
DO NOT SHARE
SLIDES AND CLASS MATERIALS
ON ONLINE SITES
Course Hero

CSEE W4823 Lab#2
Verilog HDL & Modelsim

Xiaofu Pei

2nd October, 2019

Acknowledgement: Jae-bin Choi & Mingoo Seok, previous TAs

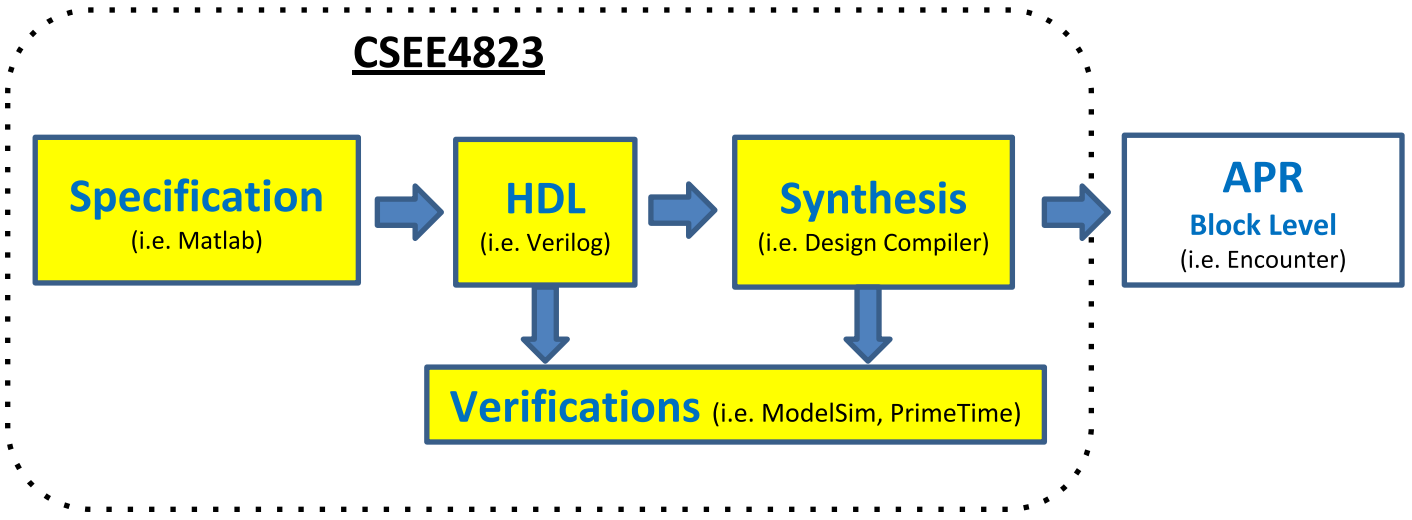
Topics

- Lab#1: Design flow & Matlab®
- Lab#2: Verilog HDL / ModelSim®
- Lab#3: Synthesis / Design Compiler®
- Lab#4: Timing and power analysis / PrimeTime®
- Lab#5: Memory Compiler
- Lab#6 and following labs: Project based lab

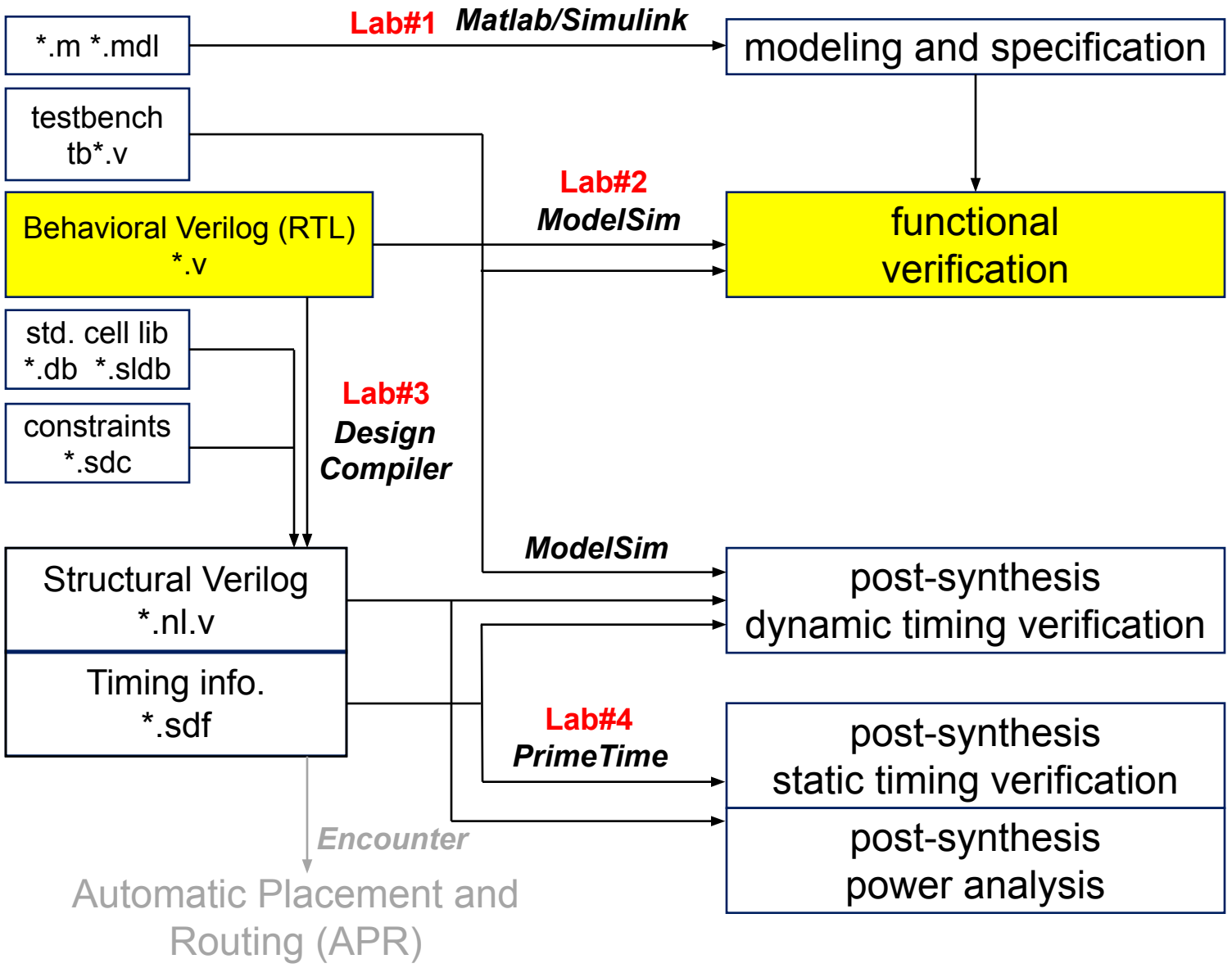
(Lab 3 and Lab 4 may be combined to meet schedule)

Design Flows

- Semi-Custom flow (**CSEE4823**, EE6920, EE6321)
  - Digital blocks
  - Need standard cells for synthesis
  - Large scale digital circuits (e.g. >10^6 transistors)
  - Reduces design time at the expense of performance compared to custom flow



Semi-Custom Flow



Hardware Description Language (HDL)

- An efficient way to design, verify, and test circuits
  - Custom design → Fine-grained but with low efficiency
  - Semi-custom design→ HDL → Higher development efficiency
- Popular HDLs: Verilog/SystemVerilog, VHDL
- If you are not familiar with Verilog HDL, you can use the following tutorials:
  - http://www.asic-world.com/verilog/veritut.html
  - IEEE Verilog Standard
  - http://www.doulos.com/knowhow/verilog_designers_guide
  - feel free to find one manual that looks good to you

Things to Know in Verilog (Design)

- Necessary:
  - Abstraction level (behavioral, gate-level)
  - Module declaration and instantiation
  - Reg, wire usage
  - If else, case statement, looping statement
  - Signed vs unsigned arithmetic, bit and logical operators
  - Procedural & continuous assignment
  - RTL style coding (look at the document uploaded)
- Helpful
  - Generate blocks
  - Iterating over 2D arrays
  - Part select
  - Parameters, localparam
  - Task

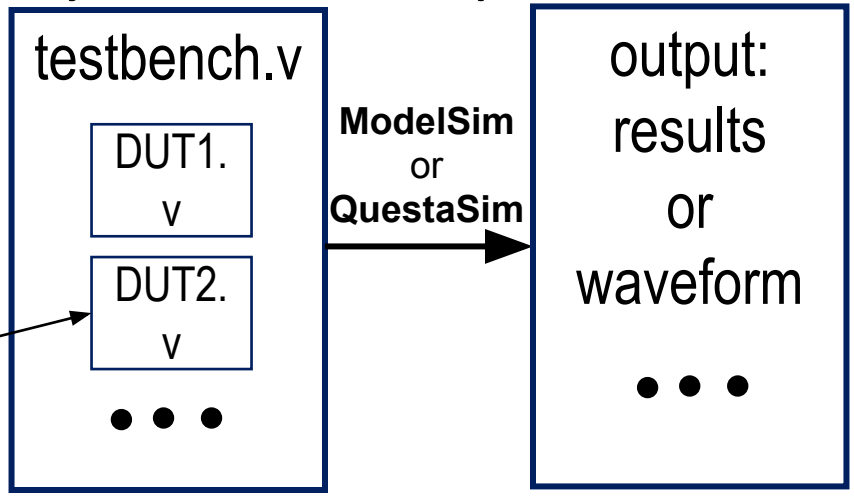
Things to Know in Verilog (Testbench)

- Many times the design is correct but the test setup is wrong
- File I/O for giving inputs and checking the results
- Initial block
- Task (makes the code readable and modular)
- Back annotation (Will see later)
- Dumping vcd for power analysis (Will see later)
- Tracing the hierarchy
  - For vcd
  - Changing the waveformat.do

Verilog HDL Simulator: QuestaSim

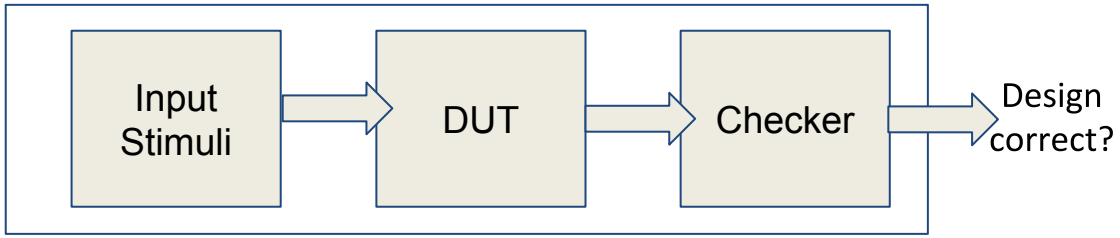
- Verilog simulator: QuestaSim by Mentor-Graphics
- Simulator I/O files

1. Testbench file (testbench.v)
   - defines input waveform
   - defines output detection
2. Source files
   - RTL-level Verilog file (*.v)
   - gate-level netlist (output of synthesis or APR)
3. How to define timing constraints?
   - RTL-level: use “#xx” expression (not affect actual circuit)
   - gate-level: include *.sdf file (output of synthesis or APR)



Testbench

- Instantiate DUT
- Generate clk
- In the initial block
  - Initialize all the inputs
  - **Delay-control**: Give inputs for testing, use small delays to reflect real inputs
  - **Event-control**: Use @(posedge clk) to time when to give inputs and check outputs
  - For loops to run through several clock cycles
  - Finally say $finish
- The testbench starts processing from the initial block
- Simulation ends at $finish or you can time the simulation in runsim.do



Debug tips

- ‘x’ means undefined
  - See if you gave reset correctly
  - Check if inputs have been initialized correctly
- ‘z’ means high impedance
  - No logic driving the wire
  - There maybe bit width mismatch in connections
- After functional verification synthesize **individual** blocks and verify them (Will see later)

To start with…

- Folders and files
  - Create your local folders

    [your_path]/project/rtl/lfsr1/ for source Verilog

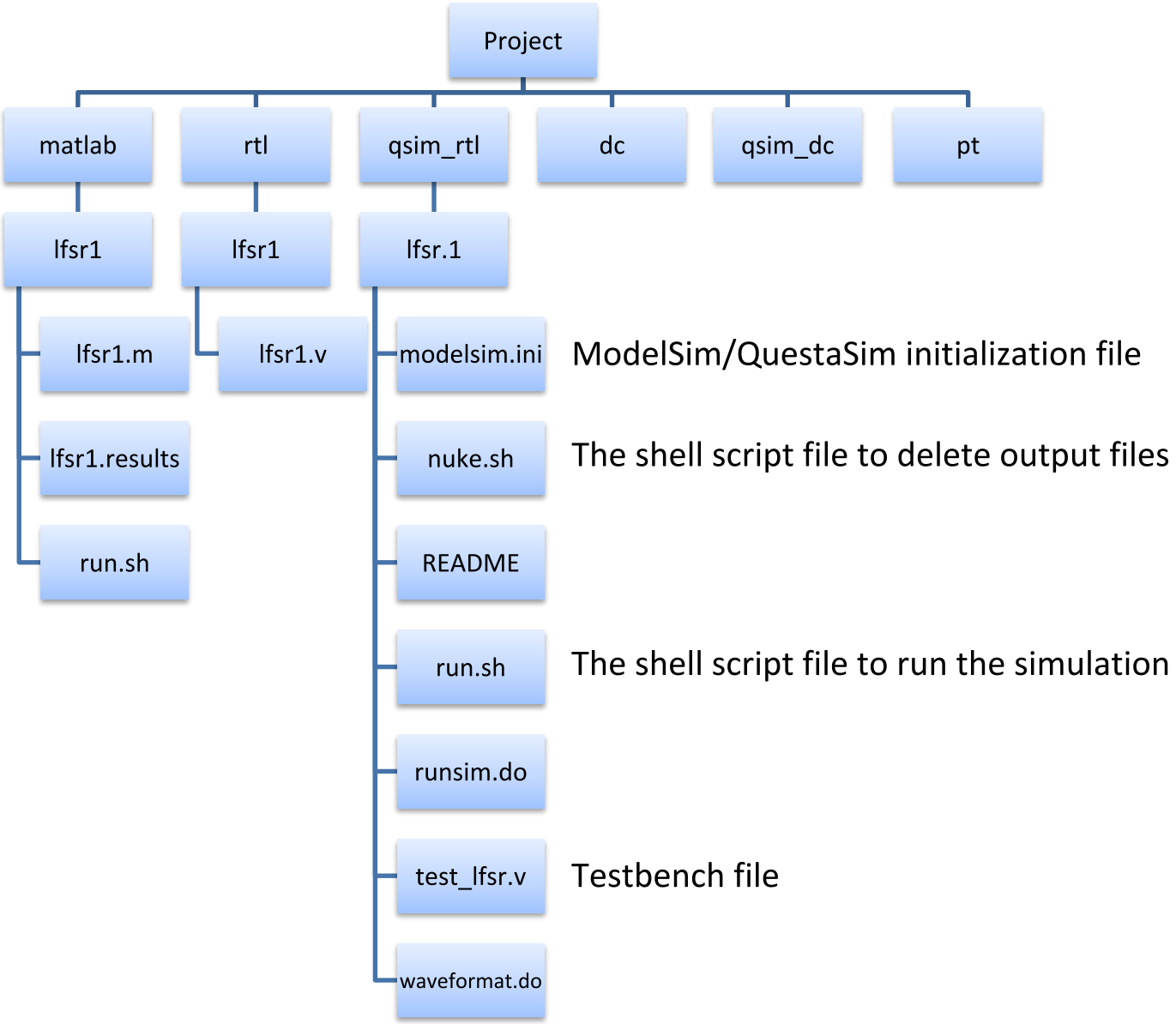
    [your_path]/project/qsim_rtl/lfsr1/ for testbench Verilog/simulation scripts/results
  - Copy source codes from

    /courses/ee6321/share/**4823-fall2020**/rtl/lfsr1/lfsr1.v

    /courses/ee6321/share/**4823-fall2020**/qsim_rtl/lfsr1/*.*

    to your local folder

Project Hierarchy



lfsr1.v

```
`timescale 1ns/1ps
module lfsr1 (clk, resetn, seed, lfsr_out);
        input clk, resetn;
        input [15:0] seed;
        output [15:0] lfsr_out;

        reg [15:0] lfsr_out;
       wire [15:0] lfsr_next;
```

- `timescale specifies the time unit and precision for the modules
  - Format: `timescale <time_unit>/<time_precision>
- input/output, wire/reg element declaration
- [Important]: Please check the difference between “reg” and “wire” elements

lfsr1.v

```
        always @(posedge clk) begin
            if (~resetn) begin
                lfsr_out <= #0.1 seed;
            end
            else begin
                lfsr_out <= #0.1 lfsr_next;
            end
        end

      assign lfsr_next = {lfsr_out[14:0], lfsr_out[15]^lfsr_out[12]^lfsr_out[5]^lfsr_out[0]};

endmodule
```

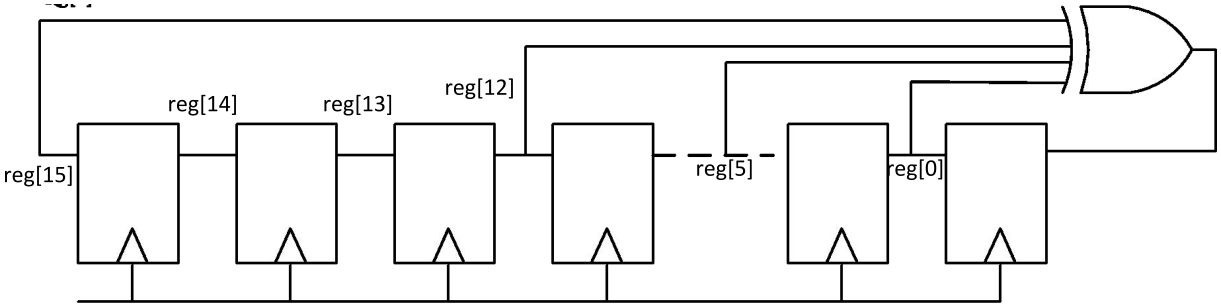
- ‘always@’: Behavioral presentation – Procedure statement
- Synchronous reset is utilized
- It is better to use the separated element for non-blocking assignment (<=) (Up to you!)
- ‘assign’: continuous statement
- #0.1 is the artificial delay (Ignored in logic synthesis)
  - Because time_unit is 1ns, the delay is 100ps

Matlab vs. Verilog

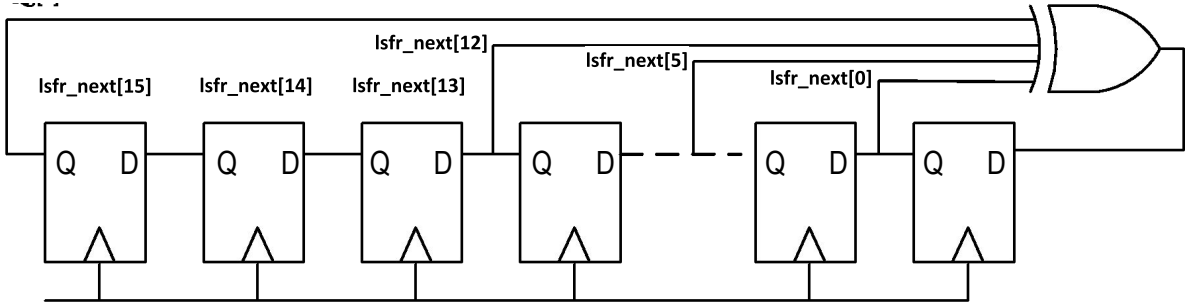
```
        always @(posedge clk) begin
            if (~resetn) begin
                lfsr_out <= #0.1 seed;
            end
            else begin
                lfsr_out <= #0.1 lfsr_next;
            end
        end

       assign lfsr_next = {lfsr_out[14:0], lfsr_out[15]^lfsr_out[12]^lfsr_out[5]^lfsr_out[0]};
endmodule
```

Matlab



Verilog



test_lfsr.v

```
`timescale 1ns/1ps
`define SD #0.010
`define HALF_CLOCK_PERIOD #0.90
`define QSIM_OUT_FN "./qsim.out"
`define MATLAB_OUT_FN "../../matlab/lfsr1/lfsr1.results"

module testbench();
	reg clk;
	reg resetn;
	reg [15:0] seed;

	integer lfsr_out_matlab;
	integer lfsr_out_qsim;

	wire [15:0] lfsr_out;

	integer i;
	integer ret_write;
	integer ret_read;
	integer qsim_out_file;
	integer matlab_out_file;
	integer error_count = 0;

	lfsr1 lfsr_0 ( .clk(clk), .resetn(resetn), .seed(seed), .lfsr_out(lfsr_out) );
```

Device-Under-Test (DUT) instantiation

- **clk**: Using a ‘reg’ because is used at the left-hand side of an always assignment
- **resetn, seed**: using a ‘reg’ in an initial block

test_lfsr.v

- **clk generator**: toggle clk every half clock period

```
always begin
	`HALF_CLOCK_PERIOD;
	clk = ~clk;
end
```

- **File I/O**

```
initial begin
	// File IO
	qsim_out_file = $fopen(`QSIM_OUT_FN,"w");
	if (!qsim_out_file) begin
		$display("Couldn't create the output file.");
		$finish;
	end

	matlab_out_file = $fopen(`MATLAB_OUT_FN,"r");
	if (!matlab_out_file) begin
		$display("Couldn't open the Matlab file.");
		$finish;
	end
```

- **$fopen** opens a disk file
  - QSIM_OUT_FN file is opened for writing
  - MATLAB_OUT_FN file is opened for reading

test_lfsr.v

```
// register setup
clk = 0;
resetn = 0;
seed = 16'd1;
@(posedge clk);

@(negedge clk);   // release resetn
resetn = 1;

@(posedge clk);   // start the first cycle
```

- Initial ‘clk’=0, ‘resetn’=0, ‘seed’=1
  - **clk** signal toggles every HALF_CLOCK_PERIOD
  - At the first falling edge of clk, **resetn** becomes 1

test_lfsr.v

```
for (i=0 ; i<256; i=i+1) begin
            // compare w/ the results from Matlab sim
            ret_read = $fscanf(matlab_out_file, "%d", lfsr_out_matlab);
            lfsr_out_qsim = lfsr_out;

            $fwrite(qsim_out_file, "%0d\n", lfsr_out_qsim);
            if (lfsr_out_qsim != lfsr_out_matlab) begin
                  error_count = error_count + 1;
            end

            @(posedge clk);  // next cycle
      end
```

- From the second rising edge of **clk**, for loop statement starts
  - You can read the file content by using **$fscanf**
  - **$fwrite** writes the data on the file
  - At the next clock's rising edge, it starts new comparison
- In this testbench, 256 comparisons are repeated

test_lfsr.v

```
            // Any mismatch b/w rtl and matlab sims?
            if (error_count > 0) begin
                  $display("The results DO NOT match with those from Matlab :( ");
            end
            else begin
                  $display("The results DO match with those from Matlab :) ");
            end

            // finishing this testbench
            $fclose(qsim_out_file);
            $fclose(matlab_out_file);
            $finish;
      end

endmodule // testbench
```

- Display the comparison result
- Close the file descriptor

Shell Script Files

- **run.sh**

```
vsim -do "runsim.do"
```

  - **vsim** command is used to invoke the VSIM simulator
  - **'-do':** instructs **vsim** to use the command specified by "cmd" or the "file"
  - Format: vsim [-do "cmd" | <file>]

- **nuke.sh**

```
\rm -rf work modelsim.ini
\rm -rf *.log *.syn *.rpt *.mr *.nl.v *.sdf *.svf *.vcd transcript *.wlf
```

  - remove generated file (log files, netlists, ...)

runsim.do

```
#####################################################
#  Modelsim do file to run simuilation
#####################################################
vlib work
vmap work work

# Include Source File and Testbench
vlog +acc -incr ../../rtl/lfsr1/lfsr1.v
vlog +acc -incr test_lfsr.v
# Run Simulator
vsim +acc -t ps -lib work testbench
do waveformat.do
run -all
```

- **vlib** command creates a design library
- **vmap** command defines a mapping between a logical library name and a directory
  - Format: vmap [<logical_name>] [<path>]
- **vlog** command compiles Verilog source code into a specified working library
  - **'-incr'**: performs an incremental compile
- **vsim** command
  - '-t': Specifies the simulator time resolution
  - '-lib': Specifies the default working library
  - Format: vsim [-arguments] [<library_name>.<design_unit>]
- **do** command executes commands contained in a macro file
- **run** command advances the simulation

Reference: https://users.ece.cmu.edu/~kbiswas/modelsim/se_man.pdf

waveformat.do

```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate /testbench/clk
add wave -noupdate /testbench/resetn
add wave -noupdate /testbench/i
add wave -noupdate -radix unsigned /testbench/lfsr_out
add wave -noupdate -radix unsigned /testbench/lfsr_out_matlab
add wave -noupdate /testbench/lfsr_out_qsim
add wave -noupdate -radix unsigned /testbench/error_count
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {3 ns} 0}
quietly wave cursor active 1
configure wave -namecolwidth 223
configure wave -valuecolwidth 89
configure wave -justifyvalue left
configure wave -signalnamewidth 0
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
configure wave -timeline 0
configure wave -timelineunits ps
update
WaveRestoreZoom {0 ns} {12 ns}
```

- **onerror** commands specifies one or more commands to be executed when a running macro encounters an error
- **add wave** commands adds the following objects to the window
- **WaveActiveNextPane** commands create a pane which contains signals

waveformat.do

```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate /testbench/clk
add wave -noupdate /testbench/resetn
add wave -noupdate /testbench/i
add wave -noupdate -radix unsigned /testbench/lfsr_out
add wave -noupdate -radix unsigned /testbench/lfsr_out_matlab
add wave -noupdate /testbench/lfsr_out_qsim
add wave -noupdate -radix unsigned /testbench/error_count
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {3 ns} 0}
quietly wave cursor active 1
configure wave -namecolwidth 223
configure wave -valuecolwidth 89
configure wave -justifyvalue left
configure wave -signalnamewidth 0
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
configure wave -timeline 0
configure wave -timelineunits ps
update
WaveRestoreZoom {0 ns} {12 ns}
```

- **TreeUpdate** command refreshes waveforms
- **WaveRestoreCursors** command restores any cursors you set during the original simulation
- **WaveRestoreZoom** command restores the Zoom range you set
- **Configure wave** commands are used only in saved Wave format files

Reference: https://users.ece.cmu.edu/~kbiswas/modelsim/se_man.pdf