ECE 327/627
Digital Hardware Systems

Tutorial 4

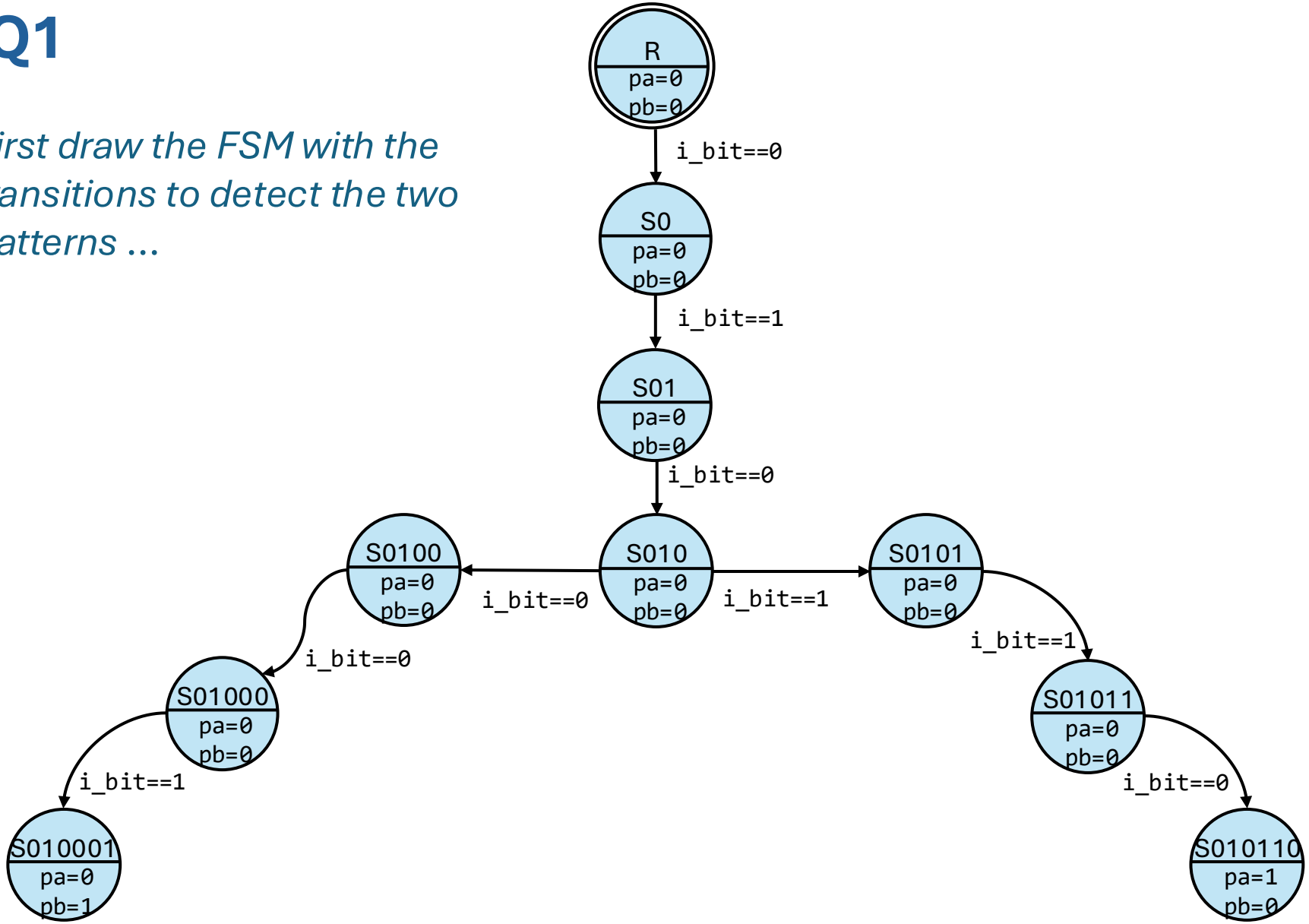
Q1

Design a single FSM to detect both the patterns given below. The input arrives one bit at a time (`i_bit`) from left to right. There are two 1-bit outputs (`pa, pb`) to indicate which of the two patterns were detected. Output goes high for one cycle after the pattern has been matched. Assume sliding window matching.

- Pattern A: 010110
- Pattern B: 010001

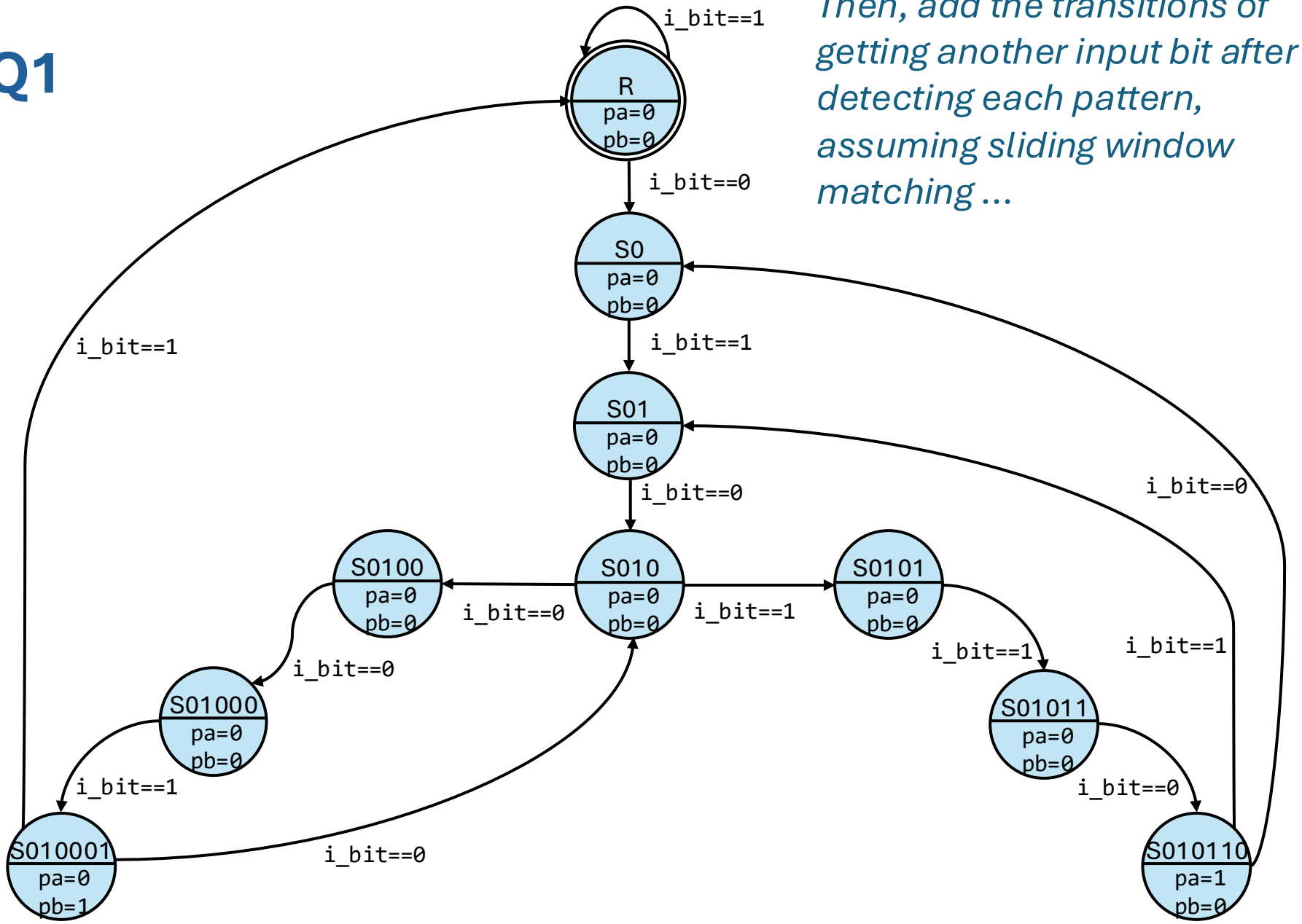
Q1

First draw the FSM with the transitions to detect the two patterns ...

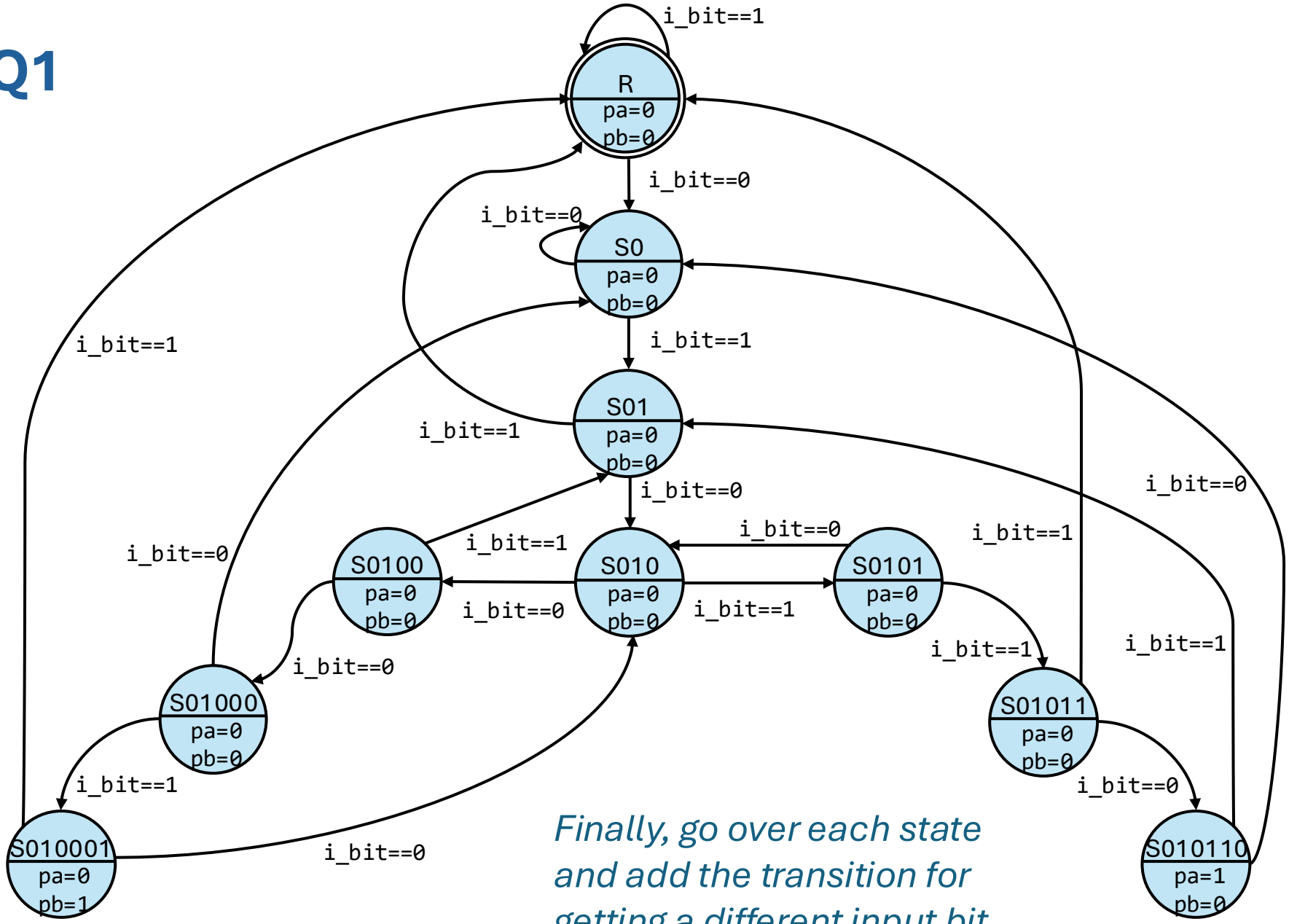


Q1

Then, add the transitions of getting another input bit after detecting each pattern, assuming sliding window matching …



Q1



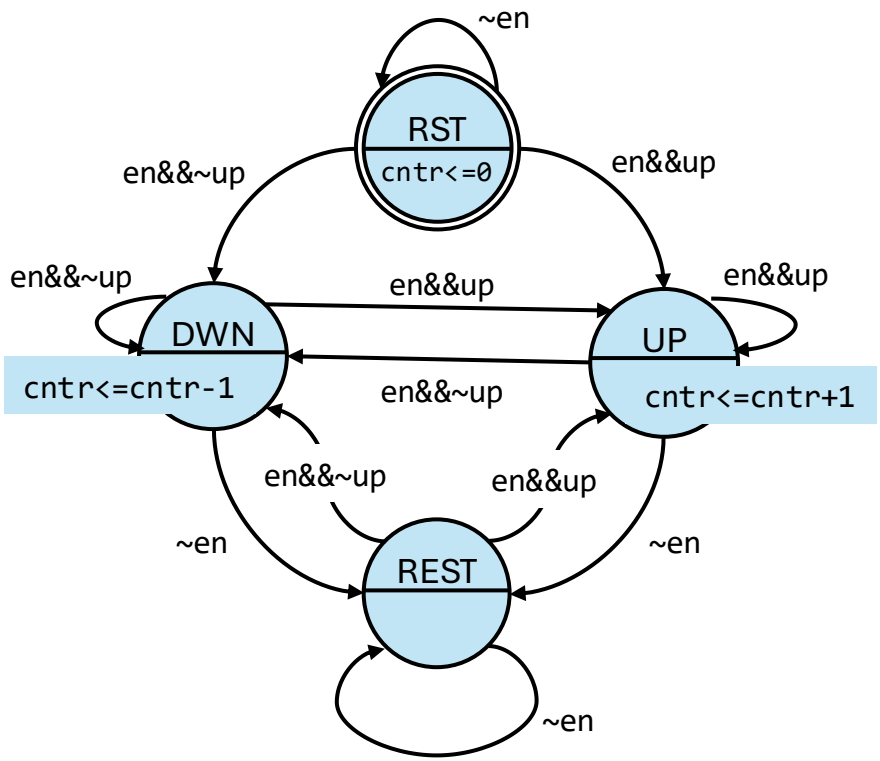
Finally, go over each state and add the transition for getting a different input bit

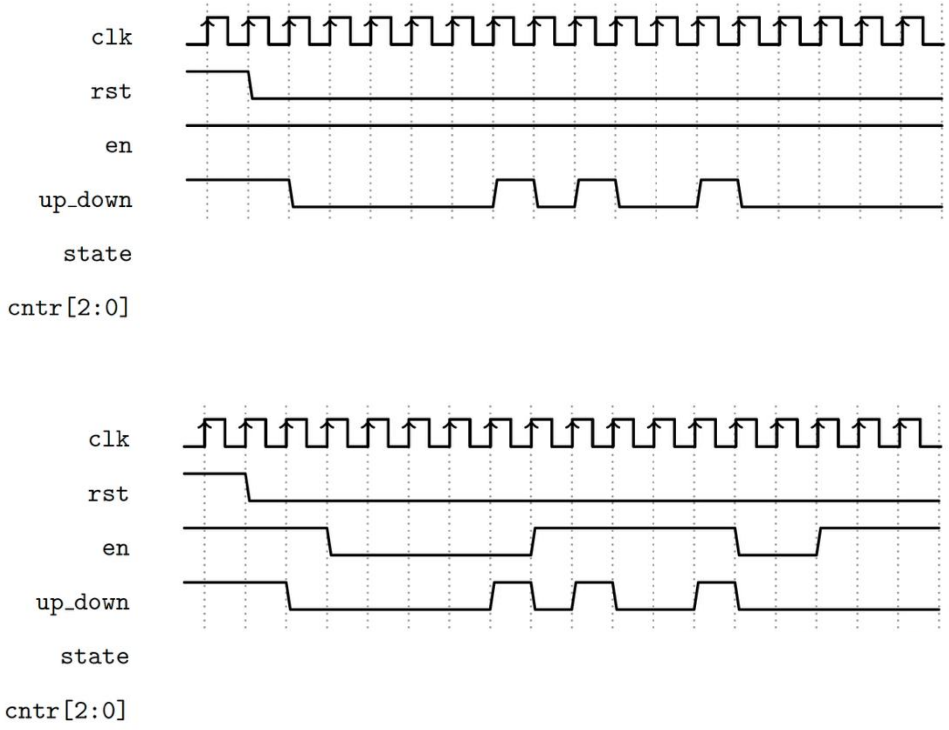
Q2

Given the following FSM for a 3-bit up/down counter

(a) Write the SystemVerilog code for it.

(b) Complete `state` and `cntr` traces for the 2 waveforms below





Q2

Given the following FSM for a 3-bit up/down counter

(a) Write the SystemVerilog code for it.

(b) Complete `state` and `cntr` traces for the 2 waveforms below

```
module up_down_cntr (
     input clk,
     input rst,
     input en,
     input up,
     output logic [2:0] cntr
);

enum {RST, UP, DWN, REST} state, next_state;
logic [2:0] cntr_w;

always_ff @ (posedge clk) begin
     if (rst) begin
          state <= RST;
          cntr <= 0;
     end else begin
          state <= next_state;
          cntr <= cntr_w;
     end
end
```

Q2

Given the following FSM for a 3-bit up/down counter

(a) Write the SystemVerilog code for it.

(b) Complete `state` and `cntr` traces for the 2 waveforms below

```
always_comb begin: state_decoder
  case (state)
    RST: next_state = (~en)? RST :
                      (up)? UP : DWN;

    UP: next_state = (en&&up)? UP :
                     (en&&~up)? DWN : REST;

    DWN: next_state = (en&&~up)? DWN :
                      (en&&up)? UP : REST;

    REST: next_state = (en&&up)? UP :
                       (en&&~up)? DWN : REST;
    default: next_state = RST;
  endcase
end
```

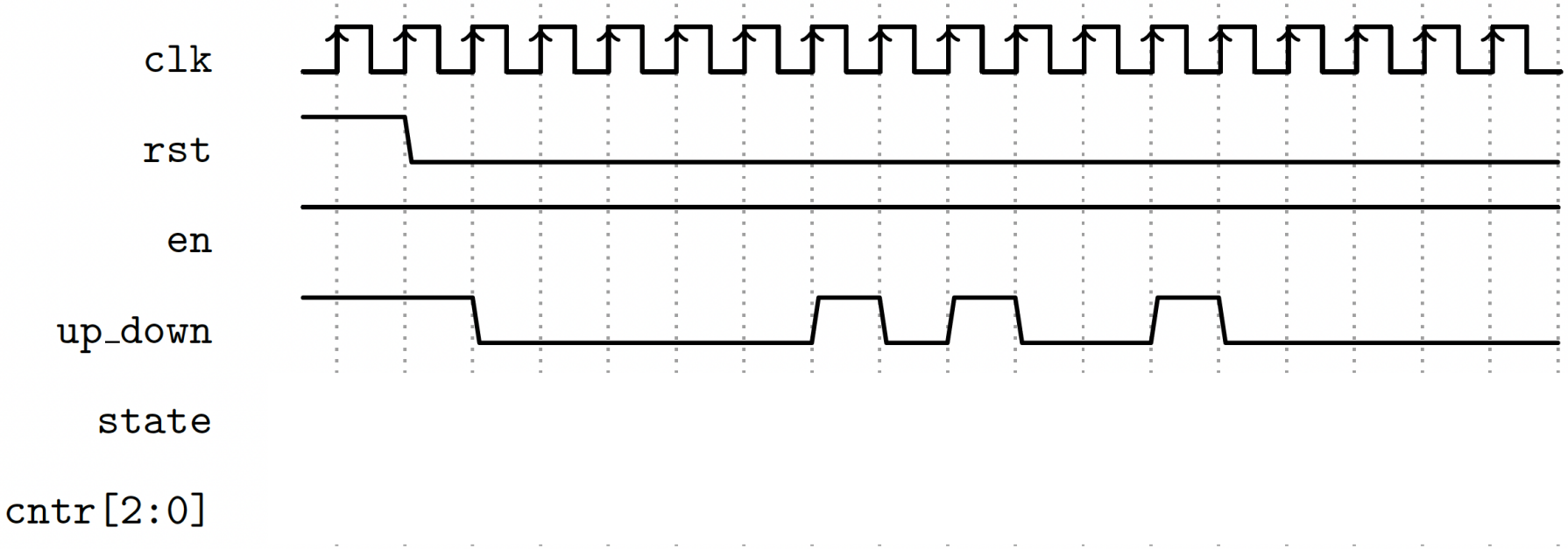
```
always_comb begin: output_decoder
  case (state)
    RST: cntr_w = 0;

    UP: cntr_w = cntr + 1;

    DWN: cntr_w = cntr - 1;

    REST: cntr_w = cntr;

    default: cntr_w = 0;
  endcase
end

endmodule
```

Q2

Given the following FSM for a 3-bit up/down counter

(a) Write the SystemVerilog code for it.

(b) Complete `state` and `cntr` traces for the 2 waveforms below

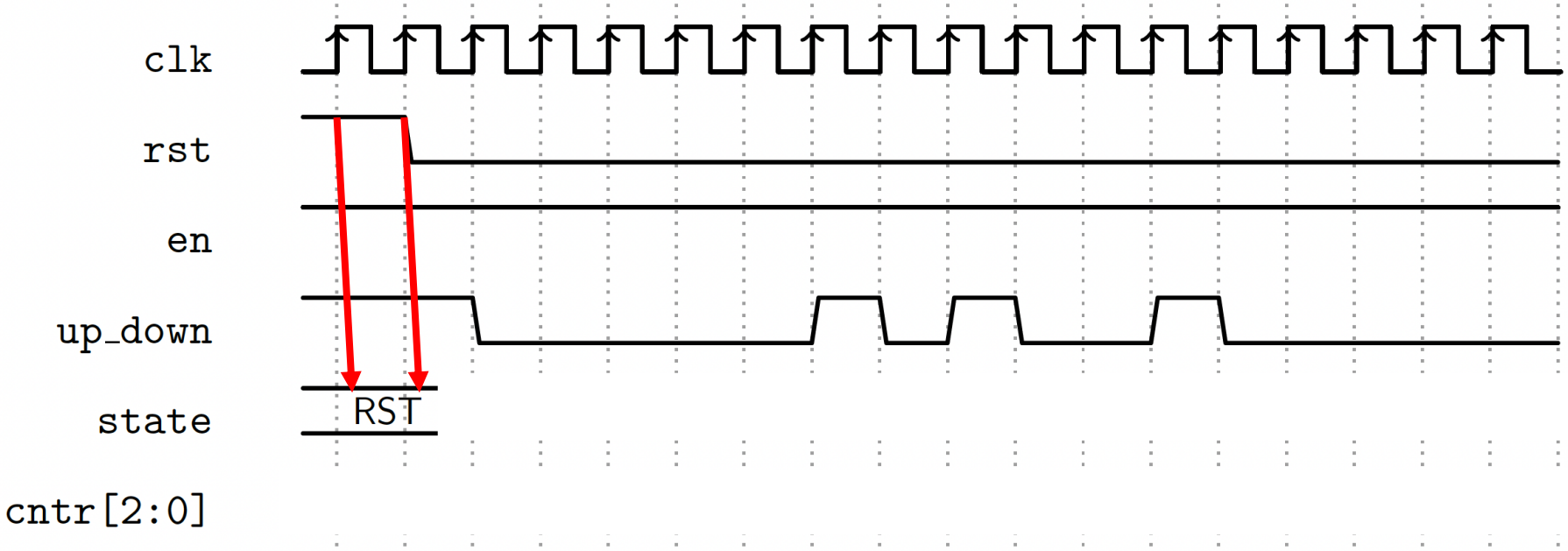


Q2

Given the following FSM for a 3-bit up/down counter

(a) Write the SystemVerilog code for it.

(b) Complete `state` and `cntr` traces for the 2 waveforms below

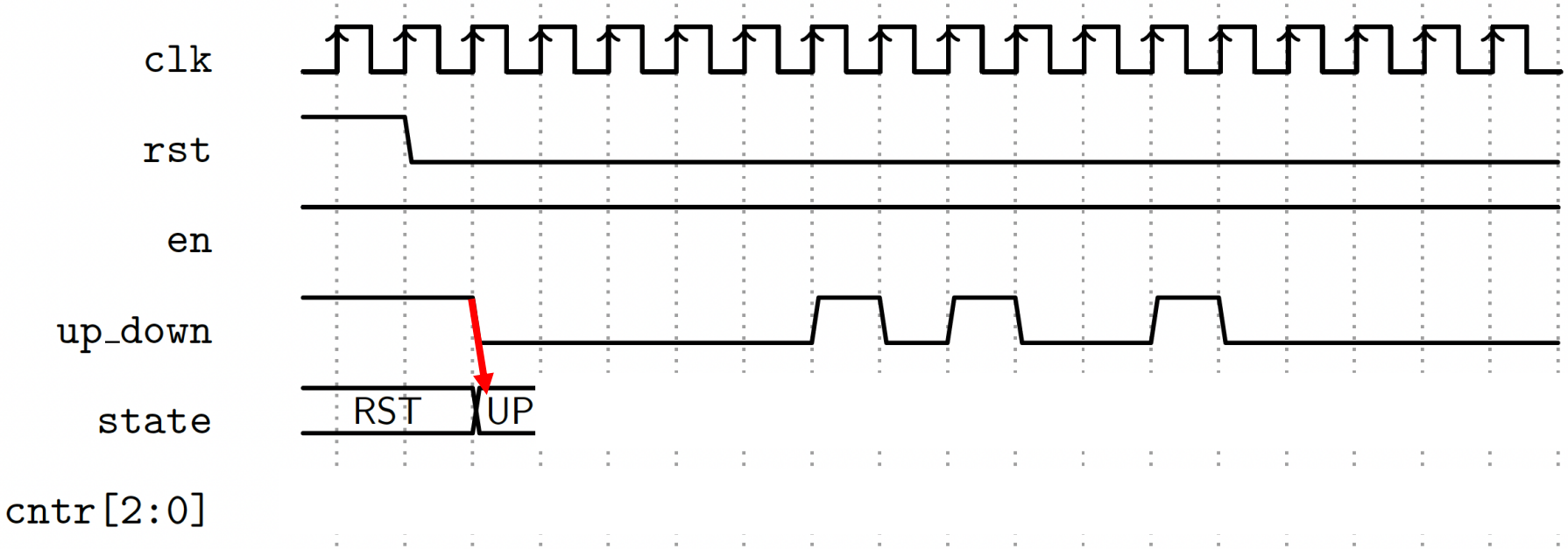


Q2

Given the following FSM for a 3-bit up/down counter

(a) Write the SystemVerilog code for it.

(b) Complete `state` and `cntr` traces for the 2 waveforms below

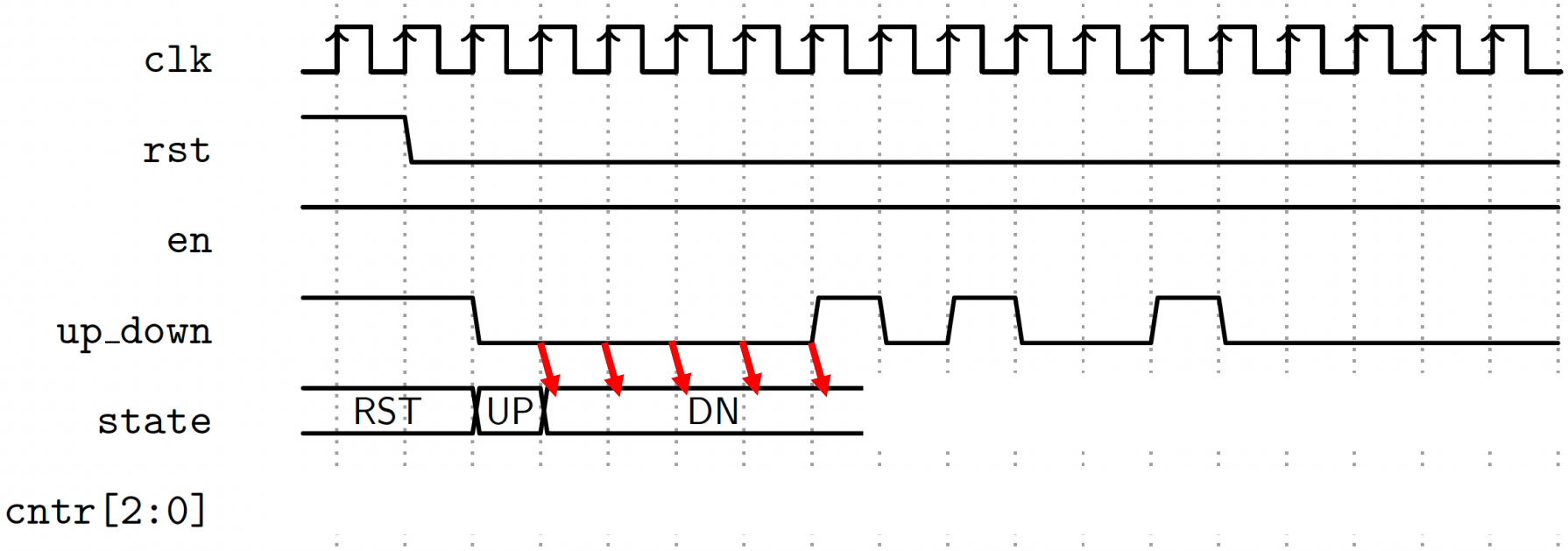


Q2

Given the following FSM for a 3-bit up/down counter

(a) Write the SystemVerilog code for it.

(b) Complete `state` and `cntr` traces for the 2 waveforms below

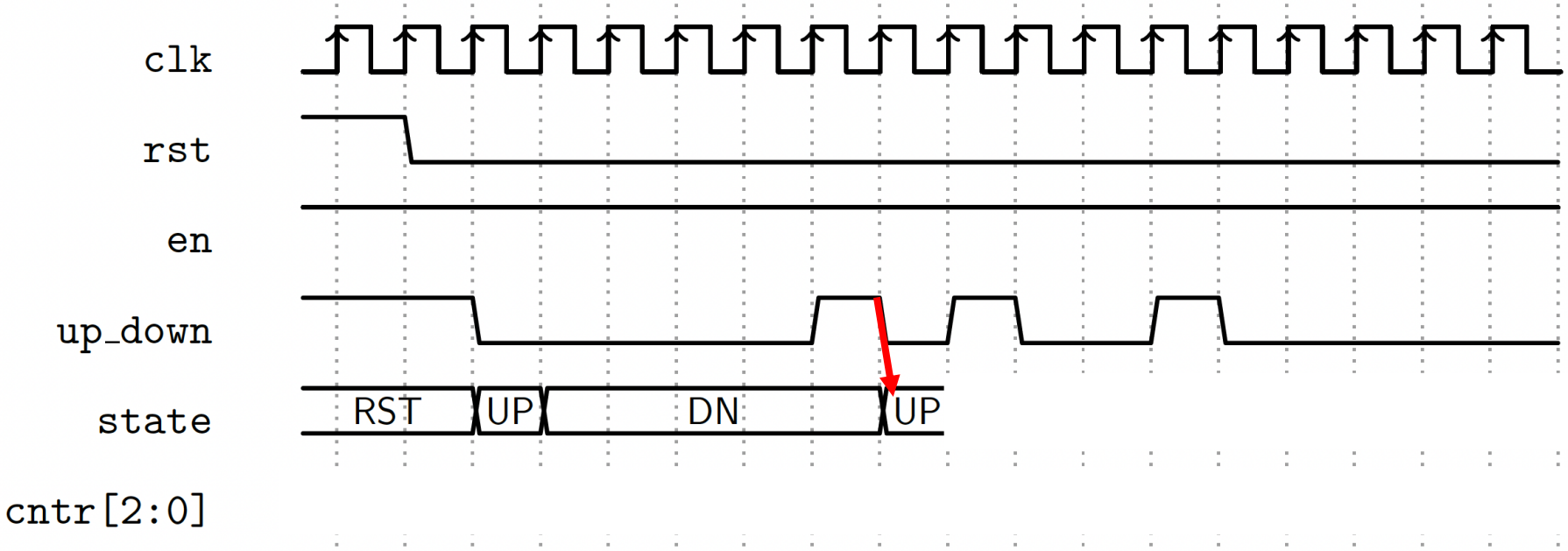


Q2

Given the following FSM for a 3-bit up/down counter

(a) Write the SystemVerilog code for it.

(b) Complete `state` and `cntr` traces for the 2 waveforms below

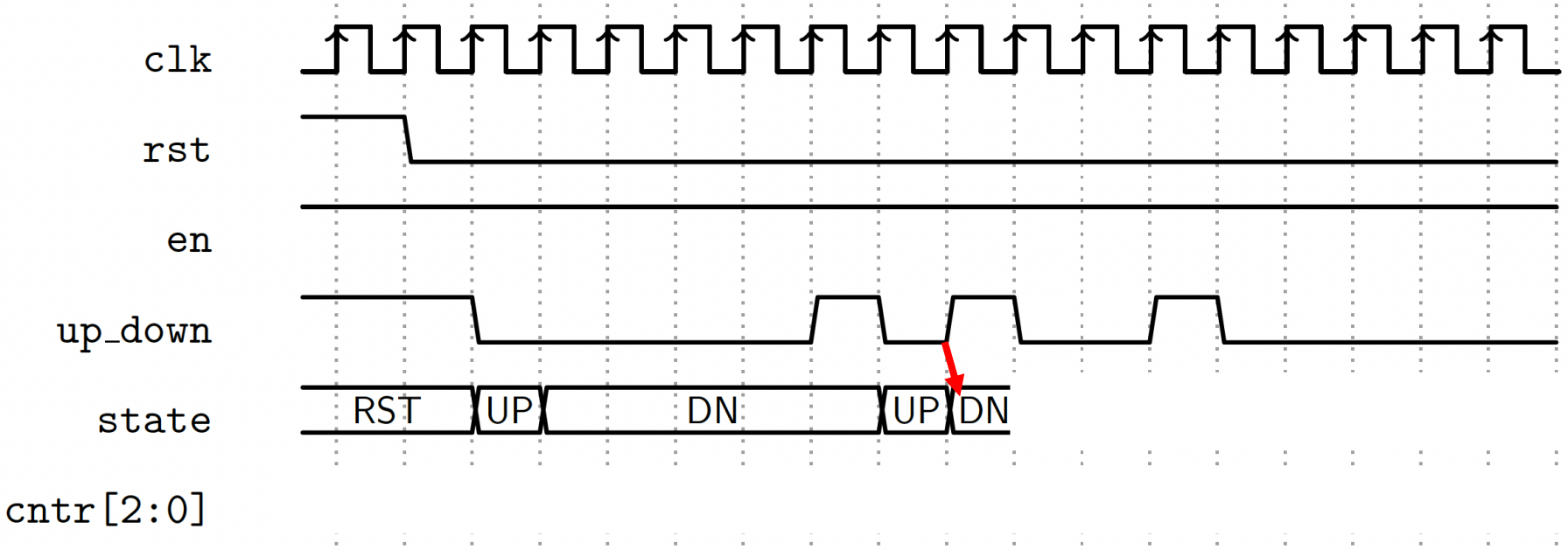


Q2

Given the following FSM for a 3-bit up/down counter

(a) Write the SystemVerilog code for it.

(b) Complete `state` and `cntr` traces for the 2 waveforms below

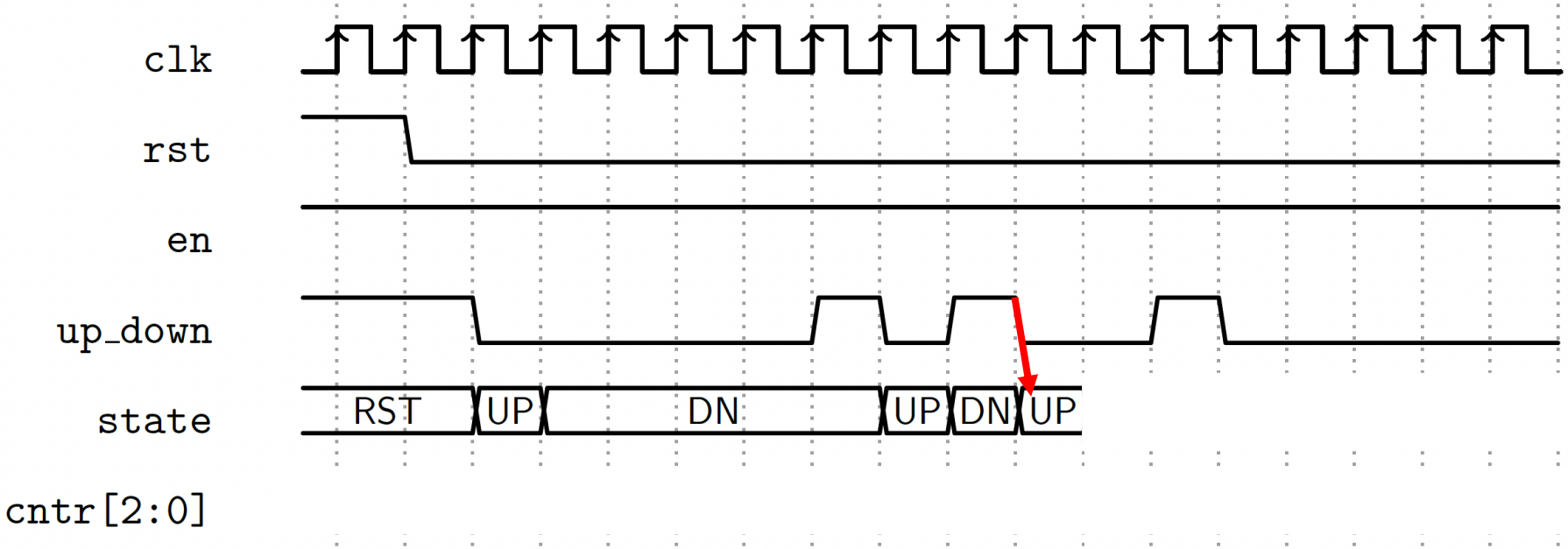


Q2

Given the following FSM for a 3-bit up/down counter

(a) Write the SystemVerilog code for it.

(b) Complete `state` and `cntr` traces for the 2 waveforms below

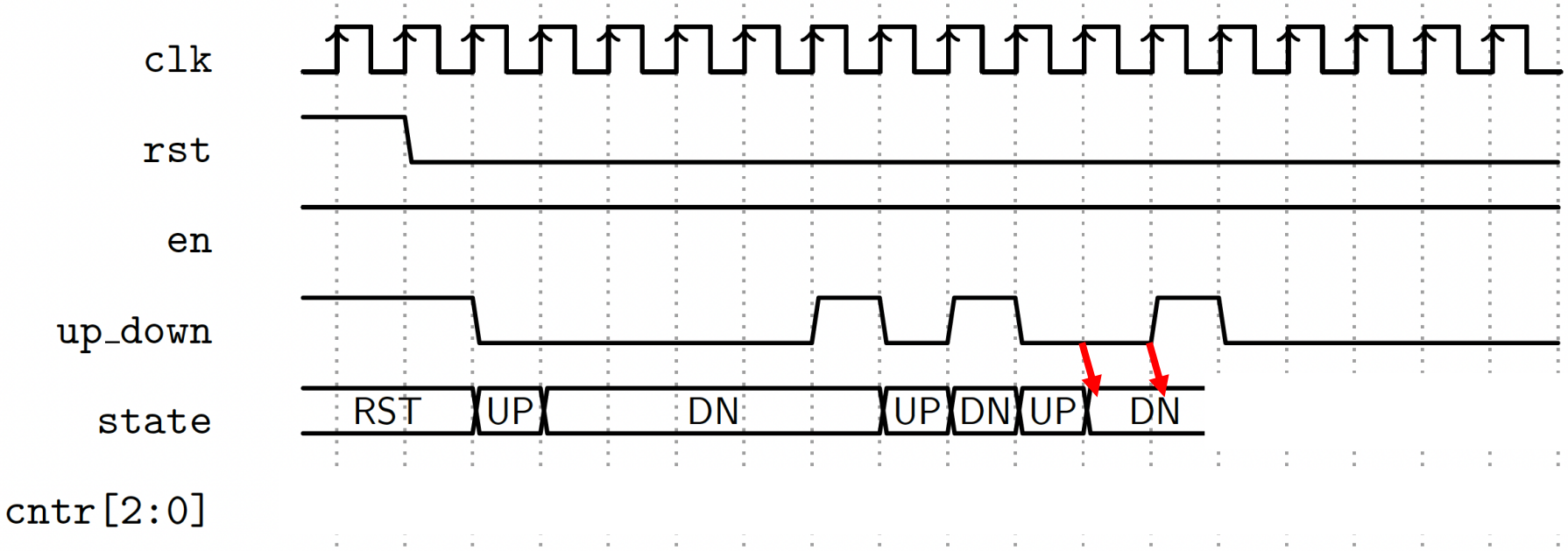


Q2

Given the following FSM for a 3-bit up/down counter

(a) Write the SystemVerilog code for it.

(b) Complete `state` and `cntr` traces for the 2 waveforms below

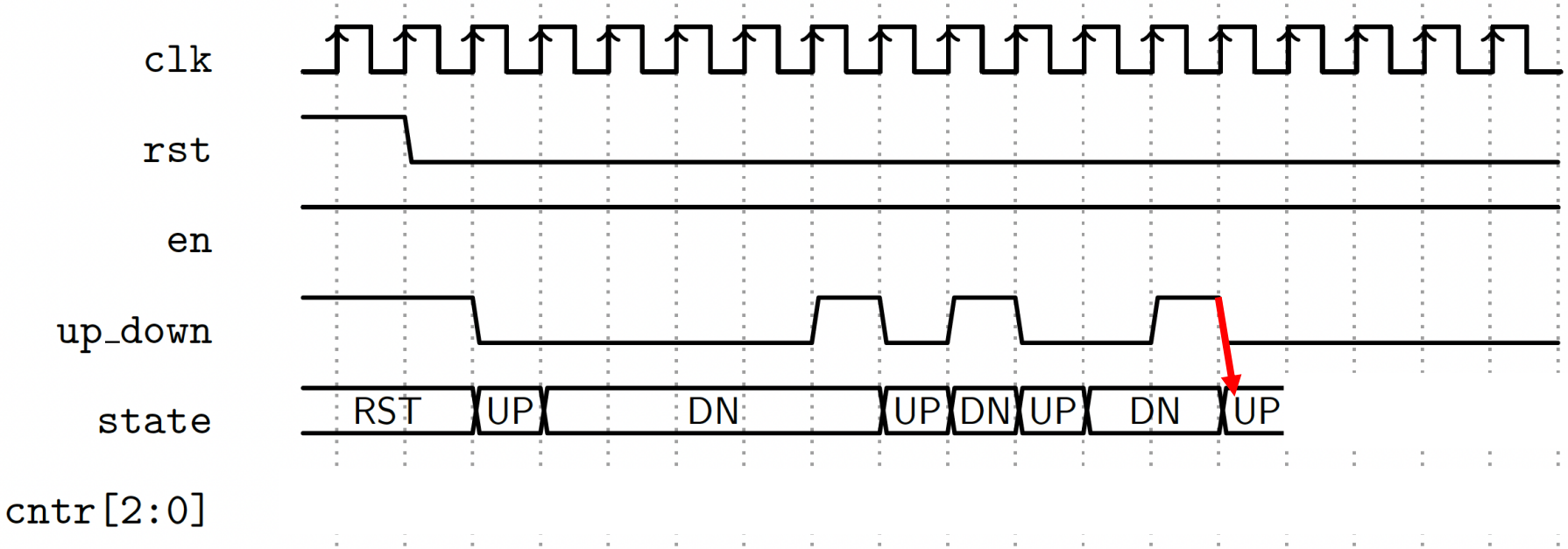


Q2

Given the following FSM for a 3-bit up/down counter

(a) Write the SystemVerilog code for it.

(b) Complete `state` and `cntr` traces for the 2 waveforms below

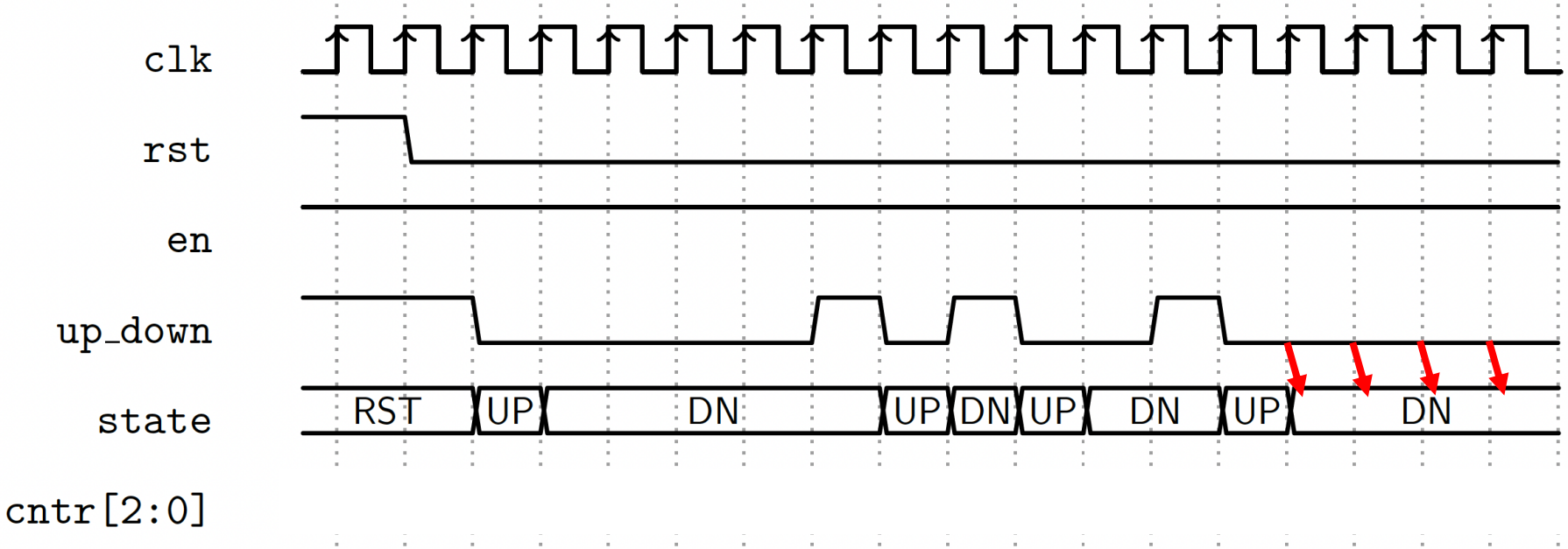


Q2

Given the following FSM for a 3-bit up/down counter

(a) Write the SystemVerilog code for it.

(b) Complete `state` and `cntr` traces for the 2 waveforms below

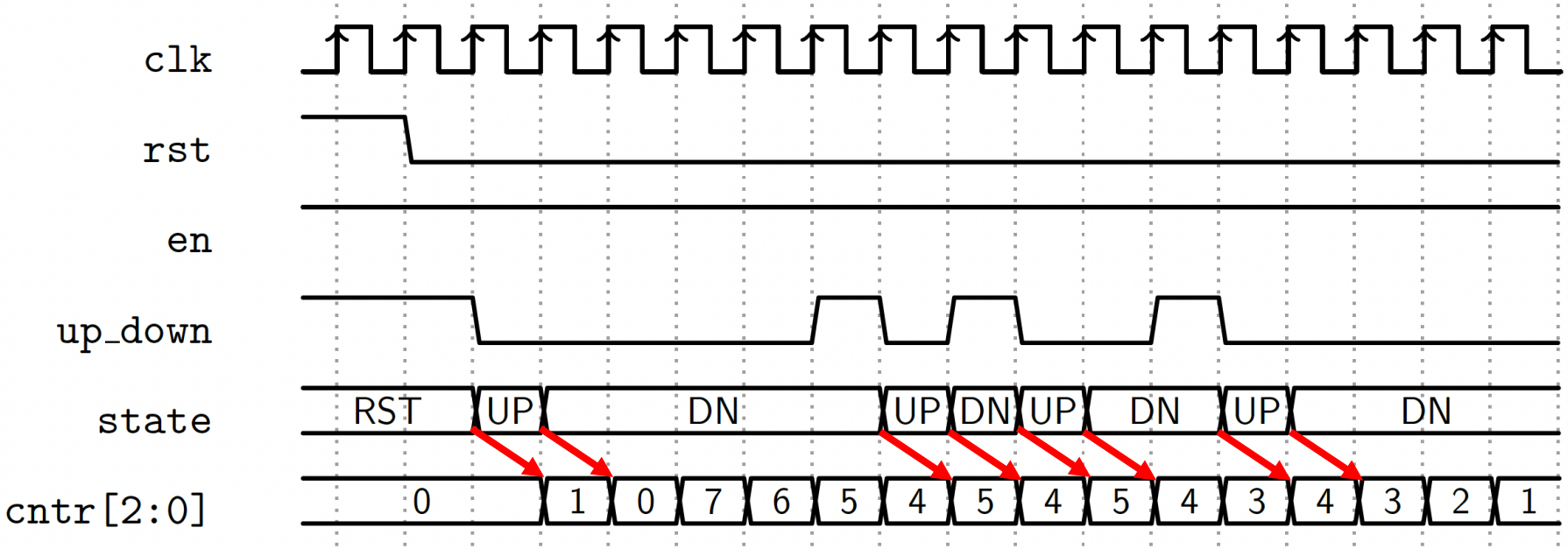


Q2

Given the following FSM for a 3-bit up/down counter

(a) Write the SystemVerilog code for it.

(b) Complete `state` and `cntr` traces for the 2 waveforms below

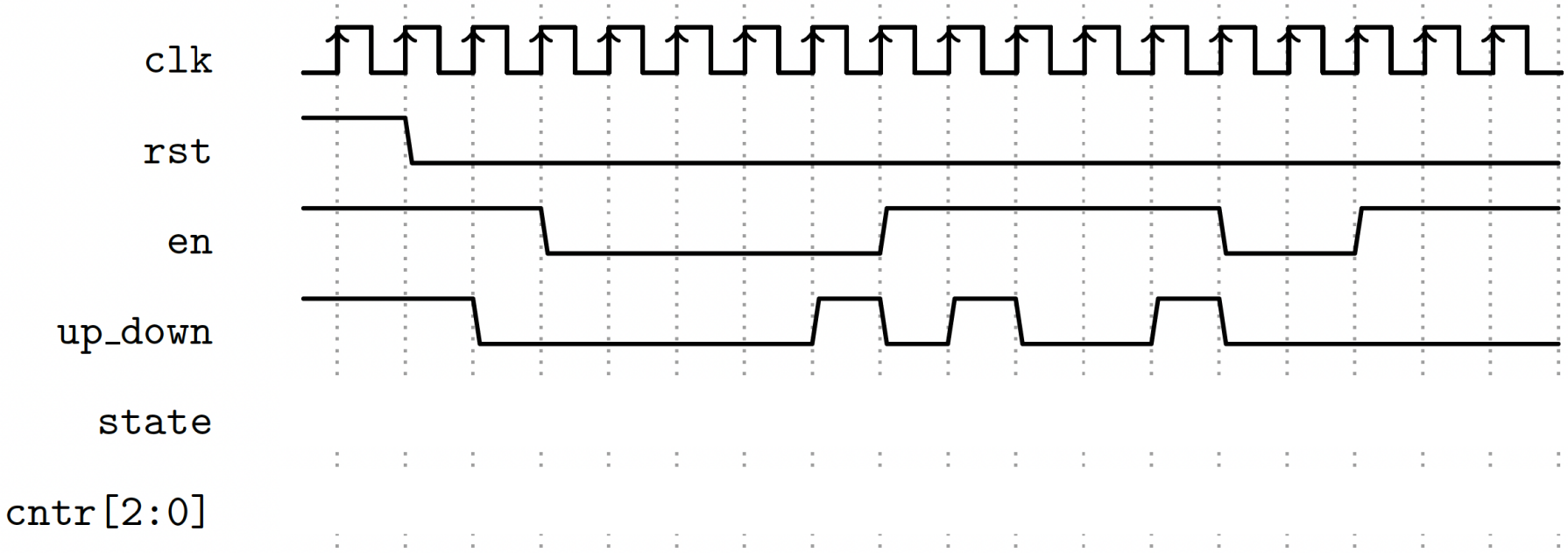


Q2

Given the following FSM for a 3-bit up/down counter

(a) Write the SystemVerilog code for it.

(b) Complete `state` and `cntr` traces for the 2 waveforms below

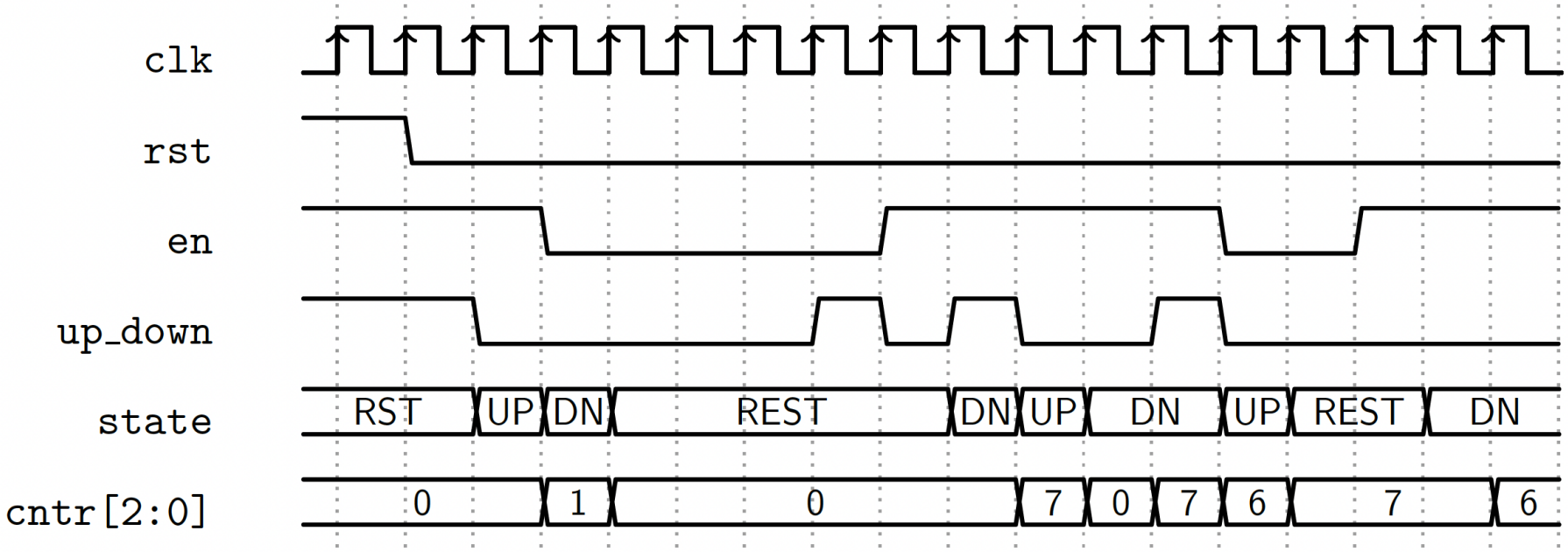


Q2

Given the following FSM for a 3-bit up/down counter

(a) Write the SystemVerilog code for it.

(b) Complete `state` and `cntr` traces for the 2 waveforms below



Q3

In many cases, large messages are sent from one device to another serially as a sequence of bytes (8-bit chunks). You are asked to design a finite-state machine (FSM) that receives a byte-serial message according to a pre-defined protocol and writes it to a memory buffer on the receiver side. The transmitter sends the message in the following format:

`[START] [LEN] [ADDR_H] [ADDR_L] [DATA_0] … [DATA_n] [CHECKSUM]`

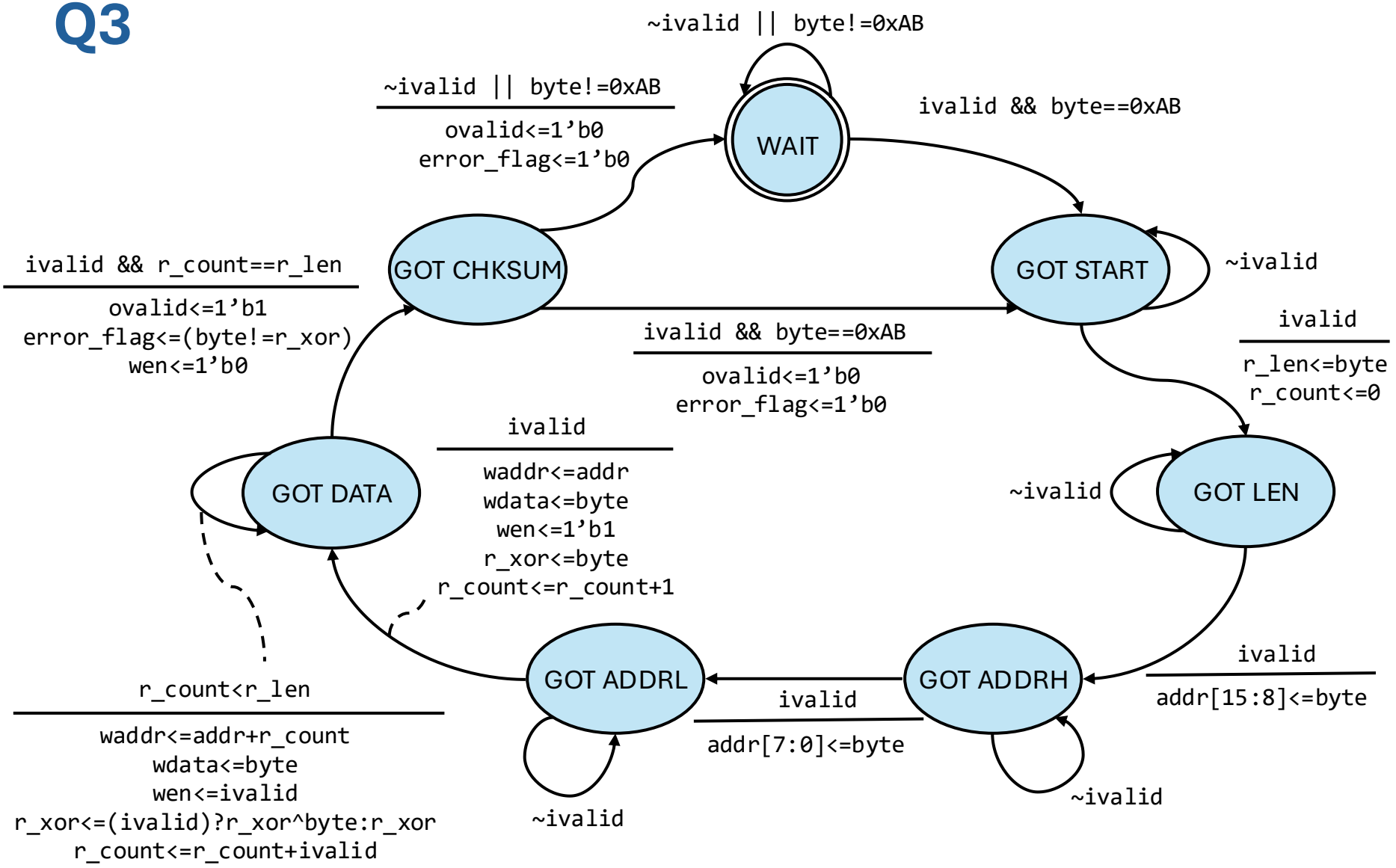
- START: Special code 0xAB indicates the beginning of a new message.
- LEN: The number of data bytes in the message.
- ADDR_H: The higher significance byte of a 16-bit address where the first data byte of the message will be stored.
- ADDR_L: The lower significance byte of a 16-bit address where the first data byte of the message will be stored.
- DATA_0, …, DATA_n: Data bytes of the message.
- CHECKSUM: The expected result of XORing all data bytes (this will not be stored in memory).

Q3

The FSM has the following interface:

- `input [7:0] byte`: Received byte
- `input ivalid`: A flag indicating that the input byte is valid
- `output [7:0] wdata`: Data to write to the receiver memory
- `output [15:0] waddr`: Address at which `wdata` will be written
- `output wen`: Enable signal for writing to the receiver memory
- `output ovalid`: A flag indicating that a complete message is written to memory
- `output error_flag`: A flag indicating whether the written message has a checksum error or not. It is set to 1'b1 if the result of XORing all the received bytes does not match the CHECKSUM value (i.e., the last byte of the message)

Q3


~ivalid || byte!=0xAB
WAIT
~ivalid || byte!=0xAB
ovalid<=1'b0
error_flag<=1'b0
ivalid && byte==0xAB
GOT START
~ivalid
ivalid
r_len<=byte
r_count<=0
GOT LEN
~ivalid
ivalid
addr[15:8]<=byte
GOT ADDRH
ivalid
addr[7:0]<=byte
~ivalid
GOT ADDRL
r_count<r_len
waddr<=addr+r_count
wdata<=byte
wen<=ivalid
r_xor<=(ivalid)?r_xor^byte:r_xor
r_count<=r_count+ivalid
ivalid
waddr<=addr
wdata<=byte
wen<=1'b1
r_xor<=byte
r_count<=r_count+1
GOT DATA
ivalid && r_count==r_len
ovalid<=1'b1
error_flag<=(byte!=r_xor)
wen<=1'b0
GOT CHKSUM
ivalid && byte==0xAB
ovalid<=1'b0
error_flag<=1'b0