Name *Roll Num*

Laboratory 6 – Advanced Finite State Machines

In this laboratory you will design an FSM of moderate complexity and implement it using gates and D registers.

Procedure:

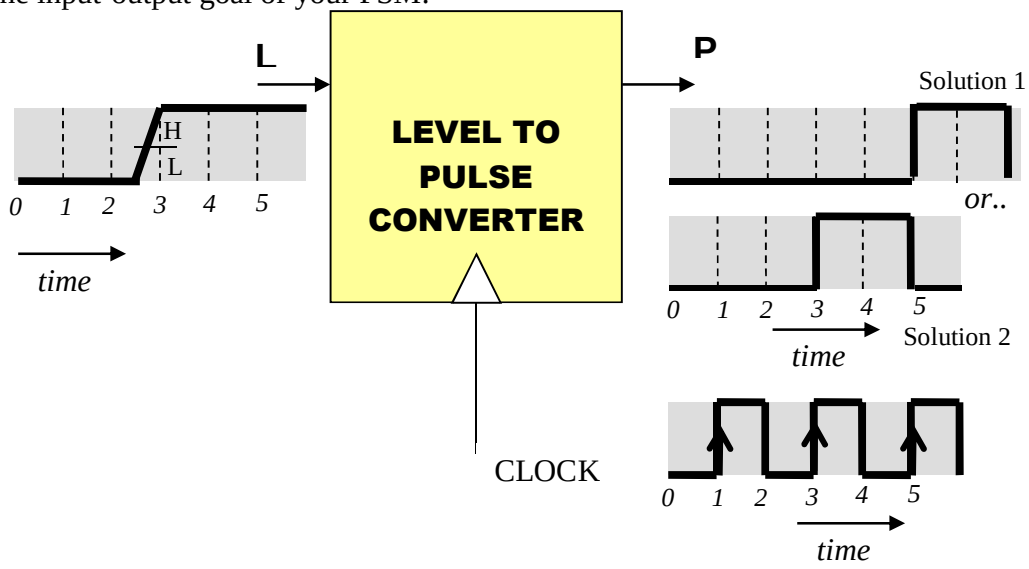
As we talked about the design process in class, we expect you to design the FSM in a logical manner. So here are a few hints – Do the following steps and discuss your write-up with a TA before proceeding to the next step:

1. Describe the FSM in words – what do you expect it to do: this will help you enumerate clearly the states of the FSM.
2. Draw a state transition diagram of the FSM which shows the allowed states of the machine and the logic that causes transitions between the states.
3. Once you have figured out the allowed states, you should also understand the amount of memory you need to store the states.
4. Work out the truth table for the FSM by specifying the "Current State", "Input" and "Next State".
5. Then use the general principle of how a sequential logic system works to design your FSM and demonstrate its operation.

Exercise: Level-to-Pulse translator: Detect a key-press

Often a synchronous (i.e. clock based) digital system may receive asynchronous inputs. A typical example is a keyboard key being pressed. The act of the key being pressed is not synchronized with the internal CPU clock: so some logic is needed to detect a user pressing a key.
An FSM offers a good solution since there are just two possible states of the key – up or down.
The objective of this exercise is to detect a transition between these states in the sequence:
(key up [=logic 0]) … (key pressed for *at least* one clock cycle [=logic 1])… (key up[logic 0]).
One complete clock cycle is defined as the period between two successive rising clock edges.
You may assume that the D registers you will use are positive clock edge sensitive.
After completing the design steps listed below, implement your FSM using the necessary IC's.
You may use a 1Hz/1kHz clock signal from the logic pulse generator as the clock, and a push button switch included in your parts kit to demonstrate the operation of your FSM.

Here is the input-output goal of your FSM:



It must take as input a digital signal level **L** which is normally in logic 0. Every time **L** goes high to logic 1 *and stays 1* for at least one clock cycle, the output **P** must be a digital pulse exactly one clock cycle wide.

*In the diagram shown above, **L** has completed a transition to logic 1 at clock edge 3. Note carefully the timing of the input **L** & output pulse **P** with respect to the clock edges (marked 0,1,2,3,4,5) It will determine how you design your FSM. Two designs are possible – they are marked differently in the timing diagram as Solution 1 and Solution 2. You can design your FSM so that it behaves according to either timing diagram. Note that the machine operates continuously, so state transitions beyond clock cycle 5 shown are similar.*

1) FSM state diagram

Draw a state diagram to determine how many states your FSM design contains**.** Label the states clearly and each state transition arrow must show the conditions on which that transition occurs.

/5
State Diagram

2) FSM Truth Table

Convert your state diagram of Part 2 into a truth table to determine the Boolean logic for state transitions.

/3
Truth table

3) Design implementation

Indicate whether your machine is expected to perform as Solution 1 (Moore) or Solution 2 (Mealy)
Build the circuit on the breadboard using the appropriate gates and registers implementing the logic worked out in Part 3 above. Use the 1 Hz pulse generator as the clock source. You can use the 'PP' positive pulse button to provide the input 'L' to your FSM. As long as the PP button is kept pressed, it produces logic 1 level. This is similar to holding down any* key on your keyboard for an extended length of time.
Hint: The D register has <u>both</u> Q and NOT(Q) outputs. You can simplify the logic a little by using the NOT(Q) output.

/7
4+3
Design + demo

* Please don't spend too much time trying to find the 'any' key on your keyboard

/15
Total

Name *Roll Num*

Reference:

Boolean Algebra Rules to remember:

In the following, the notation used is as follows:

(•) represents an AND operation

(+) represents an OR operation

(′) represents the NOT operation

(1a)	$x \cdot y$	$=$	$y \cdot x$
(1b)	$x + y$	$=$	$y + x$
(2a)	$x \cdot (y \cdot z)$	$=$	$(x \cdot y) \cdot z$
(2b)	$x + (y + z)$	$=$	$(x + y) + z$
(3a)	$x \cdot (y + z)$	$=$	$(x \cdot y) + (x \cdot z)$
(3b)	$x + (y \cdot z)$	$=$	$(x + y) \cdot (x + z)$
(4a)	$x \cdot x$	$=$	x
(4b)	$x + x$	$=$	x
(5a)	$x \cdot (x + y)$	$=$	x
(5b)	$x + (x \cdot y)$	$=$	x
(6a)	$x \cdot x'$	$=$	0
(6b)	$x + x'$	$=$	1
(7)	$(x')'$	$=$	x
(8a)	$(x \cdot y)'$	$=$	$x' + y'$
(8b)	$(x + y)'$	$=$	$x' \cdot y'$

Truth tables for the basic gate operations are:

AND (all high = high, else low)

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

OR (any high = high, else low)

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

NOT (inverter)

Input = 1	Output = 0
Input = 0	Output = 1

XOR (different = high, same = low)

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

NAND = (AND)′

NOR = (OR)′

XNOR = (XOR)′