V: Sequential Logic: State Machines

1: The State Machines

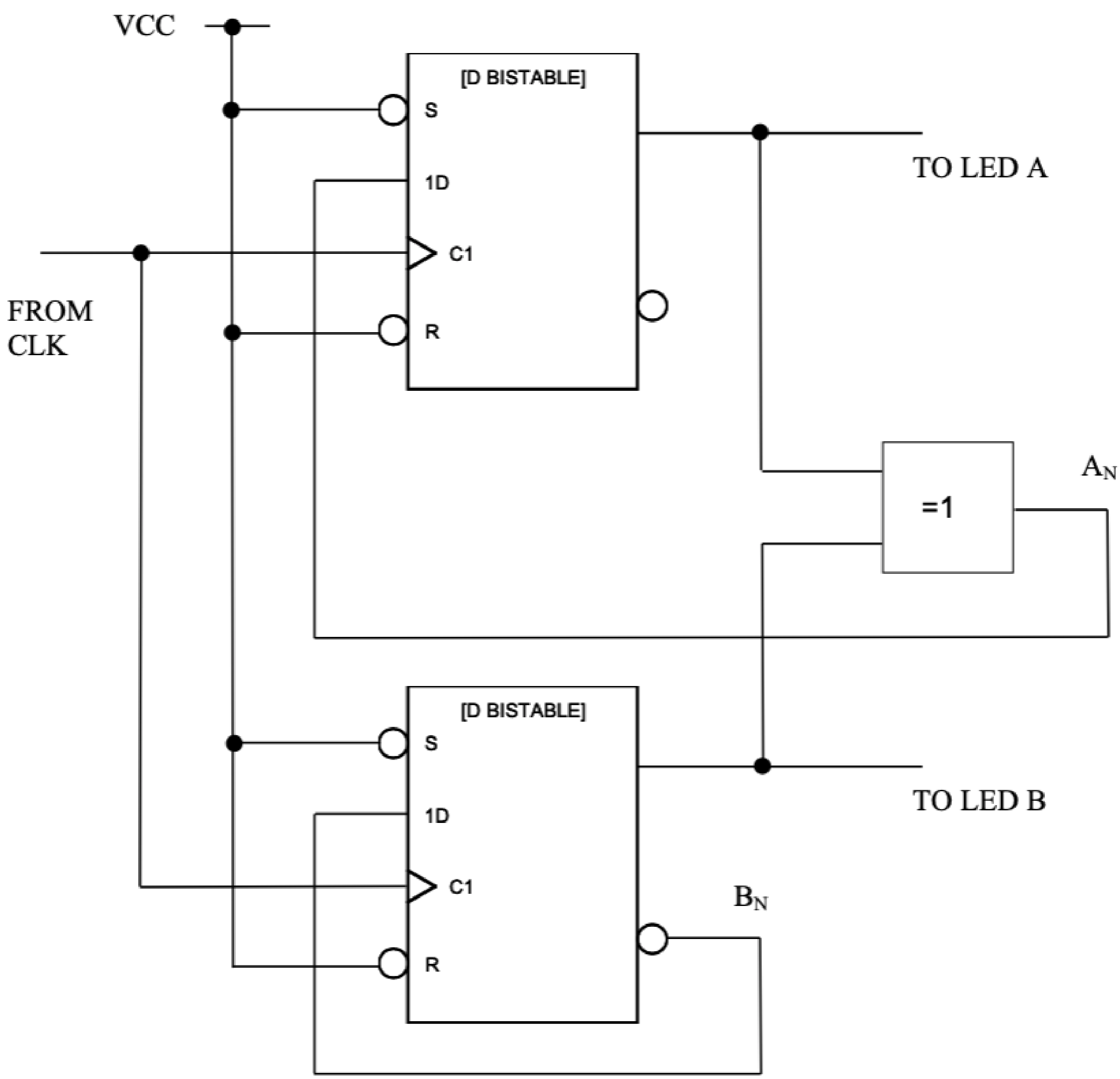
- Suppose that your are asked to designed a system that lights a pair of LEDs in the following sequence, which your will recognize as a two-bit binary counter.
- The system advances from one state to the next at every clock edge.

LED A	LED B
0	0
0	1
1	0
1	1
then repeating	
0	0
0	1
... etc	

- The fundamental shape of this system is the state machine. Two D-type flip-flop can store the state of the outputs, and a function of these outputs is fed back to the input, to form the new outputs after the next clock edge.
- We use the following way to deduce what the function is:

Current state before clock edge		New state after clock edge	
LED A	LED B	LED A_N	LED B_N
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

- From this, we can find that $B_N = \bar{B}$ and $A_N = A \oplus B$
- Then we can build the circuit:

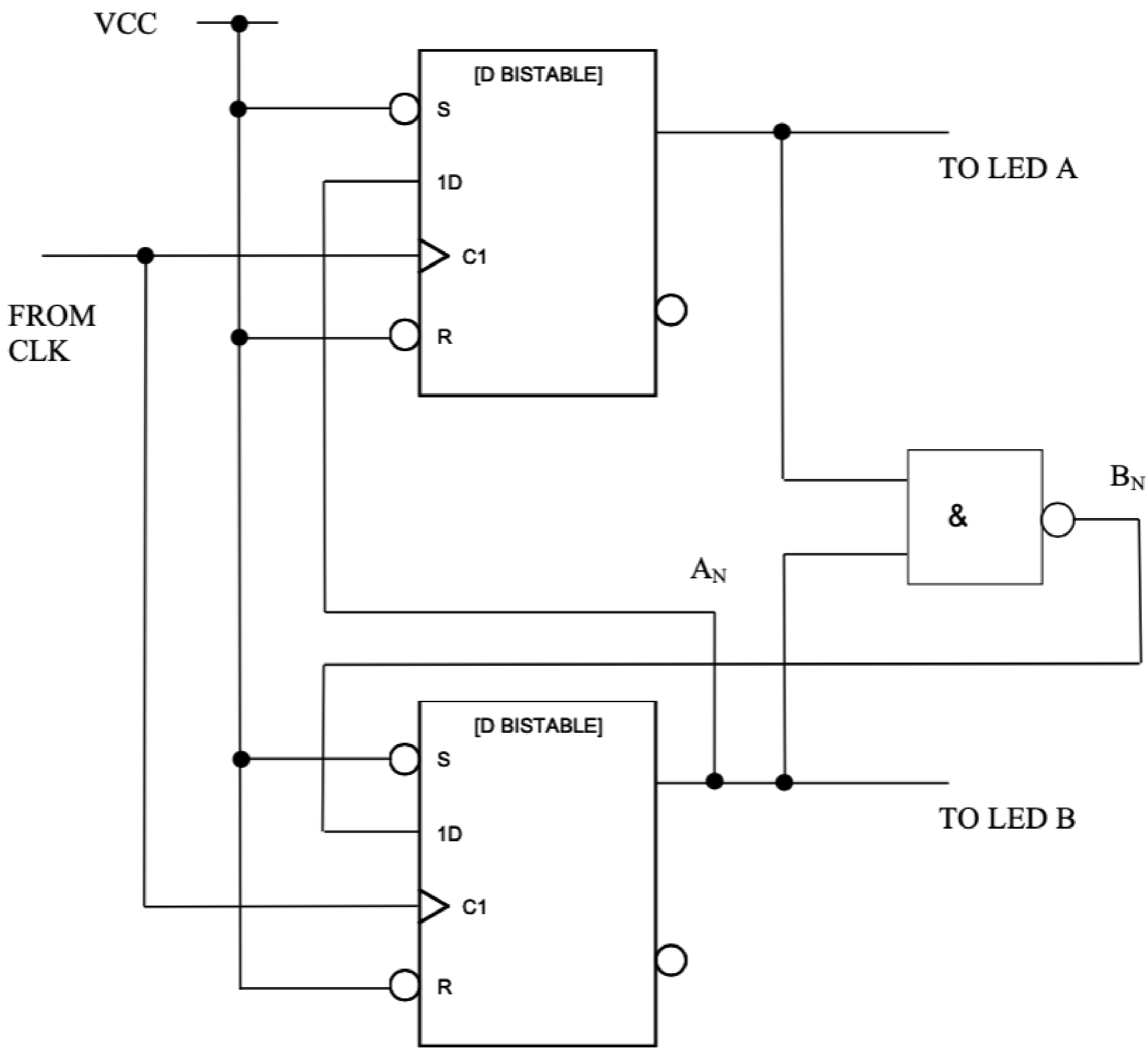


2: Example_1

- Design a two-bit counter which omit "0", (3,2,1,3,2,1...)

Current state		New state	
A	B	A_N	B_N
1	1	1	0
1	0	0	1
0	1	1	1

- The we find that $A_N = B$ and $B_N = \overline{A.B}$



- Noted that we use the NAND gate instead of OR-EXCLUSIVE gate, cause when A and B are both logic '0', it will advance to A=0 and B=1 next clock edge, then will be correct. However if it is

OR-EXCLUSIVE gate, it will remain '0' '0' forever.

3: Example_2

- Design a system that generates 3,2,1,0,3,2,1,0

LED A	LED B
1	1
1	0
0	1
0	0
then repeating	
1	1
1	0
... etc	

- No OR-EXCLUSIVE

Current state		New state	
A	B	A_N	B_N
1	1	1	0
1	0	0	1
0	1	0	0
0	0	1	1

- We can find that $A_N = \bar{A}.\bar{B} + A.B$ and $B_N = \bar{B}$

$$A_N = \bar{A} \,.\, \bar{B} \;+\; A \,.\, B \qquad (1)$$

$$= \overline{\overline{\bar{A} \,.\, \bar{B}}} \;+\; A \,.\, B$$ *double inversion cancels out, but gives opportunity to apply de Morgan's Law*

$$= \overline{A + B} \;+\; A \,.\, B$$ *de Morgan's Law* (2)

$$= \overline{\overline{\overline{A + B} \;+\; A \,.\, B}}$$ *another double inversion*

$$= \overline{(A + B) \;.\; \overline{A \,.\, B}}$$ *de Morgan's Law* (3)

- We use the last one cause it have less gates and types of functions.