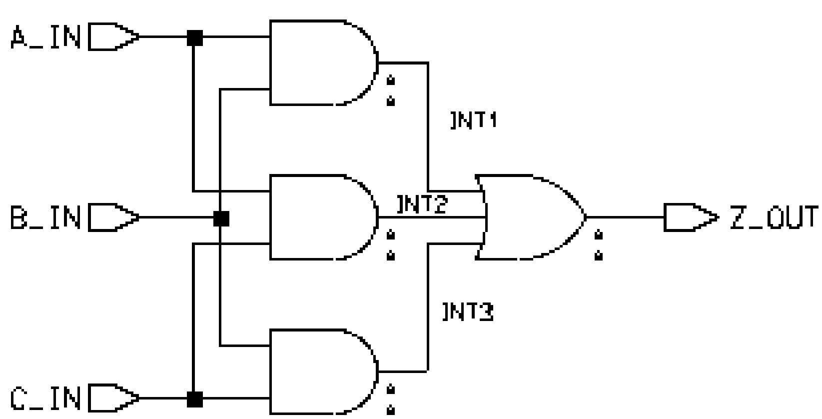
ECE C03
LAB#5
Design of combinational and sequential circuits using VHDL

Spring 1998
Assigned: Tuesday, May 26, 1998
Due: Thursday, June 4, 1998
at the BEGINNING of class

1 Introduction

In this assignment you will learn how to use the Mentor Graphics tools to enter, compile, and simulate digital circuits described using VHDL. To start with, you will begin by entering a simple combinational circuit using VHDL to represent a majority detector. You will simulate this by using a command file in the simulator. Then you will describe a small finite state machine representing a counter, and simulate that using VHDL commands. After this, you will be on your own to represent the 4-bit adder from Lab 2, and the finite state machine from Lab 4.

1.1 Majority detector



This circuit implements the majority function. It has three inputs, and one output. the resulting output is always high when a majority (2 of the 3) of the inputs are true, and false otherwise. To start with, you need to create a directory in the ecec03 hierarchy that is for lab #5. Create the directory *lab5* in the same way that you created any of your other directories, either by using the design manager menus or the UNIX system commands. Change into this directory, and using your favorite editor, notepad in the design manager, or from within Design Architect select Open VHDL and enter the following VHDL and save with the filename *majority.vhd.*

```
entity AND2_OP is
```

```
 port
  (A, B : in BIT;
   Z    : out BIT);
end AND2_OP;

architecture MODEL of AND2_OP is
begin
 Z <= A and B;
end MODEL;

entity OR3_OP is
 port
  (A, B, C : in BIT;
   Z       : out BIT);
end OR3_OP;

architecture MODEL of OR3_OP is
begin
 Z <= A or B or C;
end MODEL;

-- Interface
entity MAJORITY is
  -- Input/Output ports
  port
   (A_IN, B_IN, C_IN : in BIT;
    Z_OUT            : out BIT);
end MAJORITY;

-- Body
architecture STRUCTURE of MAJORITY is
 -- Declare logic operators
 component AND2_OP
  port (A, B : in BIT; Z : out BIT);
 end component;
 component OR3_OP
  port (A, B, C : in BIT; Z : out BIT);
 end component;

 -- declare signals to interconnect logic operators
 signal INT1, INT2, INT3 : BIT;
begin
 -- Connect logic operators to describe schematic
 A1 : AND2_OP port map (A_IN, B_IN, INT1);
 A2 : AND2_OP port map (A_IN, C_IN, INT2);
 A3 : AND2_OP port map (B_IN, C_IN, INT3);
```

```
 O3 : OR3_OP port map (INT1, INT2, INT3, Z_OUT);

end STRUCTURE;
```

After entering and saving this in the *lab5* directory, you need to create the working directory for the VHDL compiler. You can do this in two ways, one from the command line, and one from within Design Architect. From the commandline, type *qhlib work* and it will create the work library where the parts of the compiled simulation will be placed. From within design architect, deselect all windows by clicking on the background, click QuickHDL and select qhlib from the menu. At the prompt enter *something/ecec03/lab5/work*

Now it is necessary to compile your design. Again, there are two ways to compile the design, one from the command line, and one from within Design Architect. To compile from the command line, you need to enter *qvhcom majority.vhd*. From within Design Architect, again from the QuickHDL menu, you need to select qvhcom. Enter the name of design, or use the navigator to select it and then click ok. If there are any errors in the output, you need to check your source, and recompile it.

Finally, you need to simulate your design. Before you can do this, you need to enter the following lines into a command file called *majority.cmd*. This will be then used to simulate the majority detector. After entering this file with an editor or notepad, you need to start the simulator. There are again two methods of doing this. The first is to type *qhsim* from the UNIX command line. The second is to select simulate from the QuickHDL menu.

Once you have started the simulator you need to select the appropriate module (in this case Entity majority). Depending on how you started the simulator, you might need to set up the signals and the wave windows. To do this, select View->Signals and then View->Wave from the simulator menu. You need to then export the signals so that the appropriate ones are shown in the waveform display. You can export then individually by selecting one, and then Wave->Selected Signals in the Signals window, or by clicking the first one, holding down shift, and then clicking the last before selecting Wave->Selected signals. This will cause those signals to be displayed in the Wave window.

To finally simulate the majority detector, select Execute Command File from the File menu. Select the majority.cmd file from the list, and click Execute. At this point, if all goes well, the output waveforms should appear in the wave window. If you wish to restart the design and run the simulation again, clicking File->Restart Design will clear the wave display.

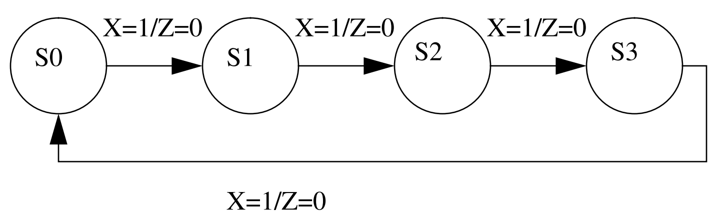
To print your waveforms, select File->Write Postscript to output a postscript file describing the waveform display. You might wish to zoom in such that the entire waveform is displayed in the output window.

```
run 10
force -freeze /a_in 1
run 10
force -freeze /a_in 0
force -freeze /b_in 1
run 10
force -freeze /a_in 1
run 10
force -freeze /c_in 1
```

```
force -freeze /a_in 0
force -freeze /b_in 0
run 10
force -freeze /a_in 1
run 10
force -freeze /a_in 0
force -freeze /b_in 1
run 10
force -freeze /a_in 1
run 10
```

1.2 Counter Finite State Machine

In this section you will enter the following source (as fsm.vhd) which describes a counter. Notice as you enter it that the commands to simulate it are written in VHDL rather than as a command file like the last part. When you have finished entering this file, compile it like you did the last one (you don't need to redo the work directory) and start the simulator with it. Make sure that you select the correct model to simulate, in this case you want to start with fsm_tester, which then invokes the fsm module. Make sure that you select the signals and set up the wave display, then simply invoke the run option. This should run the simulation in the VHDL file and output a display to the wave window. As above, if there are any errors in your simulation or the compilation stage, you will need to go back and check your initial VHDL file.




```
package states is
type state is (S0, S1, S2, S3); -- state can take one of these values.
attribute ENUM_ENCODING : STRING;
attribute ENUM_ENCODING of state : type is "00 01 10 11";
end states;

use work.states.all;

entity test is
  port (X, clock: in bit;
        Z : out bit);
end test;

architecture trial of test is
signal ST : state;
begin
```

```
process
begin
  wait until clock'event and clock = '1';
  if X='0' then Z<='0';
  else
    case ST is
    when S0 => ST <=S1; Z <= '0';
    when S1 => ST <=S2; Z <= '0';
    when S2 => ST <=S3; Z <= '0';
    when S3 => ST <=S0; Z <= '1';
    end case;
  end if;
end process;
end trial;

entity tester is
end;

architecture test1 of tester is
  signal clk, inp, outp : bit := '0';
  component test
   port (X : in bit;
         clock: in bit;
         Z: out bit);
  end component;
  for U1: test use entity work.test(trial);
begin
  U1: test port map (inp, clk, outp);
  clock_tic: process begin
  loop
   clk <= '1';
   wait for 5 ns;
   clk <= '0';
   wait for 5 ns;
end loop; end process;

state_changes: process
  begin
   inp <= '1' after 0 ns,
          '0' after 10 ns,
              '1' after 20 ns,
              '1' after 30 ns,
              '0' after 40 ns,
              '0' after 50 ns,
              '0' after 60 ns,
              '0' after 70 ns,
```

```
              '1' after 80 ns,
              '1' after 90 ns,
              '0' after 100 ns,
              '0' after 110 ns,
              '1' after 120 ns,
              '0' after 130 ns,
              '1' after 140 ns,
          '0' after 150 ns,
              '0' after 160 ns,
              '1' after 170 ns;
  wait;
  end process;
end test1;
```

2.3 Four Bit adder

For this section you are on your own. You need to create the four-bit adder from Lab 2 and simulate it similar to the majority circuit in Section 2.1. To do this you will want to build up the components much like the majority circuit, and similar to the way that you first built full adders in lab 2 to complete the final 4 bit ripple carry adder.

To simulate this, you will need to create your own command file, from the inputs given in Lab 2 and simulate it. Print the final output waveform from the wave window to the printer.

2.4 Pattern recognizer

In this section, you need to recreate the finite state machine from Lab 4. To do this, you will need to create something similar to the VHDL for the second section. You will want to use the state assignements that you used in the final construction of the FSM.

For simulation purposes, use the do file that was provided for Lab 4, and construct the VHDL code to test your FSM. Run the simulation using the same techniques provided for the second section and print your final output waveform.

3 Questions

TURN IN the following items for a grade (**stapled** in this order):

- √ A listing of the VHDL file for each circuit (4 total).
- √ A simulation trace of each circuit (4 total) clearly marked as to which circuit the trace corresponds to.

Be sure to include your name and section number on each page.