EE 443
Digital and Computer Analysis and Design Laboratory

Lab *#4 Part 1: Basic components and Register File*
2/28/13

Prepared by:
Matt Van Veldhuizen

Lab Participation
Provide the percent participation in lab by each lab partner (normally, 50%, 50% for 2 partners):

Lab member name	**Percent participation**
Matt Van Veldhuizen	100%

Does your solution work the way it's supposed to work? ☐YES ☐NO[1]

[1]If your answer is NO, please explain in your report.

Instructor/TA comments and grading:

1. Objective and Background

The goal of this lab was to design a simple 16-bit microprocessor system. This 16-bit system is to be based on the 32-bit RISC machine that was presented to us in our textbook. This is the first of many parts of creating the microprocessor. Part one consisted of creating the register file. This lab was also set up to refresh my knowledge of flip-flops, latches, registers, multiplexers, and decoders. Also refresh my knowledge of VHDL.

2. Equipment

Altera Quartus II

3. Procedure

The first task was to create the register file and this included many steps. The first of these was to create an 8-bit register component called REG8. This component was to have specified inputs and outputs, to be used when connecting all of the components together. This can be seen in Figure 1. After this was completed the next component to be created was a 4-input 4-bit multiplexer, called MUX4x4, as seen in Figure 2. The next step was to create a 3-8 decoder called DCD3x8, as seen in Figure 3. And the last step was to create the 8x16-bit register file, to be called REG8x16. And like the first component this file had specific inputs and outputs to be labeled. This can be seen in Figure 4. Once these files were created, the register file was to be compiled with the Cyclone II EP2C35F672C6 board. The chip resources can be seen in Figure 5, and the timing delays can be seen in Table 1. The last task of this lab was to provide simulations to answer some questions about the registry file as seen in Figures 6 and 7.

4. Results

Creating all of the different components was the easiest part of this lab, I had a hard time trying to figure out how to connect all of the components together in the REG8x16 file so that it works as it should. After talking with some of the other students I figured out how to accomplish this and I was able to get it working correctly. After that all that was required was to run the timing simulations to figure out some of the questions.

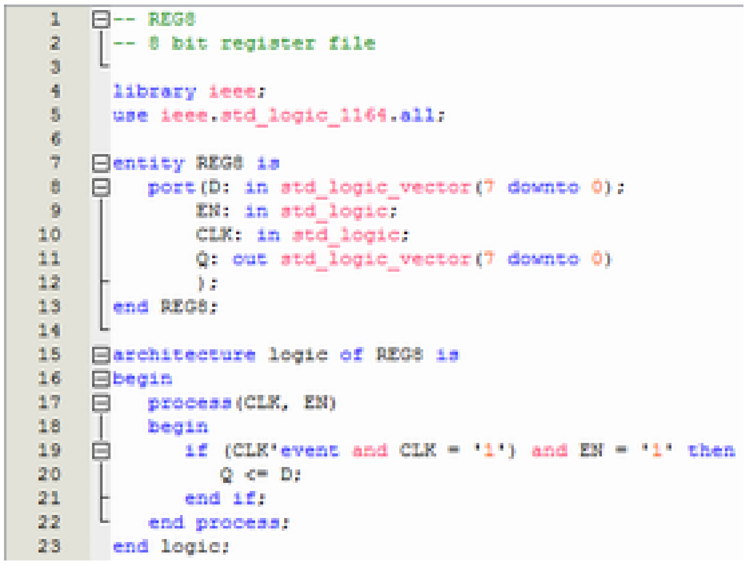
5. Discussion and Questions

The timing delay that determined the maximum clock frequency was the connection between ADD_R1[1] and DOUT2[8], this had a time of 14.033ns, as seen in Table 1. The minimum step time for the data lines when writing the data to the register file was 4.616ns, as seen in Table 2. When I wrote data into two different registers and read the contents of these registers simultaneously, the delay between the the rising edge of the clock for the first register is at 161.59ns and the data is stable at 173.73ns therefore the delay is 12.14ns, as seen in Figure 8. The rising edge of the clock for the second register is at 161.59ns and the data is stable at 171.08ns therefore the delay is 9.49ns, as seen in Figure 9. There was a difference between these two times, the first register I read from became stable after the second register, became stable. This could be explained by how the register file is laid out. Where the second register happens to be closer to the output so it requires less time to travel and outputs before the first register, which then has to wait for the second register to finish before it can be outputted.

6. Conclusion

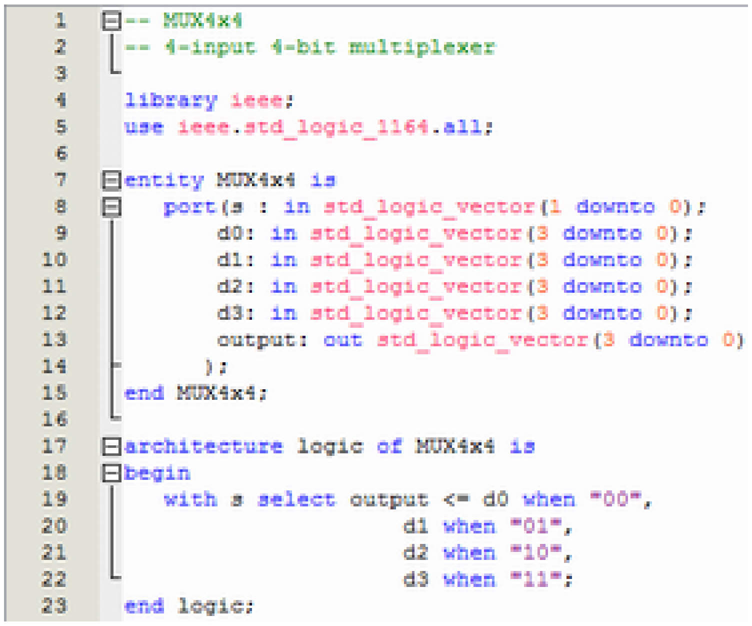
The point of this lab was to refresh our knowledge of flip-flops, latches, decoders, registers, and multiplexers. So that we can then start to create a 16-bit RISC style microprocessor. The first step of this process was to create the register file, which this lab covered.

7. Attachments



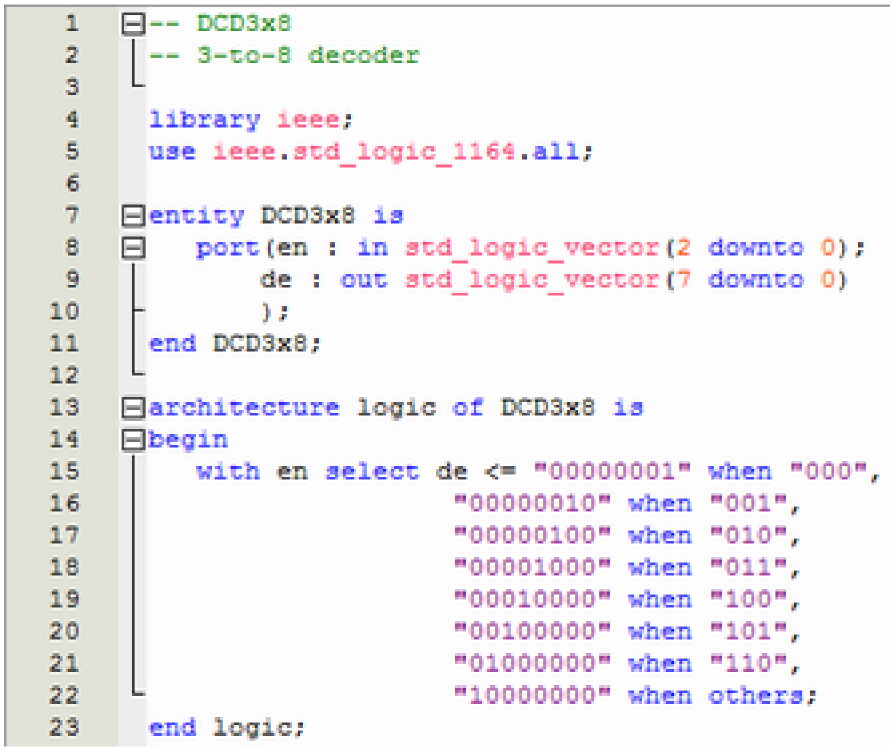
```vhdl
-- REG8
-- 8 bit register file

library ieee;
use ieee.std_logic_1164.all;

entity REG8 is
   port(D: in std_logic_vector(7 downto 0);
        EN: in std_logic;
        CLK: in std_logic;
        Q: out std_logic_vector(7 downto 0)
        );
end REG8;

architecture logic of REG8 is
begin
   process(CLK, EN)
   begin
      if (CLK'event and CLK = '1') and EN = '1' then
         Q <= D;
      end if;
   end process;
end logic;
```

Figure 1: REG8.vhdl



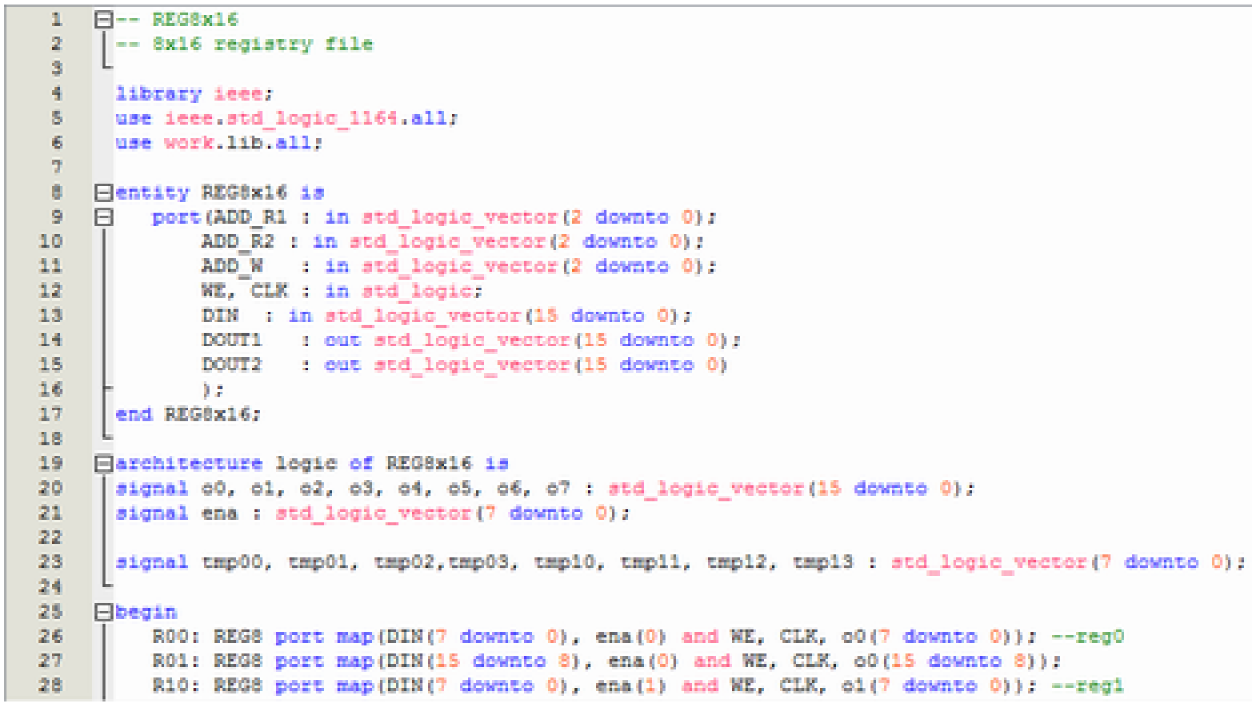
```vhdl
-- MUX4x4
-- 4-input 4-bit multiplexer

library ieee;
use ieee.std_logic_1164.all;

entity MUX4x4 is
   port(s : in std_logic_vector(1 downto 0);
        d0: in std_logic_vector(3 downto 0);
        d1: in std_logic_vector(3 downto 0);
        d2: in std_logic_vector(3 downto 0);
        d3: in std_logic_vector(3 downto 0);
        output: out std_logic_vector(3 downto 0)
       );
end MUX4x4;

architecture logic of MUX4x4 is
begin
   with s select output <= d0 when "00",
                           d1 when "01",
                           d2 when "10",
                           d3 when "11";
end logic;
```

Figure 2: MUX4x4.vhdl



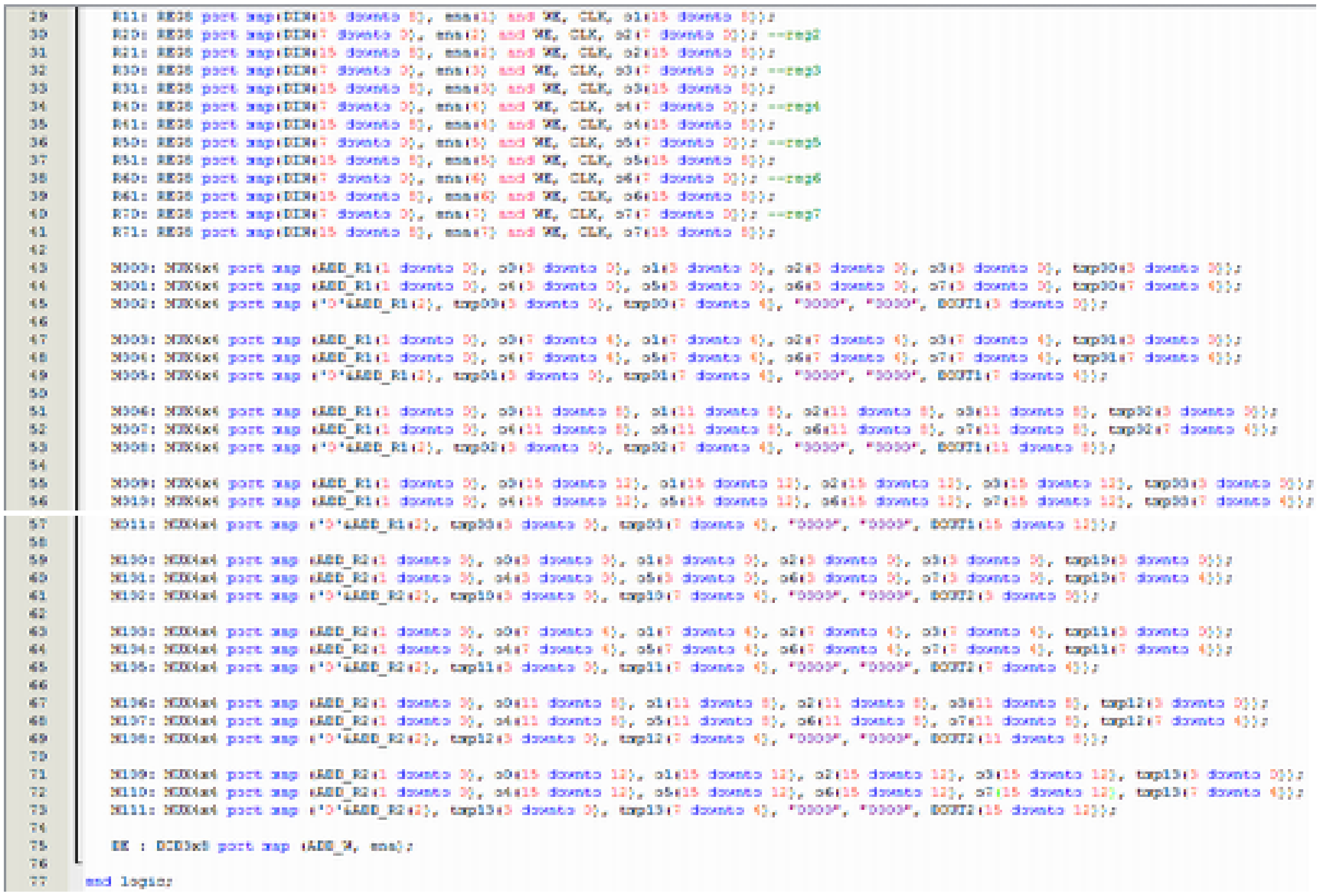
```vhdl
-- DCD3x8
-- 3-to-8 decoder

library ieee;
use ieee.std_logic_1164.all;

entity DCD3x8 is
    port(en : in std_logic_vector(2 downto 0);
         de : out std_logic_vector(7 downto 0)
         );
end DCD3x8;

architecture logic of DCD3x8 is
begin
    with en select de <= "00000001" when "000",
                        "00000010" when "001",
                        "00000100" when "010",
                        "00001000" when "011",
                        "00010000" when "100",
                        "00100000" when "101",
                        "01000000" when "110",
                        "10000000" when others;
end logic;
```

Figure 3: DCD3x8.vhdl



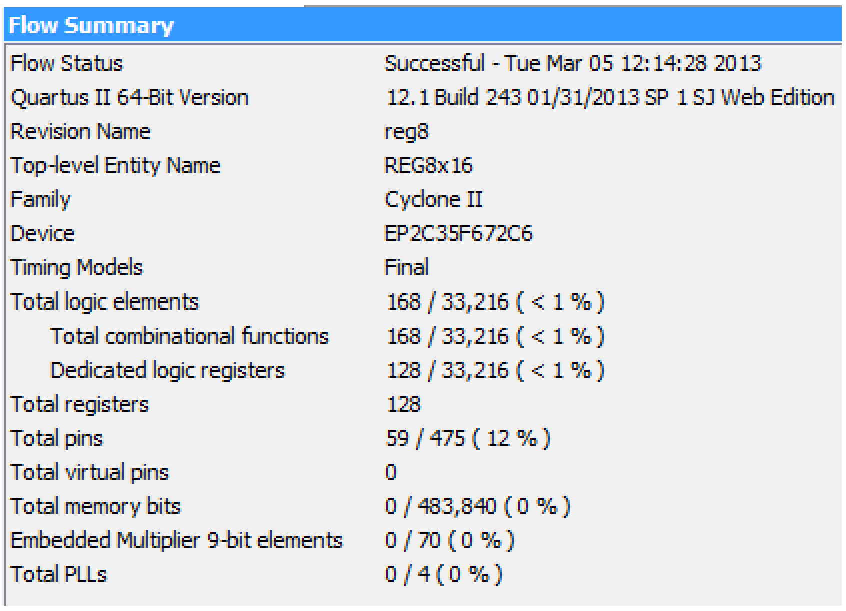
```vhdl
-- REG8x16
-- 8x16 registry file

library ieee;
use ieee.std_logic_1164.all;
use work.lib.all;

entity REG8x16 is
   port(ADD_R1 : in std_logic_vector(2 downto 0);
        ADD_R2 : in std_logic_vector(2 downto 0);
        ADD_W    : in std_logic_vector(2 downto 0);
        WE, CLK : in std_logic;
        DIN   : in std_logic_vector(15 downto 0);
        DOUT1    : out std_logic_vector(15 downto 0);
        DOUT2    : out std_logic_vector(15 downto 0)
        );
end REG8x16;

architecture logic of REG8x16 is
signal o0, o1, o2, o3, o4, o5, o6, o7 : std_logic_vector(15 downto 0);
signal ena : std_logic_vector(7 downto 0);

signal tmp00, tmp01, tmp02,tmp03, tmp10, tmp11, tmp12, tmp13 : std_logic_vector(7 downto 0);

begin
   R00: REG8 port map(DIN(7 downto 0), ena(0) and WE, CLK, o0(7 downto 0)); --reg0
   R01: REG8 port map(DIN(15 downto 8), ena(0) and WE, CLK, o0(15 downto 8));
   R10: REG8 port map(DIN(7 downto 0), ena(1) and WE, CLK, o1(7 downto 0)); --reg1
```

Figure 4: REG8x16.vhdl



```
    R11: REG8 port map(DIN(15 downto 8), ena(1) and WE, CLK, o1(15 downto 8));
    R20: REG8 port map(DIN(7 downto 0), ena(2) and WE, CLK, o2(7 downto 0)); --reg2
    R21: REG8 port map(DIN(15 downto 8), ena(2) and WE, CLK, o2(15 downto 8));
    R30: REG8 port map(DIN(7 downto 0), ena(3) and WE, CLK, o3(7 downto 0)); --reg3
    R31: REG8 port map(DIN(15 downto 8), ena(3) and WE, CLK, o3(15 downto 8));
    R40: REG8 port map(DIN(7 downto 0), ena(4) and WE, CLK, o4(7 downto 0)); --reg4
    R41: REG8 port map(DIN(15 downto 8), ena(4) and WE, CLK, o4(15 downto 8));
    R50: REG8 port map(DIN(7 downto 0), ena(5) and WE, CLK, o5(7 downto 0)); --reg5
    R51: REG8 port map(DIN(15 downto 8), ena(5) and WE, CLK, o5(15 downto 8));
    R60: REG8 port map(DIN(7 downto 0), ena(6) and WE, CLK, o6(7 downto 0)); --reg6
    R61: REG8 port map(DIN(15 downto 8), ena(6) and WE, CLK, o6(15 downto 8));
    R70: REG8 port map(DIN(7 downto 0), ena(7) and WE, CLK, o7(7 downto 0)); --reg7
    R71: REG8 port map(DIN(15 downto 8), ena(7) and WE, CLK, o7(15 downto 8));

    M000: MUX4x4 port map (ADD_R1(1 downto 0), o0(3 downto 0), o1(3 downto 0), o2(3 downto 0), o3(3 downto 0), tmp00(3 downto 0));
    M001: MUX4x4 port map (ADD_R1(1 downto 0), o4(3 downto 0), o5(3 downto 0), o6(3 downto 0), o7(3 downto 0), tmp00(7 downto 4));
    M002: MUX4x4 port map ('0'&ADD_R1(2), tmp00(3 downto 0), tmp00(7 downto 4), "0000", "0000", DOUT1(3 downto 0));

    M003: MUX4x4 port map (ADD_R1(1 downto 0), o0(7 downto 4), o1(7 downto 4), o2(7 downto 4), o3(7 downto 4), tmp01(3 downto 0));
    M004: MUX4x4 port map (ADD_R1(1 downto 0), o4(7 downto 4), o5(7 downto 4), o6(7 downto 4), o7(7 downto 4), tmp01(7 downto 4));
    M005: MUX4x4 port map ('0'&ADD_R1(2), tmp01(3 downto 0), tmp01(7 downto 4), "0000", "0000", DOUT1(7 downto 4));

    M006: MUX4x4 port map (ADD_R1(1 downto 0), o0(11 downto 8), o1(11 downto 8), o2(11 downto 8), o3(11 downto 8), tmp02(3 downto 0));
    M007: MUX4x4 port map (ADD_R1(1 downto 0), o4(11 downto 8), o5(11 downto 8), o6(11 downto 8), o7(11 downto 8), tmp02(7 downto 4));
    M008: MUX4x4 port map ('0'&ADD_R1(2), tmp02(3 downto 0), tmp02(7 downto 4), "0000", "0000", DOUT1(11 downto 8));

    M009: MUX4x4 port map (ADD_R1(1 downto 0), o0(15 downto 12), o1(15 downto 12), o2(15 downto 12), o3(15 downto 12), tmp03(3 downto 0));
    M010: MUX4x4 port map (ADD_R1(1 downto 0), o4(15 downto 12), o5(15 downto 12), o6(15 downto 12), o7(15 downto 12), tmp03(7 downto 4));
    M011: MUX4x4 port map ('0'&ADD_R1(2), tmp03(3 downto 0), tmp03(7 downto 4), "0000", "0000", DOUT1(15 downto 12));

    M100: MUX4x4 port map (ADD_R2(1 downto 0), o0(3 downto 0), o1(3 downto 0), o2(3 downto 0), o3(3 downto 0), tmp10(3 downto 0));
    M101: MUX4x4 port map (ADD_R2(1 downto 0), o4(3 downto 0), o5(3 downto 0), o6(3 downto 0), o7(3 downto 0), tmp10(7 downto 4));
    M102: MUX4x4 port map ('0'&ADD_R2(2), tmp10(3 downto 0), tmp10(7 downto 4), "0000", "0000", DOUT2(3 downto 0));

    M103: MUX4x4 port map (ADD_R2(1 downto 0), o0(7 downto 4), o1(7 downto 4), o2(7 downto 4), o3(7 downto 4), tmp11(3 downto 0));
    M104: MUX4x4 port map (ADD_R2(1 downto 0), o4(7 downto 4), o5(7 downto 4), o6(7 downto 4), o7(7 downto 4), tmp11(7 downto 4));
    M105: MUX4x4 port map ('0'&ADD_R2(2), tmp11(3 downto 0), tmp11(7 downto 4), "0000", "0000", DOUT2(7 downto 4));

    M106: MUX4x4 port map (ADD_R2(1 downto 0), o0(11 downto 8), o1(11 downto 8), o2(11 downto 8), o3(11 downto 8), tmp12(3 downto 0));
    M107: MUX4x4 port map (ADD_R2(1 downto 0), o4(11 downto 8), o5(11 downto 8), o6(11 downto 8), o7(11 downto 8), tmp12(7 downto 4));
    M108: MUX4x4 port map ('0'&ADD_R2(2), tmp12(3 downto 0), tmp12(7 downto 4), "0000", "0000", DOUT2(11 downto 8));

    M109: MUX4x4 port map (ADD_R2(1 downto 0), o0(15 downto 12), o1(15 downto 12), o2(15 downto 12), o3(15 downto 12), tmp13(3 downto 0));
    M110: MUX4x4 port map (ADD_R2(1 downto 0), o4(15 downto 12), o5(15 downto 12), o6(15 downto 12), o7(15 downto 12), tmp13(7 downto 4));
    M111: MUX4x4 port map ('0'&ADD_R2(2), tmp13(3 downto 0), tmp13(7 downto 4), "0000", "0000", DOUT2(15 downto 12));

    DE : DCD3x8 port map (ADD_W, ena);

end logic;
```

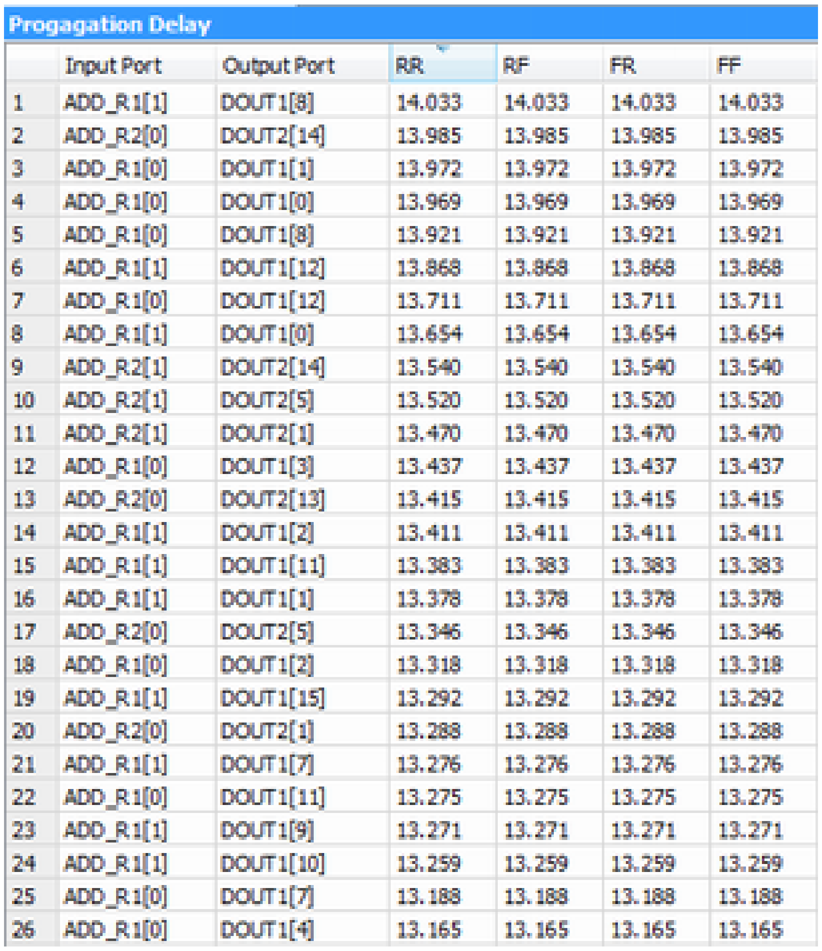
Figure 4: REG8x16.vhdl continued



Flow Summary	
Flow Status	Successful - Tue Mar 05 12:14:28 2013
Quartus II 64-Bit Version	12.1 Build 243 01/31/2013 SP 1 SJ Web Edition
Revision Name	reg8
Top-level Entity Name	REG8x16
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	168 / 33,216 (< 1 %)
Total combinational functions	168 / 33,216 (< 1 %)
Dedicated logic registers	128 / 33,216 (< 1 %)
Total registers	128
Total pins	59 / 475 (12 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Figure 5: Chip Resources

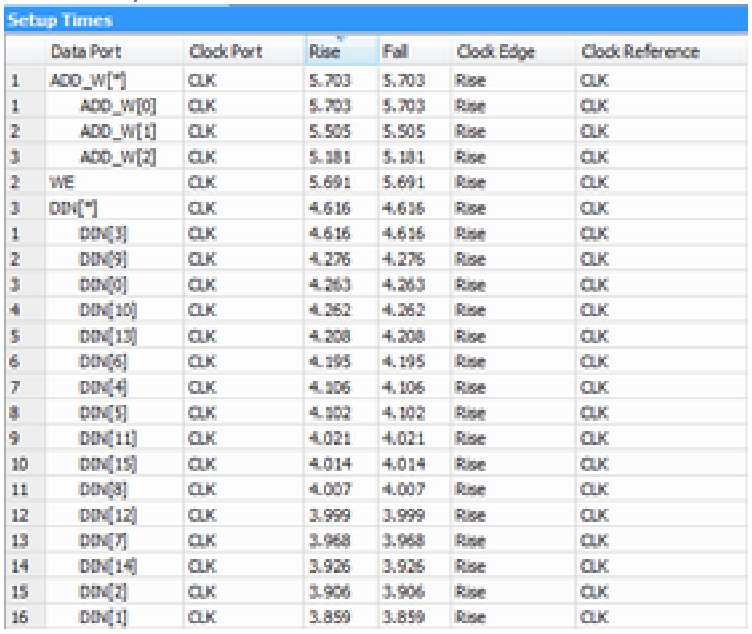
Table 1: Timing Delays



Progagation Delay

	Input Port	Output Port	RR	RF	FR	FF
1	ADD_R1[1]	DOUT1[8]	14.033	14.033	14.033	14.033
2	ADD_R2[0]	DOUT2[14]	13.985	13.985	13.985	13.985
3	ADD_R1[0]	DOUT1[1]	13.972	13.972	13.972	13.972
4	ADD_R1[0]	DOUT1[0]	13.969	13.969	13.969	13.969
5	ADD_R1[0]	DOUT1[8]	13.921	13.921	13.921	13.921
6	ADD_R1[1]	DOUT1[12]	13.868	13.868	13.868	13.868
7	ADD_R1[0]	DOUT1[12]	13.711	13.711	13.711	13.711
8	ADD_R1[1]	DOUT1[0]	13.654	13.654	13.654	13.654
9	ADD_R2[1]	DOUT2[14]	13.540	13.540	13.540	13.540
10	ADD_R2[1]	DOUT2[5]	13.520	13.520	13.520	13.520
11	ADD_R2[1]	DOUT2[1]	13.470	13.470	13.470	13.470
12	ADD_R1[0]	DOUT1[3]	13.437	13.437	13.437	13.437
13	ADD_R2[0]	DOUT2[13]	13.415	13.415	13.415	13.415
14	ADD_R1[1]	DOUT1[2]	13.411	13.411	13.411	13.411
15	ADD_R1[1]	DOUT1[11]	13.383	13.383	13.383	13.383
16	ADD_R1[1]	DOUT1[1]	13.378	13.378	13.378	13.378
17	ADD_R2[0]	DOUT2[5]	13.346	13.346	13.346	13.346
18	ADD_R1[0]	DOUT1[2]	13.318	13.318	13.318	13.318
19	ADD_R1[1]	DOUT1[15]	13.292	13.292	13.292	13.292
20	ADD_R2[0]	DOUT2[1]	13.288	13.288	13.288	13.288
21	ADD_R1[1]	DOUT1[7]	13.276	13.276	13.276	13.276
22	ADD_R1[0]	DOUT1[11]	13.275	13.275	13.275	13.275
23	ADD_R1[1]	DOUT1[9]	13.271	13.271	13.271	13.271
24	ADD_R1[1]	DOUT1[10]	13.259	13.259	13.259	13.259
25	ADD_R1[0]	DOUT1[7]	13.188	13.188	13.188	13.188
26	ADD_R1[0]	DOUT1[4]	13.165	13.165	13.165	13.165

Table 2: Setup Times



Setup Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	ADD_W[*]	CLK	5.703	5.703	Rise	CLK
1	ADD_W[0]	CLK	5.703	5.703	Rise	CLK
2	ADD_W[1]	CLK	5.505	5.505	Rise	CLK
3	ADD_W[2]	CLK	5.181	5.181	Rise	CLK
2	WE	CLK	5.691	5.691	Rise	CLK
3	DIN[*]	CLK	4.616	4.616	Rise	CLK
1	DIN[3]	CLK	4.616	4.616	Rise	CLK
2	DIN[9]	CLK	4.276	4.276	Rise	CLK
3	DIN[0]	CLK	4.263	4.263	Rise	CLK
4	DIN[10]	CLK	4.262	4.262	Rise	CLK
5	DIN[13]	CLK	4.208	4.208	Rise	CLK
6	DIN[6]	CLK	4.195	4.195	Rise	CLK
7	DIN[4]	CLK	4.106	4.106	Rise	CLK
8	DIN[5]	CLK	4.102	4.102	Rise	CLK
9	DIN[11]	CLK	4.021	4.021	Rise	CLK
10	DIN[15]	CLK	4.014	4.014	Rise	CLK
11	DIN[8]	CLK	4.007	4.007	Rise	CLK
12	DIN[12]	CLK	3.999	3.999	Rise	CLK
13	DIN[7]	CLK	3.968	3.968	Rise	CLK
14	DIN[14]	CLK	3.926	3.926	Rise	CLK
15	DIN[2]	CLK	3.906	3.906	Rise	CLK
16	DIN[1]	CLK	3.859	3.859	Rise	CLK

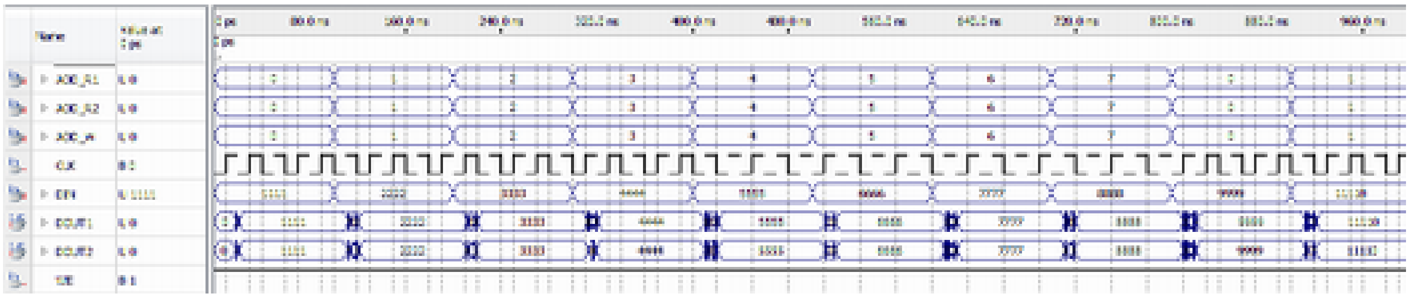


Figure 6: Simulation 1

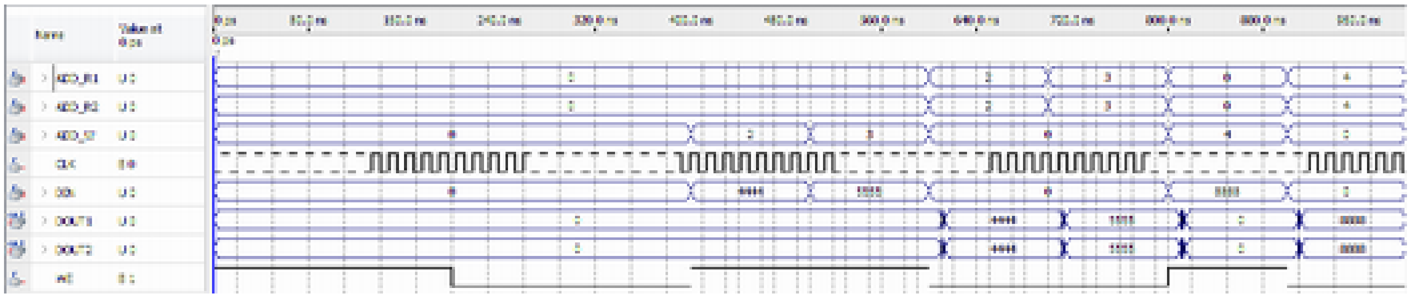


Figure 7: Simulation 2

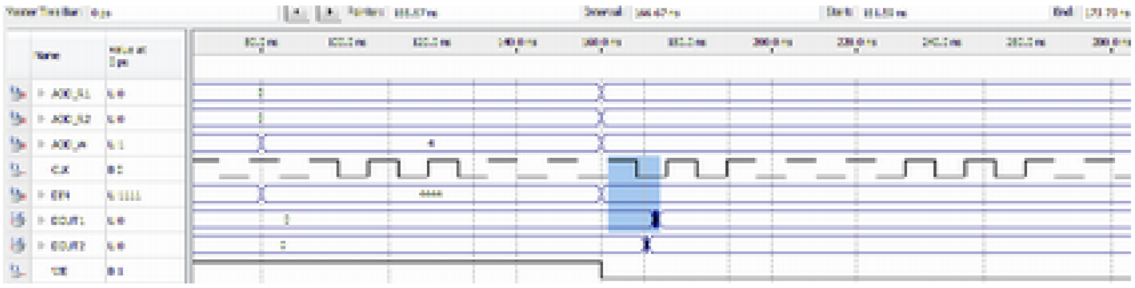


Figure 8: Delay from Clock Rising Edge to Stable Data Output for the First Register

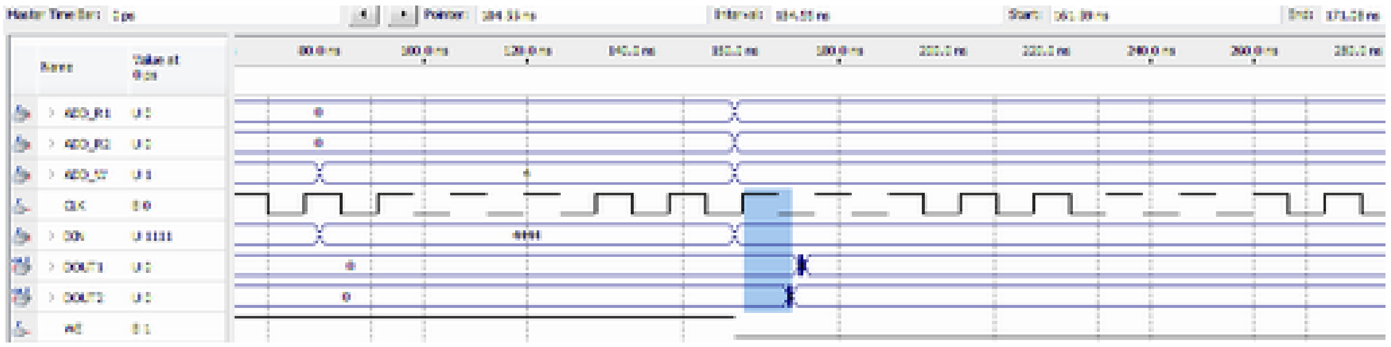


Figure 9: Delay from Clock Rising Edge to Stable Data Output for the Second Register