Assignment 5 - Phase II

Name - Zubair Nadaph
Ankit Taskar
Group no. - 4
Due Date - 11/16/2017

Objective

This lab is based on developing a VHDL code for a 4-bit Central Processing Unit and test a certain instruction sets. VHDL code for individual components were developed and burnt onto Cyclone IV FPGA DE2-115 Development kit. A bunch of different instruction sets were tried and tested and the desired output was obtained.

List of Components Used

- DE2 115 Development Board
- Quartus II CAD Tool
- DE2 115 Parallel Cable for FPGA
- Board Power Supply
- DE2 115 user manual
- Tutorials

Experimental Approach

Using VHDL, the code was developed for 4-bit CPU.The opcode for the 4 bit CPU is 15 bit (using flexibility with code of ALU), this is further divided into one 7 bits and a byte in order to segregate instruction from direct data and RAM address[1]. One byte is reserved for direct data and RAM address. Another 7 bits consist of instruction which is given to the instruction register, hence every operation is synchronous with clock. The output of instruction register are given as a mode selector to the ALU, select line for MUX and read/write to the scratchpad RAM[1]. The inputs to the MUX are the direct data and the value from given RAM address, the output of the multiplexer is connected to the ALU. By this selecting the mode through the instruction register, all the desired results are found. The ALU is connected to the Accumulator in order to update the result of operation with clock, which is then connected to the BCD - 7-segment decoder. The output of the accumulator is also provided back to the ALU, scratchpad RAM and ‘program counter[1]. The outputs of PC are also connected to 7-segment display through BCD. For the jump instructions 3 additional switches were required for JMPZ, JIFZ and JIFN instructions. The following was implemented using VHDL.

VHDL Code

-- Code for combining all the components

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

entity cpu is
port(sw : in std_logic_vector(17 downto 0);
      key: in std_logic_vector(3 downto 0);
      hex0 : out std_logic_vector(0 to 6);
      hex1 : out std_logic_vector(0 to 6);
      hex2 : out std_logic_vector(0 to 6);
      hex3 : out std_logic_vector(0 to 6);
      hex4 : out std_logic_vector(0 to 6);
      hex5 : out std_logic_vector(0 to 6);
      hex6 : out std_logic_vector(0 to 6);
      hex7 : out std_logic_vector(0 to 6);
      ledg : out std_logic_vector(7 downto 0);
      CLOCK_50: in std_logic);
end cpu;

architecture behavior of cpu is

   COMPONENT debounce
      GENERIC (   bouncetime  : INTEGER := 50000);
      PORT (   CLK, RST, sw  : IN STD_LOGIC;
               outp, invoutp : OUT STD_LOGIC );
   END COMPONENT;

   component alu
      port(data1, data2: in std_logic_vector(3 downto 0);--data1=A and data2=B
            dataout: out std_logic_vector(3 downto 0);
            mode: in std_logic_vector(4 downto 0));
   end component;

   component numdisp
      port(c: in std_logic_vector(3 downto 0);
           display: out std_logic_vector(0 to 6));
   end component;

   component RAMFinal
   port(address: in std_logic_vector(3 downto 0);
         ReadWrite: in std_logic;
         dataIn:in std_logic_vector(3 downto 0);
         dataOut: out std_logic_vector(3 downto 0));
   end component;

   component program_counter
   port(clk, arst: in std_logic;
         jmps: in std_logic_vector(2 downto 0);
         datain: in std_logic_vector(3 downto 0);
         dataout: out std_logic_vector(3 downto 0));
   end component;
```

```
   component hex_ff
   port(clock, reset : in std_logic;
         datain : in std_logic_vector (5 downto 0);
         dataout: out std_logic_vector (5 downto 0));
   end component;

   component octa_ff
   port(clock, reset : in std_logic;
         datain : in std_logic_vector (7 downto 0);
         dataout: out std_logic_vector (7 downto 0));
   end component;

   component mux
   port(data1, data2: in std_logic_vector(3 downto 0);
         switch: in std_logic;
         dataout: out std_logic_vector(3 downto 0));
   end component;


signal pq, rs, mu, acc, ramout, alo: std_logic_vector(3 downto 0);
signal mn: std_logic_vector(6 downto 0);
signal pp: std_logic_vector(2 downto 0);
signal clk1: std_logic;
signal inst, instout, accout, oddout, accin: std_logic_vector(7 downto 0);

begin
D0: debounce GENERIC MAP (bouncetime => 100000)               --default is 50000 (1ms@50MHz)
      PORT MAP (CLOCK_50, key(1), key(0), clk1, LEDG(0));     -- debouncing key(1)
F0: octa_ff port map(pq(0), key(1), sw(10 downto 3), oddout);  -- register to load data and ram addr
R1: RAMFinal port map(oddout(3 downto 0), instout(6), accout(3 downto 0), ramout);-- ram
      mn <= sw(17 downto 11);                                 -- even 7bit opcode
      inst <= mn or "00000000";                               -- converting the even opcode to byte with MSB = '0'
      rs(0) <= pq(0) xor '1';                                 -- inverting the lsb of program counter to give out of synchronism clock to acc
M0: mux port map(oddout(7 downto 4), ramout, instout(5), mu);  --mux
F1: octa_ff port map(pq(0), key(1), inst, instout);           -- instruction register
A0: alu port map(accout(3 downto 0), mu, alo, instout(4 downto 0));  -- this is alu; A = acc, B = mn(output of mux) and mode = instout(4 downto 0)
      accin <= alo or "00000000";                             -- converting the input to accumulator to 8 bit
F2: octa_ff port map(rs(0), key(1), accin, accout);           -- This is accumulator with input form ALU (alo) and output as acc
N0: numdisp port map(accout(3 downto 0), hex6);               -- displaying the result
P1: program_counter port map(clk1, key(1), sw(2 downto 0), accout(3 downto 0), pq );
N1: numdisp port map(pq, hex7);                               -- displaying the program counter
N2: numdisp port map(sw(10 downto 7), hex5);                  -- data
N3: numdisp port map(sw(6 downto 3), hex4);                   -- addr
N4: numdisp port map(sw(14 downto 11), hex0);                 -- displaying the mode of the ALU
N5: numdisp port map(mu, hex1);                               --displaying output of mux
N6: numdisp port map(alo, hex2);                              --displaying output of alu

end behavior;
```

-- Code for Program Counter

```
library ieee;
use ieee.std_logic_1164.all, ieee.numeric_std.all;

entity program_counter is
   port(clk, arst: in std_logic;                    -- clock and asynchronous reset
        jmps: in std_logic_vector(2 downto 0);      --three aditional bits for jump in the sequence: <jmp><jifz><jifn>
        datain: in std_logic_vector(3 downto 0);
        dataout: out unsigned(3 downto 0));
end program_counter;

architecture behavior of program_counter is
begin
      process(clk)
      variable dataout_v: unsigned(3 downto 0);
         begin
            if (arst='0') then
               dataout_v := "0000";                 --asynchronous resetting the counter
            elsif rising_edge(clk) then
               if(jmps = "100") then                --unconditional jump setting the program counter to "0000"
                  dataout_v := "0000";
               elsif(jmps = "010" and datain ="0000") then --jump if zero, setting the program counter to "0000"
                  dataout_v := "0000";
               elsif(jmps = "001" and datain = "1111") then --jump if negative, setting the program counter to "0000"
                  dataout_v := "0000";
               else
                  dataout_v := dataout_v + 1;       --up counting
               end if;
            end if;
         dataout <= dataout_v;
      end process;
end behavior;
```

--Code for Number Display

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

-- this component is used to lit the leds of 7 segment display. Similar to BCD, but displays hex numbers in their original form
entity numdisp is
port(c: in std_logic_vector(3 downto 0);
      display: out std_logic_vector(0 to 6));
end numdisp;

architecture behavior of numdisp is
begin
   with c select
      display <= "1001111" when "0001",
                 "0000001" when "0000",
                 "0010010" when "0010",
                 "0000110" when "0011",
                 "1001100" when "0100",
                 "0100100" when "0101",
                 "0100000" when "0110",
                 "0001111" when "0111",
                 "0000000" when "1000",
                 "0000100" when "1001",
                 "0001000" when "1010",
                 "1100000" when "1011",
                 "0110001" when "1100",
                 "1000010" when "1101",
                 "0110000" when "1110",
                 "0111000" when others;
end behavior;
```

-- Code for Multiplexer

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

entity mux is
   port(data1, data2: in std_logic_vector(3 downto 0);
        switch: in std_logic; -- select line
        dataout: out std_logic_vector(3 downto 0));
end mux;

architecture behavior of mux is
begin
      with switch select
         dataout <= data1 when '0',
                    data2 when others;
end behavior;
```

-- Code for Octa Flip Flop

```
library ieee;
use ieee.std_logic_1164.all, ieee.numeric_std.all;

-- this component is replicate of general octa-d flip flop which is also use as accumulator
entity octa_ff is
port(clock, reset : in std_logic;
      datain : in unsigned (7 downto 0);
      dataout: out unsigned (7 downto 0));
end octa_ff;

architecture behavior of octa_ff is
begin
      process(clock)
      variable dataout_v: unsigned(7 downto 0);
      begin
            if (reset='0') then
               dataout <= "00000000"; -- asynchronous reset
            elsif rising_edge(clock) then
               dataout <= datain;
            end if;
         --dataout <= dataout_v;
      end process;
end behavior;
```

-- Code for RAM

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee. std_logic_unsigned.all;
use ieee.numeric_std.all;

entity RAMFinal is
port(address: in std_logic_vector(3 downto 0);
      ReadWrite: in std_logic;
      dataIn:in std_logic_vector(3 downto 0);
      dataOut: out std_logic_vector(3 downto 0)
      );                                          --Declaring Basic Ports
end entity RAMFinal;

Architecture behavior of RAMFinal is

type memory is array (15 downto 0) of std_logic_vector (3 downto 0);
signal memory_type: memory;
signal AddressOut:integer range 0 to 100;

begin

process( address, dataIn, ReadWrite)

begin
      AddressOut<= conv_integer(address);  -- indexing the elements of array
      if(ReadWrite = '1') then
         dataOut <= memory_type(AddressOut);  --storing data to given address
      elsif (ReadWrite<= '0') then
         memory_type(AddressOut) <= DataIn;   -- retrieving the stored Data.
      else
         dataOut <= "0000";           -- Default
end if;
end process;
end behavior;
-- End of Architecture.
```

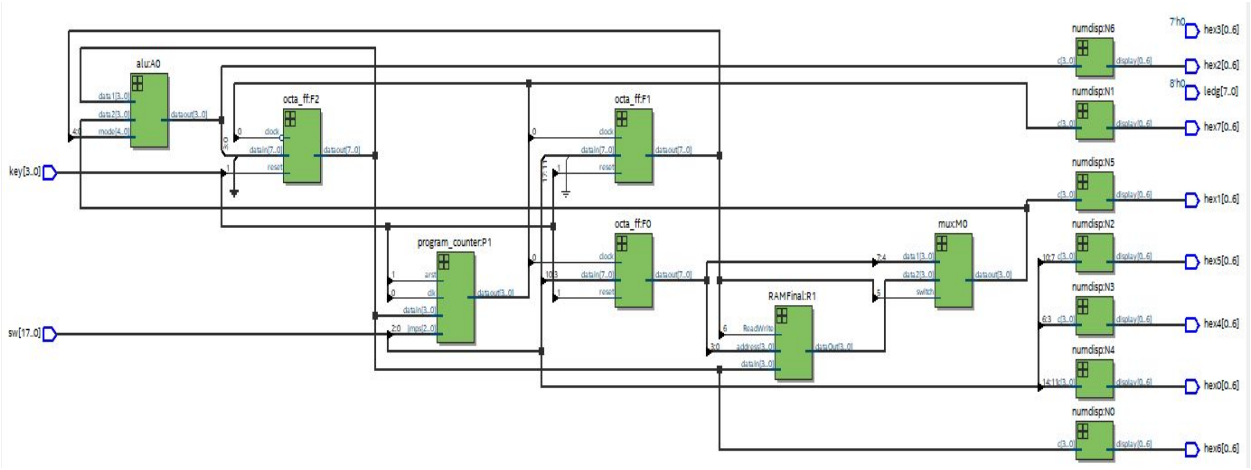
--Code for ALU

```
library ieee;
use ieee.std_logic_1164.all, ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity alu is
port(data1, data2: in unsigned(3 downto 0);--data1=A and data2=B
      dataout: out unsigned(3 downto 0);
      mode: in std_logic_vector(4 downto 0));
end alu;

architecture behavior of alu is
begin
   with mode select
      dataout <= not data1 when "10000", --logical inversion of A
                  data1 or data2 when "11110",
                  data1 and data2 when "11011",
                  not data2 when "10101", -- logical inversion of B
                  data1 - data2 -1 when "00110",
                  data1 + data2 when "01001",
                  data2 when "11010", -- copying B at the output
                  data1 + data1 when "01100",
                  data1 when "11111", -- copying A at the output
                  data2 + data2 when "01010",
                  "0000" when others;

end behavior;
```

-- Code for Debounce

```
module debounce(
    CLK,
    RST,
    sw,
    outp,
    invoutp);

parameter bouncetime = 50000;
parameter clkwidth = $clog2(bouncetime);

input CLK;
input RST;

input sw;
output reg outp;
output reg invoutp;

reg [(clkwidth-1):0] count;
reg lsw; //last switch state

always@(posedge CLK or negedge RST)
begin
      if(RST==0) begin
         count <= bouncetime;
      end
      else begin
         lsw<=sw;

         if(lsw != sw) begin
            count <= bouncetime;
         end
         else

         if (count == 0) begin
            outp <= sw;
            invoutp <= ~sw;
         end
         else begin
            count <= count - 1;
         end;

      end
end

endmodule
```

Results



μ Run the Following Program

Instruction	ACC Disp.	PC Disp.	Notes
LDI 7	7	2	
STORE 3	7	4	
LDI 6	6	6	
LOAD 3	7	8	
ANDI 3	3	A	
JIFN	3	C	
ADD 3	A	E	
NOP	A	0	
JMP	A	0	Constraints
LDI 15	F	2	
JIFN	F	0	

!!! PUT THESE RESULTS IN THE REPORT!!!

Fig 1. Executed Instruction Sets.

While developing the code a few changes were done to the previously designed circuit in lab four. The accumulator was given an inverted clock as it was observed that accumulator got update with rising edge of clock and since, the clock were given through lsb of program counter it required 3 pulses to get rising edge for accumulator. Inversion of lsb provided the rising edge in just two clock cycles. Also, the RAM address and the direct data to the multiplier were given using an octa flip-flop as it was observed that the opcode needs to be updated simultaneously with data and address. Lastly, the FPGA wasn't debounced properly causing it miss a pulse or two. As a result, a

debounce code had to be implemented which was avoided in phase I of this lab. As one incorrect operation ment that all the instructions had to been to done from start.

Conclusion

The code developed was successful in generating the desired result for the given instruction set. All the instruction sets required two clock cycles. We tried playing around with different instructions and the desired output was obtained. Thus, the system was successfully designed and implemented.

FTQs

Zubair Nadaph

Byte 0					Byte 1	
M	MUX	W’	CN	S	DB	RAM
1	1	1	0	0000	0100	1101

The above instruction based on the design of ALU component (no CN was used) would result in inversion of data present at the given RAM address (1101).

Ankit Taskar

Byte 0					Byte 1	
M	MUX	W’	CN	S	DB	RAM
1	0	0	0	0000	0110	1001

The above instruction would cause the value of DB to be inverted.

Reference

[1] ADSD Lab Report 4