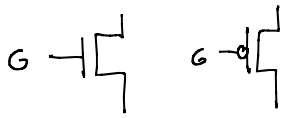
Lec-6-HW-1-devices

Problems, PP, Chp 3:
3.1 (n and p transistors),
3.2 (cmos inverter),
3.3 (how many 2-input functions?),
3.5 (trans. ckt. => truth table),

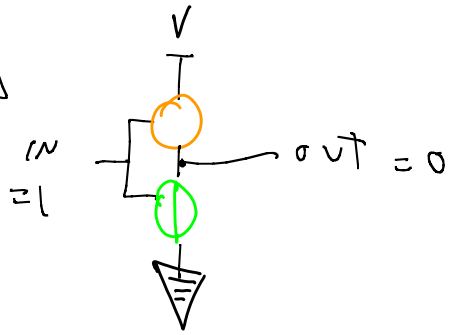
Reading: PP, Chapter 3:
§ 3.1 (transistors),
§ 3.2 (OR, NOR, AND, NAND, and DeMorgan),
§ 3.3 (DEC, MUX, FA, PLA)

3.1



	N	P
G = 0	not cond.	conducting
G = 1	cond.	not.

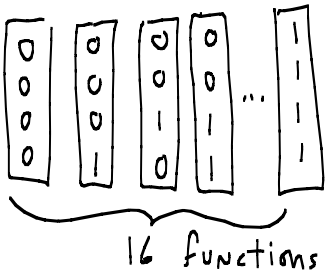
3.2



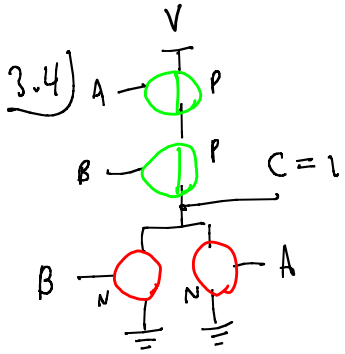
3.3

A	B	f(A,B)
0	0	x_0
0	1	x_1
1	0	x_2
1	1	x_3

⇒ how many 4-tuples of $\{0,1\}^4$? ⇒

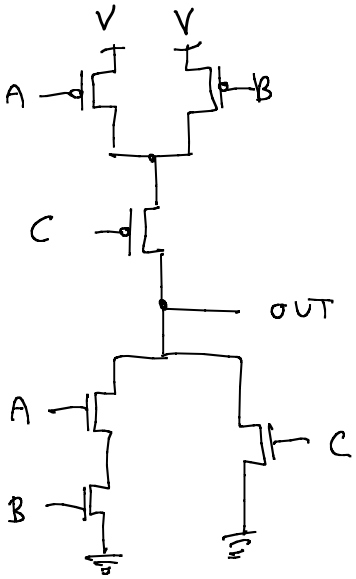


3.4



A = 0
B = 0

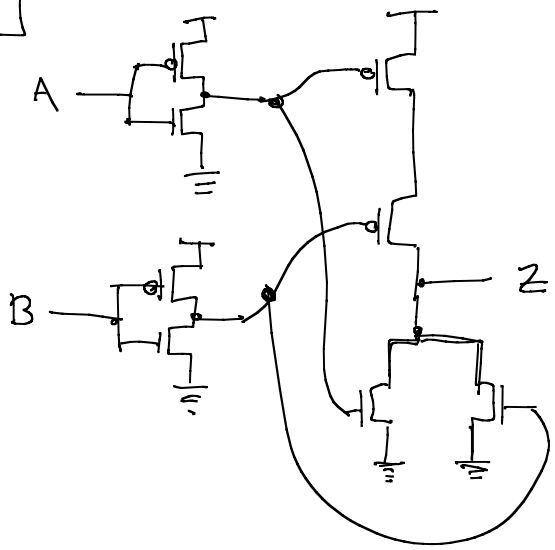
3.5



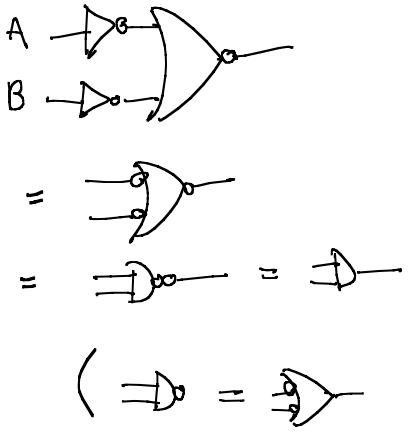
A	B	C	OUT
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

3.6 (as 3.5, but tricky ckt.),
3.7 (fix broken trans. NAND ckt.),
3.8 (label ckt. to match func.),
3.9 (expression to truth-table),

3.6

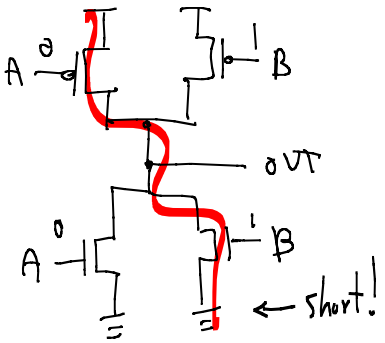


A	B	$\bar{A}$	$\bar{B}$	Z = AND(A,B)
0	0	1	1	0
1	0	0	1	0
0	1	1	0	0
1	1	0	0	1

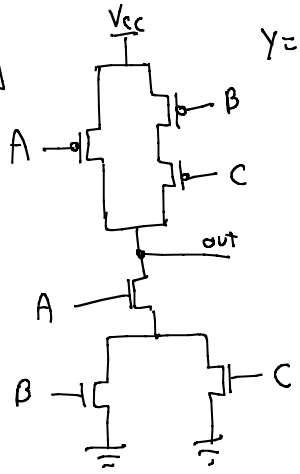


(= Demorgan)

3.7



3.8



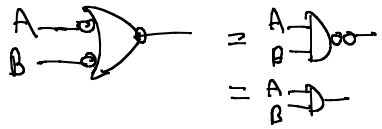
$Y = \overline{A \cdot (B+C)}$

A	B	C	
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Bottom ½-ckt. :
= 0 iff $A(B+C)$
= $\overline{A(B+C)}$

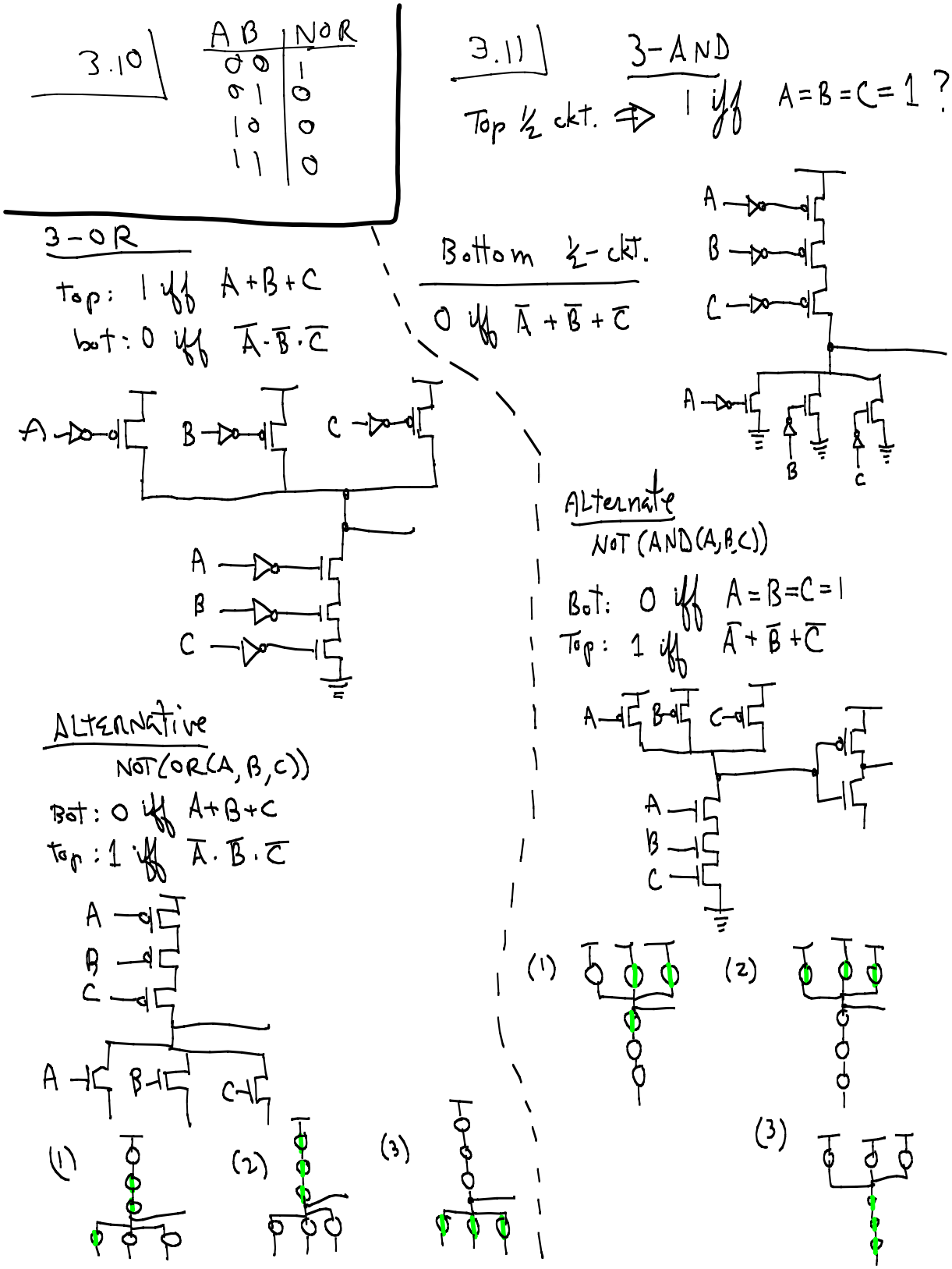
Top ½-ckt.
= 1 iff
$\bar{A} + (\bar{B} \cdot \bar{C})$

3.9

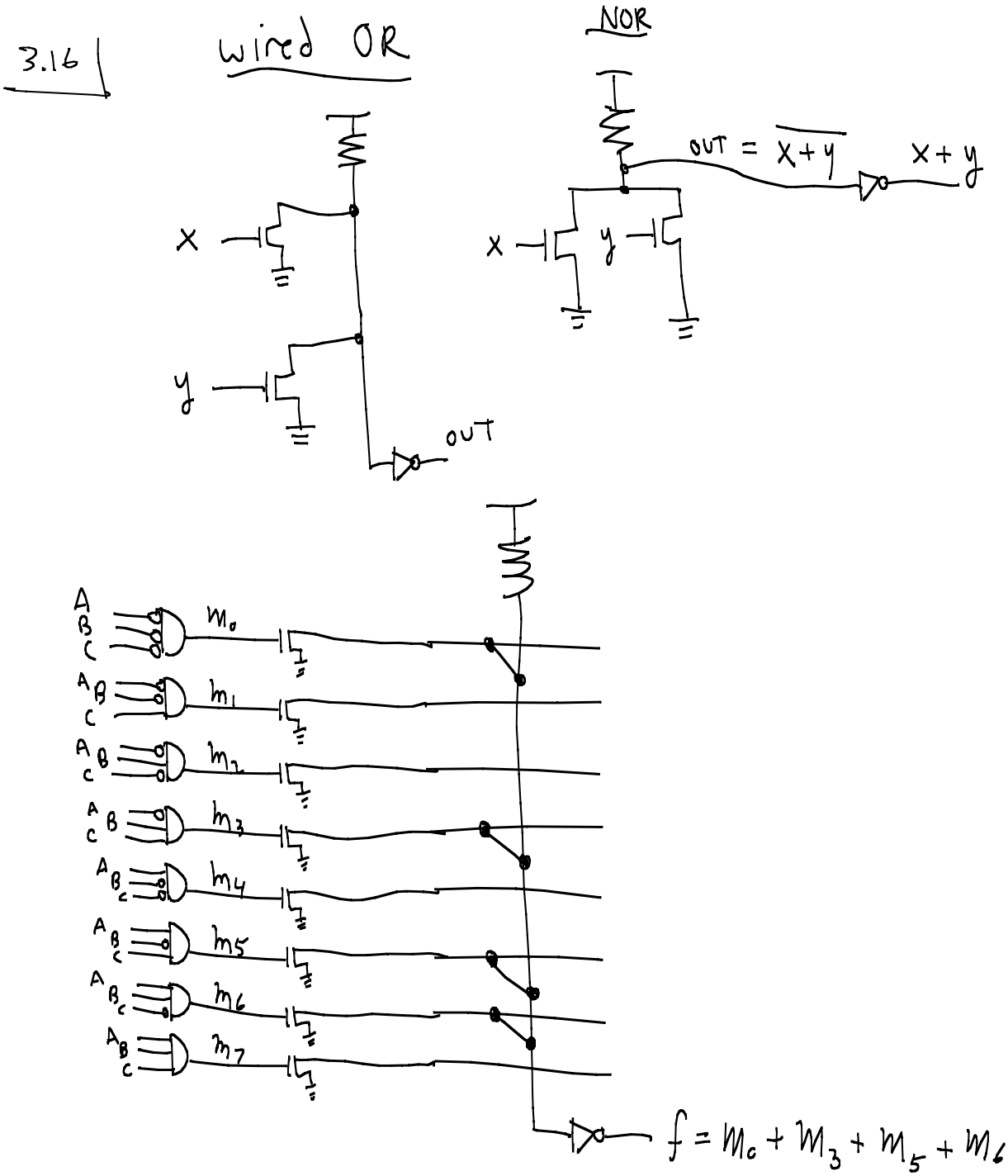


A	B	
0	0	0
0	1	0
1	0	0
1	1	1

3.10 (NOR truth-table),
3.11a (trans. ckt. for 3-AND, 3-OR),
3.11b (conduction diagram for 3.11a using input vectors),



3.16 (truth-table to PLA [connect parts of fig. 3.17])

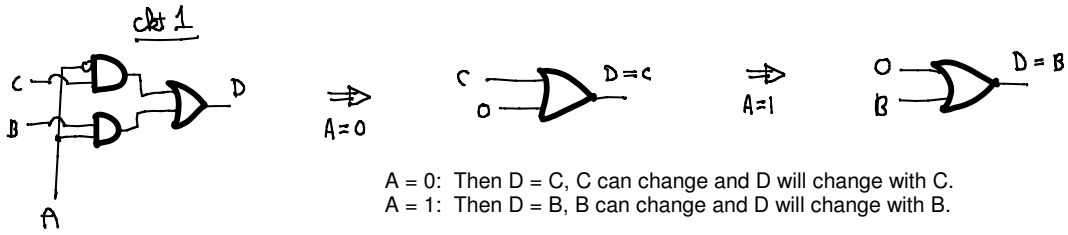


$$\bar{f} = m_1 + m_2 + m_4 + m_7$$

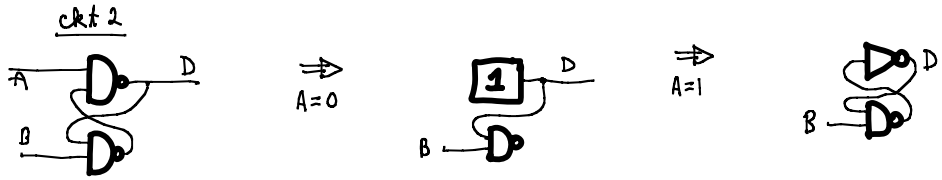
$$= \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$

$$f = (A+B+\bar{C})(A+\bar{B}+C)(\bar{A}+B+C)(\bar{A}+\bar{B}+\bar{C})$$

3.19



A = 0: Then D = C, C can change and D will change with C.
A = 1: Then D = B, B can can change and D will change with B.

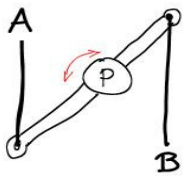


A = 0: D is forced to 1, regardless of B

B = 0: Then D is forced to 0.
B = 1: Then remains D = 1.

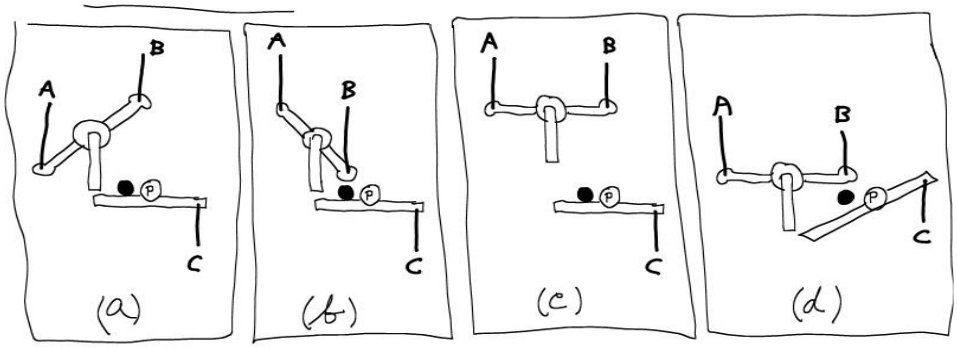
If B changes, then D will remain = 0 thereafter, regardless of how B changes, assuming A remains = 1.

Device 1

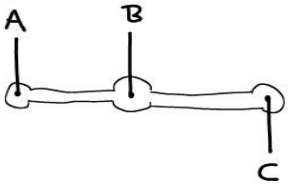


P is a fixed pivot.

Device 2



Device 3



(Problem)----------
Basic CMOS gates are NOR, NAND, NOT. Electric comes with three basic gates:

Components.schematic.{AND}
Components.schematic.{OR}
Components.schematic.{BUF}

Invert the outputs to get (NAND, NOR, NOT):

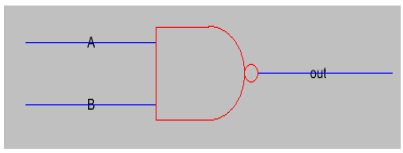
^{output crosshair}
Edit.TechnologySpecific.TogglePortNegation

(There must be a wire already attached to the gate's output for this to work.)

Make a testbench which sets all possible inputs for the NOR. Record the input and output results as a truth table. You will need reg types to drive the gate's inputs: in your test bench define a "reg" for each input, use "assign" to connect the reg-type to the input wire, and assign values to those regs in an "initial" statement. Do the same for the NAND. Turn in the truth tables on paper and checkin the circuits to you branch (perhaps use a separate library).

SOLN
Here's testbench code for both gate types:

```
/**/ reg Asrc; reg Bsrc;
/**/ assign A = Asrc; assign B = Bsrc;
/**/ initial begin
/**/        Asrc = 0; Bsrc = 0; #1 $display("(A, B) = (%b, %b), out = %b", A, B, out);
/**/   #1 Asrc = 0; Bsrc = 1; #1 $display("(A, B) = (%b, %b), out = %b", A, B, out);
/**/   #1 Asrc = 1; Bsrc = 0; #1 $display("(A, B) = (%b, %b), out = %b", A, B, out);
/**/   #1 Asrc = 1; Bsrc = 1; #1 $display("(A, B) = (%b, %b), out = %b", A, B, out);
/**/   $finish;
/**/ end
```



SOLN
Here's the output using NAND:

(A, B) = (0, 0), out = 1
(A, B) = (0, 1), out = 1
(A, B) = (1, 0), out = 1
(A, B) = (1, 1), out = 0

(Problem)----------
In Electric, implement a NAND-NAND latch. Test its behavior by driving your circuit with all possible sequences of inputs. For latches, the history of inputs is important, not just the current input, because latches have state. That is, they remember what happened before. Do the same for the NOR-NOR latch, and comment on the difference between the NAND-NAND latch and the NOR-NOR latch. Turn your answers in on paper and check in your testbench and its output.

Note: For the NAND-NAND latch, we are interested in 3-step sequences that begin and end with A=1, B=1 (for NOR-NOR, A=0, B=0). For instance, here is a 3-step NAND-NAND sequence: (1,1), (0,1), (1,1). Here is another that is particularly interesting:

```
Asrc = 1;
Bsrc = 1;
#1
Asrc = 0;
Bsrc = 0;
#1
Asrc = 1;
Bsrc = 1;
```

That sequence might give different results from this one. Why? Does this mean we should use random delays for signal propagation through our gates to be more physically realistic? HINT--metastability problem.

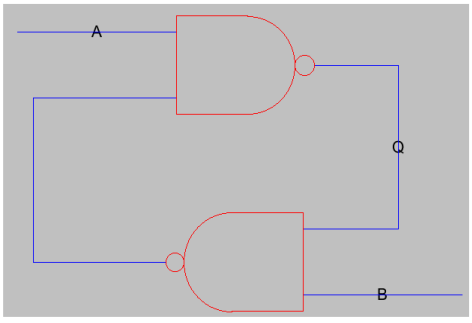
```
Asrc = 1;
Bsrc = 1;
#1
Bsrc = 0;
Asrc = 0;

#1
Asrc = 1;
Bsrc = 1;
```

SOLN
The difference between the two latch types is that the NAND latch is stable (keeps its last state) for input (A,B) = (1,1) while the NOR latch is stable for (0,0). For both types, (0,1) sets Q to 1 and (1,0) resets Q to 0.

The input sequence (0,0), (1, 1) to the NAND latch should result in an unknown value for Q. Symmetrically, the sequence (1,1), (0,0) should make Q unknown for the NOR latch. However, this happens for some verilog simulated circuts and not others.

For me, unknowns did not occur for these latches, except for the first state. For the NAND latch, inputs (A, B) are (-S, -R). For the NOR latch, (A, B) = (R, S). (Setting "R" to 1 forces Q to 0; setting "S" to 1 forces Q to 1. "R" is for "reset"; "S" is for "set". "-S" means "NOT(S)", and setting "-S" to 0 forces Q to 1.)



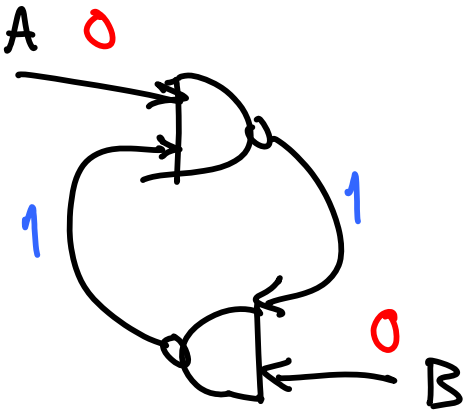
SOLN
The NOR-NOR solution is similar.

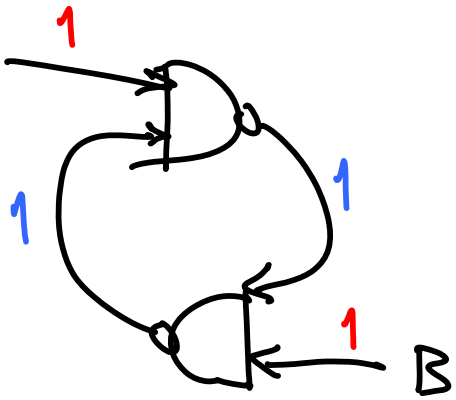
```
/**/ reg Asrc; reg Bsrc;
/**/ assign A = Asrc; assign B = Bsrc;
/**/ initial begin
/**/    #1 Asrc = 1; Bsrc = 1;
/**/    #1 $display("(A, B) = %b%b Q=%b", A, B, Q);
/**/    #1 Asrc = 0; Bsrc = 1;
/**/    #1 $display("(A, B) = %b%b Q=%b", A, B, Q);
/**/    #1 Asrc = 1; Bsrc = 1;
/**/    #1 $display("(A, B) = %b%b Q=%b", A, B, Q);
/**/    #1 Asrc = 1; Bsrc = 0;
/**/    #1 $display("(A, B) = %b%b Q=%b", A, B, Q);
/**/    #1 Asrc = 1; Bsrc = 1;
/**/    #1 $display("(A, B) = %b%b Q=%b", A, B, Q);
/**/    #1 Asrc = 0; Bsrc = 0;
/**/    #1 $display("(A, B) = %b%b Q=%b", A, B, Q);
/**/    #1 Bsrc = 1; Asrc = 1;
/**/    #1 $display("(A, B) = %b%b Q=%b", A, B, Q);
/**/    #1 Asrc = 0; Bsrc = 0;
/**/    #1 $display("(A, B) = %b%b Q=%b", A, B, Q);
/**/    #1 Bsrc = 1; Asrc = 1;
/**/    #1 $display("(A, B) = %b%b Q=%b", A, B, Q);
/**/    $finish;
/**/ end
```

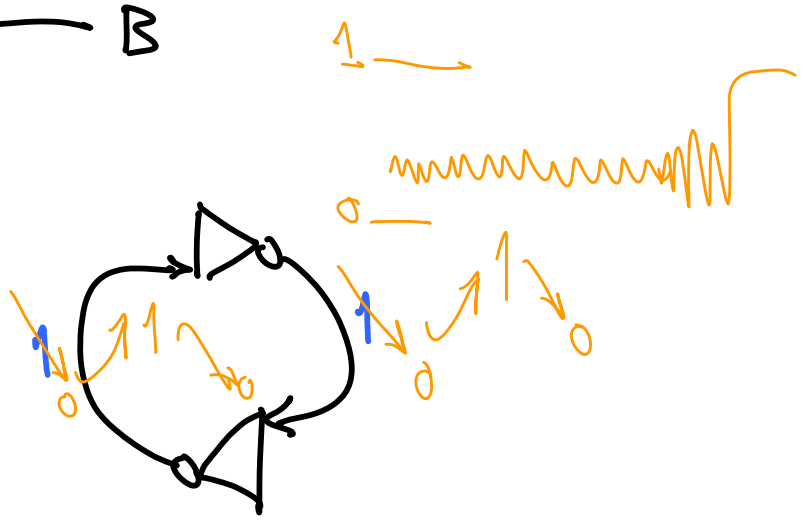
SOLN
Testbench output:

```
(A, B) = 11 Q=x
(A, B) = 01 Q=1
(A, B) = 11 Q=1
(A, B) = 10 Q=0
(A, B) = 11 Q=0
(A, B) = 00 Q=1
(A, B) = 11 Q=1
(A, B) = 00 Q=1
(A, B) = 11 Q=1
```

Note that the simulation does not show unknowns for the indeterminate input (0,0).


A
0
1
1
0
B


1
1
1
1
B


1
0
1
1
0
1
0
1
0