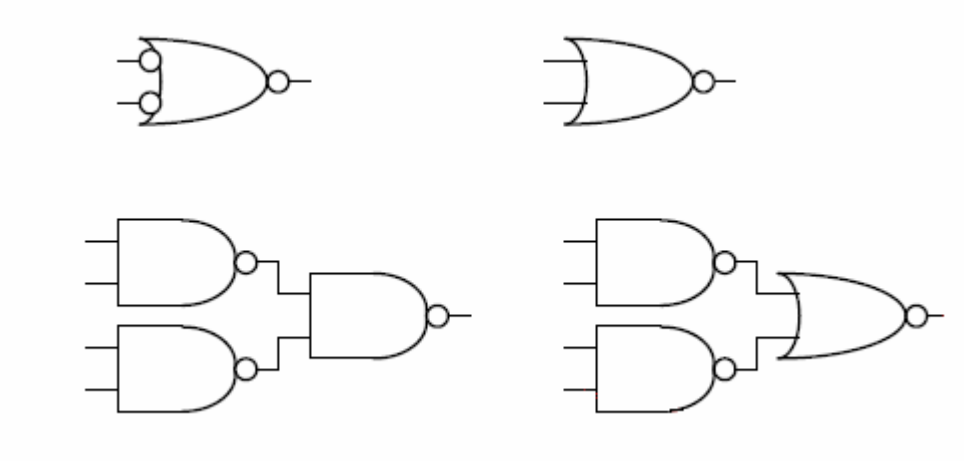
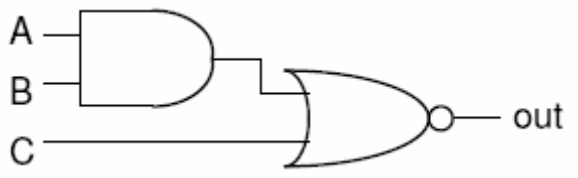
Problem 1 (15 points)

1.a. Circle the gate-level circuits that DO NOT implement a Boolean AND function.



1.b. Show that a 2-to-1 MUX is universal (i.e. that any Boolean expression can be implemented with a collection of 2-to-1 muliplexers).

1.c. Write a Verilog module that implements the following combinational logic.




```
module (A, B, C, out);
     input     A, B, C;
     output    out;




endmodule
```

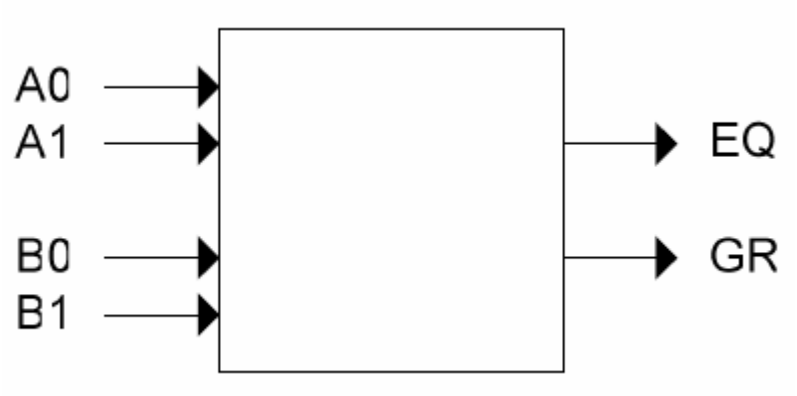
1.d. Write a Verilog module that implements a D flip-flop with reset.

```
module (D, Q, Clock, Reset);
     input     D, Clock, Reset;
     output    Q;









endmodule
```

Problem 2 (10 points)

Generate a truth table with appropriate don't-cares for the circuit shown below. It has two 2-bit unsigned inputs, A = $\{A_1,A_0\}$ and B = $\{B_1,B_0\}$, and two outputs, EQ and GR.

EQ is 1 if A = = B and 0 otherwise. GR is 0 when A > B and GR is 1 when A < B. Your solution should allow for an efficient implementation.



Problem 3 (10 points)

You are to implement the following Boolean operation over three inputs:
out = ~(*a* • *b*) • ~(*b* • *c*)

3.a. Implement this efficiently using NAND, NOR and INV gates.

3.b. How many n-channel and p-channel transistors are used in this gate-level implementation?

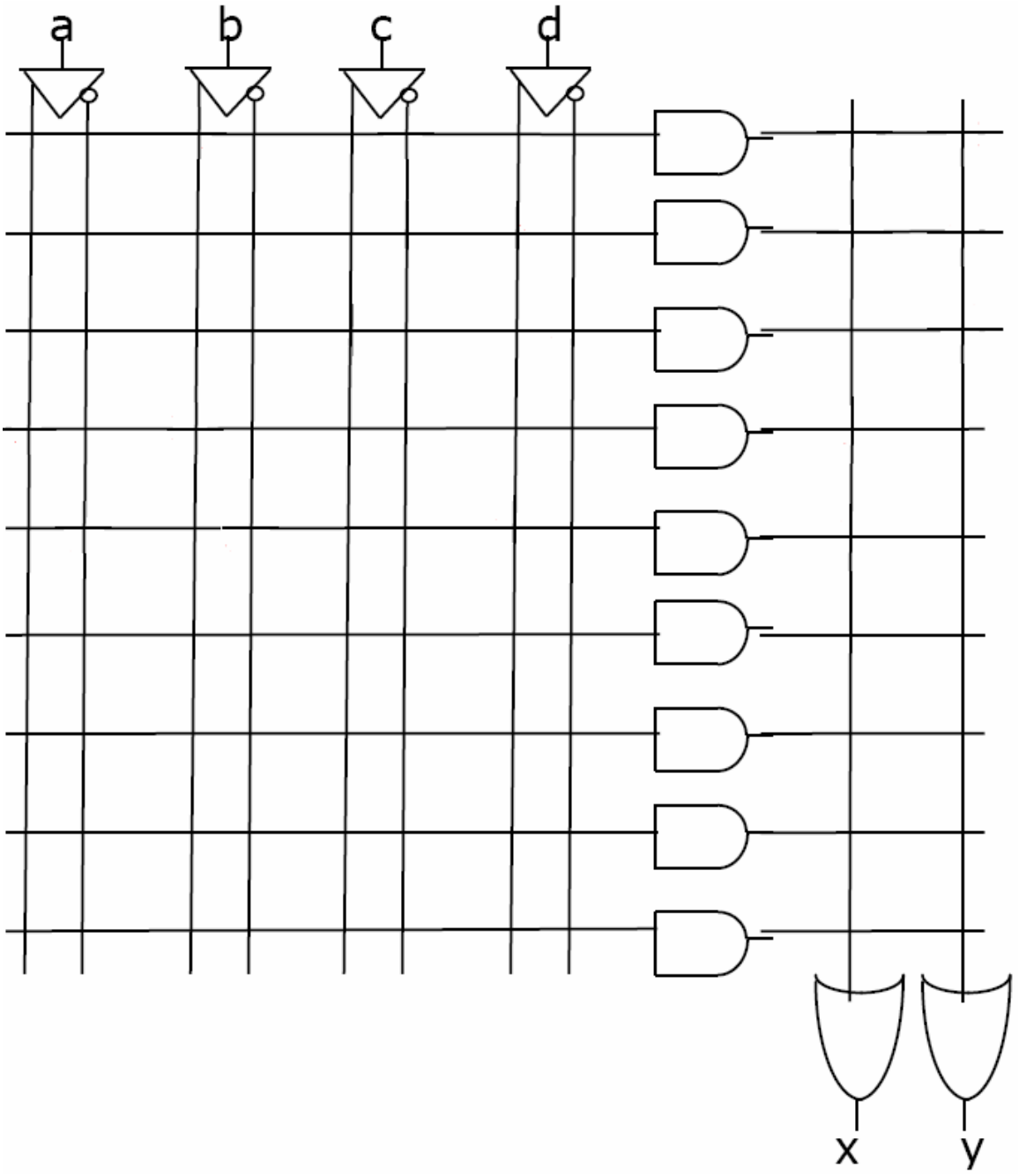
3.c. How much can you reduce this number of by implementing the operation directly at the transistor level? Draw a transistor level schematic.

Problem 4 (15 points)
The following is a truth table for a 4-input, 2-output logic function.
Inputs: **a, b, c, d**
Outputs: **x, y**

a	b	c	d	x	y
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	-	1
0	0	1	1	1	0
0	1	0	0	1	1
0	1	0	1	1	1
0	1	1	0	0	1
0	1	1	1	0	-
1	0	0	0	-	0
1	0	0	1	1	1
1	0	1	0	-	1
1	0	1	1	0	0
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	1	-
1	1	1	1	1	0

4.a. Compute minimal sum-of-product (SOP) expressions for **x** and **y** using Karnaugh maps.

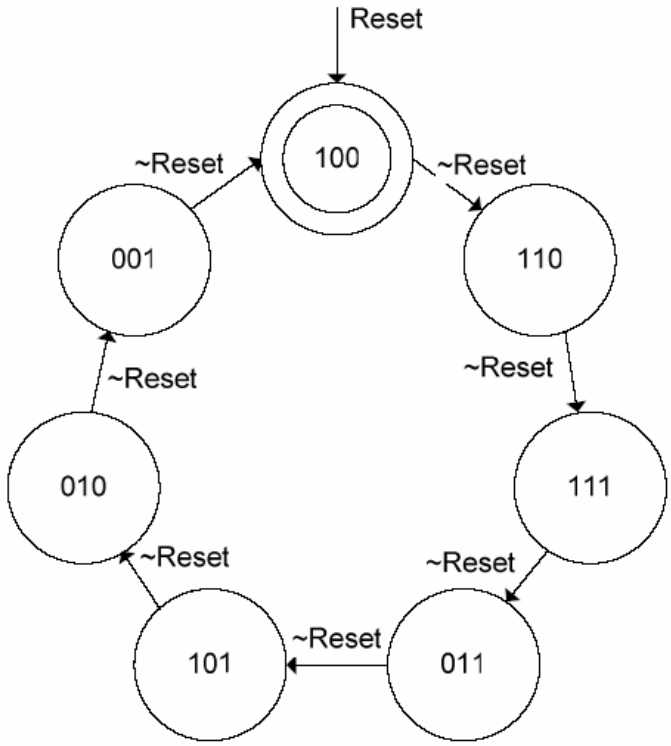
4.b. In this part, you will implement your logic on a PLA on the diagram below. You want to MINIMIZE the number of AND gates (rows) on the PLA. Mark the utilized connections on the PLA diagram for computing **x** and **y**.
Note: You may not find use for all the rows or columns on the PLA.



Problem 5 (20 points)

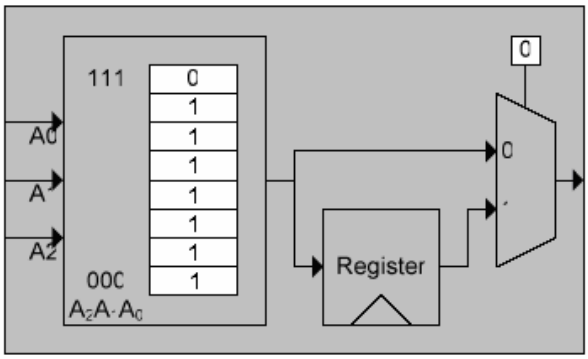
You are to implement a 3-bit counter with the following (somewhat unusual) state transition diagram.

	$C_2C_1C_0$
After reset:	1 0 0
After 1 Clock Cycle:	1 1 0
After 2 Clock Cycles:	1 1 1
After 3 Clock Cycles:	0 1 1
After 4 Clock Cycles:	1 0 1
After 5 Clock Cycles:	0 1 0
After 6 Clock Cycles:	0 0 1
After 7 Clock Cycles:	1 0 0
After 8 Clock Cycles:	1 1 0
Sequence then repeats	



5.a. Draw the schematic diagram for your 3-bit counter. You may only use 1-bit flip flops and primitive gates. Make sure to implement the "Reset" input, as your flip-flops do not have a built in reset input.

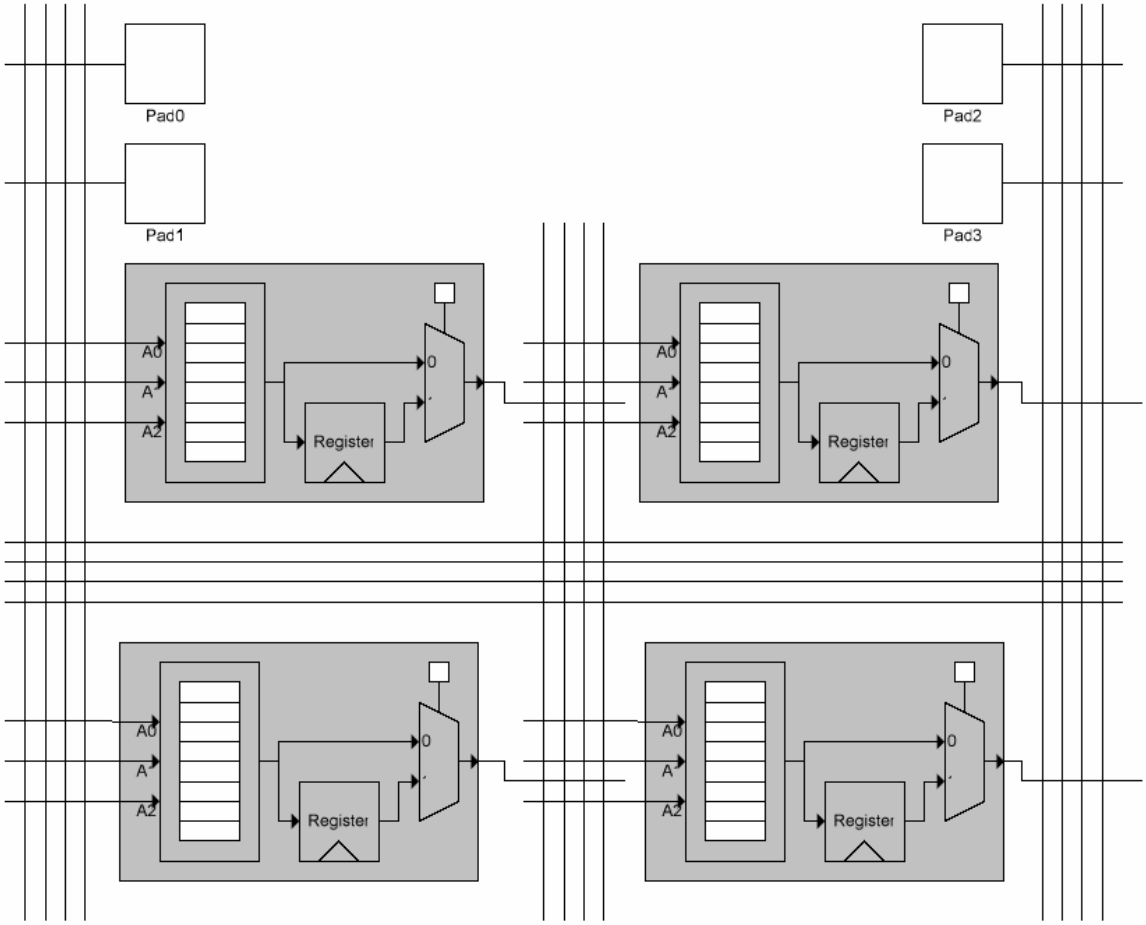
5.b. Shown below is an example CLB from an FPGA filled in to implement a NAND gate. Notice that not only is the 3-LUT filled in, but the control bit for the MUX is set.



In this problem you must implement your counter in the simplified FPGA below. Fill in the white boxes with either 1 or 0 to indicate both the programming of the 3-LUTs and the mux control bits. Indicate connected wires with an X as with PLAs.

In addition to configuring the CLBs, you must make sure to route all the signals you use, including Reset, and to configure the four I/O pads at the top.

Each signal which must connect to the outside world must be connected to an I/O pad. Reset is the only input and the three bits of the counter $C_2C_1C_0$ are the only outputs. In the white I/O pad box, write in the name of the signal connected to it. Each I/O pad would be connected to a pin on an FPGA chip.



Problem 6 (30 points)

In this problem you will be working with a new combination lock. You will design the controller for a lock with 11 buttons labeled 0-9 and Reset. To open the lock, the user must press the correct numbered buttons in sequence. The Reset button should return the lock to a default state at any time.

The lock must respond with either the Open or Locked output at all times. The lock should not output Open until one of the combinations has been entered. Once the lock is opened and outputting Open, it should continue to do so until Reset.

If at any time the user enters even a single wrong digit, the lock should output Error and continue to do so until it is Reset. Whenever the lock is not open, it should assert the Locked output. The lock may report both Error and Locked, but never Open and Locked or Open and Error.

You may assume that the inputs from the buttons are high for only one cycle when the button is pushed, regardless of the clock speed and how long the button was held down.

As a twist, the lock must respond with Open to any of three different combinations:

- 0-1-4
- 0-2-9-4
- 0-2-8

Entering any one of the above three combinations should cause the lock to Open.

6.a. Draw the bubble and arc diagram for your Moore machine implementation of the controller.

I. Label and name all states appropriately.

II. Label all arcs with the buttons that will cause that transition.

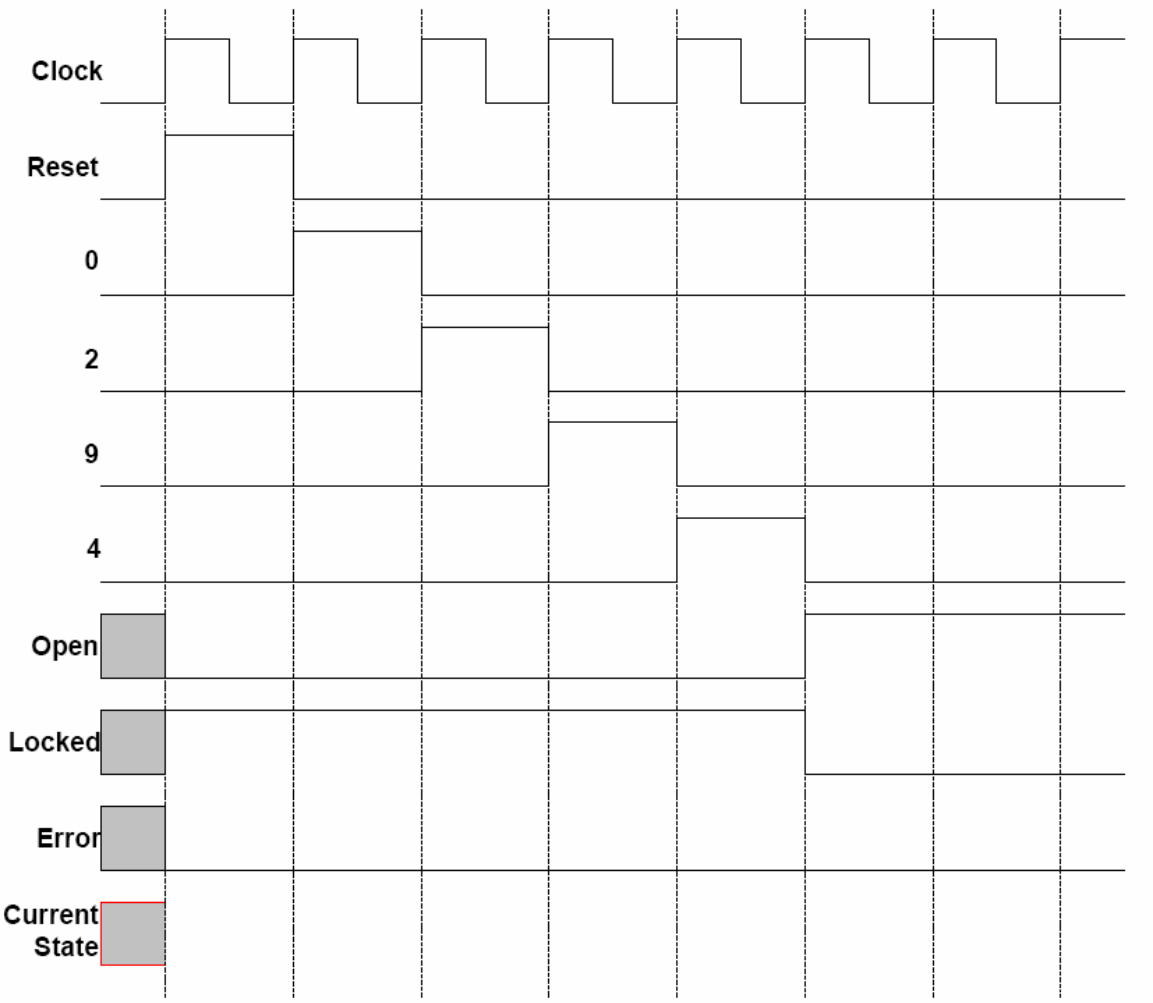
a. You may label arcs with ranges of buttons to save time

III. Label all states with the outputs that should be asserted in that state.

a. Outputs are assumed to be unasserted unless otherwise marked.

6.b. Fill in the current state values for each cycle in the timing diagram below. You should use the state names from your bubble-and-arc diagram in part (a).

Note that the gray boxes in the following diagram stand for Don't Know/Don't Care.



6.c. Fill in the Verilog shell given below with an implementation of your controller.

Note that we have added a signal called Input for your use. It is the OR of all the 10 number inputs. That is to say, it will be 1'b1 when ANY of the number buttons are pressed.

```
module LockFSM(Numbers, Reset, Clock, Open, Locked, Error);
     input [9:0]    Numbers;
     input          Reset;
     input          Clock;

     output         Open, Locked, Error;

     wire           Input;

     assign Input = (|Numbers);









endmodule
```