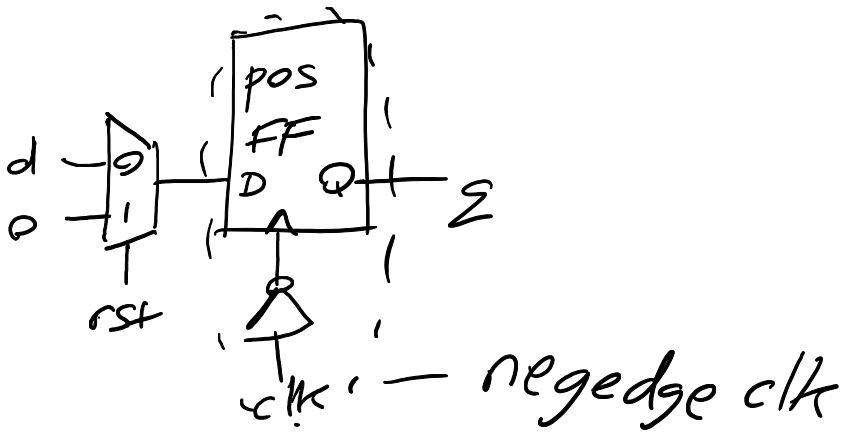
EECS 151/251A Midterm 1 Review Problems

Sp19 Midterm #6

6. Sequential Verilog [3pts]

Using the template below, write the Verilog for a module that implements a *negative* edge-triggered flip-flop with synchronous reset.

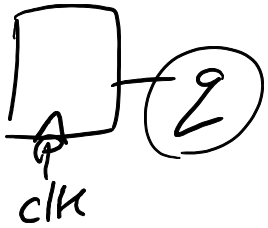
```
module FF_RST(q, d, clk, rst)
```



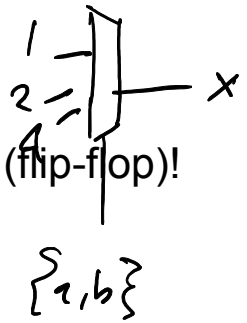

```
endmodule
```

Sp19 Midterm #6

```
module FF_RST(q, d, clk, rst);
  input d, clk, rst;
  output reg q;
  always @(negedge clk) begin
    q <= rst ? 0 : d;
  end
endmodule
```




```
reg [x:0] x;
always @* begin
  if(a)
    x = 1
  else if(b)
    x = 2
  else
    x = 4
end
```



- Inputs and outputs are wire-type nets by default
  - wire-type nets can only be assigned using an assign statement
- If you want to assign a net in an always block, it must be a reg-type net
- A reg-type net doesn't mean the signal it refers to comes from a register (flip-flop)!
  - Assignment inside always @(posedge/negedge clk) = register
  - Assignment inside always @(*) = wire
- A reg-type net used as register refers to the output of the register

Sp19 Midterm #7

7. Verilog Generator[3pts]

Assume we have previously defined a Verilog module DFF that describes a flip-flop. We instantiate that module to implement a 2-bit wide register below:

```
module Reg (Q, D, clk)
  input [1:0] D, clk;
  output [1:0] Q;
  DFF ff1( .q(Q[1]), .d(D[1]), .clk(clk)),
      ff0( .q(Q[0]), .d(D[0]), .clk(clk));
endmodule
```

Write a new version of this module that is a generator for an N-bit wide register using instances of DFF.

Sp19 Midterm #7

```
module Reg#(parameter w)(
  input [w-1:0] Q,
  output [w-1:0] D,
  input clk
); parameter w;
  genvar i;
  generate for (i=0; i < w; i = i + 1) begin
    DFF ff(.q(Q[i]), .d(D[i]), .clk(clk));
  end endgenerate
endmodule
```

integer

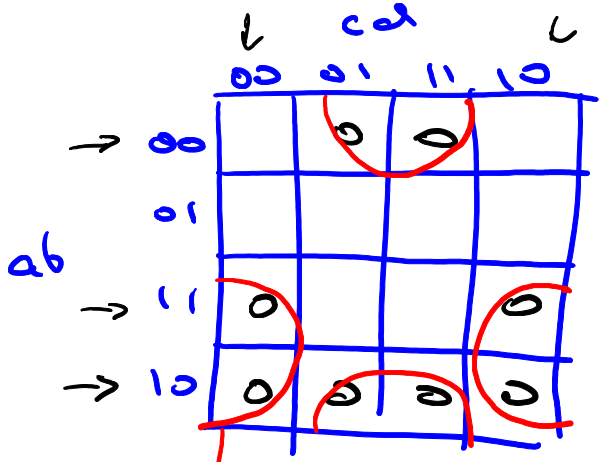
= 1

Reg #(.w(5)) r(.Q(), .D(), .clk());

ff_0
ff_1
:
ff_9

- Generate-for and generate-if are *macros*, they generate hardware instances
- Parameters are used to make certain aspects of a module *generic* and *reusable*
  - Parameters are by default Verilog integers

Sp19 Midterm #10/12



10. Boolean Algebra [4pts]

Using the laws of Boolean algebra, derive a simplified expression for $\overline{F}$ given $F = ab + bc$.

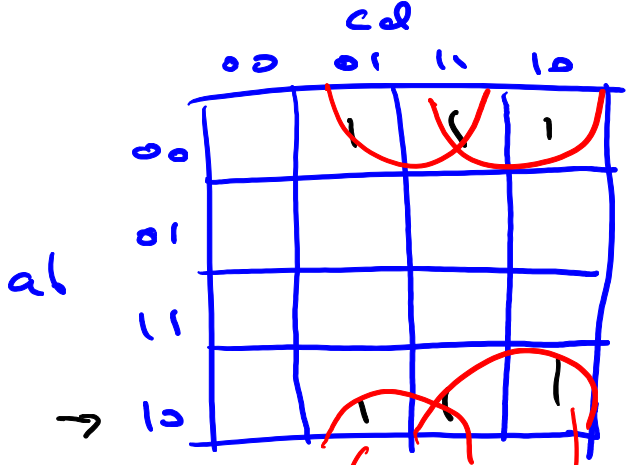
$\overline{F} = a\overline{d} + \overline{b}\,d$ $\quad F = (\overline{a} + d).(b + \overline{d})$

12. K-Maps [3pts]

Use a K-map to simplify the following expression and leave in products-of-sums form:
$F = (\overline{a} + b)(\overline{a} + \overline{b} + d)(a + b + \overline{d})$

$\overline{F} = a\overline{b} + ab\overline{d} + \overline{a}\,\overline{b}\,d$

Sp19 Final #7



7. Combinational Logic [7pts]

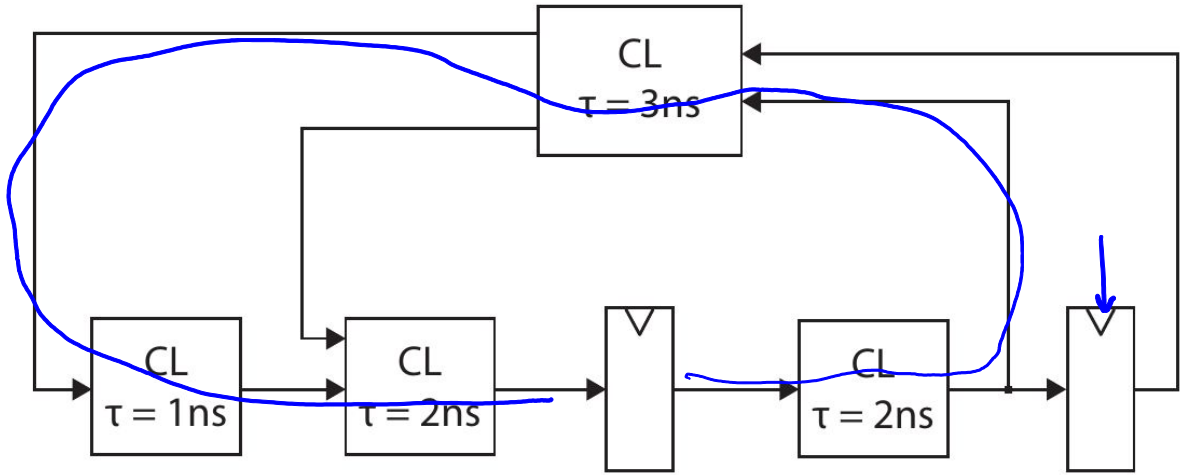
Consider the following Boolean function: $\bar{a}\ \bar{b}\ d + \bar{b}\ c\ \bar{d} + a\ \bar{b}\ d$.

(a) Use a K-map to simplfy.

$\bar{b} \cdot d + b \cdot \bar{c}$

Sp19 Midterm #21

21. Circuit Timing [3pts] For the circuit below what is the maximum clock frequency for correct operation?

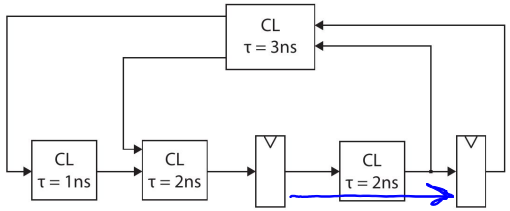


Flip-Flop/Reg Timing:

$$\tau_{clk \to q} = \tau_{setup} = \tau_{hold} = 1ns$$

Sp19 Midterm #21

21. Circuit Timing [3pts] For the circuit below what is the maximum clock frequency for correct operation?



Setup time check:

$$T_{clk} > t_{clk-q} + t_{setup} + t_{prop,max}$$

Hold time check:

$$t_{hold} < t_{clk-q} + t_{prop,min}$$

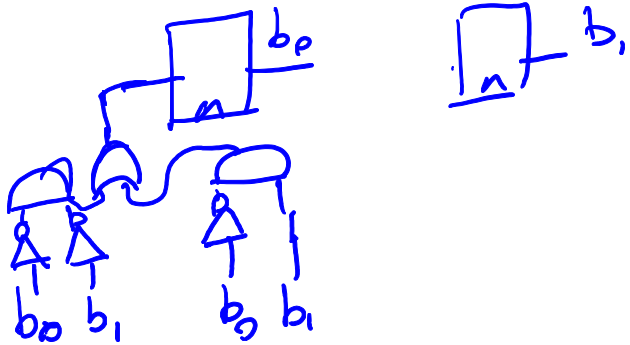
$$T_{clk} > 1ns + 1ns + max(2, 2+3+2, 2+3+1+2, 3+1+2, 3+2)$$

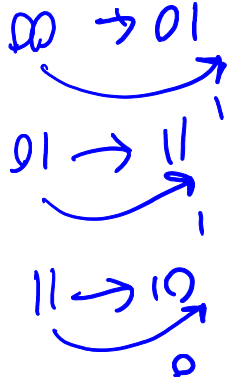
$$T_{clk} > 10ns$$

Sp19 Final #6

6. Logic Design [5pts]

Design a 2-bit gray-code counter with the following sequence: 00, 01, 11, 10. Draw the circuit using flip-flops and simple logic gates.





Fa15 MT2 #5

5. [12pts/12pts] MIPS microarchitecture.

Consider the design of the single-cycle MIPS processor, as we we discussed in class. Suppose you want to pipeline this design using *2 pipeline stages*. Functionality is divided into 2 pipeline stage as shown below:

stage 1	stage 2
instruction fetch/decode	ALU operation
regfile access	Dmem access

In the space below, explain what *hazards* will result from this pipelining (prior to any measures to remove the hazards). For each hazard, i) describe why it occurs, ii) write a short MIPS assembly language instruction sequence to demonstrate a case when the hazard would occur, and iii) describe what you can do to deal with the hazard. Write your answers in the space provided. (You might not need all answer spaces.)

Fa15 MT2 #5

(a) Hazard 1

i. **Control hazard occurs on every branch instruction because the branch comparison is done in stage 2 and the branch target address is needed in stage 1.**

ii. *Any insruction sequence with a branch.*

iii. **1) Stall the pipeline to allow the branch instruction to complete, 2) implement branch prediction.**

(b) Hazard 2

i. **Data hazard occurs on r-type instruction dependent on the immediately preceeding instruction, because ALU output is available sometime in stage 2 but needed at the beginning of stage 2.**

ii.

```
add $1, $2, $3
add $4, $5, $1
```

iii. **1) Stall the pipeline to allow r-type to store results back to register file, 2) Selectively feed output of ALU to ALU input register.**

Fa15 MT2 #5

(c) Hazard 3

i. **Data hazard on load from memory when an instruction is dependent on load instruction immediately preceeding. Result from memory load is available at end of stage 2 but needed at beginning of stage 2.**

ii.

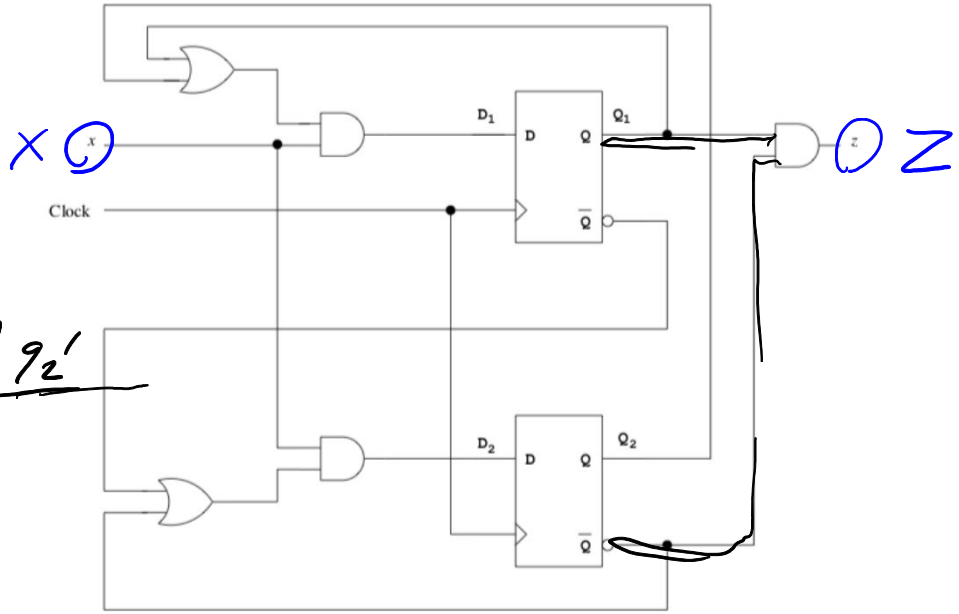
```
lw $1, 0($1)
add $4, $5, $1
```

iii. **1) Selectively direct output of memory to input register of ALU, 2) stall the load instruction.**

Fa18 MT1 #2

[PROBLEM 2] Finite State Machines (10 Pts)

Consider the following circuit with *x* the input and *z* the output.



$D_1 = x(Q_1 + Q_2)$

$D_2 = x(\bar{Q}_1 + \bar{Q}_2)$

$z = Q_1 Q_2$

input x	$Q_1 Q_2$	$Q_1' Q_2'$
0	0 0	0
0	1 0	
0	1 1	

- Mealy or Moore Machine?
- Write boolean expressions for FF inputs D1 and D2 and the system output z
- Derive the state transition table for the circuit

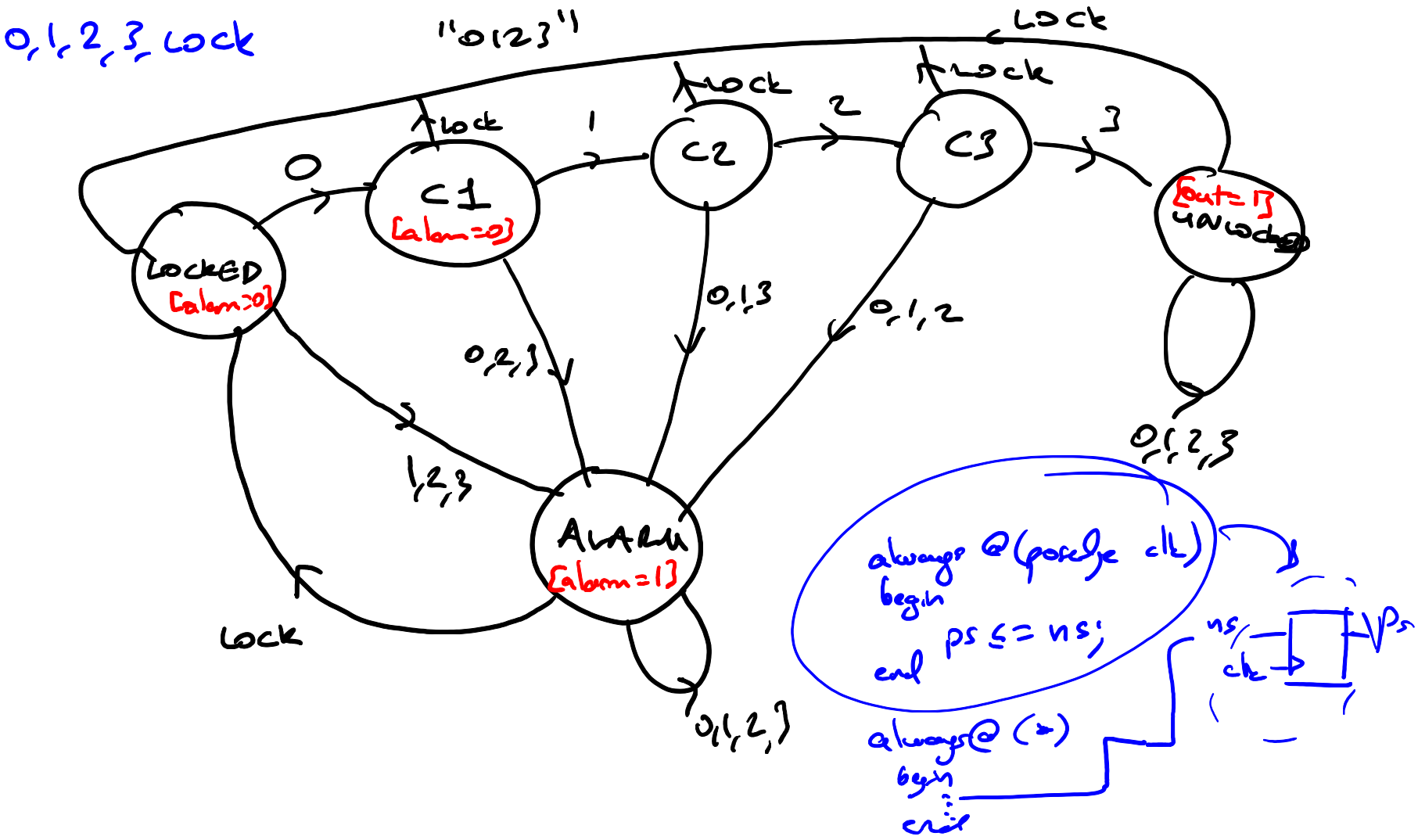
Fa17 MT1 #3

In this problem you will design a finite state machine for a safe with a keypad lock. The safe has a number pad with numbers between 0 and 3 and a LOCK button. Whenever the user presses a button, the corresponding signal goes high for a single clock cycle.

To unlock the safe, the user must put in the correct 4-digit code in the correct order. If the user enters an incorrect number, an alarm buzzes immediately and the user must press the LOCK button to return to the initial locked state before they are allowed to try again. (This is a very bad design for a safe, as you could quickly discover the combination by trial and error). Once all 4 numbers are input correctly, the safe unlocks by setting OUT to high and remains unlocked until the user presses the LOCK button. Additionally, the user may press the LOCK button at any time in order to reset the combination and return to the locked state.

Inputs:
0, 1, 2, 3, and LOCK – 1 bit; high for a single clock cycle when corresponding button is pressed
Outputs:
OUT – 1 bit; high whenever the safe is unlocked
ALARM – 1 bit; high whenever the alarm is sounding

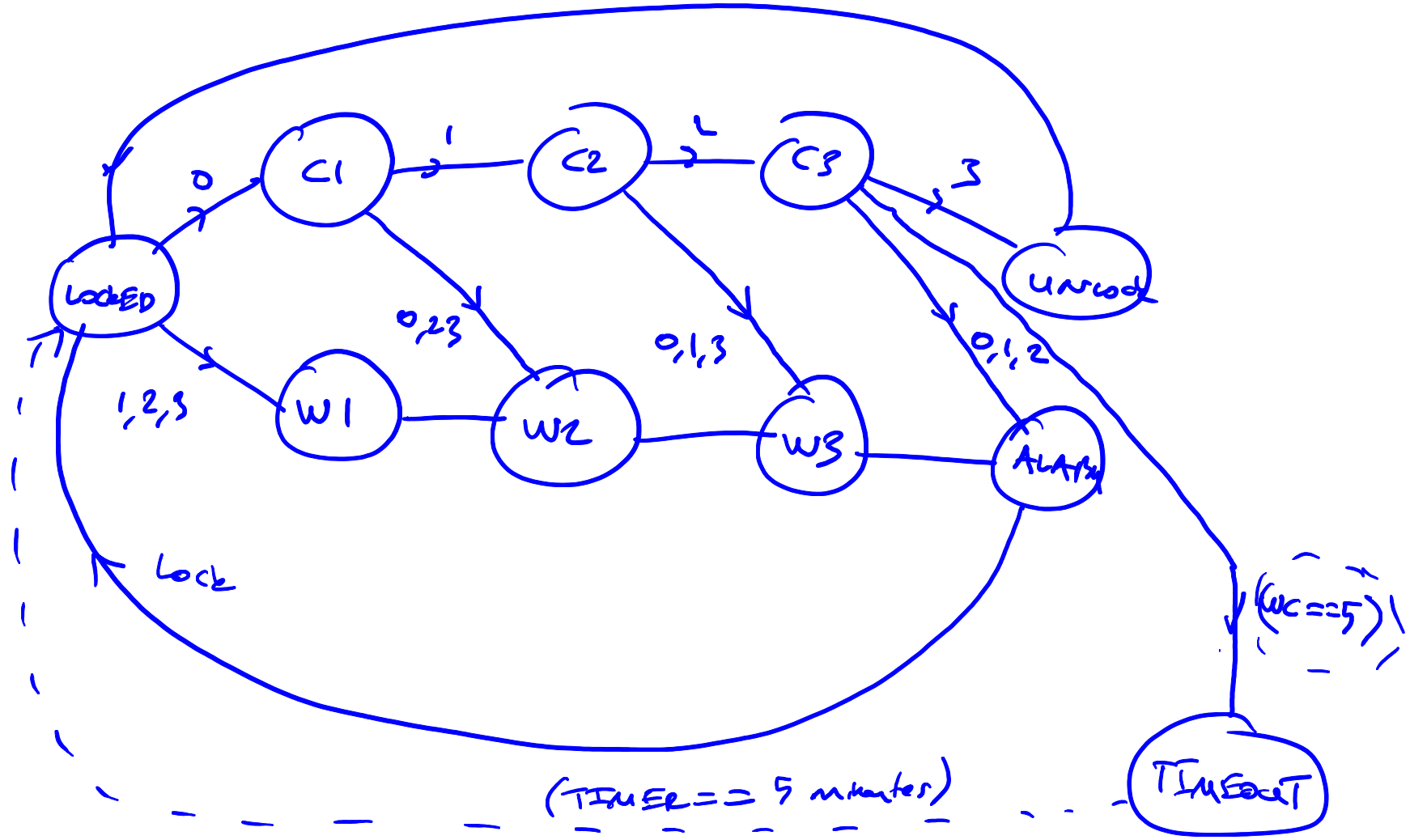
a) Draw an FSM that describes the functionality above assuming that the correct code is **0123**. Your implementation should be in the style of a Moore machine. (7 Pts)


0,1,2,3, lock
"0123"
lock
lock
lock
lock
0
1
2
3
C1
[alarm=0]
C2
C3
[out=1]
LOCKED
[alarm=0]
0,2,3
0,1,3
0,1,2
0,1,2,3
1,2,3
ALARM
[alarm=1]
lock
0,1,2,3
always @(posedge clk)
begin
ps <= ns;
end
always@(*)
begin
ns
clk
ps

Fa17 MT1 #3

c) Now we consider how to improve the security of our safe. In the previous implementation, a user could brute-force the combination by iterating through each number until they do not get an alarm. In the worst case, this would require only (4+4+4+4) = 16 attempts.
Instead of sounding the alarm immediately, we can wait until a full combination has been entered, similar to how your phone's lock works. The user would not know which of the inputs was wrong, only that the entire 4-digit combination is incorrect, increasing the worst case number of attempts to (4x4x4x4) = 256.

Draw a revised FSM based on this behavior. You are not allowed to use counters in your design. (5 Pts)

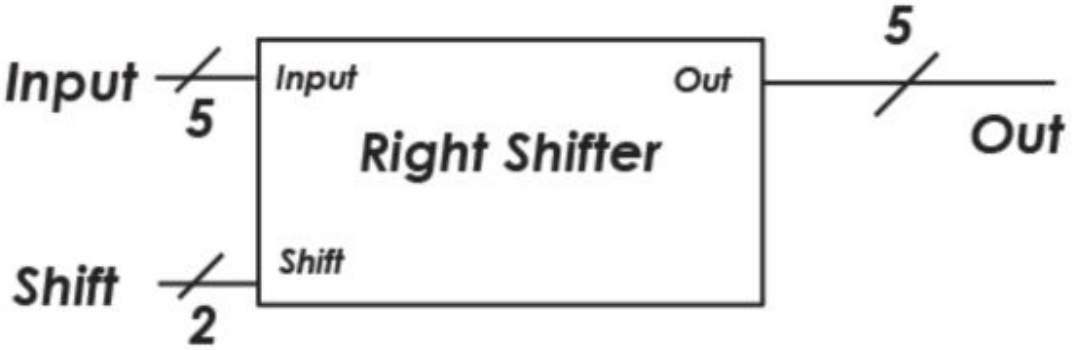

LOCKED
C1
C2
C3
UNLOCK
W1
W2
W3
ALARM
TIMEOUT
0
1
2
3
1,2,3
0,2,3
0,1,3
0,1,2
Lock
(wc==5)
(TIMER == 5 minutes)

Fa17 MT1 #3

d) We would like to go a step further and include a timeout feature which triggers whenever 5 consecutive, incorrect 4-digit codes are entered. In this state, the safe does not accept codes for 5 minutes and cannot be unlocked, even if the correct code is entered.
Describe the high-level digital constructs (e.g. register, timer, etc) you would use to implement this functionality – you are not required to write any Verilog. (3 Pts)

Sp16 MT1 #1

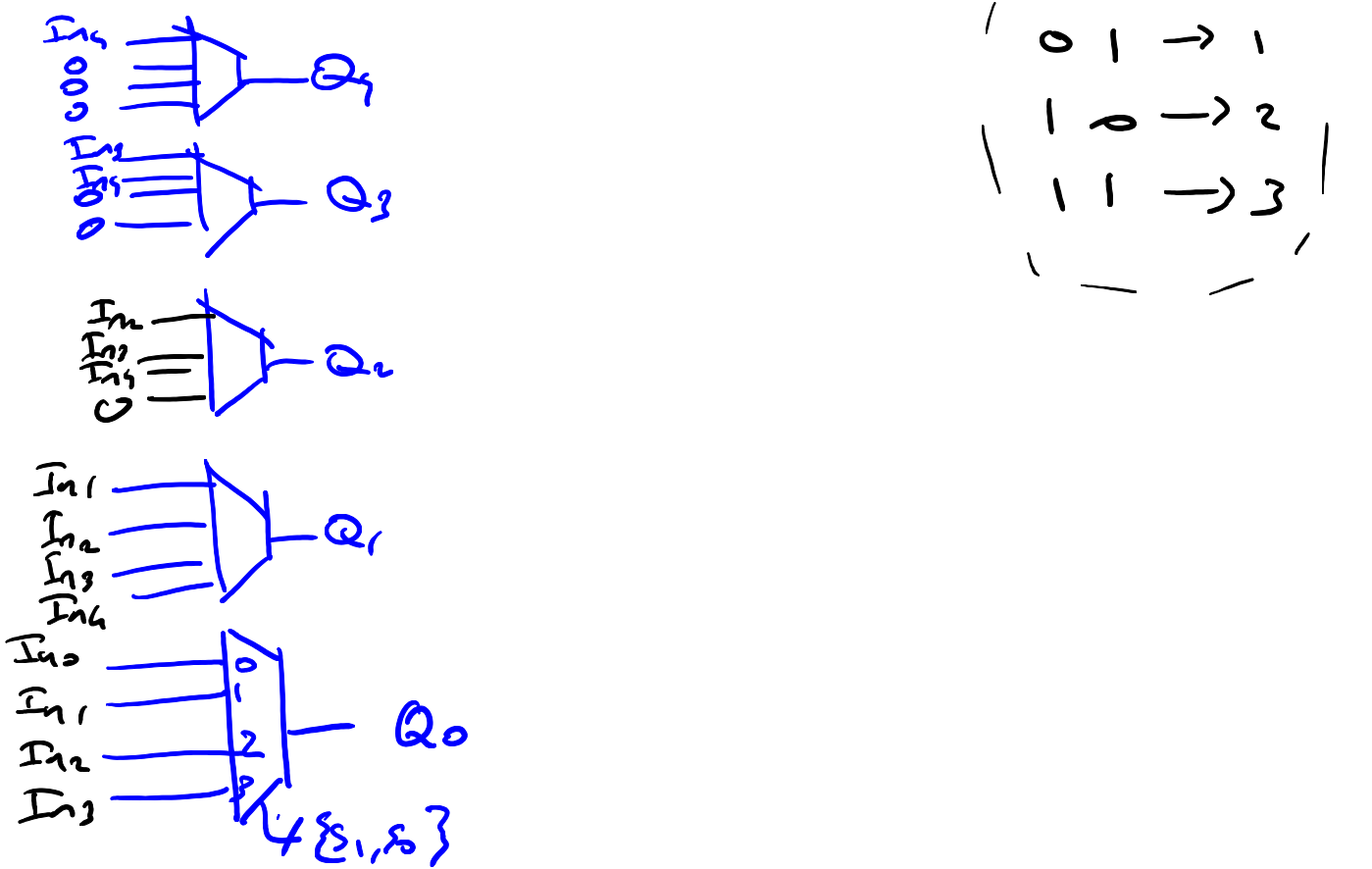
A block designed to *logically* shift its input to the right (from msb to lsb) is drawn below. Its inputs consist of a 5-bit number to be shifted (*Input*), and a two-bit number (*Shift*) indicating the number of bits the 5-bit number should be shifted to the right.



a.) Design the logical circuit that performs the shift function described above. You are allowed to use inverters, AND, OR, NOR, NAND gates, and muxes. **(6 points)**

In_4 In_3 In_2 In_1 In_0 Q_4 Q_3 Q_2 Q_1 Q_0

S_1	S_0	
0	0	→ 0
0	1	→ 1
1	0	→ 2
1	1	→ 3



Sp16 MT1 #1

b.) Suppose now that we need to design a circuit that performs an *arithmetic* right shift. You may assume that the input number is in 2's complement representation. Please complete the design using the same constraints as a). **(4 points)**

Zeros are replaced w. In4

Sp16 MT1 #1

c.) Next we will expand the circuit to support both logical and arithmetic shifts. A new 1-bit input is added to the block called "*Arithmetic*." If "*Arithmetic*" is high, then the shifter will perform a arithmetic right shift. If "*Arithmetic*" is low, then the block will perform a logical right shift. Design the logic for the new programmable shifter. Make sure that you keep the area to a mininum (that is, reuse if you can). **(5 points)**