CS 207 Digital Logic - Spring 2020
Lab 10

James Yu

Apr. 22, 2020

Objective

1. Go on with sequential circuit design, in particular counters.

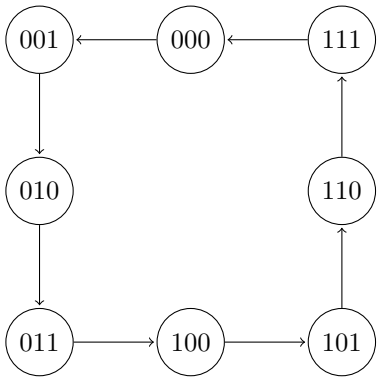
Lab Exercise Submission

1. You should name all source code as instructed. Mis-named files will not be recognized.
2. You should submit all source code files with an extension "`.v`".
3. You should submit all source code directly into the sakai system below. **Do not compress them into one folder.**

   `https://sakai.sustech.edu.cn/portal/site/ebf68254-68c5-4cfe-9d42-b26758f854ee`
4. Lab exercises should be submitted before the deadline, typically one week after the lab session. No late submission policy applies to lab exercises.

1 Three-bit binary counter

During the lecture we introduced the state diagram of the 3-bit binary counter, which loops from binary `0` to `7` and then back to `0` upon clock pulses. The state diagram is as follows:



We use Q to output the internal state. This module can be implemented as follows:

count.v

```
module count(Q, clk);
output reg [2:0] Q;
input clk;

always @(posedge clk)
case(Q)
    3'b000: Q <= 3'b001;
    3'b001: Q <= 3'b010;
    3'b010: Q <= 3'b011;
    3'b011: Q <= 3'b100;
    3'b100: Q <= 3'b101;
    3'b101: Q <= 3'b110;
    3'b110: Q <= 3'b111;
    3'b111: Q <= 3'b000;
    default: Q <= 3'b000;
endcase
endmodule
```

This module can be tested by the following testbench:

mealy.v

```
module count_tb;
reg CLK;
wire [2:0] Q;

initial begin
    CLK<=1'b0;
end

always @(CLK)
```

```
begin
    #1 CLK<=!CLK;
end

count count_1(Q, CLK);

initial begin
    $dumpfile("count.vcd");
    $dumpvars(0, count_tb);
end

initial begin
    #20 $finish;
end
endmodule
```

2 Exercise

2.1 Exercise 1

Base on the above code, implement a 4-bit binary counter with preset data and reset control. The module should follow the design below:

count4.v

```
module count4(Q, data, load, reset, clk);
output [3:0] Q;
input [3:0] data;
// Module Code
endmodule
```

and save in file `count4.v`. In this module, 1-bit `load` is the preset signal. Setting it to HIGH makes the counter load the number stored in `data` into its current state. 1-bit `reset` is the reset signal, which reset the state to `0` when set to HIGH.

Assignment

Save the source code in **`count4.v`**. Upload this file to Sakai under **Assignments → Lab Exercise 10**.

2.2 Exercise 2

Implement a modulo-M 8-bit counter. The counter takes an input M, which defines the radix of the counter. For example, when $M = 16$, the counter should start counting from `0` to `1` then to `2` until `F`, after which the counter resets to `0`. The module should follow the design below:

countm.v

```
module countm(Q, M, clk);
// Module Code
endmodule
```

and save in file `countm.v`. Note that M will not exceed $2^8 = 256$ during the test.

Assignment

Save the source code in **`countm.v`**. Upload this file to Sakai under **Assignments → Lab Exercise 10**.

Assignment

When you finished Lab Exercise 10, there should be two `.v` files in the Sakai system: **`count4.v`** and **`countm.v`**.