Testing

ENGR 210 / CSCI B441
“Digital Design”

Midterm Review

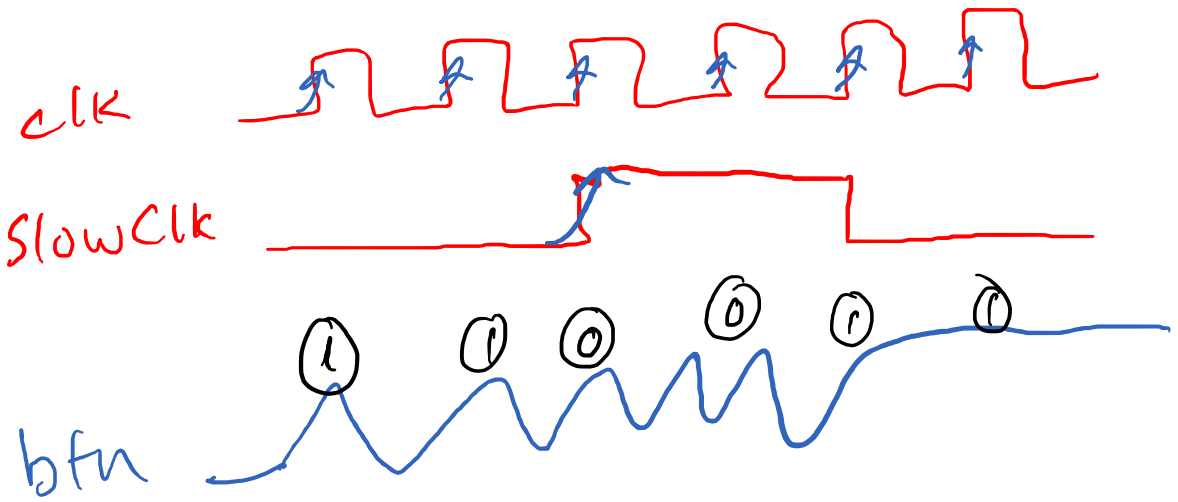
Andrew Lukefahr

Non-Exam Announcements

@todo: Add video requirements to P4 docs.

- P4 is out
  - Keeps “demo” requirement. (Pure-Remote students exempted)

- Upload Box Number to Canvas



P4 slowClk Questions vs. DeBounced

Exam Procedure

phone photos are fine → → convert to PDF

- Post Question 1 to Canvas on Friday
- Question 1 due back on Canvas **by Monday, 9.25am.** (graded)

- Post Questions 2+ to Canvas on Monday, 9.25am.
- Questions 2+ due back **on Monday, 10.45am.** ← strict

download PDF, annotate, & upload PDF

→ writing on blank paper. is fine

Exam Procedure

- Open Book
- Open Notes
- Open PDFs
- Open Lecture Videos

- Please no broader internet!

→ I will be on this Zoom for Exam.

"Honor Pledge"

- I'm going to ask everyone to sign the following pledge on the exam.

I have neither **given** nor **received** unauthorized aid on this examination, nor have I concealed any knowledge of unauthorized aid by others.

Signature: _____

- If you don't to sign it, I won't grade it.

General Topics

- Truth Tables / Boolean Logic / Logic Gates
- Multiplexers / Demultiplexers
- ALU
- ~~RS~~, D Latch Circuits
- D Flip Flops
- State Machines
  - Moore Type
  - Mealy Type

Textbook Chapters Covered

- Section 1.2
- Section 2.2 (skip 2.2.2)
- Section 2.3
- Section 2.4
- Section 3.1
- Section 3.2
- Section 4.1
- Section 4.2
- Section 4.3

SystemVerilog on Exam (?)

Testbenches on Exam!

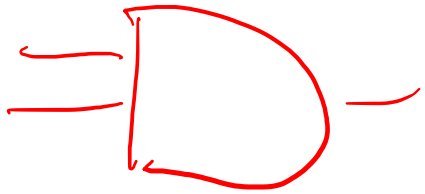
- Oh, yes!
- Big part of the exam is writing code.

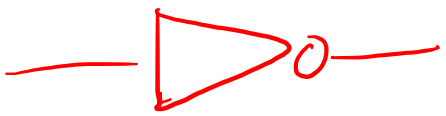
- Don't care about minor syntax errors
  - Missing commas / semi-colons (`;`)

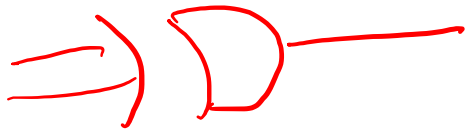
- Minor point deductions for:
  - Blocking vs. Non-blocking operators = vs <=
  - `logic` vs. `wire`
  - Missing delays (`#`)

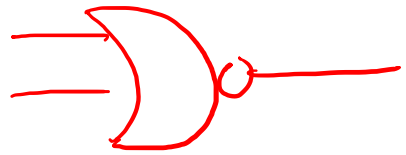
- Major point deductions for logical errors!

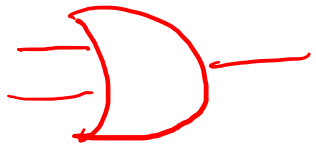
Logic Gates











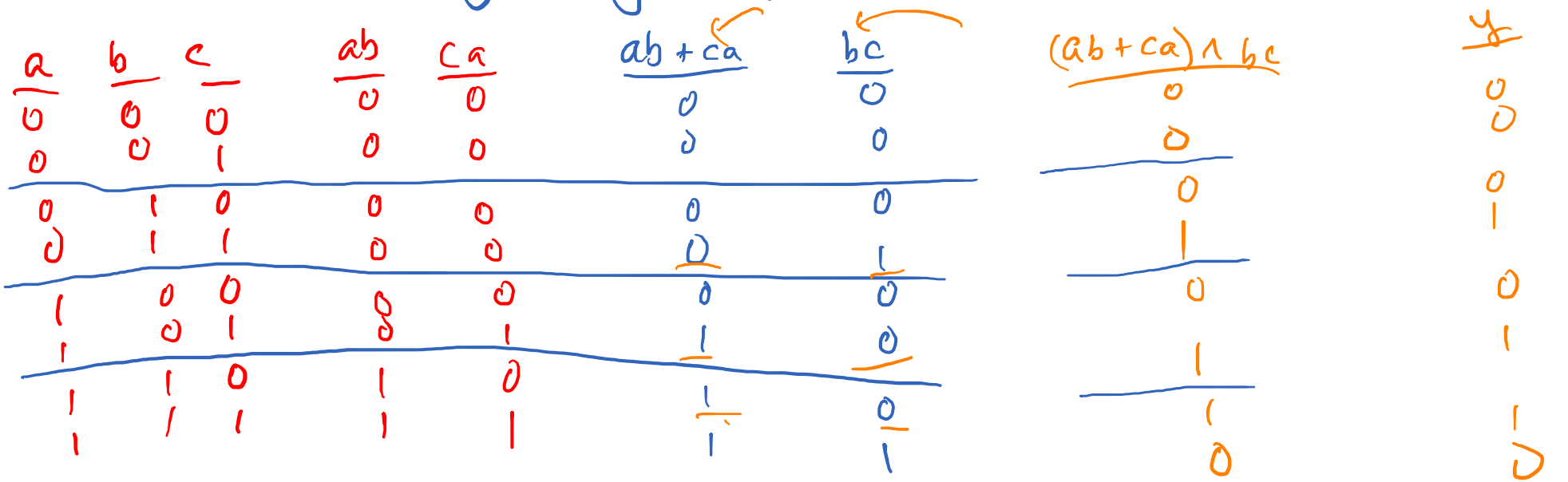
Boolean Equation -> Truth Table

XOR	
A B	X
0 0	0
0 1	1
1 0	1
1 1	0

Write the truth table for the following Boolean equation:

$$y = (ab + ca) \oplus bc$$

```
assign y = (a&b | c&a) ^ b&c;
```



Truth Table -> Boolean Equation

A	B	C	D	X	Y
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	0	0
0	1	1	0	0	0
0	1	1	1	1	0
1	0	0	0	0	0
1	0	0	1	0	1
1	0	1	0	0	0
1	0	1	1	1	0
1	1	0	0	1	0
1	1	0	1	1	0
1	1	1	0	1	1
1	1	1	1	1	0

$y_0 =$

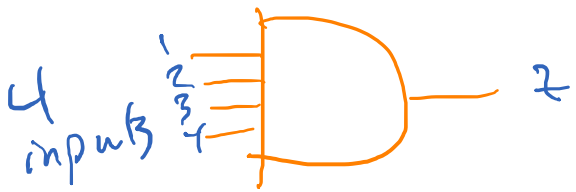
y_1

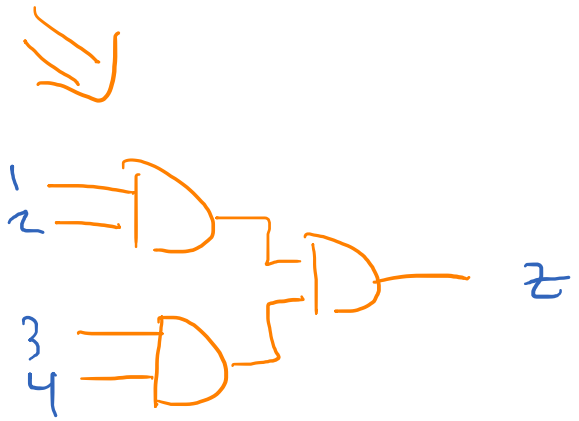
y_2

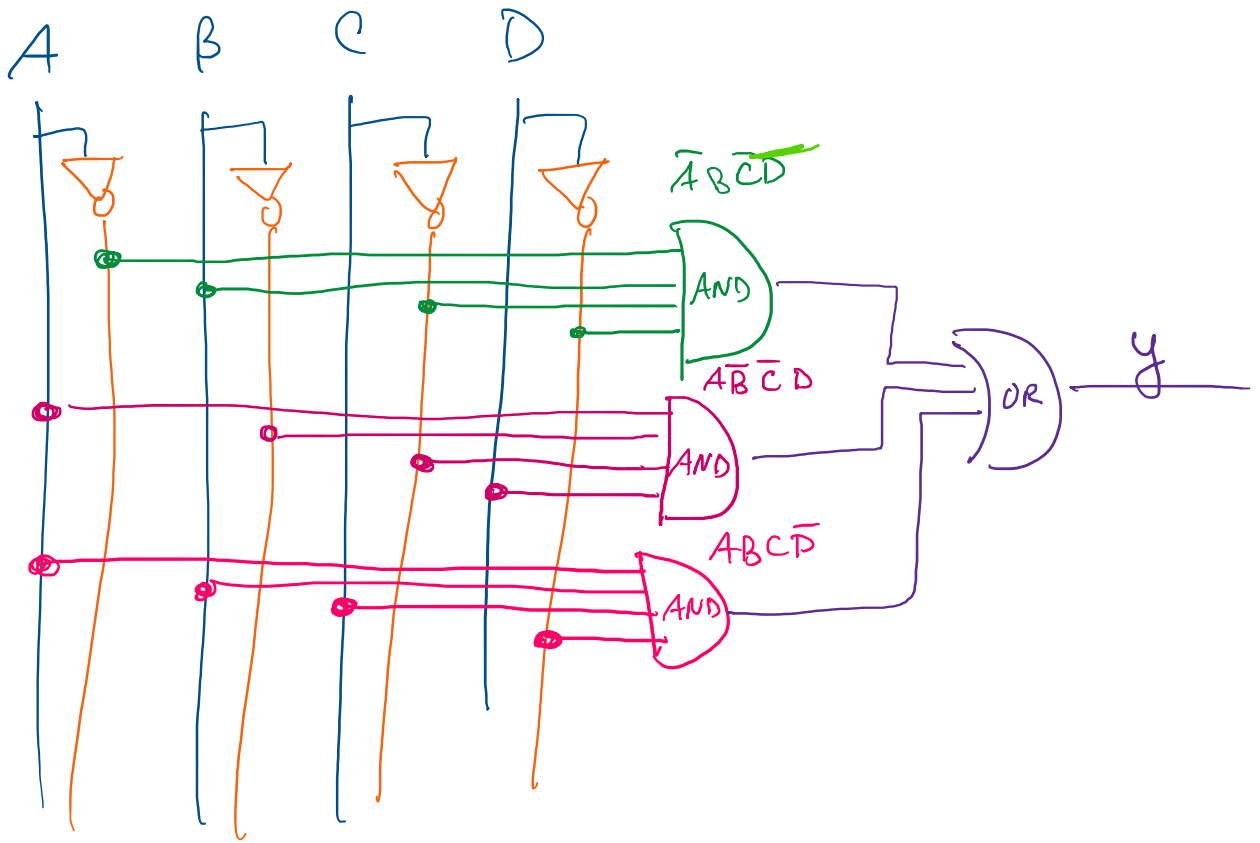
$y = (\bar{a} \cdot b \cdot \bar{c} \cdot d \; | \; a \cdot \bar{b} \cdot \bar{c} \cdot d \; | \; a \cdot b \cdot c \cdot \bar{d})$

* We need to minimize for exam.

$y = \bar{A}B\bar{C}\bar{D} + A\bar{B}\bar{C}D + ABC\bar{D}$

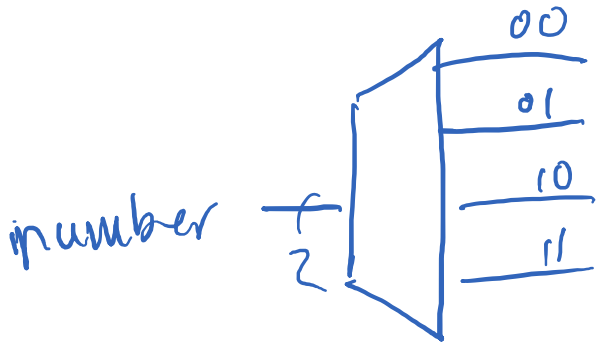


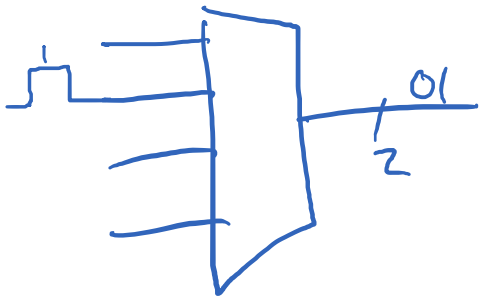




4-2 Encoder

- An **Encoder** is a combinational circuit that performs the reverse operation of Decoder. It has maximum of 2^n input lines and ‘n’ output lines. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes 2^n input lines with ‘n’ bits. It is optional to represent the enable signal in encoders.





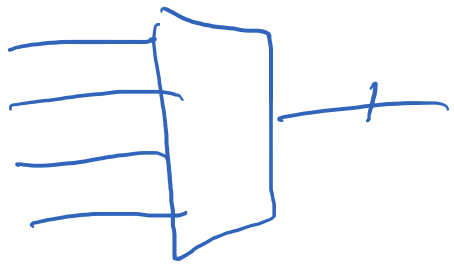
4-2 Encoder

- Please implement the following Verilog module of an 4-2 encoder. The valid signal is 0 if all wires in x are 0, and 1 if any wires in x are 1. You may assume a maximum of one of the wires in x will be 1 at a time.

```
module encoder (
    input [3:0] x,
    output [1:0] y,
    output valid
);
```

x[0] ⇒ y == 00

x[3] ⇒ y == 11



priority

```
module encoder (
        input [3:0] x,
        output [1:0] y,
        output valid
);
```

logic
logic

```
assign y = (x[0] ? 2'h0 :
           (x[1] ? 2'h1 :
           (x[2] ? 2'h2 :
            x[3] ? 2'h3)));
```

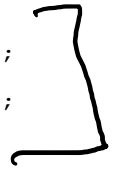
```
assign valid = x[0] | x[1] | x[2] | x[3];
assign valid = |x; // or trick
```

always_comb begin
valid = x[0] | x[1] | x[2] | x[3]; // also works!
y = 3'h0; // default
if (x[0] == 1'h1) y = 2'h0; // encode x[0] as y=0
else if (x[1] == 1'h1) y = 2'h1 // encode x[1] as y=1
else if (x[2] == 1'h1) y = 2'h2;
else if (x[3] == 1'h1) y = 2'h3;
end
end module

make priority encoder

4-2 Encoder

```
module encoder (
              input [3:0] x,
              output [1:0] y,
              output valid
);
assign valid = x[0] | x[1] | x[2] | x[3]; // OR -> assign valid = |x;

-- AND ONE OF EITHER --
logic [1:0] yR; assign y = yR;
always_comb begin
              case(x)
              4'h1: yR = 2'h0;
              4'h2: yR = 2'h1;
              4'h4: yR = 2'h2;
              4'h8: yR = 2'h3;
endcase end
-- OR --
assign y[0] = x[3] | x[1];
assign y[1] = x[3] | x[0];
endmodule
```



4-2 Priority Encoder

- A “Priority” encoder provides a solution if two or more of the input wires are 1 at the same time. It “Prioritizes” the lower-order wires and ignores the higher-order wires. For example, assume both x[0] and x[2] are 1 at the same time. As 0 < 2, a priority encoder encodes the input x[0] and ignores x[2]. Similarly, if x[2] and x[3] are 1 at the same time, 2 < 3, and thus it encodes x[2] and ignores x[3].
- Please implement the following Verilog module of an 4-to-2 priority encoder. *Multiple wires in x will be true at the same time.* You may assume a working encoder module as specified above if desired.

Valid stays the same.

```
module priority_encoder (
      input [3:0] x,
      output [1:0] y,
      output valid
);
```

```
module priority_encoder (
      input [3:0] x,
      output [1:0] y,
      output valid
);




endmodule
```

```
module prio_encoder (
    input   [3:0] x,
    output  [1:0] y,
    output        valid
);

assign valid = x[0] | x[1] | x[2] | x[3]; // or |x

logic [1:0] yR;
assign y = yR;
always_comb begin
    if (x[0]) yR = 2'h0;
    else if (x[1]) yR = 2'h1;
    else if (x[2]) yR = 2'h2;
    else if (x[3]) yR = 2'h3;
end

endmodule
```

Priority Encoder Testbench

```
module priority_encoder_tb();
logic [3:0] x;
wire [1:0] y;
wire valid;
priority_encoder pe0(.x(x), .y(y), .valid(valid) );



endmodule
```

```
module priority_encoder_tb();
logic [3:0] x;
wire [1:0] y;
wire valid;
priority_encoder pe0(.x(x), .y(y), .valid(valid) );
```

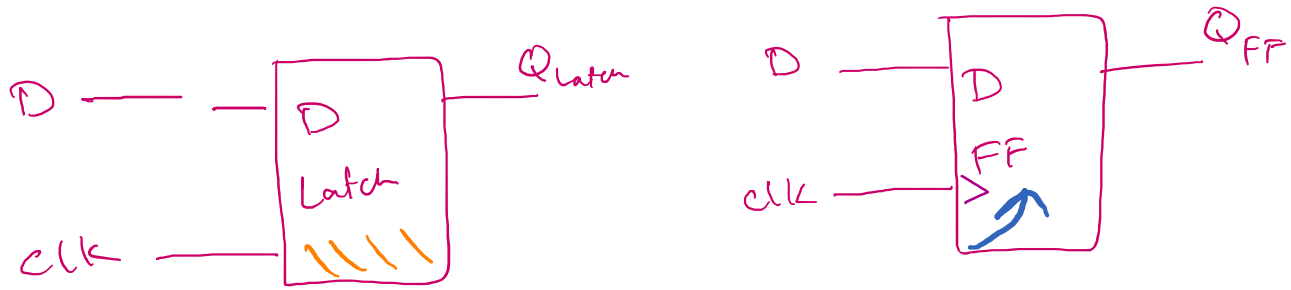
4 bits, 2^4 input values = 16 input values

```
initial begin
    for(int i=0; i<16; ++i) begin
        x = i[3:0];
        #1
        if(x[0]) assert (y==2'h0) else $fatal(1, "bad y0");
        else if(x[1]) assert (y==2'h1) else $fatal(1, "bad y1");
        else if(x[2] ...
        else if(x[3] ...
        if(x==4'h0) assert (valid==0) else $fatal(...)
        else        assert (valid==1) else $fatal(...)
    end
    $display("@@@Passed");
end
endmodule
```

```
module prior_encoder_tb();
logic [3:0] x;
wire [1:0] y;
wire valid;
prio_encoder pe0(.x(x), .y(y), .valid(valid) );

initial begin
    for(int i = 0; i < 16; i++) begin
        x=i;
        #1;
        if (x == 4'h0)
            assert(valid == 1'h0) else $fatal(1, "ERR");
        else
            assert(valid == 1'h1) else $fatal(1, "ERR");

        if (x[0] == 1)
            assert(y==2'h0) else $fatal(1, "ERR");
        else if (x[1] == 1)
            assert(y==2'h1) else $fatal(1, "ERR");
        else if (x[2] == 1)
            assert(y==2'h2) else $fatal(1, "ERR");
        else if (x[3] == 1)
            assert(y==2'h3) else $fatal(1, "ERR");
    end
    $display("@@@Passed");
    $finish;
 end
endmodule
```

See next slide

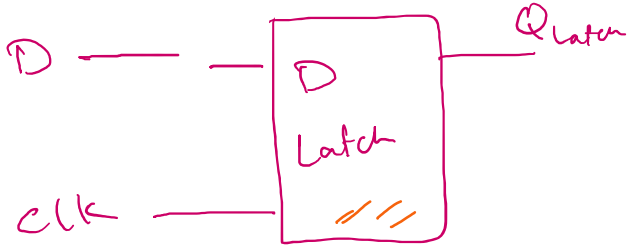
Timing Diagram

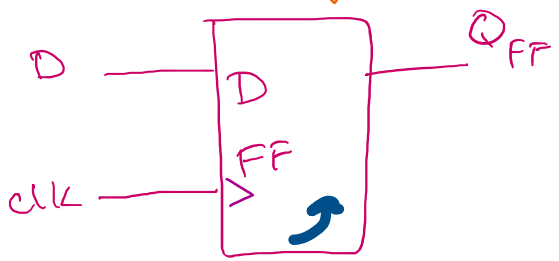


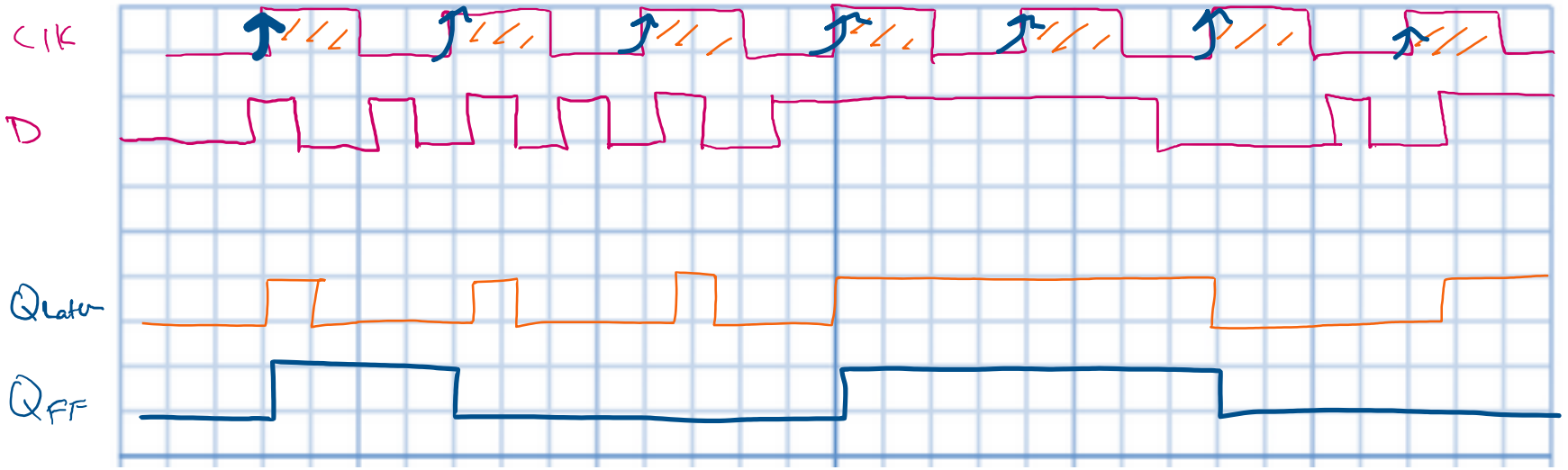
Timing Diagram

output follows input when clk == 1

ff: output matches input when clk = ↑ rising







4 cycle countdown timer

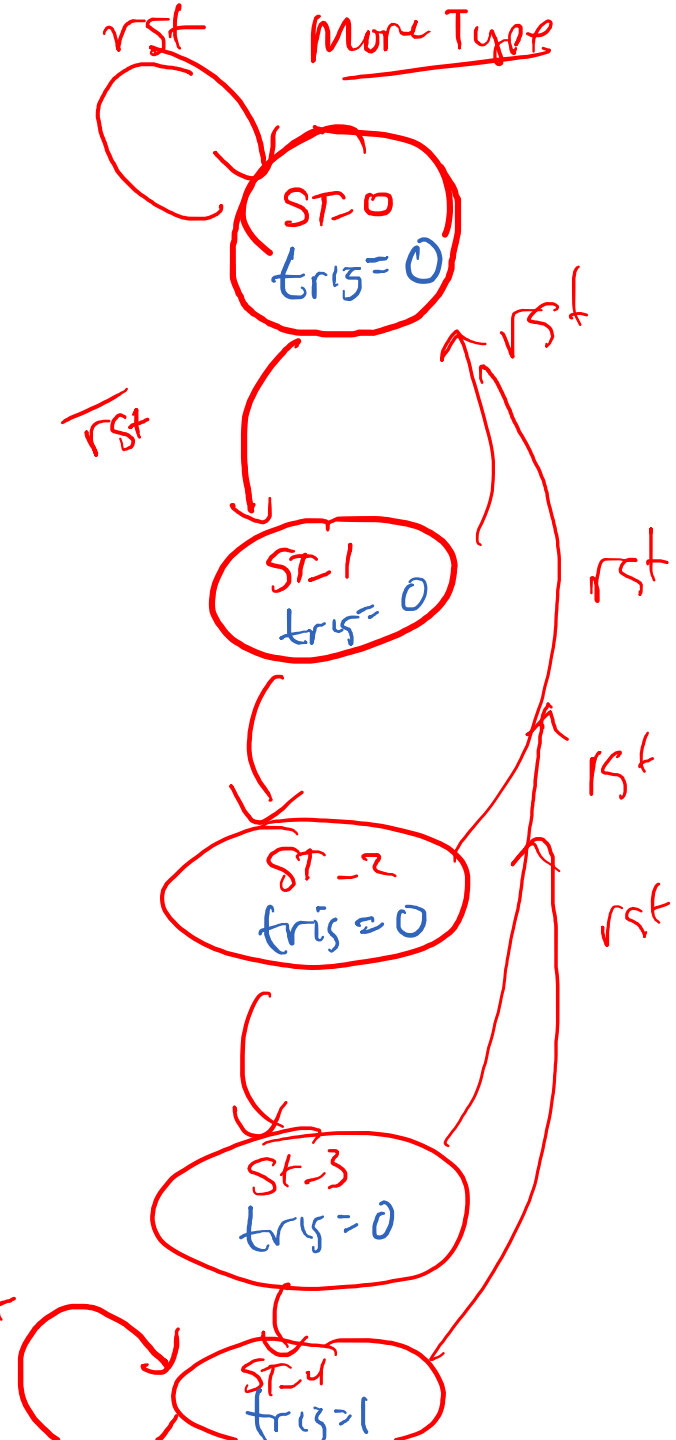
- Write the Verilog module for a countdown timer
  - `Input:  rst`
    - `When 1, should reset the timer to 4`
  - `Input: clk`
    - `Should advance the state every positive edge`
  - `Output:  trigger`
    - `Should be 0 at reset, then flip to 1 4 cycles after reset is lowered.  Should stay 1 until rst==1 again.`

```
module countdown_timer
      input clk, rst,
    → output trigger
);
```

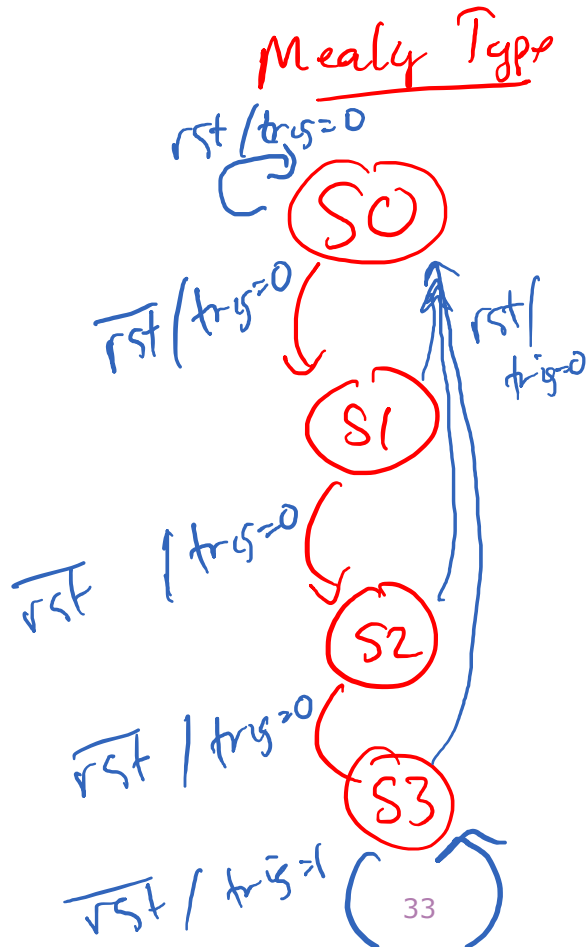
```
enum {ST_0, ST_1, ST_2, ST_3} state, nextState;

always_ff @(posedge clk) begin
    if (rst) state <= ST_0;
    else     state <= nextState;
end

always_comb
    trigger = 0;
    nextState = state;
    → advance State
    → set trigger

endmodule
```

See next slides

@todo: fix me!
→ Moore vs. Mealy



(This turned out to be wrong)



```
module countdown_mealy(
    input clk, rst,
    output logic trigger
);
enum { ST_0, ST_1, ST_2, ST_3, ST_4 } state, nextState;
always_ff @(posedge clk)  begin
    if (rst) state <= ST_0;
    else     state <= nextState;
end
always_comb begin
    nextState = state; // default
    trigger = 'h0; //default
    case(state)
        ST_0: begin
            nextState = ST_1; //rst case handled by always_ff
            if (1) trigger = 'h0; //Mealy-type's can have an if guard with an output.
        end
        ST_1: begin
            nextState = ST_2;
        end
        ST_2: begin
            nextState = ST_3;
        end
        ST_3: begin
            nextState = ST_4;
        end
        ST_4: begin
            nextState = ST_4;
            if (~rst) trigger = 'h1; //Mealy-type with input as guard
        end
    endcase
end
endmodule
```

```
module countdown_moore(
             input clk, rst,
             output logic trigger
);
enum { ST_0, ST_1, ST_2, ST_3, ST_4 } state, nextState;
always_ff @(posedge clk)  begin
    if (rst) state <= ST_0;
    else     state <= nextState;
end
always_comb begin
    nextState = state; // default
    trigger = 'h0; //default
    case(state)
        ST_0: begin
            nextState = ST_1; //rst case handled by always_ff
            trigger = 'h0; //Moore-type's never have an if() guard on outputs
        end
        ST_1: begin
            nextState = ST_2;
            trigger = 'h0;
        end
        ST_2: begin
            nextState = ST_3;
            trigger = 'h0;
        end
        ST_3: begin
            nextState = ST_4;
            trigger = 'h0;
        end
        ST_4: begin
            nextState = ST_4;
            trigger = 'h1;
        end
    endcase
end
endmodule
```

```
module countdown_tb();
logic clk, rst;
wire moore_trigger, mealy_trigger;
countdown_moore mo0( .clk, .rst, .trigger(moore_trigger) );
countdown_mealy me0( .clk, .rst, .trigger(mealy_trigger) );

always #5 clk = ~clk;

task check (
    input moT,
    input meT);
    assert(moore_trigger == moT) else $fatal(1, "Bad Moore");
    assert(mealy_trigger == meT) else $fatal(1, "Bad Mealy");
endtask

initial begin
    clk = 'h0; rst = 'h1;
    @(negedge clk);
    rst = 'h0;
    @(negedge clk);
    check('h0, 'h0);
    @(negedge clk);
    check('h0, 'h0);
    @(negedge clk);
    check('h0, 'h0);
    @(negedge clk);
    check('h1, 'h1);
    @(negedge clk);
    check('h1, 'h1);

    $display("@@@Passed");
    $finish;
end

endmodule
```