University of California
College of Engineering
Department of Electrical Engineering
and Computer Sciences

J. Rabaey

G. Alexandrov, N. Narevsky, V. Iyer

MoWe 4-5:30pm

Mo, Oct. 2, 6:00-7:30pm

EECS 151/251A: FALL 17—MIDTERM 1

NAME	Last **SOLUTION** First

SID	

Problem 1 (15):
Problem 2 (15):
Problem 3 (20):

Total (50)	

[PROBLEM 1] Logic optimization and gate construction (15 Pts)

a) For the Truth Table below, write *Out* as a sum-of-products. In addition, construct the 4 variable K-map for this truth table, and derive **a simplified Boolean expression** for *Out*. (5 Pts)

A	B	C	D	**Out**
0	0	0	0	**1**
0	0	0	1	**1**
0	0	1	0	**0**
0	0	1	1	**1**
0	1	0	0	**1**
0	1	0	1	**0**
0	1	1	0	**0**
0	1	1	1	**0**
1	0	0	0	**1**
1	0	0	1	**1**
1	0	1	0	**0**
1	0	1	1	**1**
1	1	0	0	**1**
1	1	0	1	**0**
1	1	1	0	**0**
1	1	1	1	**0**

The un-simplified sum-of-products can be written directly:

$$Out = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}\,C\,\overline{D} + \overline{A}\,B\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}D + A\overline{B}C\,\overline{D} + AB\overline{C}$$

For the K-map: AB is on the vertical axis and CD is on the horizontal axis.

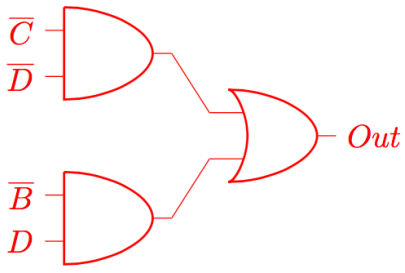
	00	01	11	10
00	1	1	1	0
01	1	0	0	0
11	1	0	0	0
10	1	1	1	0

We can directly write the simplified expression.

$$Out = \overline{C}\,\overline{D} + \overline{B}D$$

+2 Pts: correct sum-of-products unsimplified expression
+1 Pts: correct K-map drawing
+2 Pts: correct K-map maxterm circling and correct final simplified form

b) Draw the transformed function implemented with 2-input logic gates (inverter, AND, OR, NOR, etc.). You can use $\overline{A}$, $\overline{B}$, $\overline{C}$, and $\overline{D}$, as inputs to your circuit. (3 Pts)



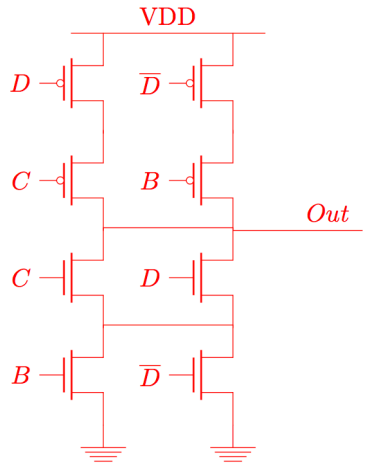
+3 Pts: correct translation of simplified equation from part a)

c) Construct a complex CMOS gate that implements the simplified function. Again feel free to use the inverted ABCD signals as inputs to your gate. (3 Pts)

We begin by constructing $\overline{Out}$.

$$\overline{Out} = \overline{\overline{C}\,\overline{D} + \overline{B}D}$$
$$\overline{Out} = (\overline{\overline{C}\,\overline{D}} \cdot (\overline{\overline{B}D})$$
$$\overline{Out} = (C + D) \cdot (B + \overline{D})$$

Then directly construct the pull-down network using $\overline{Out}$ and find the complementary pull-up network.



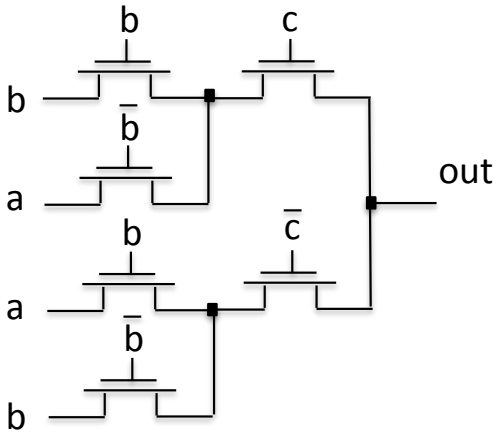
+1 Pts: correct equation for $\overline{Out}$ or correct assignment of inputs to complex gate
+1 Pts: correct pull-down network corresponding to $\overline{Out}$ or to equation in part a) if $\overline{Out}$ was incorrectly derived or not derived
+1 Pts: correct pull-up network, complementary to drawn pull-down network

d) An interesting way to implement any logic function is to realize it as *a memory module* with the inputs serving as the memory address. Describe how you imagine the logic function described by the truth table of part a) could be implemented that way (draw a simple diagram). For an extra, show how you could minimize the area of your circuit. (4 Pts)

+1 Pts: mention that one can use a LUT, a ROM, or any array of flip-flops/latches to implement the logic function
+2 Pts: specify the inputs into the memory block as an address in the form of {A,B,C,D}
+1 Pts: include a diagram that elaborates/clarifies the points you made
++1 Pts: mention that you can minimize area by simplifying the logic function and only use a 3-LUT or a 3-bit memory block

[PROBLEM 2] Logic Circuits (15 pts)

a) Derive the Truth Table of the logic circuit presented in Figure 1. Can you describe the logic function it performs? (2Pts)



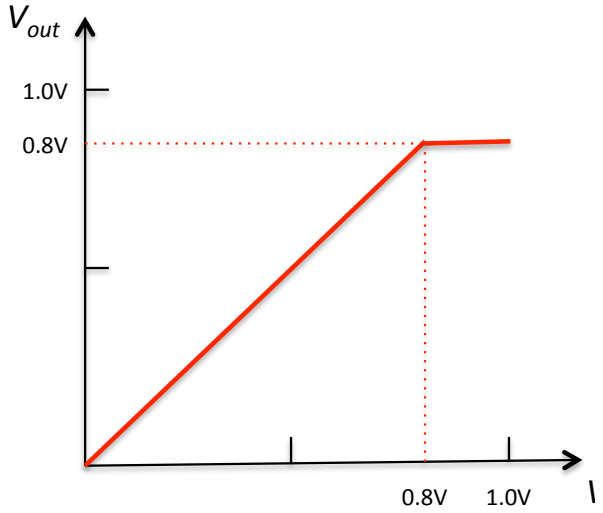
a	b	c	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Majority gate

+1 Pts: correct truth table
+1 Pts: majority gate, or similar description

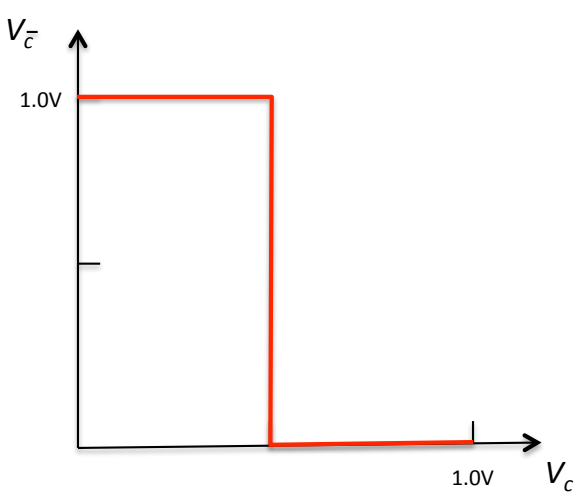
b) Would you describe the circuit as being *static*? Explain your answer. (1 Pt)
Yes – the output is always connected to either VDD or GND (via the inputs) and never to both at the same time
+1 Pts: reasonable explanation

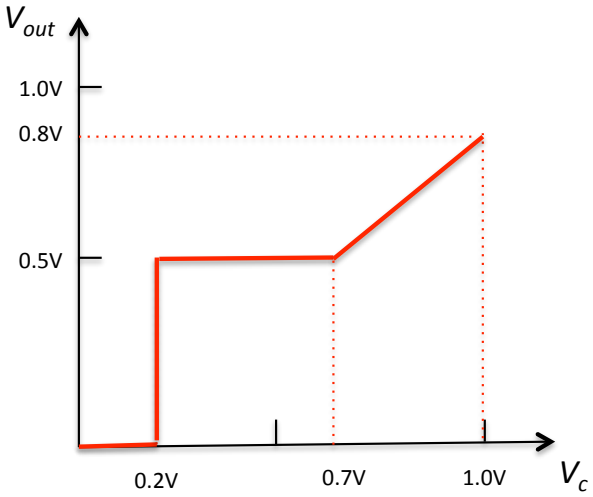
c) Draw the voltage transfer characteristic of V_{out} as a function of V_a assuming that *b* is equal to 1 (or is at VDD) and *c* is equal to 0 (is at GND). Assume that the transistors have a threshold of 0.2V and the supply voltage VDD is equal to 1.0V. All transistors can be considered as ideal switches with the same on-resistance (and infinite off-resistance). You may assume that the inverted signals $\overline{b}$ and $\overline{c}$ are generated from *b* and *c*, respectively, using ideal inverters with the switching threshold at VDD/2 (= 0.5V). (3 Pts)

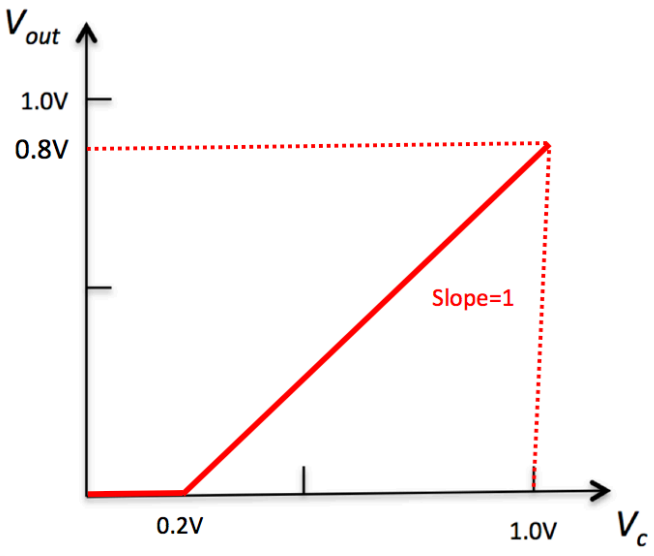


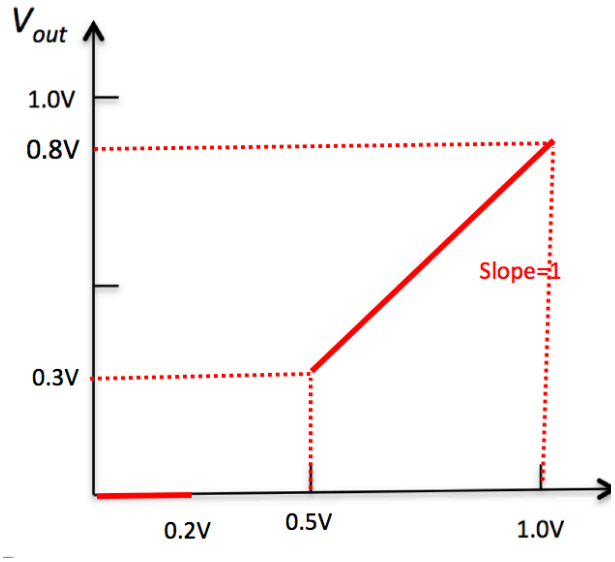
+1 Pts: slope of 1 for 0 < Va < 0.8
+1 Pts: Vo,max = 0.8V
+1 Pts: output goes flat after Va = 0.8V

d) Do the same for V_{out} as a function of V_c assuming that a and $\overline{b}$ are equal to 1 (or are at VDD). Again, assume that $\overline{c}$ is generated from c using an ideal inverter (with the switching threshold in the middle) It is worth drawing the VTC for $\overline{c}$ as well. (4 Pts)









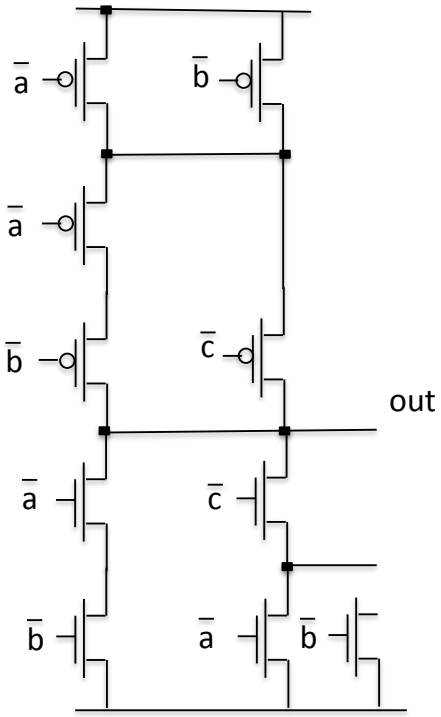
Note: We accepted any of the above three answers because the behavior of the circuit for 0.2 < Vc < 0.5 is not well defined
+1 Pts: output = 0 only at 0 < Vc < 0.2
+1 Pts: Vo,max = 0.8V
+1 Pts: slope of 1
+1 Pts: reasonable answer for 0.2 < Vc < 0.5 (see plots above)

e) Does this realization have some problems that you may identify? If so, discuss and describe ways on how you would tackle the problem. (1 Pt)
+1 Pts: The output does not go rail-to-rail due to the threshold loss. A full transmission gate realization would help.

f) Realize the same function in a more traditional *complementary* style. **Minimize** the number of transistors used in your implementation, and draw the circuit diagram. HINT: It is ok to use the inverted $signals$ $\overline{a}$, $\overline{b}$ and $\overline{c}$ as primary inputs. (4 Pts)

$\overline{Out} = \bar{A} \cdot \bar{B} + \bar{C} \cdot (\bar{A} + \bar{B})$

$Out = \overline{\bar{A} \cdot \bar{B} + \bar{C} \cdot (\bar{A} + \bar{B})}$



+2 Pts: minimize expression and find out_bar
+1 Pts: 10 transistors in implementation
+1 Pts: gate implementation matches expression

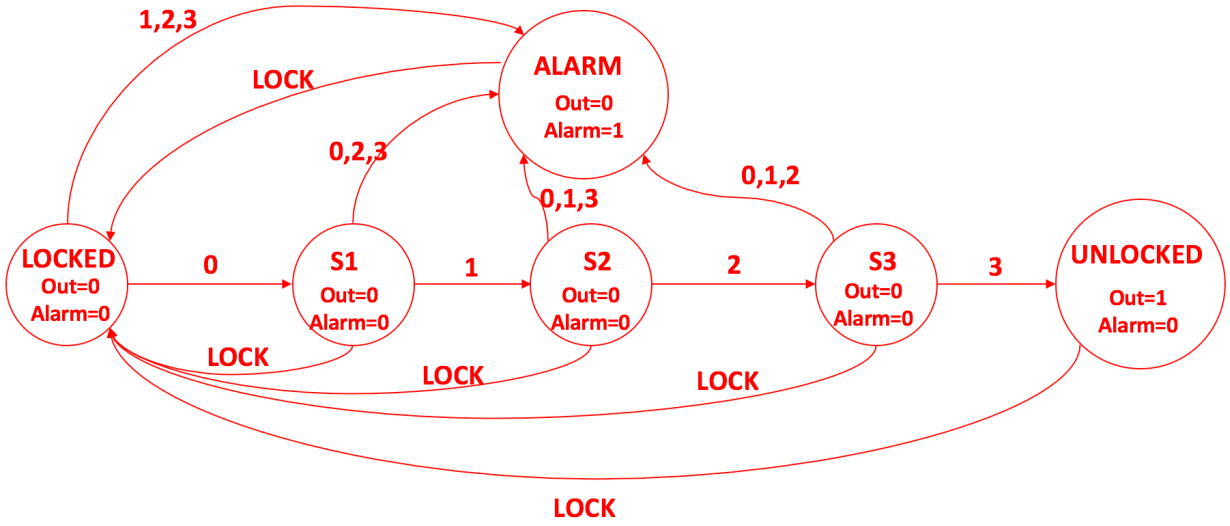
[PROBLEM 3] Finite State Machines (20 Pts)

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)



+1 Pts: out=1 in unlocked state
+1 Pts: alarm=1 in alarm state
+2 Pts: 3 intermediate states
+1 Pts: every state has LOCK transition back to LOCKED state
+1 Pts: correct transitions for input 0123
+1 Pts: correct transitions to alarm state

b) Write the Verilog that corresponds to the FSM you drew in (a). (5 Pts)

```
module fsm (
        input clk,
        input zero,
        input one,
        input two,
        input three,
        input lock,
        output alarm,
        output out
        );
```

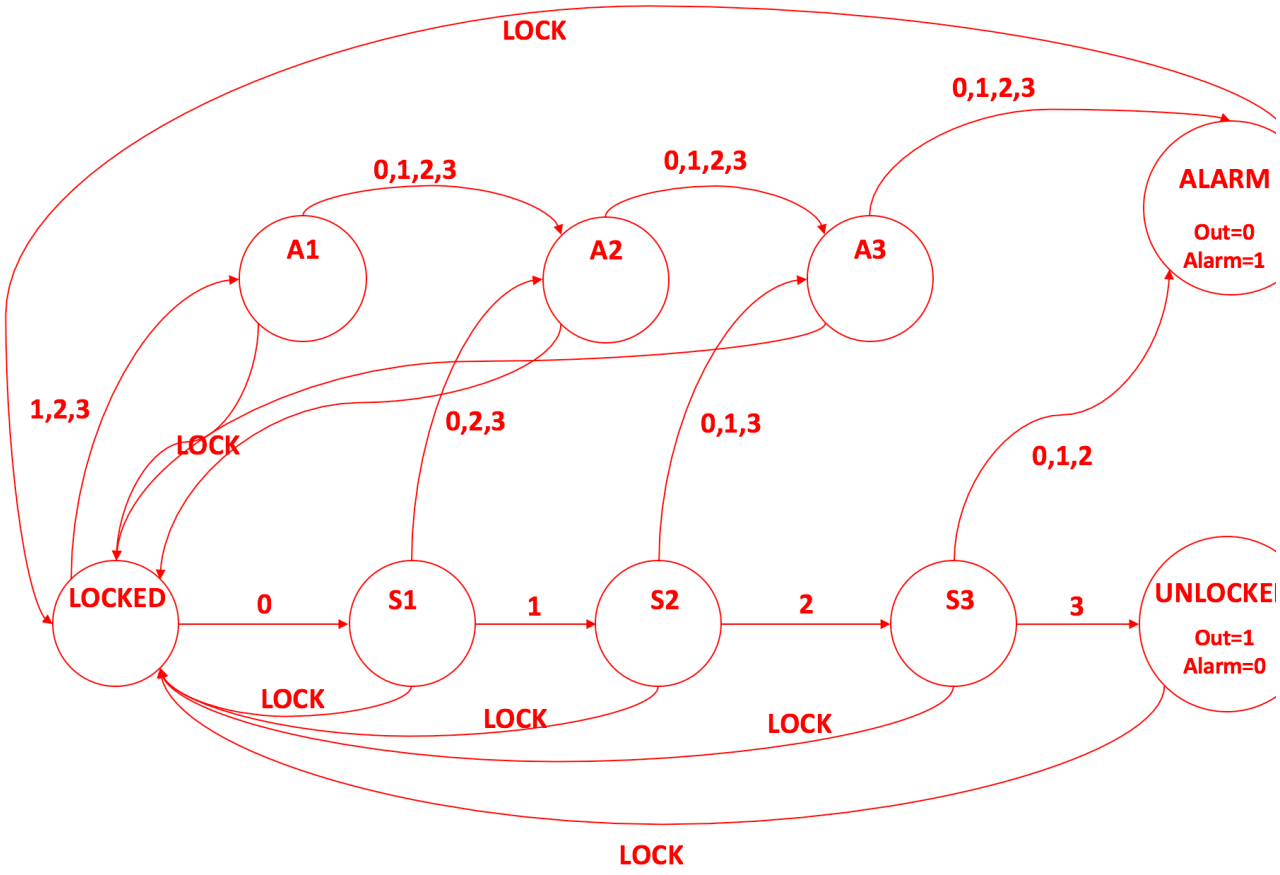
```
module fsm (
    input clk,
    input zero,
    input one,
    input two,
    input three,
    input lock,

    output alarm,
    output out
    );

    parameter   LOCKED  = 3'd0,
                S1      = 3'd1,
                S2      = 3'd2,
                S3      = 3'd3,
                UNLOCKED= 3'd4,
                ALARM   = 3'd5;

    reg [2:0] CS, NS;

    assign alarm    = (CS == ALARM);
    assign out      = (CS == UNLOCKED);

    always @(posedge clk) begin
        CS <= NS;
    end

    always @ (*) begin
        NS = CS; // default to staying in the same state
        case (CS)
            LOCKED: begin
                if (zero) NS = S1;
                else if (one || two || three) NS = ALARM;
            end
            S1: begin
                if (one) NS = S2;
                else if (zero || two || three) NS = ALARM;
                else if (lock) NS = LOCKED;
            end
            S2: begin
                if (two) NS = S3;
                else if (zero || one || three) NS = ALARM;
                else if (lock) NS = LOCKED;
            end
            S3: begin
                if (three) NS = UNLOCKED;
                else if (zero || one || two) NS = ALARM;
                else if (lock) NS = LOCKED;
            end
            UNLOCKED, ALARM: begin
                if (lock) NS = LOCKED;
            end
            default: NS = LOCKED;
        endcase
    end
endmodule
```

+1 Pts: 3 bit register for storing state
+ 1 Pts: assigning out, alarm correctly
+1 Pts: always@(posedge clk) block
+1 Pts: “case-like” statement
+1 Pts: correct logic for state transitions

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)

If an output is not defined inside a block, it is low in that state.



+3 Pts: correct intermediate alarm states
+1 Pts: intermediate alarm states lock transitions
+1 Pts: correct transitions to new states

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)
Include a register to keep track of the number of incorrect attempts. Reset this to 0 whenever the correct code is entered, or when the lockout period has ended.
Whenever this register reaches 5, enter a LOCKOUT state where you enable a counter that counts down from 5 minutes. While the lockout counter is not 0, the FSM accepts no inputs and only transitions out of the LOCKOUT state back to LOCKED state once the counter finishes.
+1 Pts: register to keep track of # incorrect attempts
+1 Pts: register/counter/timer for 5 minute countdown
+1 Pts: reasonable description of how these function with respect to each other