Northwestern University
Electrical and Computer Engineering
ECE303: Advanced Logic Design, Spring 02
Prof. Hai Zhou

May 30, 2002
Handout #9
Due: Jun 6

Lab 5 Finite State Machine Design and Simulation by VHDL

In this lab, you will use Mentor Graphics tools to enter, compile, and simulate a finite state machine using VHDL. The finite state machine is the one you designed in Lab 4, whose functionality is as follows.

> A finite state machine has one input (X) and two outputs (Z_1 and Z_2). An output $Z_1 = 1$ occurs every time the input sequence 101 is observed, provided the sequence 011 has never been seen. An output $Z_2 = 1$ occurs every time the input 011 is observed. Note that once $Z_2 = 1$, $Z_1 = 1$ can never occur. You need to implement the machine in the Mealy design style.

Based on this description, the interface can be defined as

```
entity string_checker is
port (X, clock:  in bit; Z1, Z2:  out bit);
end string_checker;
```

In Lab 4, you have come up with a state diagram for this machine. That can be one architecture of `string_checker`. So we can have

```
architecture diagram of string_checker is
--decleration of type and variable
type state is (S0, S1, S2, S3, S4, S5, S6, S7);
begin
process
variable current:  state := S0;
begin
-- Please fill in your design
...
end process;
end diagram;
```

In Lab 4, you have also come up with a gate-and-FF design of the machine. So we can also define the other architecture of the `string_checker` to describe that design.

```
architecture gate_and_FF of string_checker is
--decleration of signals signal QA, QB, QC: bit := '0';
begin
-- Please fill in your design
...
end gate_and_FF;
```

After you complete the two designs, you can include the following test bench to test your designs.

```
entity test_bench is
end;
```

```
architecture test1 of test_bench is
signal clk, input, diagram_Z1, diagram_Z2, gate_Z1, gate_Z2:  bit := '0';
component string_checker
port (X, clock:  in bit; Z1, Z2:  out bit);
end component;
for U1:  string_checker use entity work.string_checker(diagram);
for U2:  string_checker use entity work.string_checker(gate_and_FF);
begin
U1:  string_checker port map (input, clk, diagram_Z1, diagram_Z2);
U2:  string_checker port map (input, clk, gate_Z1, gate_Z2);
clock_tic:  process begin
loop
clk <= '1';
wait for 5 ns;
clk <= '0';
wait for 5 ns; end loop;
end process;

input_changes:  process
begin
input <= '1' after 0 ns,
'1' after 10 ns,
'1' after 20 ns,
'0' after 30 ns,
'0' after 40 ns,
'1' after 50 ns,
'0' after 60 ns,
'1' after 70 ns,
'0' after 80 ns,
'0' after 90 ns,
'0' after 100 ns,
'1' after 110 ns,
'1' after 120 ns,
'0' after 130 ns,
'1' after 140 ns,
'0' after 150 ns,
'1' after 160 ns,
'1' after 170 ns;
wait;
end process;
end test1;
```

You need to type in your designs and the test bench in a file (say `checker.vhd`). This can be done by using your favorite editor or the notepad in the design manager, or from within Design Architect select Open VHDL and save it to a file.

After this, 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 design 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/ece303/lab5/work`.

Now it is necessary to compile your design. Again, there are two ways to compile the design, one from the commandline, and one within Design Architect. To compile from the commandline, you need to enter `qvhcom checker.vhd`. From within Design Architect, again from the QuickHDL menu, you need to select qvhcom. If there are any errors in the output, you need to check your source and re-compile it.

Finally, you need to simulate your design. You can type `qhsim` from the UNIX command line or select simulate from the QuickHDL menu. Once you have started the simulator you need to select the appropriate module (here it is entity test_bench). 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 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 display in the Wave window. Simply invoke the run option to start the simulation.

To print waveforms, select File/Write Postscript to output a postscript file describing the waveform display.

You need to turn in a print-out of the VHDL source file and the waveforms.