ECE745: ASIC Verification
Lab Assignment #1

Questa SystemVerilog Tutorial

PART 1: Getting Started with the Environment

Lab Introduction

In this laboratory exercise, you are going to complete the Questa tutorial at NCSU and get yourself acquainted with some of the fundamental components (interface, program body and top-level integration) of a testbench. You will also learn to recognize a few basic data-types and their usage in a testbench environment. The basic idea is to a) Get the feel for a typical testbench using SystemVerilog and b) to get comfortable with the Questa verification environment c) To get acquainted with the DUT that will be used in all the Laboratory exercises henceforth.

Tool Introduction

QuestaSim is part of the Questa Advanced Functional Verification Platform and is the latest tool in Mentor Graphics tool suite for Functional Verification. The tool provides simulation support for latest standards of SystemC, SystemVerilog, Verilog 2001 standard and VHDL. This tool is an advancement over Modelsim in its support for advanced Verification features like coverage databases, coverage driven verification, working with assertions, SystemVerilog constrained-random functionality.

Step 1: Establishing the Design Environment for compilation

One time setup for a given directory used for simulation:

Each time you create a directory for simulations you would have to do the following

```
prompt%> add questasim63 OR prompt%> add modelsim
```

Copy the modelsim.ini file that comes with this tutorial into the directory. This file sets up the necessary defaults for the Questa tool.

Create the library into which all the design units will be compiled. Do this by performing

```
prompt%> vlib mti_lib
```

(Note that the name “mti_lib” corresponds to the variable “work” within the modelsim.ini file and is the library to which all the source code would be compiled to create a design single entity). Note that in some cases, if the compilation seems to crash for a reason you think is incorrect, it would be advisable to delete the “mti_lib” directory (Use: \rm –rf mti_lib OR vdel –all) and re-create it as shown above.

Setup for simulations within a directory for a given session:

We assume the previous step has already been followed. Let us assume a directory has been setup up correctly and you come into this directory for a future simulation. You would still need to run the following commands each time you start a set of simulations for a given design within a directory.

set paths to the Modelsim tool:

```
prompt%> add questasim63 OR prompt%> add modelsim
```

set environment variable MODELSIM to modelsim.ini

```
prompt%> setenv MODELSIM modelsim.ini
```

At this point, all the path settings are good to go for the executables associated with Questa. It is assumed here that user is aware of the requirements for remote access from a Windows platform. If not, the information can be obtained from the remote access page.

Compilation and Simulation

The Questa tool enables compilation of multiple design/verification/modeling units (each of which might be in a different language) into a common library (called the working library) and a common design representation. This enables each individual unit of the entire simulation to be compiled independently and incremental compilation to be performed. At this point it must be stated that the compilation of the source code can either be done within the simulation environment (GUI) or on the command prompt. The simulation though MUST be performed within the simulation environment. In the interest of simplicity, we shall be performing the compilation of the source code on the command prompt of the UNIX/Linux terminal and providing a basic understanding of the tool capabilities.

All the work should be done within the same folder say *<PATH TO FOLDER>/Simulation/*

To begin the compilation and simulation process, please download the following into a directory of your choice *(<PATH TO FOLDER>/Simulation/)*

The Protected and unprotected Verilog RTL for the Design Under Test (DUT):

- **Basic_ALU.vp**

The SystemVerilog (SV) Testbench for this RTL:

- **Basic_ALU.tb.sv**
- **Basic_ALU.if.sv**
- **Basic_ALU.test_top.sv**

This corresponds to a very basic ALU block to introduce you to the tools.

The SystemVerilog (SV) Testbench for this RTL:

- **Basic_ALU.if.sv** the creation and use an interface to the DUT with a clocking block and a modport.

- **Basic_ALU.tb.sv** the creation of a program which provides constrained stimulus to the DUT. This code has been written to provide the user a very basic introduction to a typical program structure with tasks and passing of signals into the DUT.

- **Basic_ALU.test_top.sv** the creation of a top level integration of the DUT, the interface and the program for sending stimulus to the design.

For this tutorial the testbench is only going to be used for the creation of constrained stimulus and for sending these stimuli into the DUT.

At this point we can either do step3 in the GUI environment or at the command prompt. We shall continue with the command prompt.

The next steps need to be performed each time a code change is performed and the changes need to be reflected in the simulations.

Step1: Compilation of source code

Compile the Verilog RTL Model: A good feature is that Verilog(protected an unprotected) source files can be compiled out of order. Thus, we begin by compiling the protected verilog files by doing:

```
prompt%> vlog *.vp
```

Which results in the message:

```
QuestaSim vlog 6.3c Compiler 2007.09 Sep 11 2007

-- Compiling module Basic_ALU


Top level modules:

        Basic_ALU
```

Which implies that the high level module in the hierarchy that was discovered is Basic_ALU.

Compile the SystemVerilog Testbench: The SV code shares the same ability to be compiled in any order. This holds as long as the includes from one file do not beat on those in another file. It is always suggested to follow an ordered compilation. Thus, SystemVerilog compilation can be done by running the commands

```
prompt%> vlog -sv -mfcu Basic_ALU.if.sv Basic_ALU.tb.sv Basic_ALU.test_top.sv
```

A successful compilation of the systemverilog files should spit out the following message:

```
QuestaSim vlog 6.3c Compiler 2007.09 Sep 11 2007
-- Compiling interface Basic_ALU_interface
-- Compiling program Basic_ALU_test
-- Compiling module Basic_ALU_test_top

Top level modules:
        Basic_ALU_test_top
```

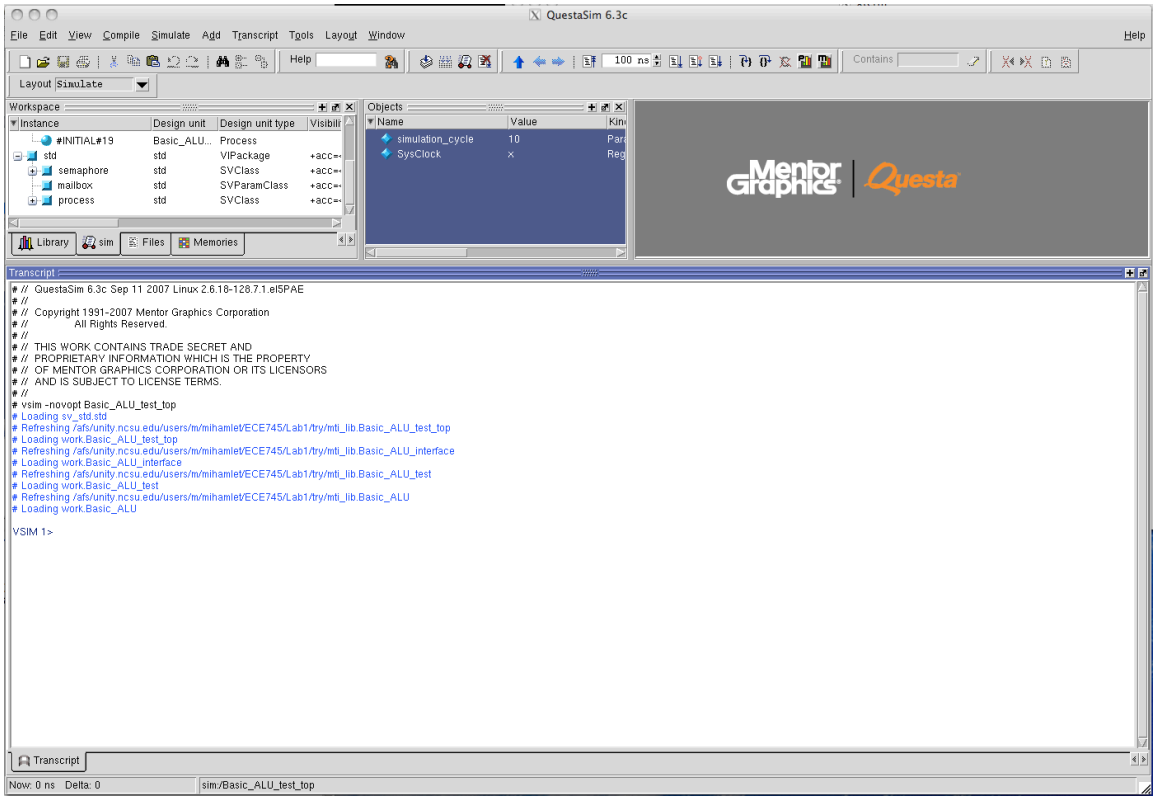
The most important thing to note in this example is that the topmost design unit is Basic_ALU_test_top which has an instance of the "testbench Basic_ALU_test" connected to the DUT "Top" through the interface "ALU_Interface".

Step2: Simulation of design

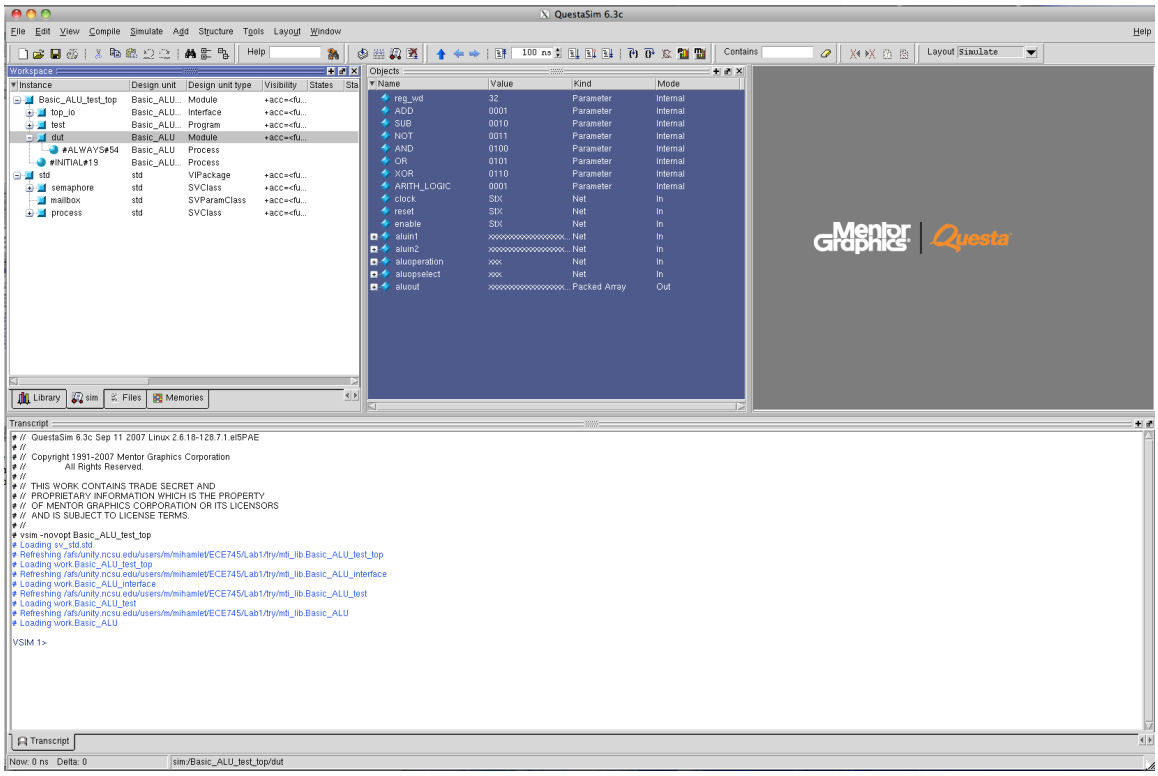
We can now simulate the design unit (testbench + DUT) by invoking the simulator (vsim) with the top level design unit (Basic_ALU_test_top).

```
% vsim -novopt Basic_ALU_test_top
```

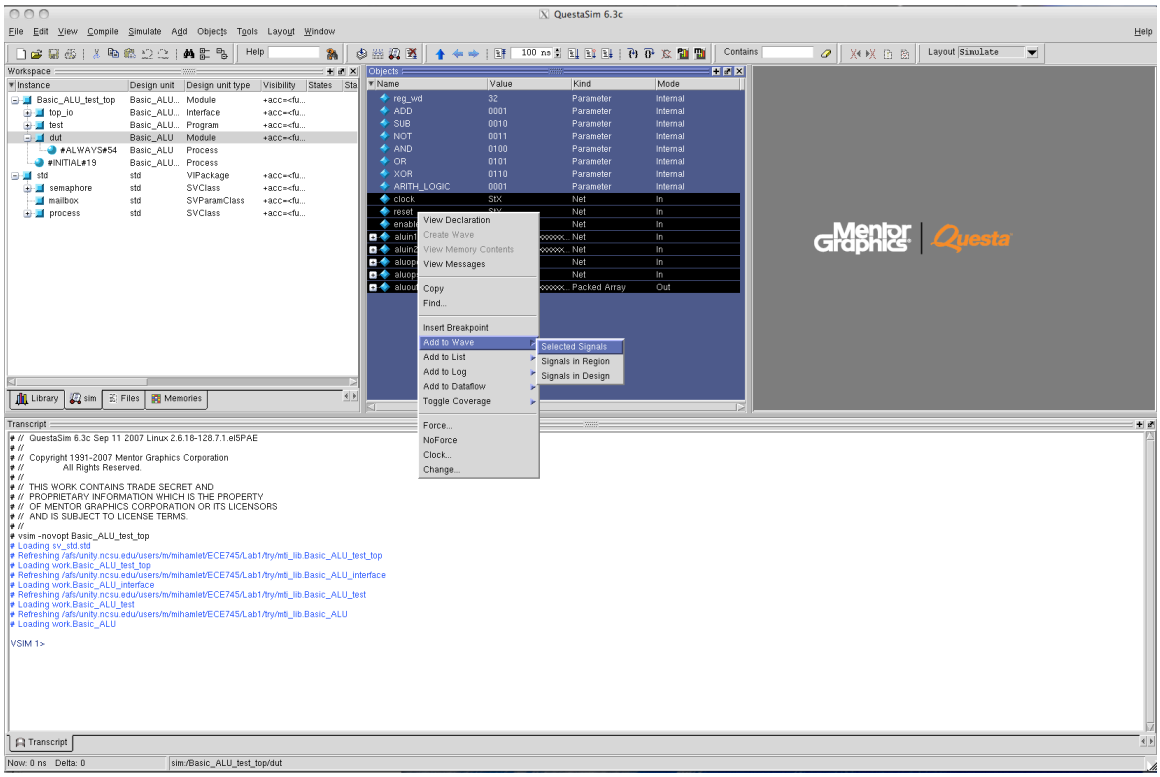
This leads to the invocation of the simulation GUI with three windows: Workspace, Objects and transcript. To begin with, note that the prompt is of the form VSIM #>, where # corresponds to the number of commands issued in the simulation mode. Also, at this point the Workspace window would have tabs corresponding the simulation hierarchy (sim), the source files (Files) and the analysis tool for the register arrays in the system if any (Memories). The objects window contains variables, wires, regs, variables etc their values and their characteristics. An example of the window at this point is shown below:



In the figure below, the hierarchy corresponding to the DUT has been expanded and its contents populate the objects window by virtue of clicking on the name DUT on the workspace window.



At this point we need to have a means of viewing the values that are of interest to us. To add the signals of interest to the waveform viewer, we would need to select these from the Objects window -> right click -> add to wave -> select signals. This can also be done for a given instance in the Workspace window but this might need to an excessive number of values which might be overkill. An example set of signals in the Object window have been highlighted and sent to the the waveform window below:



The execution of the above leads to the appearance of the waveform window. To begin with, this window could be a part of (docked with) the GUI but can be separated (undocked) from the GUI frame by hitting the button. This creates a new window with the signals and variables of choice populating the frame to the left. There is going to be a need to adjust the frames to make the the names and signal values appear in a manner favorable to you. We leave it up to you to play around with the waveform window. If the design is not too big, then you can run the command below to log all signals at all levels of hierarchy. If the command below was not executed prior to a simulation run, only those signals specified in the waveform window would be logged and hence any new signal of interest would require restarting of the simulation:

```
VSIM#> log -r /*
```

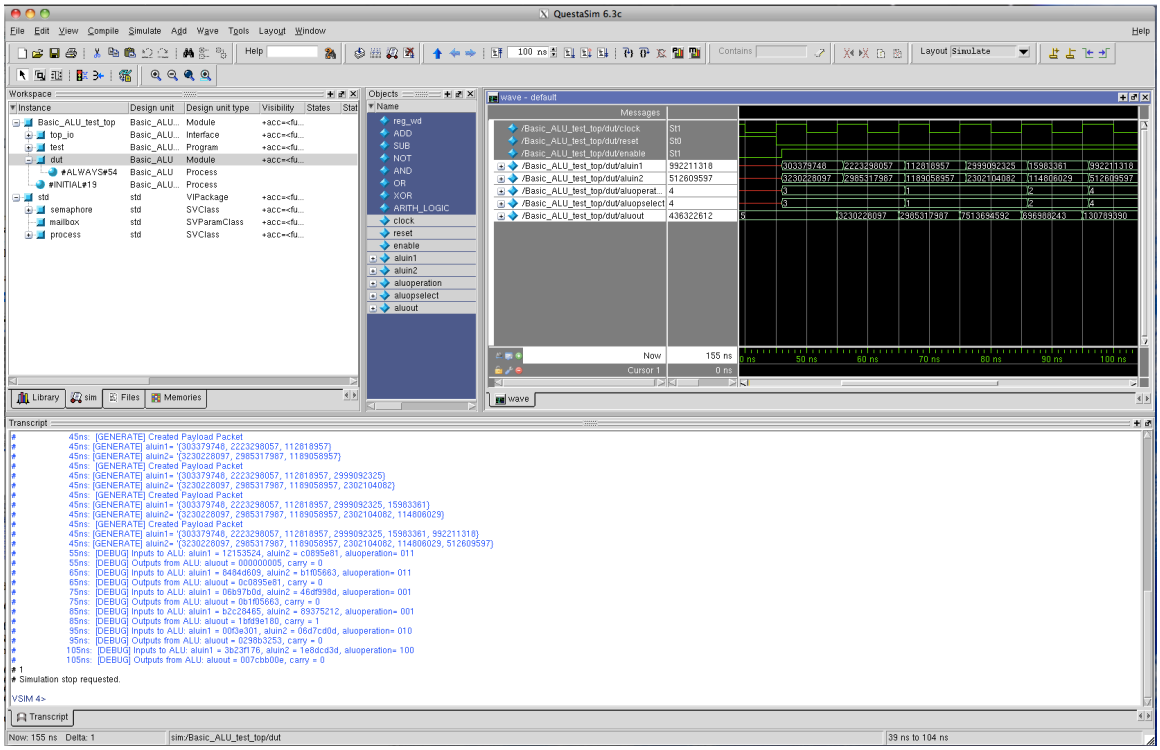
To run the simulation we need to run the command (for a 200ns advancement of simulation time OR a full run)

```
VSIM#> run 200ns (or any other time that you would like) OR  VSIM#> run -all (for a full run of the simulation)
```

It must be noted that the testbench must have the ability to finish (using $finish, exit) at some point during the simulation to do a "run -all". Else, the simulation would run iterate forever.

On issuing the above command, the contents of the waveform window could be zoomed in. To zoom use the standard zoom buttons (maginifying glass). A useful

shortcut is to use hit "f" to zoom full (the 200ns that the simulation has been run). The waveform window that you see should mirror what you see below for the case where we have add clock, reset, enable, aluin1, aluin2, aluoperation, aluopselect, etc. from the object window to the waveform. The figure also shows you one other useful feature where you can change the radix of the numerical representation to one that you are comfortable with.



To end a simulation run

```
VSIM#> exit
```

Also, if changes were made to the code while the GUI was up, you can recompile (on the UNIX command prompt if you so wish) and then re-start the simulation by running (assuming we want to recompile ALL the files).

```
VSIM#> vlog *.sv -sv -mfcu
VSIM#> vlog *.vp
VSIM#> vlog *.v
VSIM#> restart -f
```

We could have recompiled just the files that were modified.

Lab 1 Questions:

1. What does `repeat(5) @(ALU_Interface.cb)` do?

2. Describe the functionality and the timing characteristics of the reset task

3. There are multiple tasks defined in the testbench provided. Based on the concepts conveyed in Lecture 1, please state the layers (in a layered testbench) that the tasks would belong to.

4. What lines would you modify to make the testbench run 10 times?

5. What set of inputs would you send in (relevant input signals only need to be stated) for the following to be executed:
   a. Add the numbers 32 and 67.

   b. Subtract 579 from 4967.

   c. Perform a NOT operation on the binary equivalent of 876

   d. AND together 579 and 982

   e. OR together 579 and 982

f. XOR together 579 and 982

6. Send in each of the inputs from the previous question into the DUT by modifying the testbench program. Also, determine whether the outputs are along expected lines. If not, state the errors.

7. Observe the operation of the clock, reset and enable signals. Determine if they are operating along expected lines and if not state the errors.

8. Locate and Document any bugs found in the space below. Be as specific as possible. You may need to run stimuli other than the ones you generated above.

Lab1 Submission Requirements:

The generated inputs and resulting outputs are shown in the QuestaSim log from $display statements within the testbench.

Make sure that these results are displayed from each of your inputs by running your program just as stated in the tutorial. The same file names (`Basic_ALU.tb.sv, Basic_ALU.test_top.sv, Basic_ALU.if.sv`) need to be used.

Follow the following steps for submissions (Solaris/Linux only please) `mkdir Lab1` (creates the directory Lab1) copy all the SystemVerilog files into the Lab1 directory Answer all the above questions in a file called `Lab1.doc` and copy this into the

Lab1 directory Zip the file using the command > `zip Lab1.zip Lab1/*`
Submit the zip using the submit utility on the course webpage.