FLEX10K FPGA that was used. The goal was to design a microprocessor that is similar to an 8051.The preliminary goal was be to get several commands to work such as *mov* and *add* instructions. The desired instructions to be executed and the data to be operated on were given to the system as inputs. The result of the executed instructions was the output. After testing the individual parts they were then combined to test functionality. The final goal was to program an FPGA with the VHDL that was written.

Overall System Description

The FPGA, once programmed with the VHDL from APPENDIX B, behaved much the same as an actual microprocessor. This includes an ALU, registers, and a system bus. Each part was designed separately and tested for functionality. After all parts were working they were be combined together forming a simple microprocessor. Figure 1 shows the basic high-level block diagram of the system.

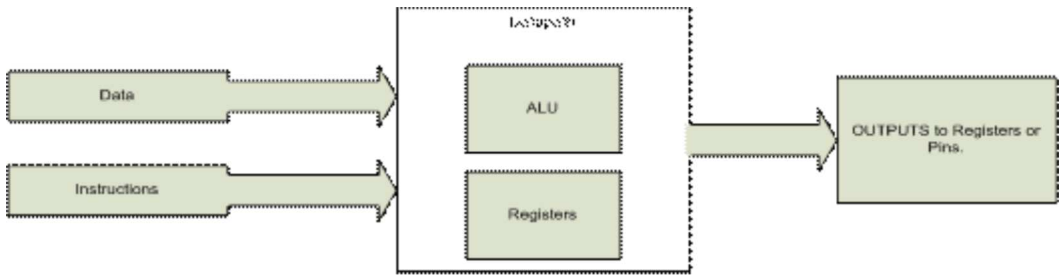


Figure 1 High-level Block Diagram

Figure 1 shows the basic parts that were be designed. The inputs and outputs go into and come out of the FPGA. The registers and ALU were actually programmed on the FPGA. The individual parts were designed separately and then integrated into one system by making blocks from the individual modules and combining together in a schematic entry module.

Inputs and Outputs

To begin with I simply used dipswitches to indicate the instruction and the data that would be manipulated. The data that was to be operated on was be fed directly into the simulation to begin with. Later different implementations for data and instruction input were examined and the dipswitches were what finally seemed to be the best fit. The outputs were the results that were stored in the various registers or various pins. They were examined for the expected behavior.

-

-

Operations

There were several different executable instructions available. The first instructions that were implemented first were *mov* and *add*. Later additions included rotates, various logic functions, and other arithmetic operations. The instructions that are normally available on an 8051 that I have implemented are *mov, add, subb, mul, orl, anl, xrl, rl, rlc, rr*, and *rrc*.

-

-

System Block Diagram

-

The block diagram in Figure 2 shows how the data and instructions are handled in the ACC/ALU module. The Accumulator accepts the data that is to be manipulated. The ALU takes the additional data that will be operated on and the

instructions for that data from the system bus. The result of the operations are then placed back in the Accumulator. The data can only be sent to the system bus from the Accumulator. The ALU can only read the additional data and instructions from the bus and cannot send any information to the bus. This process is shown in a flowchart in Figure 3.

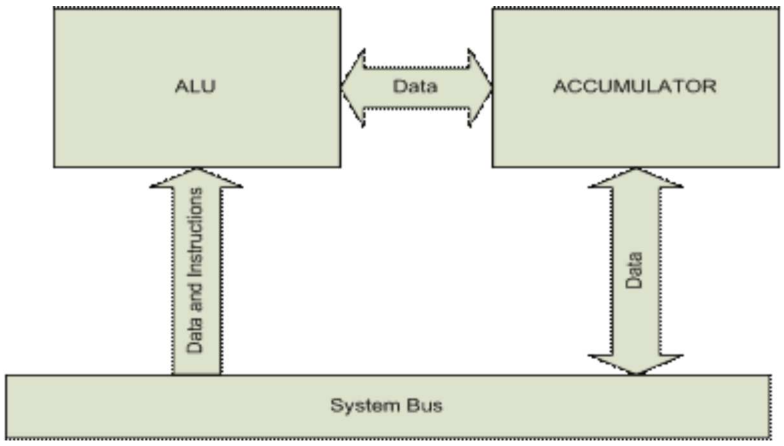


Figure 2 Accumulator and ALU subsystem

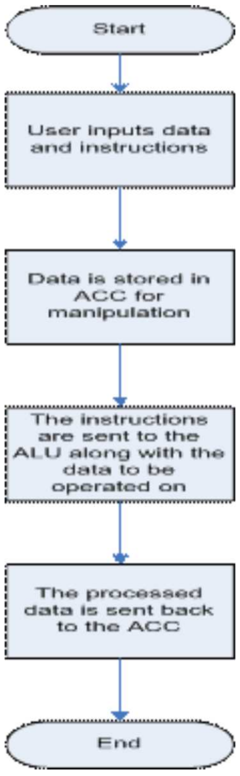

Start
User inputs data and instructions
Data is stored in ACC for manipulation
The instructions are sent to the ALU along with the data to be operated on
The processed data is sent back to the ACC
End

Figure 3 Accumulator and ALU flowchart

The block diagram for the registers and ports is shown in Figure 4. The data and instructions come to and from the ports and registers in a manner similar to the accumulator; however the ports also have their data fed out to output port pins on the board.

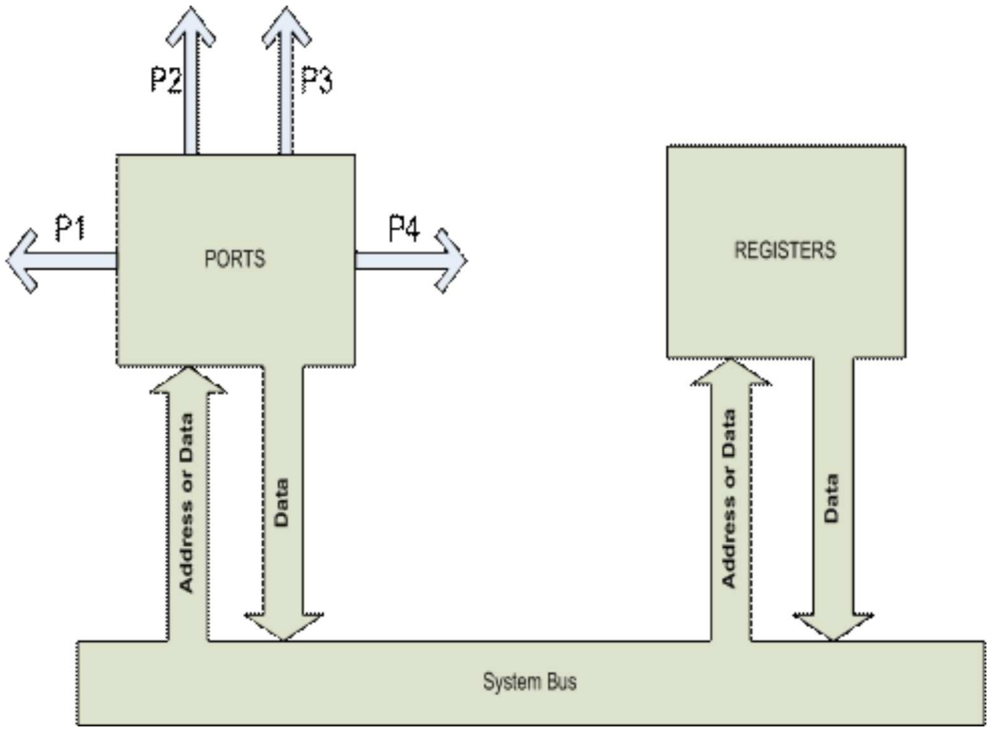


Figure 4 Ports and Registers Block Diagram

Results

The final outcome of my project was that I got all the aforementioned instructions and functions working properly. I tested it on the FPGA and went through all of the available instructions that I implemented and they all behaved exactly as they were supposed to behave. The biggest problem that I ran into as far as the instructions go was the debouncing issue I faced with the pushbutton on the UP2 board. I had to find a way around it. I ended up setting up the input procedure to press down a dipswitch, then push the pushbutton and then pull back up the dipswitch. Otherwise everything worked fine and I had no problem proving the functionality once it was programmed onto the FPGA.

Conclusion

I found this project to be very intense and involved. I was able to figure out about halfway through this semester how much I was going to be able to accomplish. The proposal that I gave at the beginning of the semester outlined a bit more than I was able to accomplish. I did end up getting most of the instructions I wished to implement finished though. This was a very successful project and I am pleased with the results.

APPENDIX A

References

<http://ieeexplore.ieee.org/iel5/92/19988/00924027.pdf?isnumber=&arnumber=924027>

<http://www.webopedia.com/TERM/F/FPGA.html>

<http://www.webopedia.com/TERM/M/microprocessor.html>

<http://www.xilinx.com/bvdocs/whitepapers/wp213.pdf>

<http://www.xilinx.com/publications/xcellonline/xcell_48/xc_pdf/xc_roman-jones48.pdf>

Huang, Han-Way. Using the MCS-51 Microcontroller. New York City: Oxford UP, 2000.

Skahill, Kevin. VHDL for Programmable Logic. Menlo Park, California: Addison-Wesley, 1996.

APPENDIX B

Code

```
--Matt Headley
--Senior Project
--IR.vhd
--IR

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity IR is
port(clk,rst,pb1:in std_logic;
irreg:in std_logic_vector(15 downto 0);
ops:out std_logic_vector(2 downto 0);
modes:out std_logic;
loc1:out std_logic_vector(3 downto 0);
loc2ordata:out std_logic_vector(7 downto 0));
end IR;

architecture rtl of IR is

signal ireg: std_logic_vector(15 downto 0);

begin

process (pb1)
begin
if(pb1='0')then --I am going to set up to feed in one instruction at a time
ireg<=irreg;    --the instruction is executed when pb1 is pressed
end if;
end process;
ops<=ireg(15 downto 13);
modes<=ireg(12);
loc1<=ireg(11 downto 8);
loc2ordata<=ireg(7 downto 0);
end rtl;




--Matt Headley
--Senior Project
--ALU.vhd
--ALU
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity ALU is
port(clk,rst: in std_logic;
                to_a, to_b, to_c, to_d, ALU_add, ALU_subb, ALU_and,
                ALU_rol, ALU_ror, ALU_rlc, ALU_rrc, ALU_or, ALU_xor, ALU_mul:       in
std_logic;
                regi:in std_logic_vector(1 downto 0);
                 num_rot: in std_logic_vector(2 downto 0);
                systembusi: in std_logic_vector(7 downto 0);
                systembuso: out std_logic_vector(7 downto 0);
                zeroflag, carryflagg: out std_logic;
                tmprl,tmprh:out std_logic_vector(6 downto 0));
end ALU;

architecture rtl of ALU is
signal remmul,carryflag,wtr_rol,wtr_ror,wtr_rlc,wtr_rrc:std_logic;
signal seg1, seg2: std_logic_vector(3 downto 0);
signal acc,breg,creg,dreg: std_logic_vector(7 downto 0);
signal acc1:std_logic_vector(15 downto 0);
        begin


        ALU_proc:process(clk,rst)
                variable rots:std_logic_vector(2 downto 0);
                begin
                if rst='0' then
                acc<= (others =>'0');
                carryflag<='0';
                elsif (clk'event and clk='1') then
                                        if ALU_add ='1' then
                                                if(((acc + systembusi)<acc) or ((acc +
systembusi)<systembusi)) then
                                                        carryflag<='1';
                                                end if;
                                                acc<= acc + systembusi;

                                        elsif ALU_subb ='1' then
                                                --if (systembusi>acc) then
                                                        acc<= acc - systembusi;
                                                        --carryflag<='1';
                                                --else
                                                        --acc<= acc - systembusi;
                                                        --carryflag<='0';
                                                --end if;

                                        elsif ALU_and ='1' then
        acc<= acc and systembusi;
elsif ALU_or ='1' then
        acc<= acc or systembusi;
elsif ALU_xor ='1' then
        acc<= acc xor systembusi;
elsif ALU_mul ='1' then
        acc1<= acc * breg;
        remmul<='1';

elsif ALU_rol ='1' then
rots:=(num_rot);
wtr_rol<='1';
elsif wtr_rol='1' then

        if ((rots)>0) then
        acc(7)<=acc(6);
        acc(6)<=acc(5);
        acc(5)<=acc(4);
        acc(4)<=acc(3);
        acc(3)<=acc(2);
        acc(2)<=acc(1);
        acc(1)<=acc(0);
        acc(0)<=acc(7);
        rots:=rots-1;
        elsif rots=0 then
        wtr_rol<='0';
        end if;
elsif ALU_ror ='1' then
rots:=(num_rot);
wtr_ror<='1';
elsif wtr_ror='1' then
        if ((rots)>0) then
        acc(7)<=acc(0);
        acc(6)<=acc(7);
        acc(5)<=acc(6);
        acc(4)<=acc(5);
        acc(3)<=acc(4);
        acc(2)<=acc(3);
        acc(1)<=acc(2);
        acc(0)<=acc(1);
        rots:=rots-1;
        elsif rots=0 then
        wtr_ror<='0';
        end if;
elsif ALU_rlc ='1' then
rots:=(num_rot);
wtr_rlc<='1';
elsif wtr_rlc='1' then
        if((rots)>0) then
        carryflag<=acc(7);
        acc(7)<=acc(6);
```

```
                                        acc(6)<=acc(5);
                                        acc(5)<=acc(4);
                                        acc(4)<=acc(3);
                                        acc(3)<=acc(2);
                                        acc(2)<=acc(1);
                                        acc(1)<=acc(0);
                                        acc(0)<=carryflag;
                                        rots:=rots-1;
                                        elsif rots=0 then
                                        wtr_rlc<='0';
                                        end if;
                                elsif ALU_rrc ='1' then
                                rots:=(num_rot);
                                wtr_rrc<='1';
                                elsif wtr_rrc='1' then
                                        if ((rots)>0) then
                                        acc(7)<=carryflag;
                                        acc(6)<=acc(7);
                                        acc(5)<=acc(6);
                                        acc(4)<=acc(5);
                                        acc(3)<=acc(4);
                                        acc(2)<=acc(3);
                                        acc(1)<=acc(2);
                                        acc(0)<=acc(1);
                                        carryflag<=acc(0);
                                        rots:=rots-1;
                                        elsif rots=0 then
                                        wtr_rrc<='0';
                                        end if;
                                elsif to_a='1' then
                                        acc<= systembusi;
                                elsif to_b='1' then
                                        breg<=systembusi;
                                elsif to_c='1' then
                                        creg<= systembusi;
                                elsif to_d='1' then
                                        dreg<=systembusi;

                                elsif remmul='1' then
                                        acc<=acc1(7 downto 0);
                                        breg<=acc1(15 downto 8);
                                        remmul<='0';
                                end if;
                        end if;
seg2<=acc(7 downto 4);
seg1<=acc(3 downto 0);
case seg1 is
when "0000" => tmprl <= "0000001";
when "0001" => tmprl <= "1001111";
when "0010" => tmprl <= "0010010";
when "0011" => tmprl <= "0000110";
when "0100" => tmprl <= "1001100";
when "0101" => tmprl <= "0100100";
when "0110" => tmprl <= "0100000";
when "0111" => tmprl <= "0001111";
when "1000" => tmprl <= "0000000";
when "1001" => tmprl <= "0001100";
when "1010" => tmprl <= "0001000";
when "1011" => tmprl <= "1100000";
when "1100" => tmprl <= "0110001";
when "1101" => tmprl <= "1000010";
when "1110" => tmprl <= "0110000";
when "1111" => tmprl <= "0111000";
end case;

case seg2 is
when "0000" => tmprh <= "0000001";
when "0001" => tmprh <= "1001111";
when "0010" => tmprh <= "0010010";
when "0011" => tmprh <= "0000110";
when "0100" => tmprh <= "1001100";
when "0101" => tmprh <= "0100100";
when "0110" => tmprh <= "0100000";
when "0111" => tmprh <= "0001111";
when "1000" => tmprh <= "0000000";
when "1001" => tmprh <= "0001100";
when "1010" => tmprh <= "0001000";
when "1011" => tmprh <= "1100000";
when "1100" => tmprh <= "0110001";
when "1101" => tmprh <= "1000010";
when "1110" => tmprh <= "0110000";
when "1111" => tmprh <= "0111000";
end case;
        end process;
        with regi select
systembuso<= acc when "00",
                     breg when "01",
                     creg when "10",
                     dreg when "11";
--systembus<= when ACC_en='0' and B_en='0' and C_en='0' and D_en='0';
zeroflag<= '1' when acc="00000000" else '0';
carryflagg<=carryflag;
end rtl;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

ENTITY regs IS
   port(Reg,WriteReg        : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
        systembuso               : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
                    systembusi                : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        RRDen,RWRen,clk           : IN STD_LOGIC);
END ENTITY regs;

ARCHITECTURE regs OF regs IS
TYPE regarr IS ARRAY (INTEGER RANGE 0 TO 7) OF
                                    STD_LOGIC_VECTOR(7 downto 0);

 signal registers : regarr;
BEGIN

   regg : PROCESS (clk) IS




   BEGIN

      IF(clk'event and clk='1') THEN
            IF(RRDen='1') THEN
         systembuso <= registers(CONV_INTEGER(UNSIGNED(Reg)));
```

```
          END IF;

                                IF (RWRen = '1') THEN
             registers(CONV_INTEGER(UNSIGNED(WriteReg))) <= systembusi;
          END IF;
      END IF;
   END PROCESS;
END regs;








library ieee;
use ieee.std_logic_1164.all;
ENTITY portreg IS
   PORT(prtWR,prtRD    : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
       WRen,RDen,clk,exin          : IN STD_LOGIC;
                   systembusi : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                   systembuso : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
       prt_in1,prt_in2,prt_in3,prt_in4 : in STD_LOGIC_VECTOR(7 DOWNTO 0);
                   reset: IN std_logic;
       prt_1,prt_2,prt_3,prt_4 : out STD_LOGIC_VECTOR(7 DOWNTO 0));


END portreg;

ARCHITECTURE portreg OF portreg IS
signal prt_1o,prt_2o,prt_3o,prt_4o : STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN

   prt : PROCESS(clk) IS

   BEGIN
                  IF reset='0' then
                  prt_1o<=(others =>'0');
                  prt_2o<=(others =>'0');
                  prt_3o<=(others =>'0');
                  prt_4o<=(others =>'0');
      ELSIF(clk'EVENT and clk='1') THEN
         IF(WRen='1' and RDen='0' and exin='0') THEN

                                     if(prtWR="00")then
                                             prt_1o<=systembusi;
                                     elsif(prtWR="01")then
                                             prt_2o<=systembusi;
                                     elsif(prtWR="10")then
```

```
                                    prt_3o<=systembusi;
                              elsif(prtWR="11")then
                                    prt_4o<=systembusi;
                              end if;

                        ELSIF(RDen='1' and WRen='0' and exin='0')THEN

                              if(prtRD="00")then
                                    systembuso<=prt_1o;
                              elsif(prtRD="01")then
                                    systembuso<=prt_2o;
                              elsif(prtRD="10")then
                                    systembuso<=prt_3o;
                              elsif(prtRD="11")then
                                    systembuso<=prt_4o;
                              end if;

                        Elsif(WRen='0' and exin='1')then

                              if(prtWR="00")then
                                    prt_1o<=prt_in1;
                              elsif(prtWR="01")then
                                    prt_2o<=prt_in2;
                              elsif(prtWR="10")then
                                    prt_3o<=prt_in3;
                              elsif(prtWR="11")then
                                    prt_4o<=prt_in4;
                              end if;

                        END IF;
                        prt_1<=prt_1o;
                        prt_2<=prt_2o;
                        prt_3<=prt_3o;
                        prt_4<=prt_4o;
          END IF;
     END PROCESS;
END portreg;
```

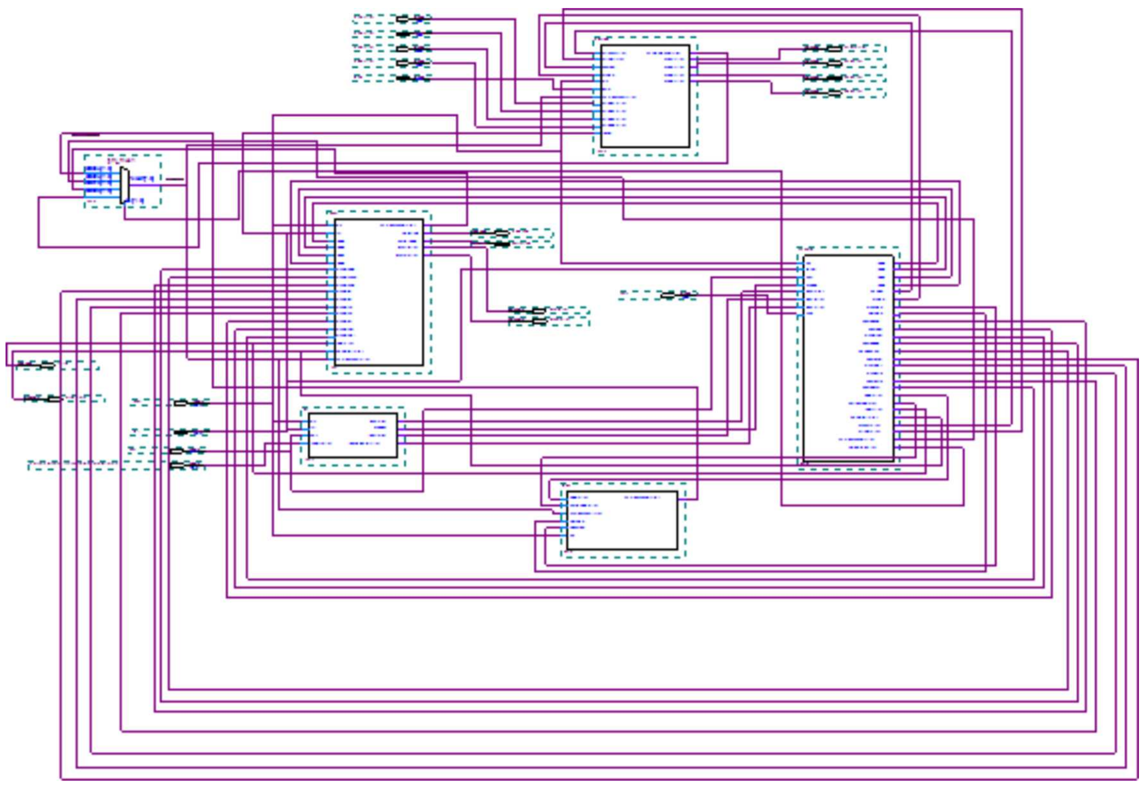
```
--Matt Headley
--Senior Project
--synchro.vhd
--synchronization module
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
ENTITY syncro IS
        PORT
        (       clk,reset,pb1,mode                                  : IN     STD_LOGIC;
                ops                                                        : IN
STD_LOGIC_VECTOR(2 downto 0);
                loc1                                                  : IN
STD_LOGIC_VECTOR(3 downto 0);
                loc2                                                  : IN
STD_LOGIC_VECTOR(7 downto 0);
                sw1                                                        : IN
STD_LOGIC;
                to_a,to_b,to_c,to_d,WRen,RDen,RWRen, RRDen,
ALU_and,ALU_or,ALU_xor,ALU_add,
                ALU_subb, ALU_rol, ALU_ror, ALU_rlc, ALU_rrc,ALU_mul          : OUT
STD_LOGIC;
                Reg,WriteReg: OUT   STD_LOGIC_VECTOR(2 downto 0);
                regi: out STD_LOGIC_VECTOR(1 downto 0);
                num_rot                 : OUT   STD_LOGIC_VECTOR(2 downto 0);
                prtWR,prtRD                                                : OUT
STD_LOGIC_VECTOR(1 downto 0);
                --systembusi                                               : IN
STD_LOGIC_VECTOR(7 downto 0);
                systembuso                                           : OUT
STD_LOGIC_VECTOR(7 downto 0);
                selecting                                             : OUT
STD_LOGIC_VECTOR(1 downto 0));
END syncro;
ARCHITECTURE a OF syncro IS
        TYPE STATE_TYPE IS (s0z, s0a, s0, s0x, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10,
s10a,s10b,s10c);
        SIGNAL state: STATE_TYPE;
        signal stater:std_logic;
BEGIN
        PROCESS (clk, reset)
        BEGIN
                IF reset = '0' THEN
                        state <= s0z;
```

```
ELSIF clk'EVENT AND clk = '1' THEN
        CASE state IS
                WHEN s0z =>
                        if sw1='0' then
                        state<=s0a;
                        else state<=s0z;
                        end if;

                WHEN s0a =>

                        IF (pb1='0') THEN
                                state <= s0;
                        else
                                state <= s0a;
                        END IF;

                WHEN s0 =>

                        IF (pb1='1') THEN
                                state <= s0x;
                        elsif(pb1='0')then
                                state <=s0;
                        END IF;

                WHEN s0x =>

                        if sw1='1' then
                        state<=s1;
                        else state<=s0x;
                        end if;
                WHEN s1 =>
                        IF (ops="000") THEN
                                state <= s2;
                        ELSIF (ops="001") THEN
                                state <= s3;
                        ELSIF (ops="010") THEN
                                state <= s4;
                        ELSIF (ops="011") THEN
                                state <= s5;
                        ELSIF (ops="100") THEN
                                state <= s6;
                        ELSIF (ops="101") THEN
                                state <= s7;
                        ELSIF (ops="110") THEN
                                state <= s8;
                        ELSIF (ops="111") THEN
                                state <=
s9;
                        END IF;

                WHEN s2 =>
```

```
                                        if(loc1>"0111")then
                                                RWRen<='1';
                                                WriteReg<=loc1(2 downto 0);
                                        elsif(loc1>"0011" and loc1<"1000")then
                                                WRen<='1';
                                                prtWR<=loc1(1 downto 0);
                                        elsif(loc1="0000")then
                                                to_a<='1';
                                        elsif(loc1="0001")then
                                                to_b<='1';
                                        elsif(loc1="0010")then
                                                to_c<='1';
                                        elsif(loc1="0011")then
                                                to_d<='1';
                                        end if;
                                IF (mode='1') THEN
                                        systembuso<=loc2;
                                        selecting<="10";
                                elsif(mode='0')then
                                        if(loc2>"00000111")then
                                                RRDen<='1';
                                                Reg<=loc2(2 downto 0);
                                                selecting<="11";
                                        elsif(loc2>"00000011" and
loc2<"00001000")then
                                                RDen<='1';
                                                prtRD<=loc2(1 downto 0);
                                                selecting<="00";
                                        elsif(loc2<"00000100")then
                                                regi<=loc2(1 downto 0);
                                                selecting<="01";
                                        end if;
                                END IF;
                                state <= s10;

                        WHEN s3 =>
                                ALU_and<='1';
                                IF (mode='0') THEN
                                        if(loc2>"00000111")then
                                                RRDen<='1';
                                                Reg<=loc2(2 downto 0);
                                        elsif(loc2>"00000011" and
loc2<"00001000")then
                                                RDen<='1';
                                                prtRD<=loc2(1 downto 0);
                                        elsif(loc2<"00000100")then
                                                regi<=loc2(1 downto 0);
                                        end if;
                                elsif(mode='1')THEN
                                        systembuso<=loc2;
                                        selecting<="10";
                                END IF;
                                state<= s10;
                        WHEN s4 =>
                                ALU_or<='1';
                                IF (mode='0') THEN
                                        if(loc2>"00000111")then
                                                RRDen<='1';
                                                Reg<=loc2(2 downto 0);
                                        elsif(loc2>"00000011" and
loc2<"00001000")then
                                                RDen<='1';
                                                prtRD<=loc2(1 downto 0);
                                        elsif(loc2<"00000100")then
                                                regi<=loc2(1 downto 0);
                                        end if;
                                elsif(mode='1')THEN
                                        systembuso<=loc2;
                                        selecting<="10";
                                END IF;
                                state<=s10;
                        WHEN s5 =>
                                ALU_xor<='1';
                                IF (mode='0') THEN
                                        if(loc2>"00000111")then
                                                RRDen<='1';
                                                Reg<=loc2(2 downto 0);
                                        elsif(loc2>"00000011" and
loc2<"00001000")then
                                                RDen<='1';
                                                prtRD<=loc2(1 downto 0);
                                        elsif(loc2<"00000100")then
                                                regi<=loc2(1 downto 0);
                                        end if;
                                elsif(mode='1')THEN
                                        systembuso<=loc2;
                                        selecting<="10";
                                END IF;
                                state<=s10;
                        WHEN s6 =>
                                if(loc2>"00000111")then
                                        state<=s10;
                                else num_rot<=loc2(2 downto 0);
                                end if;

                                if(loc1="0000")then
                                        ALU_rrc<='1';
                                elsif(loc1="0001")then
                                        ALU_rlc<='1';
                                elsif(loc1="0010")then
                                        ALU_ror<='1';
```

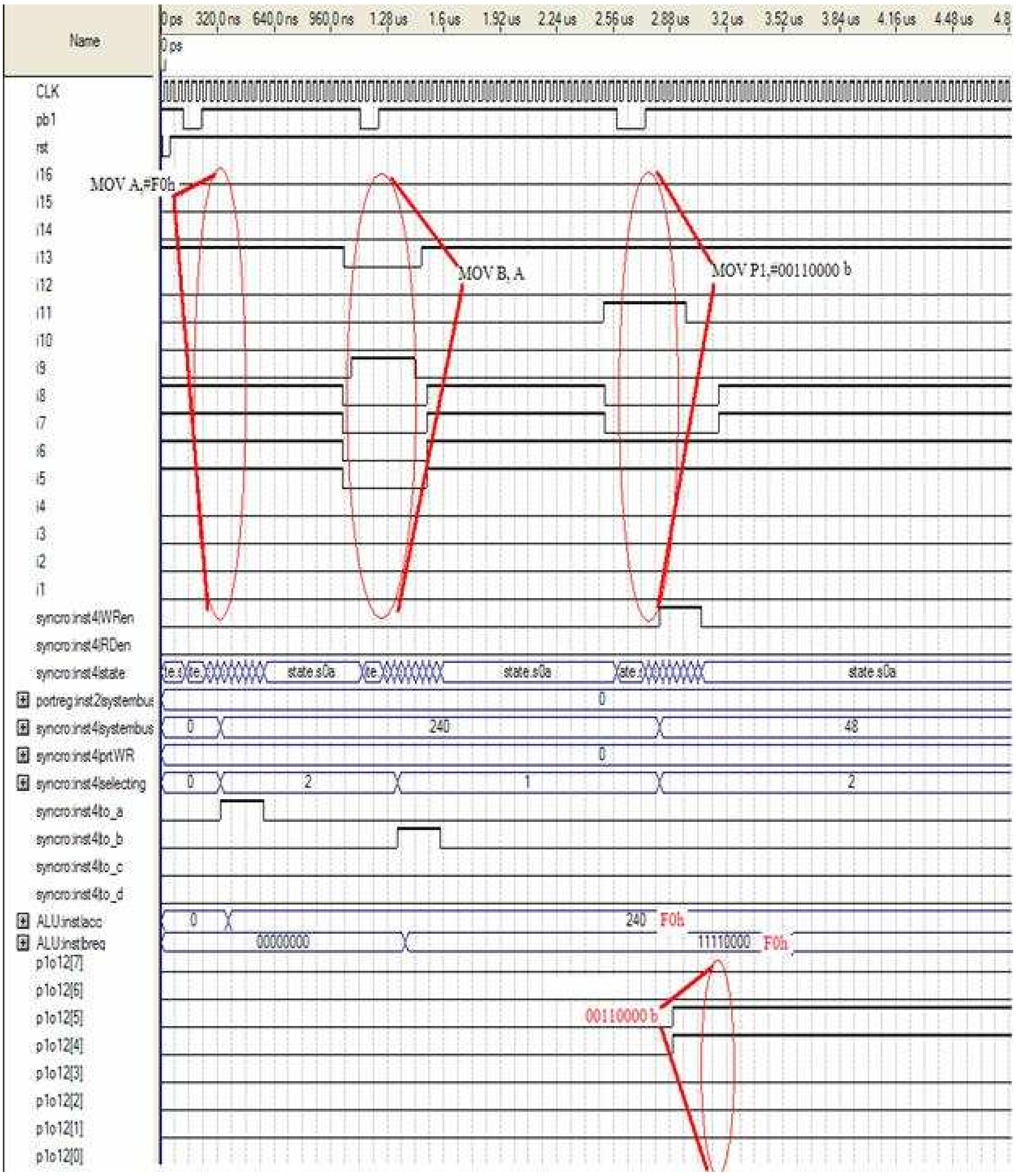
```
				elsif(loc1="0011")then
					ALU_rol<='1';
				else
					ALU_rrc<='0';
					ALU_rlc<='0';
					ALU_ror<='0';
					ALU_rol<='0';
				end if;
				state<=s10;
			WHEN s7 =>
				ALU_add<='1';
				IF (mode='0') THEN
					if(loc2>"00000111")then
						RRDen<='1';
						Reg<=loc2(2 downto 0);
					elsif(loc2>"00000011" and
loc2<"00001000")then
						RDen<='1';
						prtRD<=loc2(1 downto 0);
					elsif(loc2<"00000100")then
						regi<=loc2(1 downto 0);
					end if;
				elsif(mode='1')THEN
					systembuso<=loc2;
					selecting<="10";
				END IF;
				state<=s10;
			WHEN s8 =>
				ALU_subb<='1';
				IF (mode='0') THEN
					if(loc2>"00000111")then
						RRDen<='1';
						Reg<=loc2(2 downto 0);
					elsif(loc2>"00000011" and
loc2<"00001000")then
						RDen<='1';
						prtRD<=loc2(1 downto 0);
					elsif(loc2<"00000100")then
						regi<=loc2(1 downto 0);
					end if;
				elsif(mode='1')THEN
					systembuso<=loc2;
					selecting<="10";
				END IF;
				state<=s10;
			WHEN s9 =>
				ALU_mul<='1';
				state<=s10;
			WHEN s10 =>
				ALU_rrc<='0';
				ALU_rlc<='0';
```

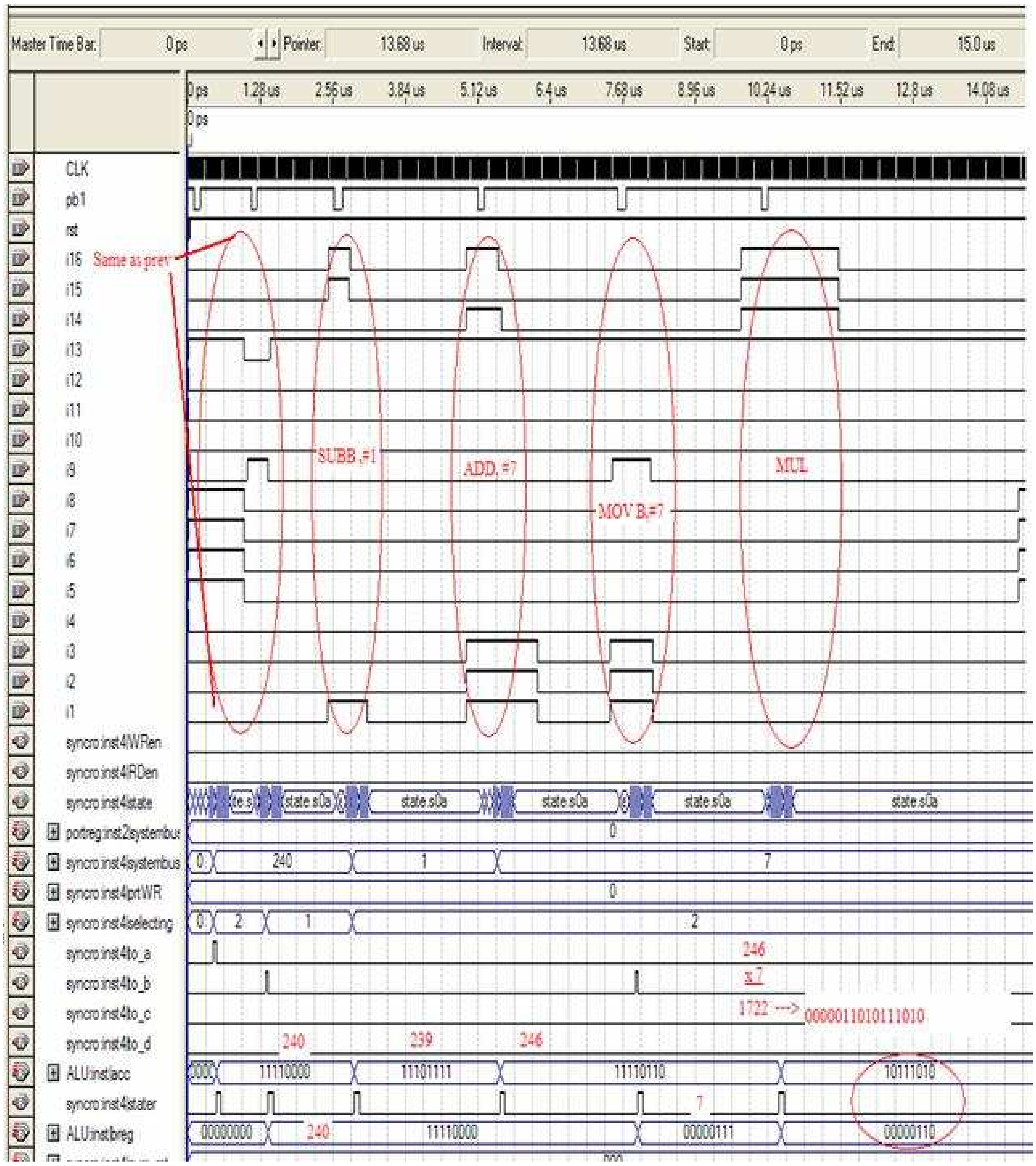
```
                                        ALU_ror<='0';
                                        ALU_rol<='0';
                                        ALU_add<='0';
                                        ALU_subb<='0';
                                        WRen<='0';
                                        RDen<='0';
                                        RWRen<='0';
                                        RRDen<='0';
                                        ALU_mul<='0';

                                        to_a<='0';
                                        to_b<='0';
                                        to_c<='0';
                                        to_d<='0';
                                        ALU_and<='0';
                                        ALU_or<='0';
                                        ALU_xor<='0';
                                        state<=s10a;
                                        stater<='1';
                                WHEN s10a =>
                                        state<=s10b;
                                WHEN s10b =>
                                        if stater='1' then
                                        stater<='0';
                                        state<=s10a;
                                        else state<=s10c;
                                        end if;
                                WHEN s10c =>

                                        state<=s0z;
                        END CASE;
                END IF;
        END PROCESS;
END a;
```



APPENDIX C

Simulations





Master Time Bar: 0 ps Pointer: 13.68 us Interval: 13.68 us Start: 0 ps End: 15.0 us