Project Assignment

In this assignment, you will need to design the BIST hardware for a circuit-under-test (CUT) that has **4 scan chains** (with Mux-D flip-flops). Your BIST hardware will need to apply 2000 pseudo-random scan patterns to the CUT and compact the response. You may work with a partner on this project or work alone.

The following directory contains all the files that you need:

/home/projects/courses/spring_19/ee382m-1/project2

You need to modify the project.v file to add the BIST hardware. You should not modify the scan_cut.v file at all. The testbench.v file has been added to help you test your design. The testbench runs the BIST hardware for the fault-free circuit (your BIST hardware should pass the fault-free circuit). Then it injects one fault at a time into the CUT and runs the BIST hardware for the faulty circuit (your BIST hardware should fail the faulty circuits). When you submit your project.v file, we will test it with a modified version of the testbench.v file that will inject a different set of faults.

Your verilog code that implements the BIST hardware need not be synthesizable. It just has to correctly simulate the way the BIST hardware should operate. A stub module is included in the project.v file which defines the I/O interface to the BIST hardware. The input signals are: **clk**, **rst**, **bistmode**, and **cut_sdo[0:3]**. The output signals are **cut_scanmode**, **cut_sdi[0:3]**, **bistdone**, **bistpass**. All flip-flops in the circuit are positive-edge triggered on the **clk** signal. When both **rst** and **bistmode** are activated, that initiates the BIST session. Your BIST hardware should start testing the CUT at that point. Your BIST hardware controls the CUT with the signals **cut_scanmode** and **cut_sdi[0:3]**. Each positive-edge of **clk** when **cut_scanmode** is activated, the scan chains shifts in the value on **cut_sdi[0:3]**. If **cut_scanmode** is not activated, then the circuit changes state as it would normally do during a system clock cycle. The values being shifted out of the scan chain come on the **cut_sdo[0:3]** signals. When your BIST hardware is finished testing the CUT, then it should set the value of **bistpass** to 1 if the CUT passed the test or 0 if the CUT failed the test. Lastly, your BIST hardware should activate the signal **bistdone** to indicate that the BIST session is done and the results are available on the **bistpass** signal. The testbench and CUT interface with the BIST hardware through these signals. You should not modify these signals in any way.
A primitive polynomial for a 16-bit LFSR is $x^{16} + x^5 + x^4 + x^3 + 1$.

The verilog code in the project directory is compiled with the following command from the UNIX command line:

```
vcs -full64 testbench.v scan_cut.v project.v
```

You can run **vcs** from the LRC 64-bit machines. First do “module load syn/vcs” to set the environment variables before you use **vcs**. More information about using Verilog on the LRC computers is given in the latter pages of this assignment handout. After compiling it, an executable file “simv” will be created which you can execute by using **./simv**.

What you need to turn in:

You need to submit a correctly working project.v file before 11:59pm on the due date on Canvas. It will be tested using a testbench to see that the BIST hardware works correctly. You also need to submit a 1-2 pages report answering the following questions:

1) How you implemented the BIST hardware
2) What challenges you faced in getting the BIST hardware to work.
3) An idea of how much hardware would be required to implement your design.

Information about Using Verilog at LRC

VCS works by compiling your Verilog source code into object files or translating them into C source files. VCS invokes a C compiler (cc, gcc, or egcs) to create an executable file that will simulate your design. This simulator can be executed on the command line and can create a waveform file.

VCS is available on ECE LRC 64-bit machines. There is an extensive user guides for vcs located in the directory: /misc/linuxws/packages/synopsys_2012/vcs-mx/doc/UserGuide.
This user guide includes another tutorial for the VCS tools.

More Information about Verilog

Note that you do not need to design your BIST hardware at the gate level. You can use high-level behavioral statements. For example, if you wanted to implement a counter, you could do the following:

```
reg [5:0] state;
wire count16;

always @(posedge clk)
begin
  if ( rst == 1 )
    state = 0;
  else
    state = state + 1;
end
assign #1 count16 = (state == 16);
```

Remember that when you assign a value to a signal that will be sampled on a clock edge, you need to have a delay. That is what the "#1" in the example above does. When state changes, the signal on count16 does not update until after 1 unit of delay.