**Verilog Lab 6 (ECS1005)**

**Objectives:**

The objectives of this lab are as follows

* Learning how to model Finite state machines (Mealy Model) in Verilog
* Revision: Counters
* Mock Exam

**Reminder: Fixing the waveforms!**

When you click the Run button, it will compile. If it opens up the waveform, then you can simply skip the next step. If the wave form does not open, then click the shell screen, right next to the console on the right side of the screen as shown below.

A screen shot of a computer

Description automatically generated

On the shell, copy and paste the following command and enter:

chmod +x run.sh && ./run.sh

This command will re-compile the project and the waveform will launch as well.

**Task 1: Modelling a Finite state machine**

Problem Statement: Develop a minimal FSM circuit that detects the sequences ‘110’ in a bit-serial data stream.

Remember that an FSM has the following representation,

* Input to the FSM is called x
* Output of the FSM is called z
* State register inputs is called next state or Y
* State register outputs is called current state or y
* The Next state is then a function of both inputs (x) and the current state (y), hence Y=F(x,y)

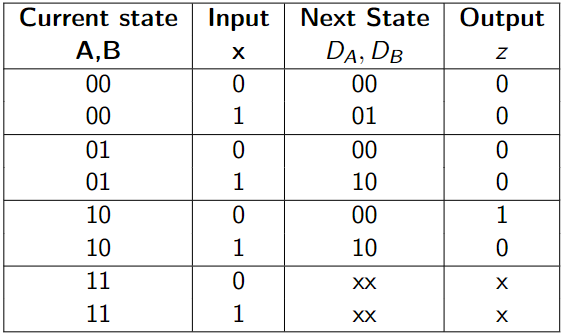
Diagram

Description automatically generated

Let’s develop the state diagram first, from that we can simply make a lookup table as well.

Text

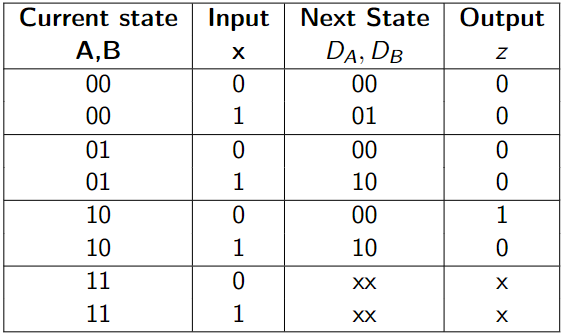
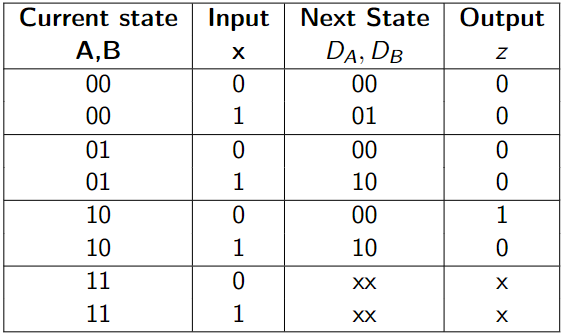
Description automatically generated



a

b

c



0/0

1/0

0/1

1/0

1/0

0/0

We learned in the class the for the synthesis (design) of an FSM circuit we use the following steps. For coding an FSM in Verilog, we stop after Step 3 to directly start coding (step 4 -7 are not needed). We need to re-check the code using a test bench which is similar to step 7 in the synthesis of the FSM anyway.

Graphical user interface, text, application

Description automatically generated

Let your starting point for the FSM finding 110 be the following code

<https://replit.com/@AyeshaKhalid5/Verilog-Lab6-FSM-Starting-point>

Graphical user interface

Description automatically generated with low confidence

Let’s now work on the various components of the FSM Verilog

* **FSM constants**

First, we define the width of the state register as a local parameter called SIZE. The names and binary codes of the three states are also defined as parameters.

Text, table

Description automatically generated

* **FSM State register**

We now define the state register of the FSM and its next value as curr\_state (y in the FSM diagram) and next\_state (Y in the FSM diagram).

A picture containing table

Description automatically generated

* **Sequential logic in the FSM**

The sequential logic of the FSM is defined below. Both curr\_state and next\_state are defined to be reg data types, however, the curr\_state becomes a Register (D-FF based register or the state register) while the next\_state is the output of the combinational block of the state machine (IFL and OFL). The state register (curr\_state) has an external asyncReset. Notice that if we want to create sequential logic or a D-FF based register we must use a clocked always block with Nonblocking assignments (<=). The Flip flop delay is represented by the #1.

A screenshot of a computer code

Description automatically generated

* **Combinational Logic (IFL+OFL) logic in the FSM**

The combinational logic in an FSM comprises of the IFL and the OFL. We use a case statement and the if-else inside an always procedural block for catering multiple cases of current state. Since there are multiple statements in each case (assigning both the next\_state and the output bit z), we must use begin and end for each. Do not forget the default since we have not used all the 4 possibilities of the 2-bit curr\_state. It is a good practice to use default even if all the cases are exhausted.

|  |  |
| --- | --- |
| Graphical user interface, text, application, email  Description automatically generated |  |

Let’s now focus on the testbench. We would like to check the working of our state machine using the given/stimulus input sequence (011101011001). Let’s first do a dry run to check the state transitions and the output z. Subject to the FSM state diagram to the input and trace state transitions and the outputs generated by filling in the following. (You will hit the sequence searched twice).

A picture containing text, clock

Description automatically generated

We declare a vector in the testbench as follows. (Notice how we use the index as 0:11 since we plan to read it from the left to right)



Insert the following loop in the initial block of the simulation after the asyncReset has been disabled in the testbench. The loop assigns the input x of the FSM the next bit from test string, every posedge of the clock.

A white background with black text

Description automatically generated

Is the output waveform as expected? Do you see output being 1 twice (2 pulses)?

A green and black line

Description automatically generated with medium confidence

Change the FSM so that it finds the string ‘1110’.

* Draw the updated FSM diagram. It now needs 4 states.
* Update the Verilog code of the FSM.
* Alter the testbench accordingly to check the presence of 1110. Put the waveform below.

**Task 2: Revision Counters**

We started with the topic of counters in the last lab. (Use lab 5 as needed).

Counters are a special case of finite-state machines because they move linearly through their discrete states (either forward or backward. Verilog allows simple up/down counters to be modelled using a single-procedural block with arithmetic operators (i.e., + and -). This enables a more compact model and allows much wider counters to be implemented in a practical manner.

Construct the following counters.

* **A 4-bit ring counter with an enable**

Let the following code be the starting point (it is a 4-bit ring counter code with test bench). Try to understand it fully before going ahead with the rest of the counters.

<https://replit.com/@AyeshaKhalid5/Verilog-Lab5-A-Ring-counter#counter.v>

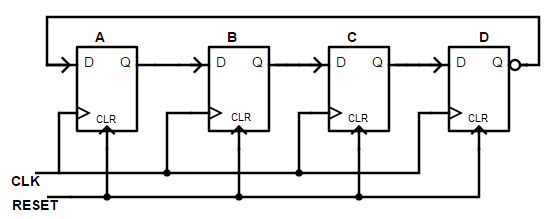
A diagram of a number system

Description automatically generated

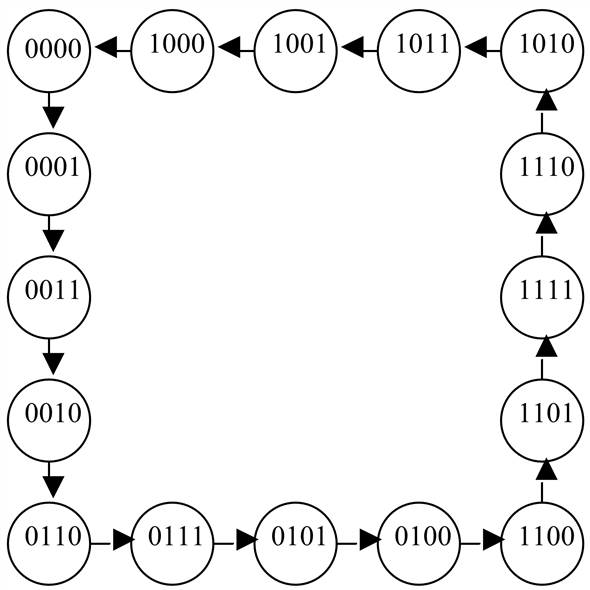
* **A 4-bit Johnson Ring Counter with an enable** (Hint: refer to lab 5). Best practice is the fork up the Repl used in the ring counter and give it a meaningful name for every new counter (e.g., lab 5 4-bit Johnson counter).

Notice that the reset value of a Johnson counter is all zeros (4’b0000). We add the inverted output of the LSB of the counter to be become the input of the MSB.

You should be able to monitor 2xN unique states in a Johnson counter (N=4 here)



* **A 4-bit Gray code counter:** The state diagram of the 4-bit Gray code counter is as shown in the Figure below.



It’s a 4 bit counter with a reset state being 4’b0000. Use a case statement inside a new always procedural block to switch the register from one state to the next based on the diagram given.

Text

Description automatically generated with low confidence

Where the Q is assigned the Q\_in, in the always block for the D-FF (Q <= #1Q\_in;). Remember Q\_in must be declared as a reg earlier.

Can you now write the code of any counter, given its state diagram or sequence?

* **A 4-bit up/down counter**: For an up counter, we use the arithmetic operator in the body of the always block (Q <= #1Q+1;). The reset value of the up-counter is kept 0. The code is provided below.

A screenshot of a computer program

Description automatically generated

For a down counter, we simply use the appropriate arithmetic operator (Q <= #1Q-1;). with a reset value of 4’b1111;. Can you also mimic a counter that counts up in increments of 2?

* **A mod-n counter**: Counters whose values “wrap around” back to zero when they reach the value n are called modulo-n counters. The most common example is a modulo-10 counter that counts from 0 up to 9 and then “wraps around” back to 0.

This can be modelled by inserting a nested if-else statement beneath of the else clause that handles the behaviour for when the counter receives a rising clock edge. This nested if-else first checks whether the count has reached its maximum value (9 for a mod10 counter). If it has, it is reset back to it minimum value. If it hasn’t, the counter is incremented as usual.

A screenshot of a computer code

Description automatically generated

See if you can get the waveform similar to the following.

A screenshot of a video game

Description automatically generated