State Machines

Introduction

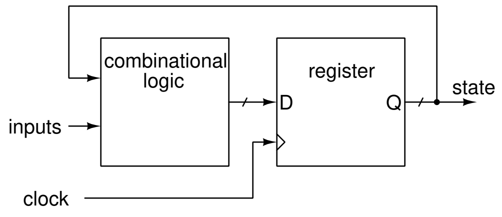
Sequential logic circuits are often designed as state machines. In this lab you will design and build a simple state machine.

For this lab you will need:

- a solderless breadboard
- your CPLD board, Byte Blaster JTAG interface and coaxial power connector,
- a 10-element LED bar array (or two LEDs)
- a 10-element 1 kΩ SIP resistor array (or two resistors $\leq$1kΩ),
- a pushbutton switch (double-throw or single-throw N.C.) and machine-pin DIP socket
- four pin-header jumpers (ideally, black, brown, red, and blue)

State Machines

A state machine consists of a register[1], whose value is the "state" of the state machine, and a combinational logic circuit that computes the next state:



On each rising edge of the clock the state changes to the next state because the register is loaded with the value of the next state. The next state is computed from the current state and the values of any inputs.

For the simple counters you will design in this lab the next state depends only on the current state. However, for a more complex counter the next state could also depend on an input such as a reset or count-direction input.

[1]A register is a group of flip-flops with a common clock.

State machines can be described by state transition diagrams. Each circle in the diagram is a state and the arrows between the states describe the conditions that cause transitions between states. For example, the state transition diagram for a 2-bit Gray-code counter would be:

$$\begin{tikzcd} 00 \arrow[r] & 01 \arrow[r] & 11 \arrow[r] & 10 \arrow[lll, bend left] \end{tikzcd}$$

The state transitions happen on the rising edge of the clock that loads the next value into the register.

Once the states and the state transition conditions are defined, it is necessary to design the combinational logic circuit that computes the next state. The behaviour of this circuit can be defined by a truth table where the current state is the input and the next state is the output.

For example, for the 2-bit Gray-code counter the truth table would be:

current state		**next state**	
A	B	A′	B′
0	0	0	1
0	1	1	1
1	1	1	0
1	0	0	0

where **A** and **B** are values of the two bits of the current state and **A′** and **B′** are the values of the two bits of the next state.

From the truth table we can derive the sum-of-products equations: $A' = \overline{A}B + AB$ and $B' = \overline{A}\,\overline{B} + \overline{A}B$[2]. From these equations were can create a schematic of the state machine in Quartus as shown in Figure 1 and program the CPLD to implement the counter.

Pre-Lab

The counter you will design in this lab will be determined by last digit of your student number[3]:

[2]You do not need to simplify these equations – CAD tools such as Quartus will do this for you

[3]This is to encourage everyone to do their own design.

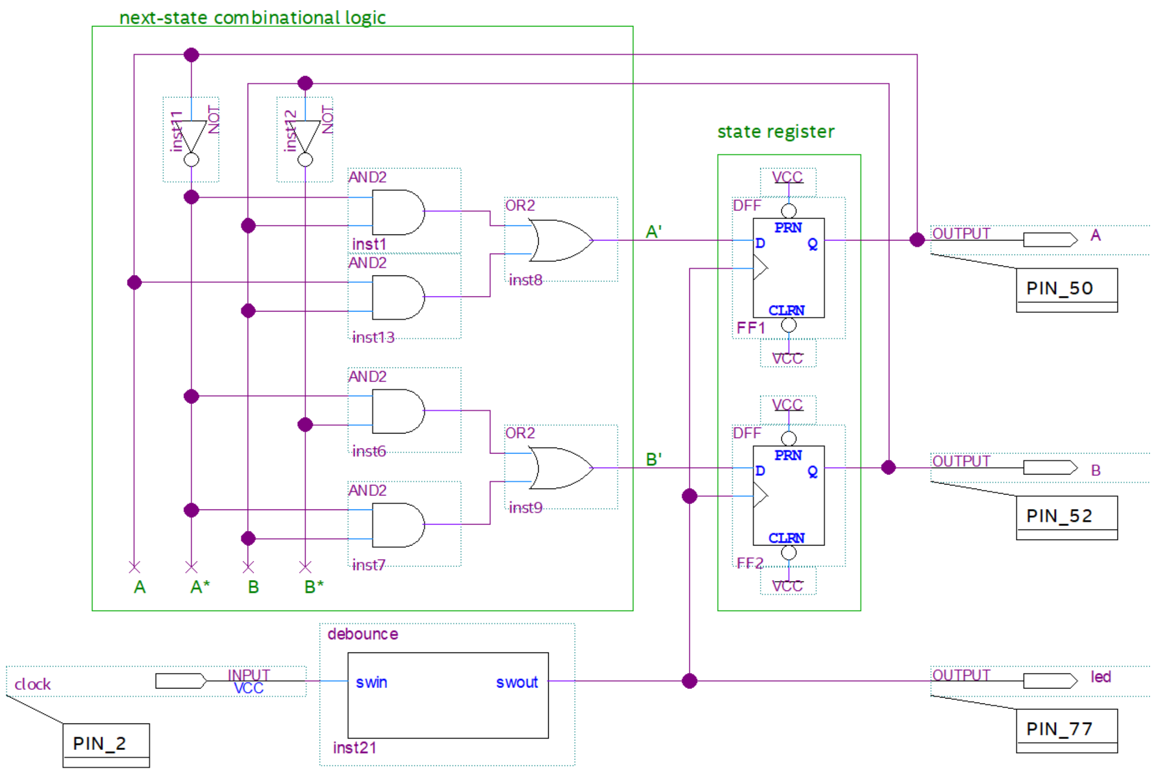


Figure 1: Schematic of 2-bit Gray-code up-counter.

digit	counter type	sequence
0 or 1	binary up-counter	0, 1, 2, 3, 0, ...
2 or 3	binary down-counter	0, 3, 2, 1, 0, ...
4 or 5	Gray-code down-counter	0, 2, 3, 1, 0, ...
6 or 7	modulo-3 up-counter	0, 1, 2, 0, ...
8 or 9	modulo-3 down-counter	0, 2, 1, 0, ...

For example, if your student number is A00123456 you will design a modulo-3 up-counter for this lab .

Write out, for *your* counter: (i) the state transition diagram, (ii) next-state truth table, (iii) the (unsimplified) sum of products expressions, and (iv) a schematic of the counter.

Your schematic should follow the general layout in Figure 1 so it's easy to understand and modify. If you've completed the schematic capture part of the procedure below, you can use a printout of your schematic[4] instead of a hand-drawn one.

If you've installed Quartus on your PC you can also follow the procedure below, complete the lab at home and bring your circuit to the lab to have it checked.

[4]If you've done some of this work on a computer, please bring a hard-copy to the lab, your instructors are old and have poor eyesight.

Procedure

Follow the general procedure in Appendix A to create a project, compile it and configure your CPLD. However, note the following additional instructions specific to this lab...

Add Files to Project

In step 4, download the **debounce.vhd** file from the course web site and copy it to your project folder. Add it to your project (**Project** > **Add/Remove Files in Project...**, **File name:** > ..., select the file, **OK**).

Select **Files** from the drop-down in the **Project Navigator** window. Right-click on the **debounce.vhd** file and select **Create Symbol Files for Current File**. This will create a **debounce.bsf** file in your project folder and you'll be able to add a **debounce** block to your schematic.

Schematic Capture

Also in step 4 you'll need to follow the procedure in Appendix B to create the schematic for your counter. Use components from the **c:\....primitives** library (**input**, **or2**, **and2**, **dff**, **not** and **output**). You'll also need to add a **debounce** component as described next.

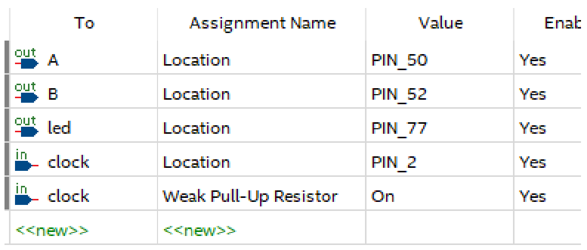
Debouncing

Mechanical switches typically produce many brief interruptions when they are switched. This "switch bounce" results in multiple signal edges. As shown in Figure 1, add a **debounce** component between the switch input (Pin 2) and the clock inputs to the flip-flops to eliminate switch bounces. This component will be available in the **Project** component library after you have added the **debounce.vhd** file to your project and created the corresponding **debounce.bsf** file in your project folder as described above.

Pin Assignments

In step 6 you'll also need to configure an internal pull-up resistor on the clock input pin. Open the Assignment Editor (**Assignments > Assignment Editor**). Double-click on **«new»** in the **To** column and enter the pin name (`clock`). Select **Weak Pull-Up Resistor** from the drop-down menu in the **Assignment Name** column. Select **On** from the drop-down menu in the **Value** column.

You should end up with the following assignments:

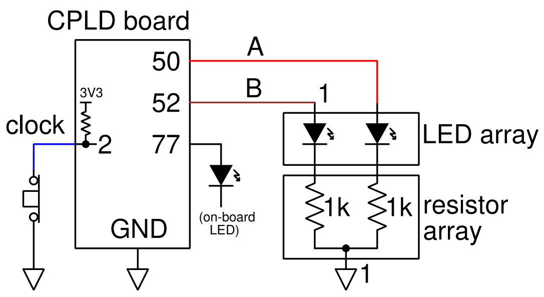


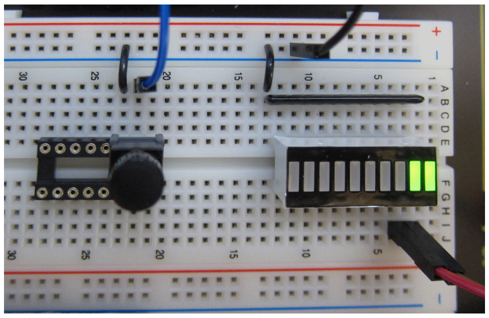
To	Assignment Name	Value	Enab
out A	Location	PIN_50	Yes
out B	Location	PIN_52	Yes
out led	Location	PIN_77	Yes
in clock	Location	PIN_2	Yes
in clock	Weak Pull-Up Resistor	On	Yes
<<new>>	<<new>>		

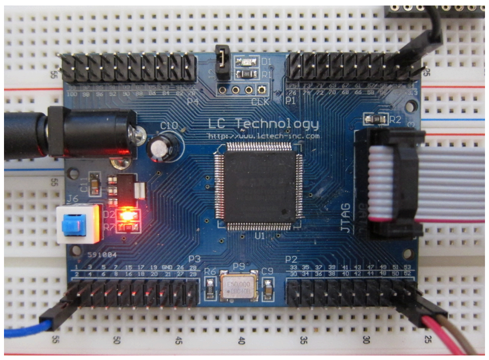
CPLD I/O

After the CPLD is programmed, the schematic and photos below show how it can be connected to a switch that generates clock pulses and to two LEDs that display the state.

The black wires are ground, blue is the clock, red and brown are the two bits of state, **A** and **B** respectively:







An internal pull-up resistor will be configured on CPLD pin 2, the clock input pin. Use the N.C. (normally closed) pushbutton switch contacts to cause a positive clock edge each time the button is pressed. Mount the pushbutton on a machined-pin DIP socket to secure it.

Connect two LEDs with 1 kΩ current-limiting resistors to CPLD pins 50 and 52 (for **A** and **B** respectively) to display the counter state. You can simplify the wiring by using the LED and resistor arrays as shown. Pin 1 of the LED array package is on the side with the part number. The pins on this side are the anodes (positive). Pin 1 of the resistor array is marked with a dot.

The schematic above also connects the **led** output to the LED on your CPLD board (pin 77) so you can verify the operation of your clock. Routing internal signals to an LED like this is a useful debugging technique.

Demonstration

Pressing the pushbutton should make the on-board LED light. On each press (rising clock edge) the count value should change to the next value. Show your working counter to the lab instructor. After your demonstration the instructor may ask you some questions or ask you to modify the count sequence to check your understanding of the concepts and the design process.

A Logic Synthesis with Quartus

1. start Quartus
2. select **File** > **New...** > **New Quartus Project** > **OK**
3. in the subsequent dialog boxes: select a folder (e.g. `H:\ELEX2117`), enter a project name (e.g. `lab2`), select an empty project, don't add files, select the Max II Family, select the specific device `EPM240T100C5`, and leave other settings at their defaults
4. add any existing design files using **Project** > **Add/Remove Files in Project**, or create new ones using **File** > **New...** You must have a top-level module whose port names correspond to the CPLD's pins. It must have the same name as the project. This may be a schematic file (e.g. `lab2.bdf`) or a VHDL file with a matching entity name (e.g. `entity lab2 is`).
5. after all the design files have been created and added to the project, select **Processing** > **Start Compilation**. Correct any errors and recompile as necessary
6. select **Assignments** > **Pin Planner** and select the correct pin in the **Location** drop-down box for each I/O pin. Note that you must compile the project before the pin names are visible in Pin Planner. Recompile the project (**Processing** > **Start Compilation**) for the assignments to take effect.
7. connect the CPLD board's coaxial power connector to a USB port and push the power button (power LED should go on). Connect the "USB Blaster" to the CPLD and a free USB port. The **POWER** and **ACT** lights should go on.
8. select **Tools / Programmer**, click on **Hardware Setup...**, select **USB-Blaster** from the drop-down and **Close**. **USB-Blaster** should appear next to **Hardware Setup...**
9. if necessary, click on **Add File..**, navigate to the location of the generated `.pof` file (typically in the `output_files` folder of the project folder) and select the `.pof` file
10. check that the **Program/Configure** checkboxes are checked and press **Start** to program the device. The progress bar should show 100%.
11. test your design

B Schematic Capture with Quartus

To create a new block diagram/schematic (`.bdf`) file:

- select **File** > **New...** > **Design Files** / **Block Diagram/Schematic File** > **OK**
- select **File** > **Save as...** and enter a file name. If this is the top-level file you must use the project name as the file name (e.g. `lab2.bdf`).
- use the "Symbol Tool" to insert the required components and I/O pins. Ensure each component has a unique instance name (e.g. **inst5**, **inst6**, ...). Double-click on pin and instance names to edit them. *Hint: Use the search box to find components quickly.*
- use the "Selection" or "Net"/"Bus" tools to connect the components. *Hint: an* × *indicates a wire that is not connected.*
- use indices in square brackets for bus names. For example, `signal[7..0]` means the 8 least-significant bits of the bus `signal`. *Hint: It is often easier to name signals (right-click >* ***Properties****) than to connect them. All signals with the same name will be connected together.*