Hardware Verification using DFT & TetraMax

CS 6745 - Testing and Verification of Digital Circuits

Anh Luong & Andrzej Forys

1 Objective

This project′s goal was to combine our new knowledge of hardware verification with our previous VLSI experiences including fabrication and post silicon testing. Throughout the Testing and Verification of Digital Circuits course, we learned the algorithms for fault detection and what happens behind the scenes of test pattern generators and verifiers. Given the Synopsys tools designed for this type of testing, Design for Test (DFT) Compiler and TetraMAX, our goal was to learn about the tools, determine the additional hardware and software needed for functionality, and apply them to previously designed hardware in Verilog.

2 Introduction

DFT and TetraMAX tools are useful for testing and verification of pre and post silicon digital circuits, including finding errors in logic and fabrication. Although these can be used for both sequential and combinational circuits, the verification of sequential systems requires the use of Scan Flip-Flops (FF). In this report we will cover the basic designs of Scan FFs, scan chain insertion using DFT, and preparing library files and verilog files for Automatic Test Pattern Generation (ATPG) with TetraMAX.

The referenced report by Edwin Jose covers the basics of using DFT and TetraMAX to perform Full Scan versus Partial Scan Analysis [1]. The tools and commands used in that report have since been updated several versions and it is no longer up to date. The workflows have also slightly changed. This report will also provide new library files and scripts to use the latest versions of the tools.

3 Background

Part of the verification process requires the use of the Synopsys Design Compiler (DC). Although our scripts and library can be used to first synthesize the design, we will not cover the DC details because we assume the user has common knowledge of this tool. This also applies to the process of creating cells that are used to generate the library. Refer to ”Digital VLSI Chip Design with Cadence and Synopsys CAD Tools” for more information about these procedures [2].

These are the versions of the tools we have been using for this project:

DC and DFT Compiler Versions D-2010.03-SP2
TetraMax Version F-2011.09-SP4 (S13 Directory, script updated for new installation)
TetraMax Version W-2004.12-SP4 (S12 Directory)

4 Synopsys Tools

Synopsys has two main tools used for this type of testing: the DFT Compiler and TetraMAX. These tools can test faults in both combinational and sequential logic [3]. In the case of sequential logic that uses FFs, a special Scan FF needs to be developed. This gives the tools or users direct control over the values inside registers. It allows for propagating values into and out of the data path [4].

4.1 DFT Compiler

The DFT Compiler uses additional features on top of the standard DC to regenerate the netlist with Scan FFs. It modifies the structural Verilog produced through DC by replacing standard FFs with Scan FFs. The tool is smart enough to update all the cells and nets associated with the new design. It runs its own test design rule checking (DRC) and is able to fix several clock, reset, and scan signal routing errors with AutoFix. The DFT tool has access to DFTMAX which allows it to create the ATPG test protocol and compressed netlist for the entire design, which is used by TetraMAX. Along with providing circuit size, number of nets, and timing delay analysis before and after scan insertion, the tool provides scan coverage details such as how many faults are testable and what types of faults are encountered [3]. Figure 1 shows the procedure flow used by the DFT Compiler.

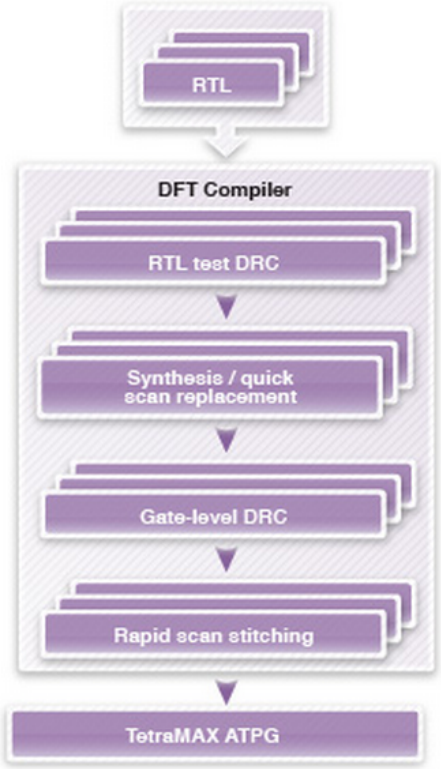

Figure 1: Design procedure in DFT Compiler [5].

4.2 Design Vision (DV)

Synopsys also has a GUI based DFT tool called Design Vision which lets the designer view a full schematic of his or her structural Verilog before and after Scan FF replacement. It can be used as an interactive command window and a schematic viewer. This can be useful in finding DRC errors, because the tool can show specific failure locations in the schematic. When using the tool to only view schematics of the netlists generated by DFT, a structural Verilog file is needed along with the test protocol defining STIL file (.spf) [3]. Figure 2 shows a section of our scan inserted design.

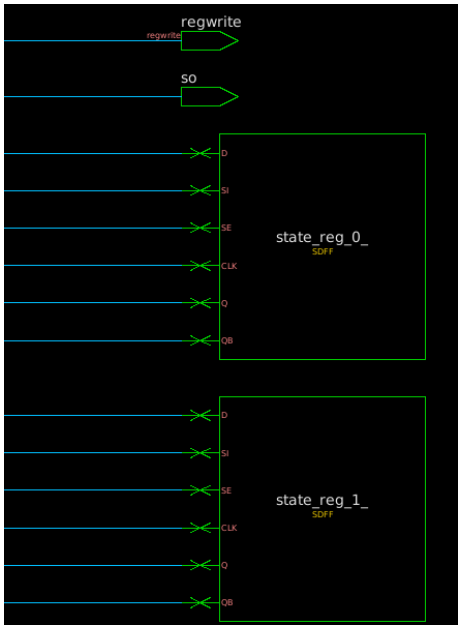

Figure 2: Scan chain in processor controller.

4.3 TetraMAX ATPG

Another Synopsys tool, called TetraMax ATPG, is used to generate the test patterns for fault analysis. This tool will determine all testable faults along with those that are unreachable or untestable due to issues like redundancy. It will generate a testbench compatible file corresponding to all the generated patterns. Although in some cases similar to the fault analysis done by the DFT Compiler, its main difference is the ability to generate test patterns for the entire design, test individual faults, and show the propagation results to the user [4]. Figure 3 shows the TetraMAX ATPG process flow.

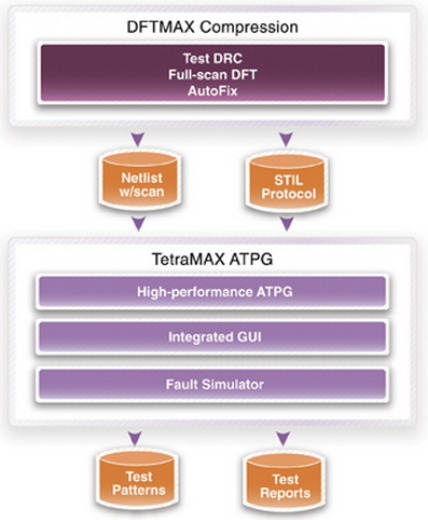

Figure 3: Process of TetraMAX ATPG [6].

5 Verification Procedures

5.1 Scan Flip-Flop Design

The difference between ordinary FFs and Scan FFs is that the latter has more than one data input pin which is activated during test modes. There are two types of Scan FFs we considered: the kind that share a clock with the rest of the integrated circuit (IC) while utilizing a ′scan enable′ signal to activate the ′scan input′ and the kinds that use a separate ′scan clock′ to switch between data sources [3]. Our implementation is based on the design using the scan enable signal.

The simplest way of designing Scan FFs like this is to add a multiplexer (MUX) on the data input line. The scan enable signal is the select bit on the MUX and it controls whether the FF receives its input from the data path or scan input. We designed 3 slightly different Scan FFs for this project. The first used a FF and a MUX from our own cell library. The second used cells from the UofUDigital_v_2 library accessible to VLSI students [2]. The third was modeled after a standard design found in [7].

The schematic, layout, and analog_extracted views need to be created for the scan cell before starting library generation. Figure 4 shows the schematic for the standard design from [7]. When finished simulating the schematic, it′s useful to independently test a behavioral view to verify a properly working Scan FF. Next, once Layout Versus Schematic Checking (LVS) shows that netlists match between the schematic and layout, the analog_extracted view can be generated [2]. This will later be used to generate the cell library file. The layout based on our own library can be seen in Figure 5.

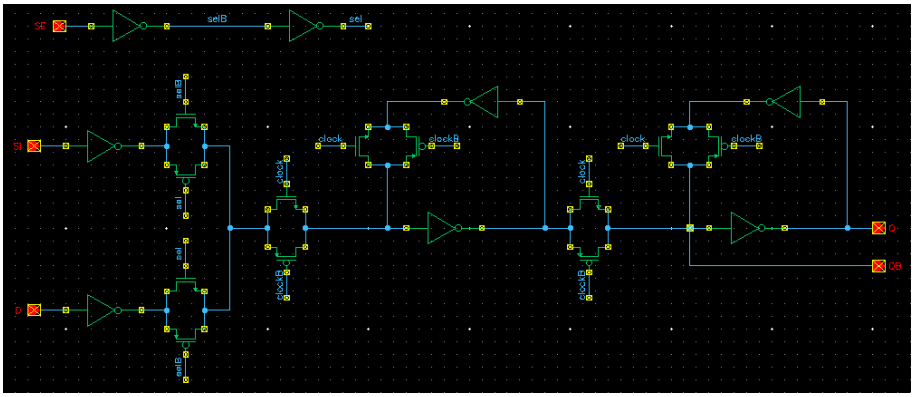

Figure 4: Schematic of Scan FF based on design in [7].

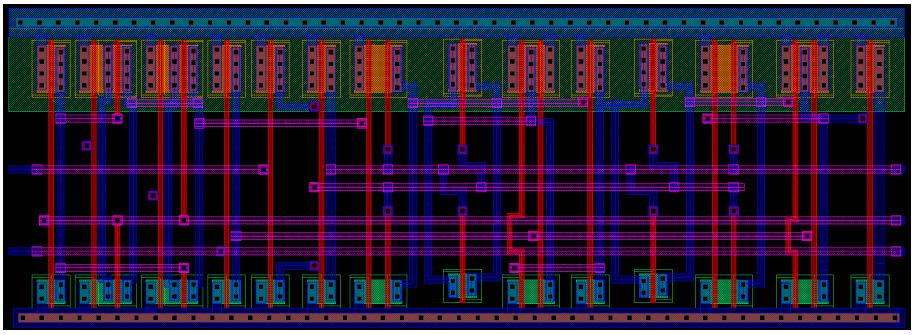

Figure 5: Scan FF layout based on our library.

5.2 Library Modifications

There are a few additions that need to be made to the Scan FF cell in the library before the .db file is produced. These will occur after the ′cad-alf2lib′ command when following the Encounter Library Characterization (ELC) process in [2].

A special test_cell() needs to be defined in the scan cell to let the tools know how to use it. It′s inserted between the initial ff() declaration and the first pin() declaration. It needs to contain all the cell pins and their corresponding direction and signal type. The direction indicates input/output/tri-state while the signal type is used for the test scan signals. Notice that the ff() declaration is also included with its next state and clock signals. The function designation refers to the generated IQ and IQN signal names for the output of the FF. Below is a snippet of the library showing the additional test_cell() information.

```
test_cell () {
        pin (D) {
        direction : input;
        }
        pin(CLK) {
        direction : input;
        }
        pin (SI) {
        direction : input;
        signal_type : test_scan_in;
        }
        pin (SE) {
        direction : input;
        signal_type : test_scan_enable;
        }
        ff (IQ,IQN) {
        next_state : "D";
        clocked_on : "CLK";
        }
        pin (Q) {
        direction : output;
        function : "IQ";
        signal_type : test_scan_out;
        }
        pin (QB) {
        direction : output;
        function : IQN;
        signal_type : test_scan_out_inverted;
        }
}
```

Another addition to the cell is a nextstate designation for pins associated with ′scan_in′ and ′scan_enable.′ See the format below:

```
...
pin(SI) {
nextstate_type : "scan_in";
direction : input;
...
pin(SE) {
nextstate_type : "scan_enable";
direction : input;
...
```

5.3 Behavioral Verilog Design Modifications

The Verilog design needs additional signals before it can be synthesized. These signals are only required for sequential circuits, where FFs reside. The Scan FF will be inserted during the compilation process to create a scan chain. Because the Scan FF contains additional signals that will need to be routed, the netlist needs inputs and outputs to allow ATPG to control the input sequence and to validate the output sequence. For this reason, the circuit needs the following pins created in the top module: test_mode, scan_input, scan_output, and scan_enable. Below is a snippet of the top module declaration.

```
module controller(input clk, reset,
                  input      [5:0] op,
                  input            zero,
                  input            si, se, test_mode,//scan in, scan enable, test_mode
                  output           so, //scan out
                  output reg       memread, memwrite, alusrca, memtoreg, iord,
                  output           pcen,
                  output reg       regwrite, regdst,
                  output reg [1:0] pcsource, alusrcb, aluop,
                  output reg [3:0] irwrite);

   parameter   FETCH1  =  4'b0001;
   parameter   FETCH2  =  4'b0010;
...
```

6 Verification Procedures

6.1 DC and DFT Setup

In our setup, the DC and DFT are run together using a single script. They could be run as two separate scripts, but problems might arise during the file export and import process from DC to DFT respectively. Therefore, we assumed that the design is already imported and ready for scan insertion. When reading the script, the breakpoint between the two compilers occurs at the "Insert Test Structures" comment. For the rest of the setup information, we will not go into each instruction in detail. This is because the "man" command can be used to view the manual entries of DC to find out functionality and usage of unclear commands. Instead we will cover a high level description of the design flow.

The DC script starts with updating the target libraries for both standard cells and the Scan FF. In our process, we differentiate between the standard cells library and the scan library but they do not necessarily need to be separated. The scan cell style is then set up. There are several designs to choose here including multiplexed FF, clocked scan, level-sensitive scan design (lssd), auxiliary clock lssd, combinational, or none. Based on our Scan FF design, only multiplexed scan flip-flop is used. We then set up the AutoFix feature for clock and reset signals. This allows the compiler to fix any test DRC errors at gate level logic introduced by the additional nets.

During the next process, we set up the test protocol, including all the parameters for clock, reset, and test_mode if they are used. Once the test protocol is created, we check the current design against the test design rules. Any major violation in this step could cause the flow to continue incorrectly and ultimately fail in the TetraMAX tool. If errors occur, the "man" command can be used with the error number in parentheses to find out more information about the violation.

Assuming errors have not occurred, we create the input signal delays. The delay allows the circuit to be fully reset before a certain input is applied. At this step, we can choose between Full Scan or Partial Scan. With Partial Scan, the commented command, "set_scan_element false {...}," blocks certain registers from being replaced with the Scan FF. Otherwise, Full Scan is assumed and the script will continue with the compilation process. This process replaces all sequential elements during optimization. This allows the design to be compiled in the future without the need of reoptimization.

Once the compilation process is completed, we can start on setting up the scan chain. This process is not needed for combinational designs because there will be no replacements. This section sets up what each signal does in the scan chain design. We also include a memory wrapper section, which is used specifically for designs with memory. Here the Scan FFs will be inserted before and after the memory block instead of replacing each individual FF cell. This allows the memory block to be tested as a single module. Next, we preview the design before the scan chain insertion process begins. We also update the design state to reflect the inserted scan chain. The test design rules are checked again to ensure no major errors occurred during the scan insertion process. Finally, we create all output reports, the new structural design with inserted scan chains, and the test protocol (STIL) file. A similar process can be done with Design Vision, while viewing the physical modifications in the schematic.

6.2 TetraMAX

The process for TetraMAX starts with importing the following: the complete netlist including scan inserted structural Verilog, the cell library (behavioral Verilog) and the scan cell library (behavioral Verilog). The design then builds and checks for synthesized violations of the testing rules. Once completed successfully, TetraMAX can generate test patterns and coverage reports. TetraMAX also allows faults to be analyzed and simulated individually. Although we did not experiment with all of the features available in TetraMAX due to conflicting version installations, our script will now work with the recently installed F-2011.09-SP4 version, assuming proper Scan FFs have been designed.

7 Results

The following results will include output information from verifying sequential or combinational Verilog designs. Due to an unresolved issue with our main Scan FF chain, our sequential design could not pass the TetraMAX DRC. Whether this was caused by tool installation issues or the scan chain itself, part of our Future Work will be to determine the root cause. We were, however, able to test combinational designs using TetraMAX and included a section of the generated test patterns. The scan cell library file (.lib) and the behavioral library file (.v) are included in Appendix A and B respectively.

The next comparison shows the size differences between our synthesized sequential controller using DC and the scan inserted version using DFT. These results show our best case Scan FF, which turned out the be the one built from the UofU_Digital_v1_2 library cells. Appendix C shows our script for running the DC and DFT Compiler.

From	Size (μm square)	Number of Nets (wires with same label)
DC:	384	90
DFT:	424	102

The next important piece of data is a high percentage of fault coverage. This means that if a fault occurs in the fabricated model, there$'$s a large chance the test patterns can pinpoint the exact area of failure. Below we outline the types of faults detected during the DFT Compiler process on our sequential controller.

Fault Type	# of faults
Detected	568
Undetectable	1
ATPG untestable	5
Not detected	0
total faults	574
test coverage	99.13%

The last snippet of output is from the TetraMAX process using our combinational adder circuit. Here we show 2 out of 15 test patterns generated by TetraMAX. This particular circuit has a total of 268 detected faults. These 15 patterns give us a test coverage of 100%, not including 14 undetectable faults for this particular circuit. Appendix D shows the script for TetraMax.

```
pattern = 0; // 200
#0 PI = 23'b11110100110000110001001;
#0;
XPCT = 8'b10110111;
MASK = 8'b11111111;
#0 ->measurePO_default_WFT;
#100; // 300

pattern = 1; // 300
#0 PI = 23'b01011100011111101010110;
#0;
XPCT = 8'b11011010;
MASK = 8'b11111111;
#0 ->measurePO_default_WFT;
#100; // 400
```

The behavioral Verilog code for the sequential controller and combinational adder can be found in Appendix E and F, respectively.

8 Future Work

The first part will be to determine what causes our scan chain to fail the TetraMAX DRC. The newer version of TetraMAX has been installed recently so we have not been able to retest our sequential designs. Another issue could be a discrepancy between the DFT Compiler and the setup files used by TetraMAX.

The next part would be getting the sequential ATPG to work so full test vectors can be generated for the controller module. We would also apply this tool flow to our entire processor design. Once completed, fabrication and actual testing of our scan enabled design is desired to validate the patterns generated by TetraMax for fault detection.

9 Conclusion

Based on what we have learned about the Synopsys testing and verification tools, along with the scripts we have developed, and the experience designing Scan FFs, we will be able to test our combinational and sequential Verilog designs. Currently we are able complete scan insertion and get a coverage analysis on our sequential designs using the DFT Compiler, although we might need slight modifications to the scan chain to complete the TetraMax process. As for combinational designs, there is no scan insertion process in the DFT Compiler, but we can still achieve an ATPG coverage and the test patterns from TetraMAX. The complete combinational output_file, called expAdd_tb_patterns.v, from TetraMAX is included in the source folder.

Only the main library files and scripts have been included in the Appendix. For a complete set of source files, refer to the zipped source folder included with this report. The files step0, step1, and step2 are used to quickly run the scripts.

step0 - sources the Synopsys environment setup file.
step1 - runs through the DC and DFT Compiler script.
step2 - runs through the TetraMAX script.

Note that modifications to the scripts are required depending on what type of Verilog design is being used.

References

[1] E. Jose, "Scan insertion and atpg using synopsys & full scan v/s partial scan analysis," 2006.

[2] *Digital VLSI Chip Design with Cadence and Synopsys CAD Tools.* Pearson Education Inc., 2010.

[3] *Synopsys DFT Compiler User Guide*.

[4] *Synopsys TetraMax User Guide*.

[5] Synopsys, *Synopsys DFT Compiler*.

[6] Synopsys, *Synopsys TetraMAX ATPG*.

[7] V. S. Ashok Kumar Suhag, "Delay testable enhanced scan flip-flop: Dft for high fault coverage," *2011 International Symposium on Electronic System Design*, 2011.

10 Appendix

A foo.lib

```
/*
delay model :                typ
check model :               typ
power model :              typ
capacitance model :      typ
other model :                typ
*/
library(foo) {

delay_model : table_lookup;
in_place_swap_mode : match_footprint;

/* unit attributes */
time_unit : "1ns";
voltage_unit : "1V";
current_unit : "1mA";
pulling_resistance_unit : "1kohm";
leakage_power_unit : "1nW";
capacitive_load_unit (1,pf);

power_supply () {
            default_power_rail : RAIL_VDD;
            power_rail( RAIL_GND, 0 );
            power_rail( RAIL_VDD, 5 );
}

/* Default attributes */
/* Fanout (in terms of capacitive load units) */
default_fanout_load : 0.3 ; default_max_fanout : 10.0 ;
/* Pin Capacitance */
default_inout_pin_cap : 0.00675 ;
default_input_pin_cap : 0.00675 ;
default_output_pin_cap : 0.0 ;
/* leakage power */
default_cell_leakage_power : 0.0 ;
default_leakage_power_density : 0.0 ;

slew_upper_threshold_pct_rise : 80;
slew_lower_threshold_pct_rise : 20;
slew_upper_threshold_pct_fall : 80;
slew_lower_threshold_pct_fall : 20;
input_threshold_pct_rise : 30;
input_threshold_pct_fall : 70;
output_threshold_pct_rise : 70;
output_threshold_pct_fall : 30;
nom_process : 1;
nom_voltage : 5;
nom_temperature : 25;
operating_conditions ( typical ) {
```

```
            process : 1;
            voltage : 5;
            temperature : 25;
            power_rail( RAIL_GND, 0 );
            power_rail( RAIL_VDD, 5 );
}
default_operating_conditions : typical;

lu_table_template(delay_template_5x5) {
            variable_1 : input_net_transition;
            variable_2 : total_output_net_capacitance;
            index_1 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0");
            index_2 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0");
}
power_lut_template(energy_template_5x5) {
            variable_1 : input_transition_time;
            variable_2 : total_output_net_capacitance;
            index_1 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0");
            index_2 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0");
}
lu_table_template(hold_template_5x3) {
            variable_1 : constrained_pin_transition;
            variable_2 : related_pin_transition;
            index_1 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0");
            index_2 ("1000.0, 1001.0, 1002.0");
}
power_lut_template(passive_energy_template_5x1) {
            variable_1 : input_transition_time;
            index_1 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0");
}
lu_table_template(setup_template_5x3) {
            variable_1 : constrained_pin_transition;
            variable_2 : related_pin_transition;
            index_1 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0");
            index_2 ("1000.0, 1001.0, 1002.0");
}
lu_table_template(width_template_5x1) {
            variable_1 : related_pin_transition;
            index_1 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0");
}

/* ---------------- *
* Design : SDFF *
* ---------------- */
cell (SDFF) {
area : 4500;
cell_leakage_power : 0.679284;
rail_connection( GND, RAIL_GND );
rail_connection( VDD, RAIL_VDD );
pin_opposite("Q", "QB");
ff (IQ,IQN) {
            next_state : "((SE SI) + (!SE D))";
            clocked_on : "(!(!CLK))";
}
```

```
test_cell () {
            pin (D) {
            direction : input;
            }
            pin(CLK) {
            direction : input;
            }
            pin (SI) {
            direction : input;
            signal_type : test_scan_in;
            }
            pin (SE) {
            direction : input;
            signal_type : test_scan_enable;
            }
  ff (IQ,IQN) {
            next_state : "D";
            clocked_on : "CLK";
  }
  pin (Q) {
            direction : output;
            function : "IQ";
            signal_type : test_scan_out;
  }
  pin (QB) {
            direction : output;
            function : IQN;
            signal_type : test_scan_out_inverted;
  }
}
pin(CLK)  {
            direction : input;
            input_signal_level : RAIL_VDD;
            capacitance : 0.0172919;
            rise_capacitance : 0.0172686;
            fall_capacitance : 0.0172919;
            rise_capacitance_range ( 0.0172252, 0.0172889) ;
            fall_capacitance_range ( 0.0172693, 0.0173121) ;
            clock : true;
            max_transition : 1.2;
            internal_power() {
            rise_power(passive_energy_template_5x1) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            values ("1.84638, 1.87525, 2.03345, 2.18017, 2.74568");
            }
            fall_power(passive_energy_template_5x1) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            values ("3.21314, 3.25617, 3.4244, 3.58177, 4.14905");
            }
            }
            timing() {
            related_pin : "CLK";
            timing_type : min_pulse_width;
            when : "!D&!SE&!SI";
```

```
            sdf_cond : "D_EQ_0_AN_SE_EQ_0_AN_SI_EQ_0 == 1'b1";
            rise_constraint(width_template_5x1) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            values ("0.763702, 0.828201, 0.970086, 1.07303, 1.38691");
            }
            fall_constraint(width_template_5x1) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            values ("0.892968, 0.967046, 1.11533, 1.22008, 1.53877");
            }
            }
}
pin(D)  {
            direction : input;
            input_signal_level : RAIL_VDD;
            capacitance : 0.0202517;
            rise_capacitance : 0.019647;
            fall_capacitance : 0.0202517;
            rise_capacitance_range ( 0.0196047, 0.0290485) ;
            fall_capacitance_range ( 0.020241, 0.0290314) ;
            max_transition : 1.2;
            internal_power() {
            rise_power(passive_energy_template_5x1) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            values ("1.85927, 1.88606, 2.02546, 2.14059, 2.5922");
            }
            fall_power(passive_energy_template_5x1) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            values ("2.58905, 2.62006, 2.77426, 2.90761, 3.38858");
            }
            }
            timing() {
            related_pin : "CLK";
            timing_type : hold_rising;
            when : "!SE";
            sdf_cond : "SE == 1'b0";
            rise_constraint(hold_template_5x3) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            index_2 ("0.06, 0.3, 0.6");
            values ( \
            "0.0, 0.0, 0.0", \
            "0.0, 0.0, 0.0", \
            "0.0, 0.0, 0.0", \
            "0.0, 0.0, 0.0", \
            "0.0, 0.0, 0.0");
            }
            fall_constraint(hold_template_5x3) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            index_2 ("0.06, 0.3, 0.6");
            values ( \
            "0.0, 0.0, 0.04875", \
            "0.0, 0.0, 0.0", \
            "0.0, 0.0, 0.0", \
            "0.0, 0.0, 0.0", \
            "0.0, 0.0, 0.0");
```

```
            }
            }
            timing() {
            related_pin : "CLK";
            timing_type : setup_rising;
            when : "!SE";
            sdf_cond : "SE == 1'b0";
            rise_constraint(setup_template_5x3) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            index_2 ("0.06, 0.3, 0.6");
            values ( \
            "0.31875, 0.21375, 0.0825", \
            "0.37125, 0.26625, 0.135", \
            "0.5325, 0.37125, 0.24", \
            "0.555, 0.45, 0.31875", \
            "0.8175, 0.65625, 0.525");
            }
            fall_constraint(setup_template_5x3) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            index_2 ("0.06, 0.3, 0.6");
            values ( \
            "0.65625, 0.495, 0.36375", \
            "0.70875, 0.60375, 0.41625", \
            "0.87, 0.765, 0.5775", \
            "1.005, 0.84375, 0.7125", \
            "1.38, 1.21875, 1.0875");
            }
            }
}
pin(Q)  {
            direction : output;
            output_signal_level : RAIL_VDD;
            capacitance : 0;
            rise_capacitance : 0;
            fall_capacitance : 0;
            rise_capacitance_range ( 0, 0) ;
            fall_capacitance_range ( 0, 0) ;
            max_capacitance : 0.362947;
            max_transition : 1.90162;
            function : "IQ";
            timing() {
            related_pin : "CLK";
            timing_sense : non_unate;
            timing_type : rising_edge;
            cell_rise(delay_template_5x5) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            index_2 ("0.025, 0.05, 0.1, 0.3, 0.6");
            values ( \
            "0.794642, 0.85843, 0.983411, 1.47749, 2.216", \
            "0.859753, 0.92356, 1.04869, 1.54274, 2.2814", \
            "0.999215, 1.06347, 1.18848, 1.68251, 2.42133", \
            "1.0991, 1.16287, 1.28787, 1.78199, 2.52075", \
            "1.40103, 1.46512, 1.58968, 2.08372, 2.8224");
            }
```

```
rise_transition(delay_template_5x5) {
index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
index_2 ("0.025, 0.05, 0.1, 0.3, 0.6");
values ( \
"0.212049, 0.27025, 0.388735, 0.873429, 1.60609", \
"0.212062, 0.270261, 0.388752, 0.87342, 1.60628", \
"0.212056, 0.270249, 0.388731, 0.873383, 1.60638", \
"0.211919, 0.270204, 0.388752, 0.873397, 1.60631", \
"0.212035, 0.270689, 0.388878, 0.873493, 1.60631");
}
cell_fall(delay_template_5x5) {
index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
index_2 ("0.025, 0.05, 0.1, 0.3, 0.6");
values ( \
"0.763702, 0.847688, 1.01205, 1.65849, 2.62194", \
"0.828201, 0.912709, 1.07703, 1.72356, 2.68686", \
"0.970086, 1.05416, 1.21854, 1.8646, 2.82858", \
"1.07303, 1.1568, 1.32098, 1.96665, 2.93059", \
"1.38691, 1.47055, 1.63436, 2.28052, 3.24412");
}
fall_transition(delay_template_5x5) {
index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
index_2 ("0.025, 0.05, 0.1, 0.3, 0.6");
values ( \
"0.239486, 0.30891, 0.449662, 1.02586, 1.90162", \
"0.239497, 0.308911, 0.449641, 1.02587, 1.90096", \
"0.239593, 0.308997, 0.44968, 1.026, 1.90111", \
"0.239997, 0.309377, 0.450007, 1.02608, 1.90162", \
"0.241425, 0.310253, 0.450402, 1.02617, 1.90118");
}
}
internal_power() {
related_pin : "CLK";
rise_power(energy_template_5x5) {
index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
index_2 ("0.025, 0.05, 0.1, 0.3, 0.6");
values ( \
"3.85901, 4.16808, 4.78988, 7.288, 11.0389", \
"3.88149, 4.19063, 4.81267, 7.31086, 11.0616", \
"4.02798, 4.33814, 4.96001, 7.45807, 11.2088", \
"4.1844, 4.4939, 5.11603, 7.61452, 11.3654", \
"4.74256, 5.05242, 5.66497, 8.1626, 11.9132");
}
fall_power(energy_template_5x5) {
index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
index_2 ("0.025, 0.05, 0.1, 0.3, 0.6");
values ( \
"3.15374, 2.84656, 2.2314, 0.247617, 3.98677", \
"3.17082, 2.87163, 2.25649, 0.222216, 3.96146", \
"3.36198, 3.05475, 2.43983, 0.038051, 3.77644", \
"3.56539, 3.25699, 2.64074, 0.161013, 3.57758", \
"4.28224, 3.97203, 3.35273, 0.872756, 2.86484");
}
}
```

```
}
pin(QB)   {
            direction : output;
            output_signal_level : RAIL_VDD;
            capacitance : 0;
            rise_capacitance : 0;
            fall_capacitance : 0;
            rise_capacitance_range ( 0, 0) ;
            fall_capacitance_range ( 0, 0) ;
            max_capacitance : 0.255931;
            max_transition : 2.23084;
            function : "IQN";
            timing() {
            related_pin : "CLK";
            timing_sense : non_unate;
            timing_type : rising_edge;
            cell_rise(delay_template_5x5) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            index_2 ("0.025, 0.05, 0.1, 0.3, 0.6");
            values ( \
            "0.578634, 0.665801, 0.837776, 1.5144, 2.522", \
            "0.643475, 0.730434, 0.902774, 1.57827, 2.58706", \
            "0.78463, 0.87052, 1.04096, 1.71503, 2.72262", \
            "0.887771, 0.972984, 1.14291, 1.81416, 2.82031", \
            "1.20173, 1.28504, 1.45249, 2.11743, 3.11921");
            }
            rise_transition(delay_template_5x5) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            index_2 ("0.025, 0.05, 0.1, 0.3, 0.6");
            values ( \
            "0.300933, 0.386874, 0.562848, 1.24025, 2.23053", \
            "0.301374, 0.386853, 0.562949, 1.24026, 2.23084", \
            "0.302073, 0.387592, 0.563185, 1.2403, 2.23055", \
            "0.305802, 0.38985, 0.564359, 1.24049, 2.23063", \
            "0.318665, 0.400004, 0.571868, 1.24257, 2.23075");
            }
            cell_fall(delay_template_5x5) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            index_2 ("0.025, 0.05, 0.1, 0.3, 0.6");
            values ( \
            "0.657902, 0.715835, 0.827302, 1.26823, 1.93043", \
            "0.723392, 0.780451, 0.892293, 1.33321, 1.99562", \
            "0.863344, 0.920551, 1.03245, 1.47331, 2.13539", \
            "0.96279, 1.02048, 1.1317, 1.5726, 2.23457", \
            "1.26559, 1.32337, 1.43455, 1.87538, 2.53717");
            }
            fall_transition(delay_template_5x5) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            index_2 ("0.025, 0.05, 0.1, 0.3, 0.6");
            values ( \
            "0.218249, 0.27038, 0.376187, 0.800053, 1.42646", \
            "0.218369, 0.269748, 0.376155, 0.800069, 1.42668", \
            "0.218317, 0.269859, 0.376014, 0.800105, 1.42646", \
            "0.218671, 0.270596, 0.376259, 0.80017, 1.42651", \
```

```
            "0.220108, 0.27147, 0.376756, 0.80025, 1.42647");
            }
            }
            internal_power() {
            related_pin : "CLK";
            rise_power(energy_template_5x5) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            index_2 ("0.025, 0.05, 0.1, 0.3, 0.6");
            values ( \
            "3.7891, 4.14936, 4.89649, 8.00794, 12.7455", \
            "3.81153, 4.16713, 4.91742, 8.03012, 12.763", \
            "3.99921, 4.35042, 5.08954, 8.18483, 12.9171", \
            "4.19431, 4.54045, 5.27395, 8.35511, 13.0813", \
            "4.90314, 5.22963, 5.94197, 8.97826, 13.6802");
            }
            fall_power(energy_template_5x5) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            index_2 ("0.025, 0.05, 0.1, 0.3, 0.6");
            values ( \
            "3.2416, 2.9453, 2.37277, 0.195615, 2.96504", \
            "3.26652, 2.97248, 2.39746, 0.222417, 2.94206", \
            "3.41397, 3.12053, 2.54626, 0.368261, 2.79364", \
            "3.57046, 3.2763, 2.7033, 0.523699, 2.63997", \
            "4.11927, 3.82483, 3.24799, 1.06815, 2.09762");
            }
            }
}
pin(SE)  {
nextstate_type : "scan_enable";
            direction : input;
            input_signal_level : RAIL_VDD;
            capacitance : 0.047487;
            rise_capacitance : 0.047487;
            fall_capacitance : 0.0471533;
            rise_capacitance_range ( 0.0463496, 0.0504435) ;
            fall_capacitance_range ( 0.0466062, 0.050495) ;
            max_transition : 1.2;
            internal_power() {
            rise_power(passive_energy_template_5x1) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            values ("2.3186, 2.34552, 2.506, 2.65134, 3.57613");
            }
            fall_power(passive_energy_template_5x1) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            values ("3.47275, 3.53407, 3.87065, 4.18244, 5.31637");
            }
            }
            timing() {
            related_pin : "CLK";
            timing_type : hold_rising;
            when : "!D&SI";
            sdf_cond : "D_EQ_0_AN_SI_EQ_1 == 1'b1";
            rise_constraint(hold_template_5x3) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
```

```
index_2 ("0.06, 0.3, 0.6");
values ( \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0");
}
fall_constraint(hold_template_5x3) {
index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
index_2 ("0.06, 0.3, 0.6");
values ( \
"0.0, 0.0, 0.04875", \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0");
}
}
timing() {
related_pin : "CLK";
timing_type : hold_rising;
when : "D&!SI";
sdf_cond : "D_EQ_1_AN_SI_EQ_0 == 1'b1";
rise_constraint(hold_template_5x3) {
index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
index_2 ("0.06, 0.3, 0.6");
values ( \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0");
}
fall_constraint(hold_template_5x3) {
index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
index_2 ("0.06, 0.3, 0.6");
values ( \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0");
}
}
timing() {
related_pin : "CLK";
timing_type : setup_rising;
when : "!D&SI";
sdf_cond : "D_EQ_0_AN_SI_EQ_1 == 1'b1";
rise_constraint(setup_template_5x3) {
index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
index_2 ("0.06, 0.3, 0.6");
values ( \
```

```
            "0.31875, 0.21375, 0.0825", \
            "0.37125, 0.26625, 0.135", \
            "0.47625, 0.37125, 0.18375", \
            "0.555, 0.39375, 0.2625", \
            "0.705, 0.6, 0.46875");
            }
            fall_constraint(setup_template_5x3) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            index_2 ("0.06, 0.3, 0.6");
            values ( \
            "0.65625, 0.55125, 0.36375", \
            "0.70875, 0.60375, 0.41625", \
            "0.87, 0.70875, 0.5775", \
            "0.94875, 0.84375, 0.65625", \
            "1.32375, 1.1625, 0.975");
            }
            }
            timing() {
            related_pin : "CLK";
            timing_type : setup_rising;
            when : "D&!SI";
            sdf_cond : "D_EQ_1_AN_SI_EQ_0 == 1'b1";
            rise_constraint(setup_template_5x3) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            index_2 ("0.06, 0.3, 0.6");
            values ( \
            "0.825, 0.66375, 0.47625", \
            "0.8775, 0.71625, 0.585", \
            "0.9825, 0.8775, 0.69", \
            "1.1175, 0.95625, 0.76875", \
            "1.38, 1.21875, 1.0875");
            }
            fall_constraint(setup_template_5x3) {
            index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
            index_2 ("0.06, 0.3, 0.6");
            values ( \
            "0.43125, 0.32625, 0.195", \
            "0.54, 0.37875, 0.2475", \
            "0.645, 0.54, 0.40875", \
            "0.72375, 0.61875, 0.4875", \
            "1.0425, 0.88125, 0.75");
            }
            }
}
pin(SI)  {
nextstate_type : "scan_in";
            direction : input;
            input_signal_level : RAIL_VDD;
            capacitance : 0.0208047;
            rise_capacitance : 0.0202066;
            fall_capacitance : 0.0208047;
            rise_capacitance_range ( 0.0201587, 0.0299656) ;
            fall_capacitance_range ( 0.0207946, 0.0299564) ;
            max_transition : 1.2;
```

```
internal_power() {
rise_power(passive_energy_template_5x1) {
index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
values ("1.85122, 1.87905, 2.01477, 2.12971, 2.57987");
}
fall_power(passive_energy_template_5x1) {
index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
values ("2.63597, 2.66673, 2.822, 2.95425, 3.43946");
}
}
timing() {
related_pin : "CLK";
timing_type : hold_rising;
when : "SE";
sdf_cond : "SE == 1'b1";
rise_constraint(hold_template_5x3) {
index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
index_2 ("0.06, 0.3, 0.6");
values ( \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0");
}
fall_constraint(hold_template_5x3) {
index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
index_2 ("0.06, 0.3, 0.6");
values ( \
"0.0, 0.0, 0.04875", \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0", \
"0.0, 0.0, 0.0");
}
}
timing() {
related_pin : "CLK";
timing_type : setup_rising;
when : "SE";
sdf_cond : "SE == 1'b1";
rise_constraint(setup_template_5x3) {
index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
index_2 ("0.06, 0.3, 0.6");
values ( \
"0.31875, 0.21375, 0.0825", \
"0.37125, 0.26625, 0.135", \
"0.5325, 0.37125, 0.24", \
"0.555, 0.45, 0.31875", \
"0.8175, 0.65625, 0.525");
}
fall_constraint(setup_template_5x3) {
index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
index_2 ("0.06, 0.3, 0.6");
```

```
            values ( \
            "0.65625, 0.495, 0.36375", \
            "0.70875, 0.60375, 0.41625", \
            "0.87, 0.765, 0.5775", \
            "1.005, 0.84375, 0.7125", \
            "1.38, 1.21875, 1.0875");
            }
            }
}
}

}
```

B foo.v

```
module SDFF(CLK,D,Q,QB,SE,SI);
  input    CLK,D,SE,SI;
  output    Q,QB;
  not(_n3,CLK);
  not(Q,QB);
  mux(_n2,SE,SI,D);
  mux(_n1,_n3,_n2,_n1);
  imux(QB,_n3,Q,_n1);
endmodule
```

C Script for DFT

```
# * Modified by Anh Luong & Andrzej Forys
# * FALL 2012
# *
# * Author: Erik Brunvand, University of Utah
# *
# * General synthesis script for Synopsys. There should be
# * some general switches and parameters set in .synopsys_dc.setup
# * but other design-specific things are set here.
# * You should look carefully at everything above the
# * "below here shouldn't need to be changed" line.
# *
# * Note that lists that continue across a line need a backslash
# * to continue to the next line (if you have a bunch of
# * different verilog files, one per line, for example, or a bunch
# * of target library files). Make SURE there isn't a space after
# * the \ because that can cause Synopsys to complain...
# *
# * Once you've modified things to your project, invoke with:
# *
# * syn-dc -f syn-script
# *
# *

#  This script assumes that the following variables are defined
#  in the .synopsys_dc.setup file. You should make sure that
#  your .synopsys_dc.setup file is configured for your
#  cell library! If you want to override or
```

```
#  add to those search paths, you can do that here...
#
#  SynopsysInstall = path to synopsys installation directory
#  synthetic_library = designware files
#  symbol_library = logic symbols for making schematics
#

# search path should include directories with memory .db files
# as well as the standard cells
# Your library path may be empty if your library will be in
# your synthesis directory because "." is already on the path.
#set search_path [list . \
#[format "%s%s"  $SynopsysInstall /libraries/syn] \
#[format "%s%s"  $SynopsysInstall /dw/sim_ver] \
#/uusoc/facility/cad_common/local/Cadence/lib/OA/UofU_Digital_v1_2]

# target library list should include all target .db files
set target_library [list UofU_Digital_v1_2.db]

# synthetic_library is set in .synopsys_dc.setup to be
# the dw_foundation library.
set link_library [concat  [concat  "*" $target_library] $synthetic_library]

###################################
# Print to screen options   #
###################################
set verbose 1     ;# 1 Write reports to screen, 0 do not write reports to screen
set verbose_dft 1 ;# 1 Write reports to screen, 0 do not write reports to screen

###################################
# Synthesis   #
###################################

# below are parameters that you will want to set for each design

# list of all HDL files in the design
set myFiles [list controller.v]
set fileFormat verilog    ;# verilog or VHDL
set basename controller   ;# Top-level module name
set myClk clk             ;# The name of your clock
set virtual 0             ;# 1 if virtual clock, 0 if real clock

# compiler switches...
#set optimizeArea 0      ;# 1 for area, 0 for speed
set useUltra 1          ;# 1 for compile_ultra, 0 for compile
                        # mapEffort, useUngroup are for
                        # non-ultra compile...
set mapEffort1 medium   ;# First pass - low, medium, or high
set mapEffort2 medium   ;# second pass - low, medium, or high
set useUngroup 1        ;# 0 if no flatten, 1 if flatten

# Timing and loading information
set myPeriod_ns 25       ;# desired clock period (sets speed goal)
set myClkLatency_ns 0.3 ; # clock network latency
```

```
set myInDelay_ns 0.25    ;# delay from clock to inputs valid
set myOutDelay_ns 0.25   ;# delay from clock to output valid
set myInputBuf INVX4     ;# name of cell driving the inputs
set myLoadLibrary UofU_Digital_v1_2 ;# name of library the cell comes from
set myLoadPin Y          ;# name of pin that outputs drive


# Control the writing of result files
set runname struct       ;# Name appended to output files

# the following control which output files you want. They
# should be set to 1 if you want the file, 0 if not
set write_v 1            ;# compiled structural Verilog file
set write_ddc 0          ;# compiled file in ddc format (XG-mode)
set write_sdf 1          ;# sdf file for back-annotated timing sim
set write_sdc 1          ;# sdc constraint file for place and route
set write_rep 0          ;# report file from compilation
set write_pow 0          ;# report file for power estimate

####################################
# DFT Switches   #
####################################
set dft_runname scan    ; # name appended to output files
set scan_library [list foo.db] ; # Library with scan chain cells
#set scancell SCANFF ; # Name of ScanFF Cell

# Setup timing variables for dft_drc command
set test_default_delay 0 ; # define time when values are applied to input ports
set test_default_bidir_delay 0 ; # Defines  the  default switching time of bidirectional p
set test_default_strobe 40 ; # default strobe time in a test cycle for output ports and bi
set test_default_period 100 ; # Defines the default length of a test vector cycle

# Setup scan chain for insert_dft
#set test_default_scan_style multiplexed_flip_flop; # Defines  the  default scan style for

#*****************************************************
#*   below here shouldn't need to be changed...      *
#*****************************************************

####################################
# remove any other designs from design compiler's memory
####################################
remove_design -all
# IMPORTING DESIGN

# analyze and elaborate the files
analyze -format $fileFormat -lib WORK $myFiles
elaborate $basename -lib WORK -update
current_design $basename

# The link command makes sure that all the required design
# parts are linked together.
# The uniquify command makes unique copies of replicated
# modules.
```

```
link
uniquify

# SETUP CONSTRAINTS

# now you can create clocks for the design
# and set other constraints
if {  $virtual == 0 } {
  create_clock -period $myPeriod_ns $myClk
} else {
  create_clock -period $myPeriod_ns -name $myClk
}
#*** add this shit
set_clock_latency $myClkLatency_ns $myClk

# Set the driving cell for all inputs except the clock
# The clock has infinte drive by default. This is usually
# what you want for synthesis because you will use other
# tools (like SOC Encounter) to build the clock tree
# (or define it by hand).
if {  $virtual == 0 } {
    set_driving_cell  -library $myLoadLibrary -lib_cell $myInputBuf [all_inputs] \
} else {
  set_driving_cell  -library $myLoadLibrary -lib_cell $myInputBuf  \
       [remove_from_collection [all_inputs] $myClk]
}

# set the input and output delay relative to myClk
if {  $virtual == 0 } {
    set_input_delay $myInDelay_ns -clock $myClk [all_inputs] \
} else {
    set_input_delay $myInDelay_ns -clock $myClk \
    [remove_from_collection [all_inputs] $myClk]
}
set_output_delay $myOutDelay_ns -clock $myClk [all_outputs]

# set the load of the circuit outputs in terms of the load
# of the next cell that they will drive, also try to fix
# hold time issues
set_load [load_of [format "%s%s%s%s%s" $myLoadLibrary "/" $myInputBuf "/" $myLoadPin]] [al
set_fix_hold $myClk

# FINISH SETUP CONSTRAINTS

# This command will fix the problem of having
# assign statements left in your structural file.
# But, it will insert pairs of inverters for feedthroughs!
set_fix_multiple_port_nets -all -buffer_constants

# COMPILING DESIGN

# now compile the design with given mapping effort
# and do a second compile with incremental mapping
# or use the compile_ultra meta-command
```

```
if {  $useUltra == 1 } {
  compile_ultra
} else {
  if {  $useUngroup == 1 } {
    compile -ungoup_all -map_effort $mapEffort1
    compile -incremental_mapping -map_effort $mapEffort2
 } else {
    compile -map_effort $mapEffort1
    compile -incremental_mapping -map_effort $mapEffort2
 }
}


#
check_design
# VIOLATIONS
report_constraint -all_violators

#*********************************************************
#* now write out the results                             *
#*********************************************************

set filebase [format "%s%s" [format "%s%s" $basename "_"] $runname]

# structural (synthesized) file as verilog
if {  $write_v == 1 } {
    set filename [format "%s%s%s" ./src/ $filebase ".v"]
    redirect change_names \
   { change_names -rules verilog -hierarchy -verbose }
    write -format verilog -hierarchy -output $filename
}

# write out the sdf file for back-annotated verilog sim
# This file can be large!
if {  $write_sdf == 1 } {
    set filename [format "%s%s%s" ./src/ $filebase ".sdf"]
    write_sdf -version 1.0 $filename
}

# this is the timing constraints file generated from the
# conditions above - used in the place and route program
if {  $write_sdc == 1 } {
    set filename [format "%s%s%s" ./src/ $filebase ".sdc"]
    write_sdc $filename
}

# synopsys database format in case you want to read this
# synthesized result back in to synopsys later in XG mode (ddc format)
if {  $write_ddc == 1 } {
    set filename [format "%s%s%s" ./src/ $filebase ".ddc"]
    write -format ddc -hierarchy -o $filename
}

# report on the results from synthesis
```

```
# note that > makes a new file and >> appends to a file
if {  $write_rep == 1 } {
    set filename [format "%s%s%s" ./src/ $filebase ".rep"]
    redirect $filename { report_timing }
    redirect -append $filename { report_area }
}

# report the power estimate from synthesis.
if {  $write_pow == 1 } {
    set filename [format "%s%s%s" ./src/ $filebase ".pow"]
    redirect $filename { report_power }
}

# print report to the screen
if { $verbose == 1 } {
    report_design
    report_hierarchy
    report_timing -path full -delay max -nworst 3 -significant_digits 2 -sort_by group
    report_timing -path full -delay min -nworst 3 -significant_digits 2 -sort_by group
    report_area
    report_cell
    report_net
    report_port -v
    report_power -analysis_effort low
}

# Design reports
set filename [format "%s%s%s" ./reports/ $filebase ".design"]
redirect $filename { report_design }

# Hierarchy reports
set filename [format "%s%s%s" ./reports/ $filebase ".design"]
redirect -append $filename { report_hierarchy }

# Timing reports
set filename [format "%s%s%s" ./reports/ $filebase ".timing"]
redirect $filename { report_timing -path full -delay max -nworst 5 -significant_digits 2 -

set filename [format "%s%s%s" ./reports/ $filebase ".timing"]
redirect -append $filename { report_timing -path full -delay min -nworst 5 -significant_di

# Report_cell
set filename [format "%s%s%s" ./reports/ $filebase ".area"]
redirect $filename { report_area }

# Report_area
set filename [format "%s%s%s" ./reports/ $filebase ".area"]
redirect -append $filename { report_cell }

# Report port
set filename [format "%s%s%s" ./reports/ $filebase ".ports"]
redirect $filename { report_port -v}

# Report net
```

```
set filename [format "%s%s%s" ./reports/ $filebase ".net"]
redirect $filename { report_net }

# Report power
set filename [format "%s%s%s" ./reports/ $filebase ".pow"]
redirect $filename { report_power -analysis_effort low }

########################################
#### Insert Test Structures ###
########################################
# Update filebase
set filebase [format "%s%s" [format "%s%s" $basename "_"] $dft_runname]

# Update target library
set target_library [list $target_library $scan_library]

# Set Scan Chain Type
set_scan_configuration -style multiplexed_flip_flop

# AutoFix for Reset and Clock
set_dft_configuration -fix_reset enable -fix_clock enable

# Set Test Protocol
set test_default_period 100
set_dft_signal -view existing_dft -type ScanClock -timing {45 55} -port clk
set_dft_signal -view existing_dft -type Reset -active_state 1 -port reset
set_dft_signal -view existing_dft -type Constant -active_state 1 -port test_mode
create_test_protocol

# DFT Check
dft_drc -verbose

# Add delay in generated clocks
create_clock clk -period 1000
set_input_delay 250 si -clock clk
set_input_delay 150 se -clock clk

# Partial Scan
#set_scan_element false {state_reg_3_}
#set_scan_element false {state_reg_2_}
#set_scan_element false {state_reg_1_}
#set_scan_element false {state_reg_0_}

# Test-Ready Synthesis
compile -scan

# Read Design & Test Protocol
# Write out the test protocol and scan-ready design
#write_test_protocol -output [format "%s%s%s" ./reports/ $filebase ".spf"]
#write -format ddc -hierarchy -output [format "%s%s%s" ./reports/ $filebase ".ddc"]

# Read design and test protocol
#read_file -format ddc [format "%s%s%s" ./reports/ $filebase ".ddc"]
#current_design [format "%s%s%s" ./reports/ $filebase ".ddc"]:$basename
```

```
#link
#read_test_protocol [format "%s%s%s" ./reports/ $filebase ".spf"]

# Specify Scan Chain
set_scan_configuration -chain_count 1
set_scan_configuration -clock_mixing no_mix
set_dft_signal -view spec -type ScanDataIn -port si
set_dft_signal -view spec -type ScanDataOut -port so
set_dft_signal -view spec -type ScanEnable -port se -active_state 1
set_scan_path chain1 -scan_data_in si -scan_data_out so

# Memory Wrapper
#set_test_point_element -type observe [get_object_name [get_pins RAM_64B/D*]] -clock_signa
#set_test_point_element -type observe [get_object_name [get_pins RAM_64B/A*]] -clock_signa
#set_test_point_element -type control_01 [get_object_name [get_pins RAM_64B/Q*]] -clock_sig
#report_test_point_element

# Scan Preview
preview_dft -show all
preview_dft -test_points all

# Scan Chain Synthesis
insert_dft

# Scan Chain Identification
set_scan_state scan_existing

# DRC & Coverage
dft_drc -coverage_estimate

# Report Scan Information
report_scan_path -view existing_dft -chain all
report_scan_path -view existing_dft -cell all

# Prepare TetraMax script
change_names -hierarchy -rule verilog
write -format verilog -hierarchy -out [format "%s%s%s" ./src/ $filebase ".v"]
write -format ddc -hierarchy -output [format "%s%s%s" ./reports/ $filebase ".ddc"]
write_scan_def -output [format "%s%s%s" ./reports/ $filebase ".def"]
set test_stil_netlist_format verilog
write_test_protocol -output [format "%s%s%s" ./reports/ $filebase ".spf"]

####################################
# Quit dc
####################################
quit
```

D Script for TetraMax

```
#########################################################################
####
#### Modified by Anh Luong & Andrzej Forys
#### University of Utah
#### Fall 2012
```

```
####
#### TetraMax Script for ECE 128
#### Performs ATPG Pattern Generation for Synopsys Generic files
#### author: tjf
#### update: wgibb, spring 2010

#### note: this script will only run in TMAX TCL mode
#### start tmax like this:   tmax -tcl
#### source tmax_atpg.tcl
##########################################################################

############################################################
#### local variables, designer must change these values ####
############################################################
set top_module controller
set synthesized_files [list ./src/${top_module}_scan.v]
set cell_lib ./UofU_Digital_v1_2.v
set scan_lib ./foo.v
set stil_file [list ./reports/${top_module}_scan.spf]

# Continue execution when command returns an error
#set_command noabort

build -force

################################################
#### read in standard cells and user's design ###
################################################
# remove any other designs from design compiler's memory
read_netlist -delete
# read in standard cell library
read_netlist $cell_lib -library
# read in scan cell library
read_netlist $scan_lib -library
# read in user's synthesized verilog code
read_netlist $synthesized_files

################################################
#### BUILD and DRC test model
################################################
report_modules -all
run_build_model $top_module
# ignoring warnings like N20 or B10
# Set STIL file from DFT Compiler
set_drc $stil_file
# run check to see if synthesized code violates any testing rules
run_drc

################################################
#### Generate ATPG (patterns)- full sequential
################################################
# capture all faults, 9 capture cycles
#set_atpg -capture_cycles 9 -full_seq_atpg
#remove_faults -all
```

```
report_summaries faults patterns
add_faults -all
# run atpg in full sequential mode for better fault coverage
report_summaries faults patterns
run_atpg full_sequential_only
# write out patterns (overwrite old files)
report_summaries faults patterns
write_patterns ./src/${top_module}_tb_patterns.v -replace -internal -format verilog_single

################################################
#### Output reports
################################################
report_patterns -all >> ./reports/${top_module}.tmax.patterns
report_violations -all >> ./reports/${top_module}.tmax.violations
report_faults -summary -collapsed >> ./reports/${top_module}.tmax.coverage

################################################
#### Analyze Faults
################################################
# up to user to run these commands, they can inspect the faults and various reasons for the
#analyze_faults -class an
#analyze_faults -class an -verbose -max 3
#analyze_faults in_a_reg_reg/p_dregscan0/q -stuck 1

# Exit the program
quit
```

E controller.v

```
module main(input clk, reset,
                input      [5:0] op//,
                //input            zero,
                //output reg       memread, memwrite, alusrca, memtoreg, iord,
                //output           pcen,
                //output reg       regwrite, regdst,
                //output reg [1:0] pcsource, alusrcb, aluop,
                );//output reg [3:0] irwrite);

  parameter   FETCH1  =  4'b0001;
  parameter   FETCH2  =  4'b0010;
  parameter   FETCH3  =  4'b0011;
  parameter   FETCH4  =  4'b0100;
  parameter   DECODE  =  4'b0101;
  parameter   MEMADR  =  4'b0110;
  parameter   LBRD    =  4'b0111;
  parameter   LBWR    =  4'b1000;
  parameter   SBWR    =  4'b1001;
  parameter   RTYPEEX =  4'b1010;
  parameter   RTYPEWR =  4'b1011;
  parameter   BEQEX   =  4'b1100;
  parameter   JEX     =  4'b1101;
  parameter   ADDIWR  =  4'b1110; // added for ADDI

  parameter   LB      =  6'b100000;
```

```
parameter   SB      =  6'b101000;
parameter   RTYPE   =  6'b0;
parameter   BEQ     =  6'b000100;
parameter   J       =  6'b000010;
parameter   ADDI    =  6'b001000; /// added for ADDI

reg [3:0] state, nextstate;
//reg       pcwrite, pcwritecond;

// state register
always @(posedge clk)
   if(reset) state <= FETCH1;
   else state <= nextstate;

// next state logic
always @(*)
   begin
      case(state)
         FETCH1:  nextstate <= FETCH2;
         FETCH2:  nextstate <= FETCH3;
         FETCH3:  nextstate <= FETCH4;
         FETCH4:  nextstate <= DECODE;
         DECODE:  case(op)
                     LB:      nextstate <= MEMADR;
                     SB:      nextstate <= MEMADR;
                     ADDI:    nextstate <= MEMADR;  // added for ADDI
                     RTYPE:   nextstate <= RTYPEEX;
                     BEQ:     nextstate <= BEQEX;
                     J:       nextstate <= JEX;
                     default: nextstate <= FETCH1; // should never happen
                  endcase
         MEMADR:  case(op)
                     LB:      nextstate <= LBRD;
                     SB:      nextstate <= SBWR;
                   ADDI:    nextstate <= ADDIWR;  // added for ADDDI
                     default: nextstate <= FETCH1; // should never happen
                  endcase
         LBRD:    nextstate <= LBWR;
         LBWR:    nextstate <= FETCH1;
         SBWR:    nextstate <= FETCH1;
         RTYPEEX: nextstate <= RTYPEWR;
         RTYPEWR: nextstate <= FETCH1;
         BEQEX:   nextstate <= FETCH1;
         JEX:     nextstate <= FETCH1;
       ADDIWR:  nextstate <= FETCH1; // added for ADDI
         default: nextstate <= FETCH1; // should never happen
      endcase
   end

always @(*)
   begin
         // set all outputs to zero, then conditionally assert just the appropriate ones
         irwrite <= 4'b0000;
         pcwrite <= 0; pcwritecond <= 0;
```

```
regwrite <= 0; regdst <= 0;
memread <= 0; memwrite <= 0;
alusrca <= 0; alusrcb <= 2'b00; aluop <= 2'b00;
pcsource <= 2'b00;
iord <= 0; memtoreg <= 0;
case(state)
   FETCH1:
      begin
         memread <= 1;
         irwrite <= 4'b0001; // change to reflect new memory
         alusrcb <= 2'b01;   // get the IR bits in the right spots
         pcwrite <= 1;       // FETCH 2,3,4 also changed...
      end
   FETCH2:
      begin
         memread <= 1;
         irwrite <= 4'b0010;
         alusrcb <= 2'b01;
         pcwrite <= 1;
      end
   FETCH3:
      begin
         memread <= 1;
         irwrite <= 4'b0100;
         alusrcb <= 2'b01;
         pcwrite <= 1;
      end
   FETCH4:
      begin
         memread <= 1;
         irwrite <= 4'b1000;
         alusrcb <= 2'b01;
         pcwrite <= 1;
      end
   DECODE: alusrcb <= 2'b11;
   MEMADR:
      begin
         alusrca <= 1;
         alusrcb <= 2'b10;
      end
   LBRD:
      begin
         memread <= 1;
         iord    <= 1;
      end
   LBWR:
      begin
         regwrite <= 1;
         memtoreg <= 1;
      end
   SBWR:
      begin
         memwrite <= 1;
         iord     <= 1;
```

```
                end
            RTYPEEX:
                begin
                    alusrca <= 1;
                    aluop   <= 2'b10;
                end
            RTYPEWR:
                begin
                    regdst   <= 1;
                    regwrite <= 1;
                end
            BEQEX:
                begin
                    alusrca     <= 1;
                    aluop       <= 2'b01;
                    pcwritecond <= 1;
                    pcsource    <= 2'b01;
                end
            JEX:
                begin
                    pcwrite  <= 1;
                    pcsource <= 2'b10;
                end
         ADDIWR:  // new state for addi writeback
        begin
          regwrite <= 1;
                end
        endcase
    end
  assign pcen = pcwrite | (pcwritecond & zero); // program counter enable

endmodule
```

F expAdd.v

```
module expAdd(Ex, Ey, Ez, B);
input [7:0] Ex, Ey;
input [6:0] B;
output [7:0] Ez;

assign Ez = Ex + Ey - B;

endmodule
```