**Digital Design (CSCE 2114) – Lab 6**

**Objective:**

1. Learn how to write VHDL code to implement some simple combinational circuits.
2. Learn how to simulate your design before implementing it on the FPGA to verify that it works.

Vhdl code Hints:

* *VHDL is* ***not*** *case sensitive.*
* *Every line ends with* ***;*** *like C language.*
* *Single bits should be put inside single quotation marks and more than one bit should be put inside double quotation marks. '0' , '1' , “00001010”*
* *Reference “VHDL Notes” on class website for more examples and syntax*

A. 3-Input Function

1. In Quartus, create a new project named lab6\_a .(license: [1800@csce-licsrv.ddns.uark.edu](mailto:1800@csce-licsrv.ddns.uark.edu)).**Please refer to earlier labs** if you don’t know how to perform this step.
2. Click on File -> New and select VHDL File.

3. Write VHDL code of the combinational function F(x1, x2, x3) = m(0, 2, 3, 4, 5, 6). You are free to optimize your circuit and then implement it. Remember, you can use simple operators like: not, and, or. Also use parenthesis to prioritize the operators. You can refer to the VHDL\_note.ppt for some examples.

|  |
| --- |
| *LIBRARY ieee ; USE ieee.std\_logic\_1164.all ;*  *ENTITY lab6\_a IS*  *PORT ( x1,x2,x3 : in std\_logic;  f : out std\_logic);*  *END lab6\_a ;*  *ARCHITECTURE Behavior OF lab6\_a IS BEGIN*  *f<= …...................*    *END Behavior ;* |

4. After finishing your code, compile your project and correct any errors.

5. Add a waveform file and add all the 3 inputs and one output nodes to your waveform list as described in an earlier lab. Apply all 8 possible inputs with 100ns intervals.

* + 1. 0-100 ns : x1=0; x2=0 ; x3=0;
    2. 100ns -200ns : x1=0; x2=0; x3=1;
    3. 200ns -300ns : x1=0; x2=1; x3=0;
    4. 300ns -400ns : x1=0; x2=1; x3=1;
    5. 400ns -500ns : x1=1; x2=0; x3=0;
    6. 500ns -600ns : x1=1; x2=0; x3=1;
    7. 600ns -700ns : x1=1; x2=1; x3=0;
    8. 700ns -800ns : x1=1; x2=1; x3=1;

6. Run the simulation and check the output/results. Your simulation results should show that the output of your design matches the truth table of the function *f*.

7. Save a screen shot of both your VHDL Code and your waveform results and include it in your lab report. Close the project.

B: Flexible 5 bit signed Add/Sub with overflow and zero flag bits

1. In Quartus, create a new project named lab6\_b .Click on File -> New and select VHDL File.
2. Write the VHDL code for a flexible 5-bit add/subtract for signed numbers. It should do both addition and subtraction based on the *add\_sub* input. The port description is as follows:
   * + 1. X, Y: 5 bit inputs.
       2. Result: 5 bit output. Either ( X+Y) or (X-Y) depending on add\_sub input.
       3. add\_sub: If this input is One, then the Result should be the sum of inputs. Else, the result is X-Y.
       4. zero\_flag: It should go high Only when the result is Zero.
       5. Overflow: It should go high only when the operation leads to an overflow, and therefor the result is wrong. Remember that with 5 bits you can show signed numbers from -16 to +15, so any data outside this range will set the *overflow* flag to high.

You can refer to vhdl\_notes.ppt for extra help as how to complete the VHDL code.

|  |
| --- |
| LIBRARY ieee ;  USE ieee.std\_logic\_1164.all ;  USE ieee.std\_logic\_unsigned.all ;  ENTITY lab6\_b IS  PORT ( add\_sub : in std\_logic;  zero\_flag : out std\_logic;  X, Y : IN STD\_LOGIC\_VECTOR(4 DOWNTO 0) ;  Result : OUT STD\_LOGIC\_VECTOR(4 DOWNTO 0) ;  Overflow : OUT STD\_LOGIC ) ;  END lab6\_b ;  ARCHITECTURE Behavior OF lab6\_b IS  SIGNAL internal\_result : STD\_LOGIC\_VECTOR(5 DOWNTO 0) ;  BEGIN    internal\_result <= ('0'&X +Y) when add\_sub=………….. else ………………….;  Result <= internal\_result( …………. downto ………..);  Overflow <= X(4) xor Y(4) xor internal\_result(4) xor internal\_result(5);  zero\_flag <=………………. when internal\_result(………downto…….) = ……………. else …………………;  END Behavior ; |

1. After finishing your code, compile your project and correct any errors.
2. Add a waveform file and add all the inputs and the output nodes to your waveform list. Change the radix of all signals to ***signed decimal.*** Apply these inputs to X and Y and add\_sub with 100ns intervals.

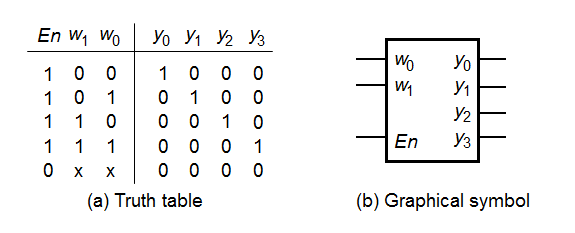
**Time interval: (x) add\_sub (y)**

* + 1. 0-100 ns : (+8) + (+8)
    2. 100ns -200ns : (-8) + (-8)
    3. 200ns -300ns : (+10) + (+6);
    4. 300ns -400ns : (-1) + (-10);
    5. 400ns -500ns : (+8) – (+8);
    6. 500ns -600ns : (+10) – (-6);
    7. 600ns -700ns : (+15) + (-16);
    8. 700ns -800ns : (+11) - ( -11);
    9. 800ns -900ns : (+5) + (+6);
    10. 900ns -1000ns : (+5) + ( -11);

1. Run the simulation and check the output/results. Your simulation results should show that the result, overflow and zero\_flag are correctly calculated based on your X, Y and add\_sub inputs. Save a screen shot of both your VHDL Code and your waveform results and include it in your lab report. Remember that in your lab report you should justify why overflow goes high or low for each case. Close the project.

C: Implementing A decoder

1. In Quartus, create a new project named lab6\_c .Click on File -> New and select VHDL File.
2. Write the VHDL code for a 2-4 Decoder with an Enable input. You can refer to the VHDL\_note.ppt for more help.



|  |
| --- |
| LIBRARY ieee ;  USE ieee.std\_logic\_1164.all ;  ENTITY lab6\_c IS  PORT ( w : IN STD\_LOGIC\_VECTOR(1 DOWNTO 0) ;  En : IN STD\_LOGIC ;  y : OUT STD\_LOGIC\_VECTOR(0 TO 3) ) ;  END lab6\_c ;  ARCHITECTURE Behavior OF lab6\_c IS  SIGNAL Enw : STD\_LOGIC\_VECTOR(2 DOWNTO 0) ;  BEGIN  Enw <= En & w ;  WITH Enw SELECT  y <= "1000" WHEN "100",  "….." WHEN "….",  "….." WHEN "….",  "….." WHEN "….",  "….." WHEN OTHERS ;  END Behavior ; |

3. After finishing your code, compile your project and correct any errors.

4. Add a waveform file and add all the inputs and outputs to your waveform list. Apply all 8 possible inputs with 100ns intervals.

* + 1. 0-100 ns : En=0; w1w0=00
    2. 100ns -200ns : En=0; w1w0=01;
    3. 200ns -300ns : En=0; w1w0=10
    4. 300ns -400ns : En=0; w1w0=11
    5. 400ns -500ns : En=1; w1w0=00;
    6. 500ns -600ns : En=1; w1w0=01;
    7. 600ns -700ns : En=1; w1w0=10
    8. 700ns -800ns : En=1; w1w0=11;

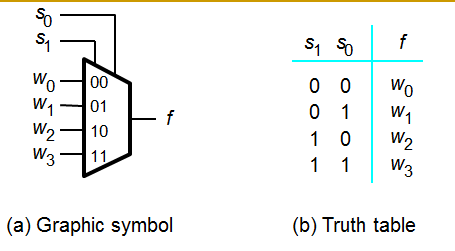
5. Run the simulation and check the output/results. Your simulation results should show that the output of your design matches the truth table of the decoder.

6. Save a screen shot of both your VHDL Code and your waveform results and include it in your lab report. Close the project.

D. Implementing a Multiplexer

1. In Quartus, create a new project named lab6\_d. Click on File -> New and select VHDL File.

2. Write the VHDL code for a 4-1 Multiplexer. You can refer to the VHDL\_note.ppt for more help.



|  |
| --- |
| LIBRARY ieee ;  USE ieee.std\_logic\_1164.all ;  ENTITY lab6\_d IS  PORT ( w0, w1, w2, w3 : in std\_logic;  s : IN STD\_LOGIC\_vector (1 downto 0) ;  f : OUT STD\_LOGIC ) ;  END lab6\_d ;  ARCHITECTURE Behavior OF lab6\_d IS  BEGIN  PROCESS ( w0, w1,w2, w3, s)  BEGIN  CASE s IS  WHEN '00' =>  f <= w0 ;  ….................    WHEN OTHERS =>  f <= w3 ;  END CASE ;  END PROCESS ;  END Behavior ; |

3. After finishing your code, compile your project and correct any errors.

4. Add a waveform file and add all the inputs and outputs to your waveform list. Assign ‘1’ to *W0*, ‘X’ to *W1*, ‘Z’ to *W2* and ‘0’ to *W3*. Check output *f* for all possible *S0, S1* inputs.

* + 1. 0-100 ns : s=00
    2. 100ns -200ns : s=01;
    3. 200ns -300ns : s=10
    4. 300ns -400ns : s=11
    5. 400ns-500 ns : s=00
    6. 500ns -600ns : s=01;
    7. 600ns -700ns : s=10
    8. 700ns -800ns : s=11

5. Run the simulation and check the output/results. Your simulation results should show that the output of your design matches the truth table of the multiplexer.

6. Save a screen shot of both your VHDL Code and your waveform results and include it in your lab report. Close the project.