University of California, Davis
Department of Electrical and Computer Engineering

EEC180B DIGITAL SYSTEMS II Spring 2006

LAB 6: Scan Testing

Objective: In this lab, you will design and implement a state machine in VHDL. You will include additional circuitry in the design so that you can use *scan path testing* to verify your state machine. You will simulate the scan path testing in ModelSim and synthesize your design so that you can see the extra resources required for the scan logic. However, you will not download this design to the Xilinx board.

Pre-lab (20 points)

Based on the state diagram given in Figure 1, with the six state names appearing in all caps, derive the next-state equations and the output equations for the finite state machine. You must use either one-hot state encoding, where each state in encoded by a single 'set' flip-flop with all other state flip-flops reset, or "almost one-hot" state encoding, where the reset state (START) is specified as all state flip-flops reset and every other state has just a single set flip-flop. Specify your state assignments for the six states.

I. Specifications

The state diagram given in Figure 1 is based on the UART Controller from some old lab. However, only some of the output signals have been included in the state diagram. The following discussion gives you some idea of what the state machine's signals are in the UART context and is included for completeness. There are four inputs: reset, TDRE, RDRF, and X. The reset signal should cause a *synchronous* reset of the system, meaning that the state machine will transition to the START state on the rising edge of the system clock or the test clock whenever the reset signal is low. The TDRE and RDRF signals represent status signals from a UART transmitter and receiver, respectively. The X signal represents the output of an 8-bit comparator, which compares a character in the TDR (transmit data register) with the carriage-return character. In this design, you will directly control each of these input signals using a simple VHDL test bench.

The outputs that are shown in the state bubbles are Moore-type, indicating that the output signal depends on that particular state independent of any input. Mealy outputs are associated with a state and an input condition and are shown next to the state transition arrows corresponding to a certain state and input combination. Note that a single signal, such as clr_addr or addr_sel, can be represented as a Moore output for one state and a Mealy output for another state. (Overall, these signals are classified as Mealy outputs, since they have some dependence on one or more input signals.)

You must implement the state machine using "one-hot" or "almost one-hot" state encoding, as mentioned in the pre-lab. Your design can use any style of VHDL coding, as long as the design implements the state diagram shown in Figure 1 and the state encoding is implemented as either "one-hot" or "almost one-hot."

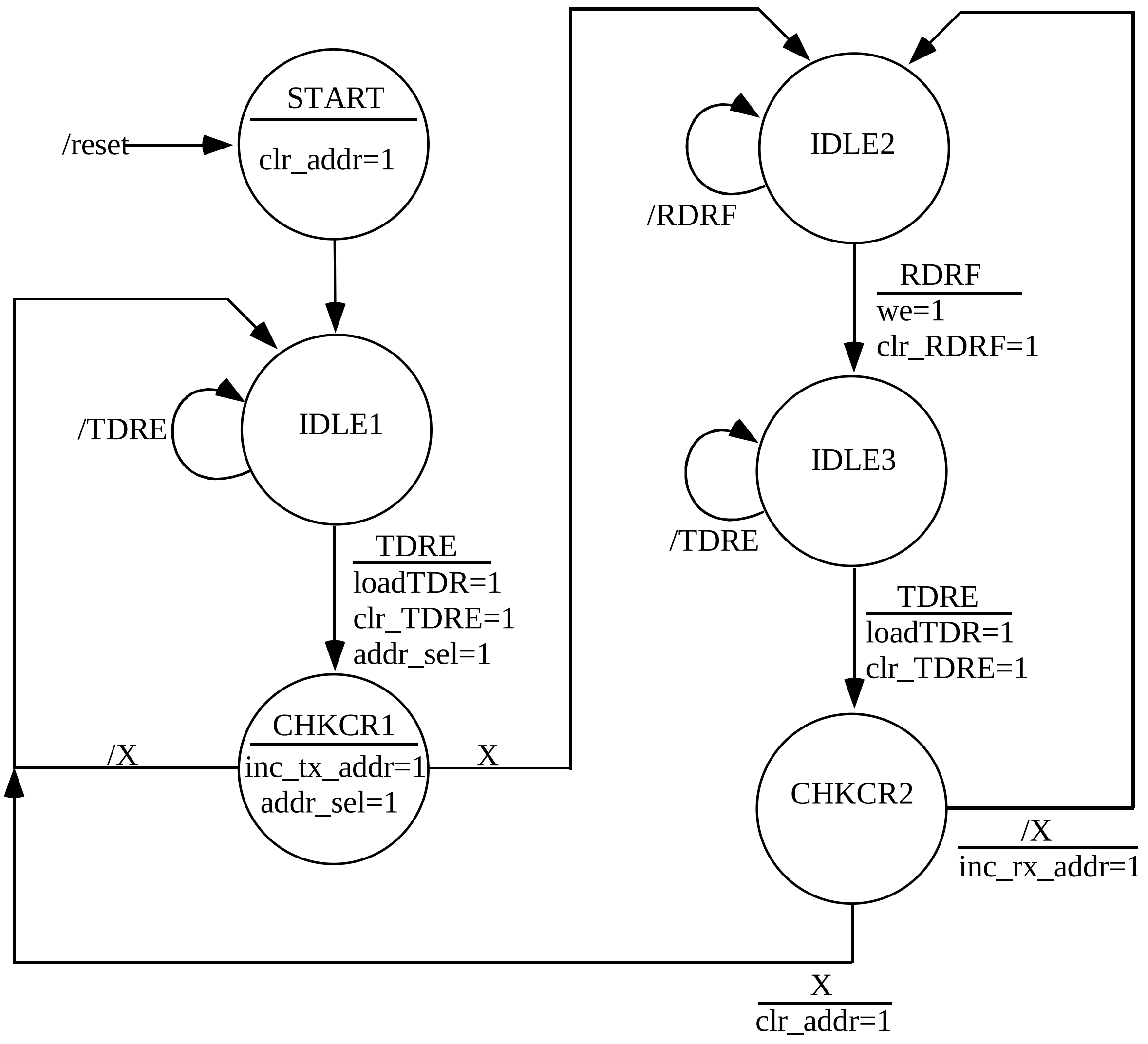

Figure 1. State Diagram of Finite State Machine

You must also implement a scan chain so that a state vector can be clocked into or out of the flip-flops by pulsing a test clock the proper number of times. Figure 2 shows an example of a simple scan chain with three flip-flops. In Figure 2, TCK denotes the test clock input while SCK means the system clock input.

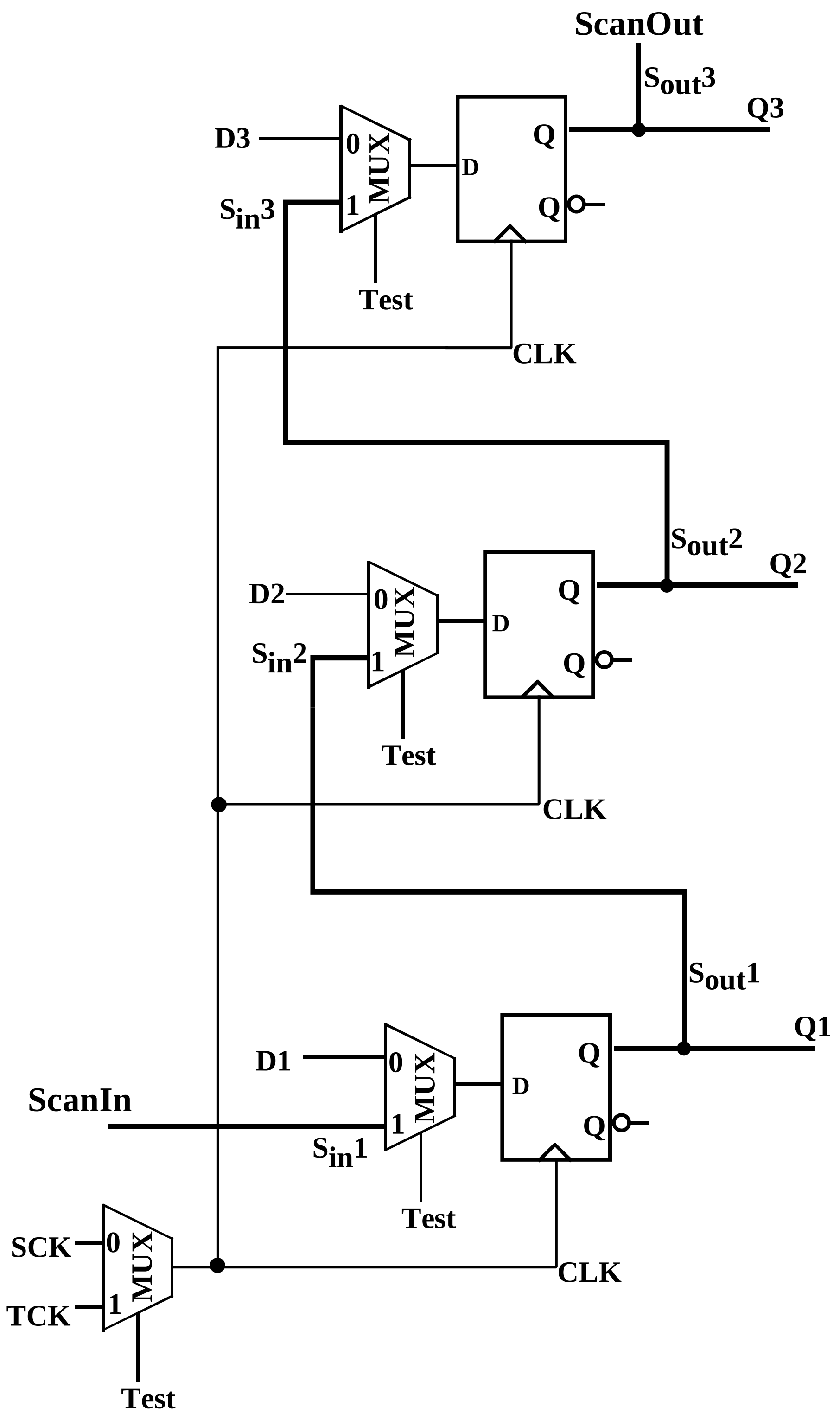

Figure 2. Simple Scan Path for Three Flip-Flops

<u>Lab Requirements</u> (55 points)

1. Model the state machine and a scan path using VHDL.

2. Write a test bench to do the following:
    a. Using the test clock, scan in a state vector to put the state machine in state IDLE1.
    b. Apply the input TDRE=1.
    c. Apply a clock pulse to the system clock to clock the state machine to its next state.
    d. Verify the system outputs.
    e. Scan out the state vector by pulsing the test clock the required number of times. Verify that the correct next state was reached given the initial state and input combination.

3. Repeat the procedure in step 2 to verify the next-state and outputs for the IDLE2 state with RDRF=1. An example test bench is included at the end of this lab write-up.

4. Simulate the two test bench cases in ModelSim and print the resulting waveforms. Analyze the simulation to verify the correct operation of your state machine and scan path circuitry.

5. Demonstrate your simulations to your TA and have him sign a verification sheet.

6. Synthesize your state machine with scan path design and print the gate-level schematic from Synplicity's Amplify tool. (Do not synthesize the test bench.) Your design **must** be synthesizable.

<u>Lab Report</u> (25 points)

Submit the following items in your lab report:

- Next-state and output equations.
- Verification sheet signed by your TA verifying simulations.
- VHDL source code for state machine and for test benches.
- Simulation waveforms for the two test bench cases.
- Gate-level schematic of your design printed from the Synplicity's Amplify.
- Answers to the following questions:

    1. When can the state machine outputs be verified during scan path testing? Can the outputs be checked while the state vector is being scanned in or scanned out? Explain.

    2. Based on the gate-level schematic of your design, verify that the input to <u>one</u> of the flip-flops corresponds to the correct next-state equation and also verify the scan path selection at the input to at least one flip-flop. Use Boolean algebra to manipulate the equations derived from the gate-level schematic into the equations you derived for your pre-lab.

/* Example test bench file for lab 6 */

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity tb2 is
end tb2;

architecture behave of tb2 is

component lab6
   port (reset, sclk, tclk, test, scan_in, TDRE, RDRF, X : in std_logic;
       clr_addr, loadTDR, clrTDRE, we, clrRDRF  : out std_logic;
       inc_tx_addr, inc_rx_addr, addr_sel, scan_out : out std_logic);
end component;

signal reset, sclk, tclk, test, scan_in, TDRE, RDRF, X : std_logic;
signal clr_addr, loadTDR, clrTDRE, we, clrRDRF  :  std_logic;
signal inc_tx_addr, inc_rx_addr, addr_sel, scan_out : std_logic;

begin

UUT : lab6
   port map (reset, sclk, tclk, test, scan_in, TDRE, RDRF, X,
           clr_addr, loadTDR, clrTDRE, we, clrRDRF,
           inc_tx_addr, inc_rx_addr, addr_sel, scan_out);

process
begin
  reset <= '0';         -- active-low reset
  sclk <= '0';          -- default
  test <= '1';          -- test mode
  scan_in <= '0';       -- default
  TDRE <= '1';          -- default
  RDRF <= '0';          -- default
  X <= '0';             -- default

  tclk <= '0';
  wait for 20 ns;
  tclk <= '1';          -- synchronous reset
  wait for 20 ns;

  reset <= '1';         -- de-assert reset
  scan_in <= '1';       -- shift in '1'

  tclk <= '0';
  wait for 20 ns;
  tclk <= '1';          -- Q0 <- '1'
  wait for 20 ns;

  scan_in <= '0';

  tclk <= '0';
  wait for 20 ns;
  tclk <= '1';          -- Q1 <- '1'
  wait for 20 ns;

  tclk <= '0';
  wait for 20 ns;
  tclk <= '1';          -- Q2 <- '1'
  wait for 20 ns;

  tclk <= '0';
  wait for 20 ns;
  tclk <= '1';          -- Q3 <- '1'
  wait for 20 ns;

  tclk <= '0';
  wait for 20 ns;
  tclk <= '1';          -- Q4 <- '1'
  wait for 20 ns;

  X <= '1';
  test <= '0';          -- normal system mode
  tclk <= '0';

  sclk <= '0';
  wait for 20 ns;
  sclk <= '1';
  wait for 20 ns;

  test <= '1';          -- back into test mode
  X <= '0';
  sclk <= '0';
  tclk <= '0';
  wait for 20 ns;
  tclk <= '1';          -- scan out the state vector from Q4 to Q0
  wait for 20 ns;

  tclk <= '0';
  wait for 20 ns;
  tclk <= '1';          -- update scan_out
  wait for 20 ns;

  tclk <= '0';
  wait for 20 ns;
  tclk <= '1';          -- update scan_out
  wait for 20 ns;

  tclk <= '0';
  wait for 20 ns;
  tclk <= '1';          -- update scan_out
  wait for 20 ns;

  tclk <= '0';
  wait for 20 ns;
  tclk <= '1';          -- extra clock cycle
  wait for 20 ns;
  wait;                 -- hang process here
end process;
end behave;

configuration tb2_cfg of tb2 is
for behave
end for;
end tb2_cfg;
```