Queen's University
Department of Electrical and Computer Engineering
ELEC 271 Digital Systems
Fall 2018

Lab 6:
Finite-State Machines

Objectives

This laboratory activity for *ELEC 271 Digital Systems* provides the opportunity for students to:

- acquire experience in applying the methodical design procedure for finite-state machines,
- confirm the equivalence of gate-level and high-level VHDL descriptions for state machines,
- use high-level VHDL syntax for a more complex state-machine description, and
- implement a complete system incorporating a state-machine controller and a datapath that consists of the example from Lab 5 that uses the hex (seven-segment) displays.

This activity involves the use of the worksheet included in the description. Each student must have a paper copy of the worksheet ready at the beginning of the laboratory session, and each student must complete it (with reasonable neatness) during the session, *otherwise no credit will be granted.*

Preparation BEFORE Your Scheduled Laboratory Session

- *You will individually prepare VHDL files and have them ready on your network storage or USB drive at the start of your scheduled laboratory session for use during the in-lab procedure. Each student will also have a separate worksheet ready at the start of the laboratory session.*

Preparation for Part 1

- In a file entitled `lab6.vhd`, prepare a template file as shown below.

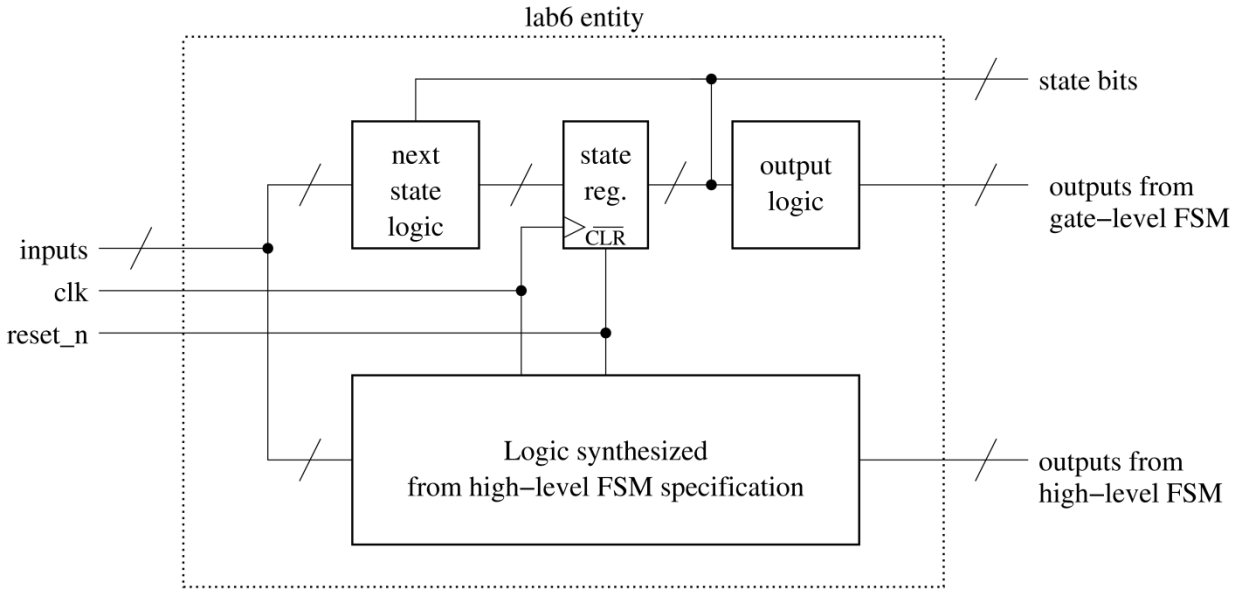
```
library ieee;
use ieee.std_logic_1164.all;

entity lab6 is
  port (
        clk, reset_n : in  std_logic;
        ..., ...     : in  std_logic;   -- primary inputs to both
        ..., ...     : out std_logic;   -- gate-level fsm primary outputs
        ..., ...     : out std_logic;   -- gate-level fsm state-bit outputs
        ..., ...     : out std_logic    -- high-level fsm primary outputs
       );
end entity;

architecture combined of lab6 is

-- for gate-level specification:

-- define internal signals q0, q1, ... for individual state flip-flop outputs
signal ... : ...;

-- define internal signals d0, d1, ... for individual state flip-flop inputs
signal ... : ...;

-- for high-level specification:

-- define state type and signal for single-process high-level specification
type ... is (...);
signal ... : ...;

begin

-- for gate-level specification:

-- define combined process for behavior of individual state flip-flops
the_state_dffs: process (clk, reset_n)
begin
  .
  .
  .
end process;

-- signal assignment statements for next-state d0, d1, ... functions
... <= ...;

-- signal assignment statements for gate-level fsm primary output functions
... <= ...;

-- signal assignment statements for gate-level fsm state-bit outputs
... <= ...;
```

(continued on next page)

- The continuation of `lab6.vhd` is below. The contents of the two pages should be in one file.

```
-- for high-level specification:

-- define the process that implements the state transitions on clock edges
the_fsm : process (clk, reset_n)
begin
   .
   .
   .
end process;

-- use signal assignment statements with when...else... syntax
-- to specify the behavior of high-level fsm outputs
... <= ...;

end architecture;
```

- The system diagram shown below reflects the intended implementation for the VHDL description.



You will be given the specifications for a Moore-type finite-state machine in the form of a detailed state diagram with input conditions for state transitions and output settings for each state.

The number of flip-flops required for the state machine will be two, for the sake of simplicity.

You will implement a process to describe the behavior of the state-register flip-flops.

You will derive optimized logic expressions for the next-state logic and output logic, and then you will implement gate-level signal assignment statements for those expressions.

Finally, you will use high-level VHDL syntax to describe behavior for an equivalent state machine, using an enumerated state type, a process for state transitions, and output signal assignments.

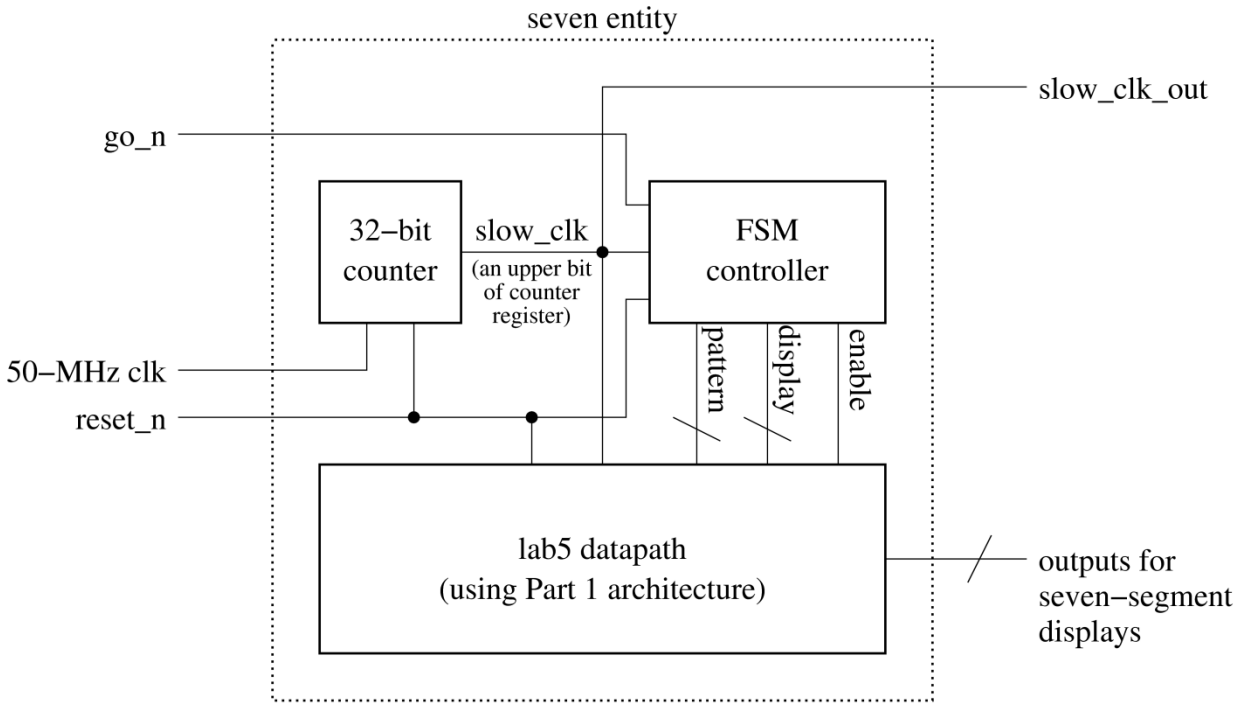
Use the suffix `_gl` for the **g**ate-**l**evel primary outputs, and `_hl` for the **h**igh-**l**evel primary outputs.

The state bit output ports from the gate-level version can be called `q1_out` and `q0_out`.

Because *both* versions of the state machine implementation have the *same* primary inputs, the primary outputs across multiple cycles should be identical, if the systems are specified correctly.

Preparation for Part 2

- For this part, the system diagram below reflects the intended implementation for the VHDL code.



- In a file entitled `my_components.vhd`, prepare a complete description using the template below.

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

package my_components is    -- declares custom components found in other files

component lab5
  port (
        -- fill in the port definitions
        -- as in the entity VHDL file
       );
end component;

component fsm
  port (
        -- fill in the port definitions
        -- as in the entity VHDL file
       );
end component;

end package;
```

The details for the port definitions should match the definitions within each entity VHDL file.

- In a file entitled `fsm.vhd`, prepare a complete description based on the template below.

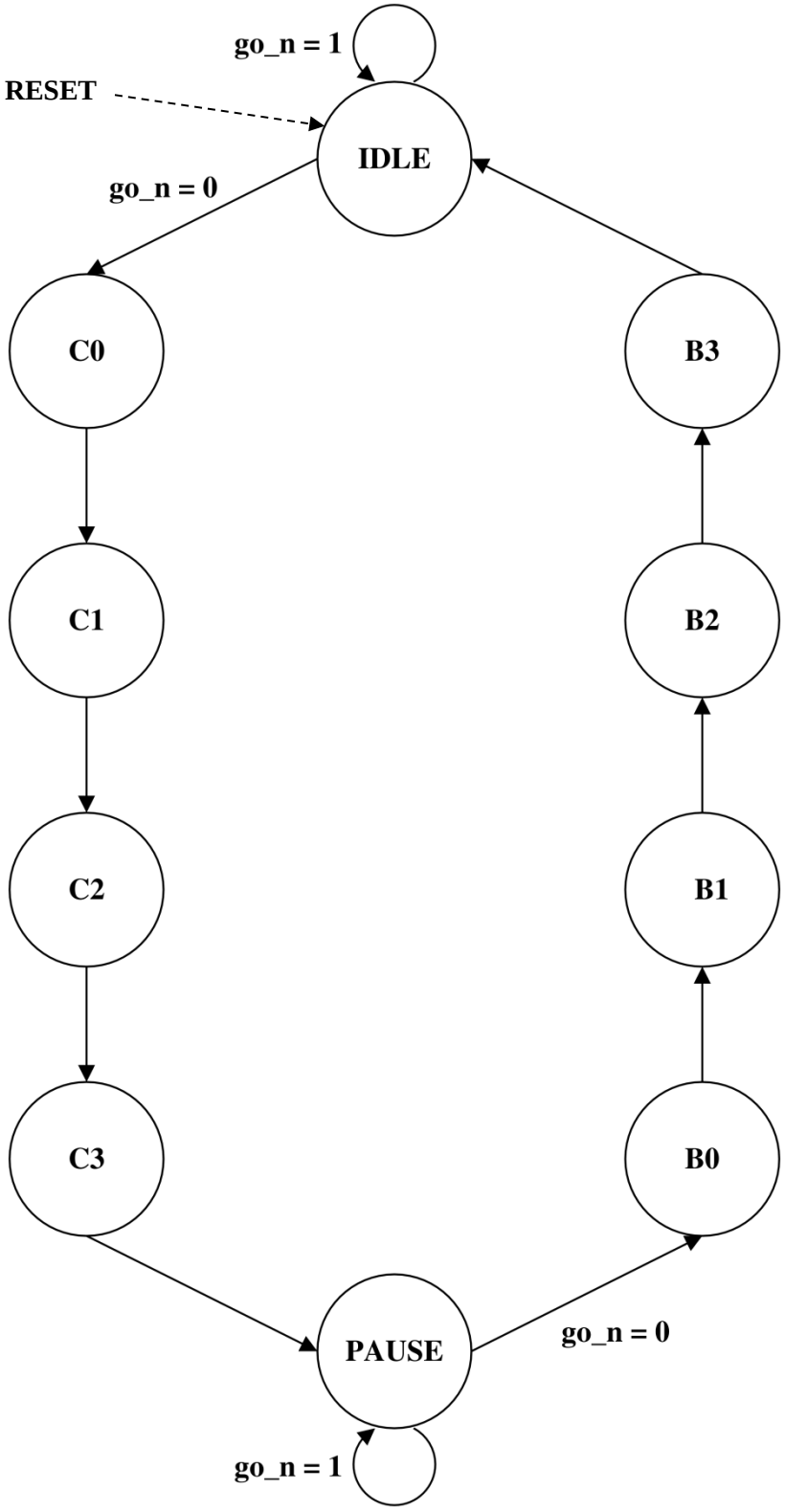
```
library ieee;
use ieee.std_logic_1164.all;

entity fsm is              -- controller for the lab5 datapath
  port (
        clk, reset_n, go_n : in std_logic;
        pattern : out std_logic_vector(2 downto 0);
        display : out std_logic_vector(1 downto 0);
        enable : out std_logic
       );
end entity;

architecture behavior of fsm is

-- define state type and signal for single-process high-level specification

type ... is (...);
signal ... : ...;

begin

-- define the process that implements the state transitions on clock edges

the_fsm : process (clk, reset_n)
begin
   .
   .
   .
end process;

-- use signal assignment statements with when...else... syntax
-- to specify the behavior of the pattern, display, and enable outputs

... <= ...;

... <= ...;

... <= ...;

end architecture;
```

The finite-state machine process will use the active-low `go_n` input to determine when to stay in the same state or when proceed to another state *in those situations where there is not an automatic transition to another state on the next clock edge*.

An active-low input is used because a pushbutton on the DE0 board will be used, and the design of the DE0 board is such that the signal value is 0 volts when the button is *pressed* (i.e., logic-0).

BEFORE YOUR LABORATORY SESSION, USE THE STATE DIAGRAM ON THE NEXT PAGE TO COMPLETE THE HIGH-LEVEL SPECIFICATION OF ALL OF THE STATE TRANSITIONS.

- The state diagram shown below is for the high-level VHDL description in the file `fsm.vhd`.



From reset, the controller remains in the IDLE state in successive clock cycles until `go_n` is 0. After that, only one cycle is spent in each of the states C0 to C3. Then, the controller remains in the PAUSE state until `go_n` is 0. After stepping through the states B0 to B3, we return to IDLE. In each of the states C0 to C3, *a specified character must <u>appear</u> on the specified hex display*. In each of the states B0 to B3, *a specified hex display must go <u>blank</u>*.

- In a file entitled `seven.vhd`, prepare a complete description based on the template below.

```
library ieee;
use ieee.std_logic_1164.all;      -- for std_logic and std_logic_vector types
use ieee.std_logic_unsigned.all;  -- needed to permit use of '+' in counter

use work.my_components.all;       -- declarations for components used below

entity seven is                   -- this is the top-level entity
  port (
        clk, reset_n, go_n : in  std_logic;
        slow_clk_out : out std_logic;
        hex_d0, hex_d1, hex_d2, hex_d3 : out std_logic_vector(6 downto 0)
       );
end entity;

architecture structure of seven is

-- define internal signals needed to connect component instances together
signal ... : ... ;

-- define internal signals related to counter process for slow clock
signal ... : ... ;

-- indicate which lab5 architecture to use when that component is instantiated
for all : lab5 use ...;

begin

-- include a 32-bit counter process based on the example in Lab 1 Part 7
-- (which was also used in Lab 5 Part 2)
the_count : process (clk, reset_n)
begin
   .
   .
   .
end process;

-- associate internal slow clock signal with a particular counter register bit
... <= ...;

-- associate the output pin for the slow clk with the internal signal
... <= ...;

the_lab5 : lab5 port map                -- instantiate a lab5 component
                (
                  -- associate the component ports with
                  -- either internal signals
                  -- or top-level entity ports
                );

the_controller : fsm port map           -- instantiate an fsm component
                     (
                       -- associate the component ports with
                       -- either internal signals
                       -- or top-level entity ports
                     );

end architecture;
```

Part 1: Gate-Level and High-Level Specification of Finite-State Machine

- Create a folder `Z:\ELEC271\LAB6`. (Or use a USB key for faster access without the network.)
- Place your template file `lab6.vhd` in the above folder.
- Start Altera Quartus II, and create a new project called `lab6` in the folder you created above.
- In the New Project Wizard dialog box, when you reach the Add Files step, use the "…" button to select your `lab6.vhd` file and be certain to click on the Add button to make it appear in the list.
- In the device selection step, choose *Cyclone III*, the *FBGA* package, *484* for the pin count, *6* for the speed grade, and then click on *EP3C16F484C6* to highlight that choice in blue. Click *Next*.
- Click *Next* to skip the EDA Tool portion, then click *Finish* to close the New Project Wizard.
- From the main menubar, select *Assignments → Device…* and then *Device and Pin Options…*
- For *Unused Pins,* change the option for 'Reserve all unused pins' to *As input tri-stated.*
- For *Voltage*, change the 'Default I/O standard' to *3.3-V LVTTL.*
- Select *OK* to close the Device/Pin Options dialog box and *OK* again to close the Device box.
- *On your worksheet*, draw the state diagram given to you at the start of the laboratory session.
- *On your worksheet*, fill in the combined state table, making appropriate choices for the assignment of state bits. *Double-check your work*.
- *On your worksheet*, derive the optimized expressions for the next-state logic. *Use appropriate care as you pursue your derivation in order to reduce the probability of making an error.*
- *On your worksheet*, derive the optimized expressions for the output logic *using appropriate care*.
- From the main menubar, select *File → Open...* and select your `lab6.vhd` file for editing.
- Ensure that you have a proper process to describe the behavior of the state bit flip-flops.
- Use your derived expressions to complete the ***g**ate-**l**evel* signal assignment statements for the next-state logic and the output logic. Use the suffix `_gl` for the related entity outputs. In addition, use `q1_out` and `q0_out` as entity outputs to allow the current state bit values to be visible.
- Using the given state diagram, complete the ***h**igh-**l**evel* VHDL description for the state machine in a process that uses `case` … `end case` syntax and `if` statements, as well as signal assignments for the outputs using `when` ... `else` syntax. Use the suffix `_hl` for the related entity outputs.
- From the main menubar, select *File → Save* to save the modified file.
- Select *Processing → Start Compilation* from the main menubar in order to synthesize the circuit.
- After synthesis, select *Assignments → Pin Planner*. Using the Terasic DE0 User Manual, make the pin assignments as indicated below.

  `reset_n`: BUTTON0 `clk`: BUTTON1 `q1_out`: LEDG9 `q0_out`: LEDG8

  …`_gl` outputs: LEDG5 and LEDG4 …`_hl` outputs: LEDG1 and LEDG0

  common primary inputs to both state machines: SW1 and SW0

- Close Pin Planner. Select *Processing → Start Compilation* **again** to resynthesize with new pins.
- Plug the USB cable into the DE0 board. Turn on the power. Select *Tools → Programmer.* Ensure "USB-Blaster" appears beside *Hardware Setup*. Press the *Start* button to program the FPGA chip.
- Use the clock/reset pushbuttons and the switches for the primary inputs to verify that both versions of the state machine implementation produce identical outputs. Also verify that state bit outputs from the gate-level implementation match the expected state bit values in your combined table. If the behavior is incorrect, review your derivations and/or your code to make corrections.
- *For credit, demonstrate your working systems on the DE0 board when you complete this part.*

Part 2: High-Level Specification of FSM Controller for Lab 5 Datapath

- Create a folder `Z:\ELEC271\SEVEN`. (Or use a USB key for faster access without the network.)
- Place your template files `my_components.vhd`, `fsm.vhd`, `seven.vhd` in the above folder.
- Place your working `lab5.vhd` from your previous laboratory activity in the above folder.
- Start Altera Quartus II, and create a new project called `seven` in the folder you created above.
- In the New Project Wizard dialog box, when you reach the Add Files step, use the "…" button to select the `my_components.vhd` file and be certain to click on the Add button to make it appear in the list. This file must be listed ***first*** in the collection of files because `seven.vhd` depends on it.
- Add the remaining files (`fsm.vhd`, `lab5.vhd`, `seven.vhd`) into the collection for this project.
- Perform the same chip-selection actions as in Part 1, *including the unused pins and voltage*.
- *On your worksheet*, record the specified output behavior for states C0 to C3 and states B0 to B3.
- Use the information from the worksheet to complete the output logic signal assignment statements in the `fsm.vhd` file. *You should have prepared the state-transition process **before** your session.*
- Recall from Lab 5 the role of the `enable` signal for the decoder and *ensure that your controller sets the value of this signal appropriately*, depending on the state and the desired behavior for it.
- In `seven.vhd`, ensure that the proper Part 1 architecture for the `lab5` component is specified.
- In `seven.vhd`, ensure that the counter process is correct. *Use counter bit 25 for the **slow clock***.
- From the main menubar, select *File → Save* to save the modified file.
- Select *Processing → Start Compilation* from the main menubar in order to synthesize the circuit.
- After synthesis, select *Assignments → Pin Planner*. Using the Terasic DE0 User Manual, make the pin assignments as indicated below.

`reset_n`: BUTTON0	`clk`: CLOCK_50 (either source will work)
`go_n`: BUTTON1	`slow_clk_out`: LEDG0

- For the hex-display pin assignments, use the same procedure as in Lab 5 with the .csv file from the course Webpage: select *Assignments → Import Assignments* and specify the .csv file.
- Close Pin Planner. Select *Processing → Start Compilation* **again** to resynthesize with new pins.
- Plug the USB cable into the DE0 board. Turn on the power. Select *Tools → Programmer.* Ensure "USB-Blaster" appears beside *Hardware Setup*. Press the *Start* button to program the FPGA chip.
- You should see the `slow_clk_out` LED blink on and off at a slow rate. This is your clock input to both the controller and the `lab5` datapath. The state machine should remain in IDLE as the slow clock runs, so nothing should appear on the hex displays (startup/reset makes them blank).
- Your finger will act as the 'circuit' that provides `go_n` input to the controller in a manner that is properly synchronized relative to the rising edge slow clock signal. In other words, *time your pressing and holding of the `go_n` button to begin before the rising edge*, with the release after the rising edge to be certain that the state machine has transitioned out of IDLE to the C0 state.
- *Note that during state C0, your controller provides inputs to the datapath in preparation for the next rising edge of the slow clock, hence no change occurs on a hex display until after that next edge. In other words, the datapath outputs will not change during C0. They change on the next clock edge that causes the controller to enter state C1. Understand how this delay arises from the edge-based behavior of the controller and the edge-based behavior of the datapath.*
- Observe the hex-display output changes that correspond to the state transitions in the controller. Experiment with the `go_n` input in both the PAUSE and IDLE states. Spend sufficient time to fully understand the timing of events – including your external input – relative to the slow clock signal.
- *For credit, demonstrate your functioning system when you complete this part.*

Part 1: Draw given state diagram below

Combined state table

	Next State				
Curr. State sym. *q1q0*	___ = ___ sym. *d1d0*	___ = ___ sym. *d1d0*	___ = ___ sym. *d1d0*	___ = ___ sym. *d1d0*	Outputs __ __

Derivation of next-state logic (two Karnaugh maps)

Derivation of output logic (two Karnaugh maps)

Part 2: Complete from the given sequences for displaying characters and displaying blank spaces

state:	C0	C1	C2	C3	B0	B1	B2	B3	
`display`:	___	___	___	___	___	___	___	___	(fill in binary values)
`pattern`:	___	___	___	___	___	___	___	___	(fill in binary values)
	'__'	'__'	'__'	'__'					(fill in letters that should appear on next clock edge)