

E2.1 – Digital Electronics II

Solutions to Problem Sheet 3 – Finite State Machines

(Question ratings: A=Easy, ..., E=Hard. All students should do questions rated A, B or C as a minimum)

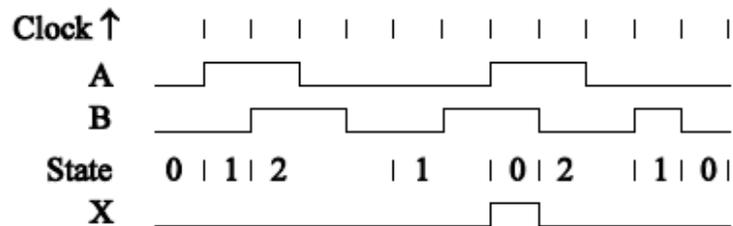
- 1B. In comparing state diagrams, you should first check the transitions and then check that the outputs are the same in each state. It can be seen that the transitions are the same for all the versions. However the outputs are incorrect in version (c) and version (e).

When output are marked on arrows they refer to the state from which the arrows originate. In version (c) therefore, state 1 has an output $Z=A$ instead of $Z=0$ as it should be.

In version (e), the output is not specified for the case $A=1$ in state 0. It must either be specified by default as in version (d) or else explicitly as in version (b).

- 2C. You should first determine the state sequence. The transitions depend on the value of A and B immediately *before* the Clock \uparrow edge. A common mistake is to use the values *after* the edge.

Note that X is only ever high in state 0 and then only if A and B are high. A common mistake is to make X high in state 2 rather than state 0: remember that outputs on transition arrows refer to the preceding state.



- 3B. This represents a 2-bit bidirectional counter whose counting sequence has only one bit changing at a time. This unit-distance property means that you can decode the outputs without risk of glitches.

DIR	S1	S0	NS1	NS0
0	0	0	0	1
0	0	1	1	1
0	1	0	0	0
0	1	1	1	0
1	0	0	1	0
1	0	1	0	0
1	1	0	1	1
1	1	1	0	1

- 4C. Since the output must go high during the fourth clock cycle in response to the value in that cycle, we must have a Mealy machine: a Moore machine would insert too much delay. If $IN=1$ during the current cycle then we want $OUT=1$ if the previous three (or more) cycles had $IN=1$. We therefore need to remember how many of the previous cycles had $IN=1$: 0, 1, 2 or ≥ 3 . We therefore need four states. Note that an unavoidable glitch possibility exists if IN goes high for three clock cycles; this can only be eliminated reliably by delaying the output for an extra cycle.

assumption that S1 is forced high whenever S0 is low; this means that the S1 flipflop's data input can be "don't care" whenever NS0 is equal to 0:

S1	S0	NS1	NS0
0	0	X	X
0	1	X	0
1	1	0	1
1	0	1	1

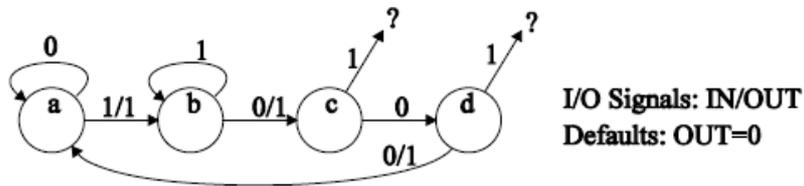
Choosing the "don't care" entries to simplify the expressions we get:

D flipflop inputs:

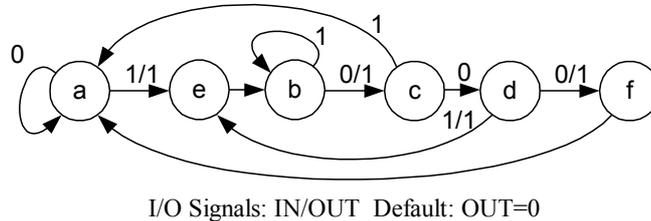
$$NS1 = \overline{S0}$$

$$NS0 = S1$$

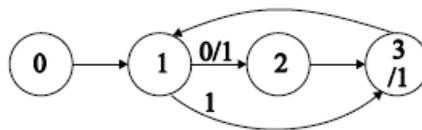
- 7C. The basic diagram is shown below; note that we *must* use a Mealy machine in order to get zero delay between IN and OUT. The only two points of difficulty is what to do if the input goes high in the middle of the double pulse sequence and whether we wish to ensure that consecutive pulses are separated by at least one clock cycle.



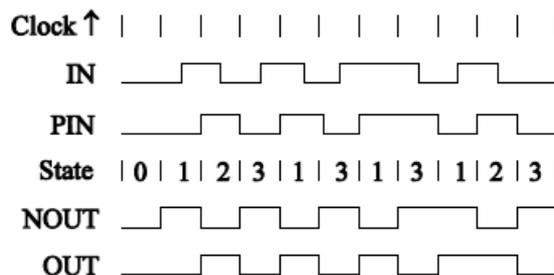
The following diagram ensures that pulses are distinct (by the addition of states e and f) and abandons pulse sequences when another input transition occurs:



- 8C.



I/O Signals: PIN/NOUT
Default: NOUT=0



9. The Verilog implementation of the FSM in Q8 is left to students. It should be fairly straight forward.

10. Here is one implementation (by me) that works very well:

```

module oneshot (
sysclk,      // Clock input to the design
trigger,     // Initial the delay time_out signal
width,       // width of pulse in sysclk cycles
pulse_out   // time_out goes high for one cycle, n clock cycles after trigger goes high
);           // End of port list

// Define number of bits in delay counter
parameter BIT_SZ = 10;

//-----Input Ports-----
input sysclk;
input trigger;
input [BIT_SZ-1:0] width;

//-----Output Ports-----
output pulse_out;

//-----Output Ports Data Type-----
// Output port can be a storage element (reg) or a wire
reg [BIT_SZ-1:0] count;

//----- Main Body of the module -----
reg [1:0] state;
parameter IDLE = 2'b00, COUNTING = 2'b01, TIME_OUT = 2'b10, WAIT_LOW = 2'b11;

// state transition

initial state = IDLE;
always @ (posedge sysclk)
    case (state)
        IDLE: if (trigger==1'b1) begin
                state <= COUNTING;
                count <= width-1;
            end
            else count <= 0;
        COUNTING: if (count==0) // time's up!
                state <= TIME_OUT;
            else
                count <= count - 1'b1;
        TIME_OUT: if (trigger==1'b0)
                state <= IDLE;
            else
                state <= WAIT_LOW;
        WAIT_LOW: if (trigger==1'b0)
                state <= IDLE;
            default: ; // do nothing
    endcase
// output logic - nothing!
    assign pulse_out = (state==COUNTING);
endmodule // End of Module oneshot

```