DIGITAL ELECTRONICS

LECTURE 6

PART I

*K-Maps Revision

By J. Mathenge

Example 1

- Minimize the following Boolean function:

$F(A,B,C,D)=\{m(0,1,2,5,7,8,9,10,13,15)\}$

A	B	C	D	OUTPUT
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

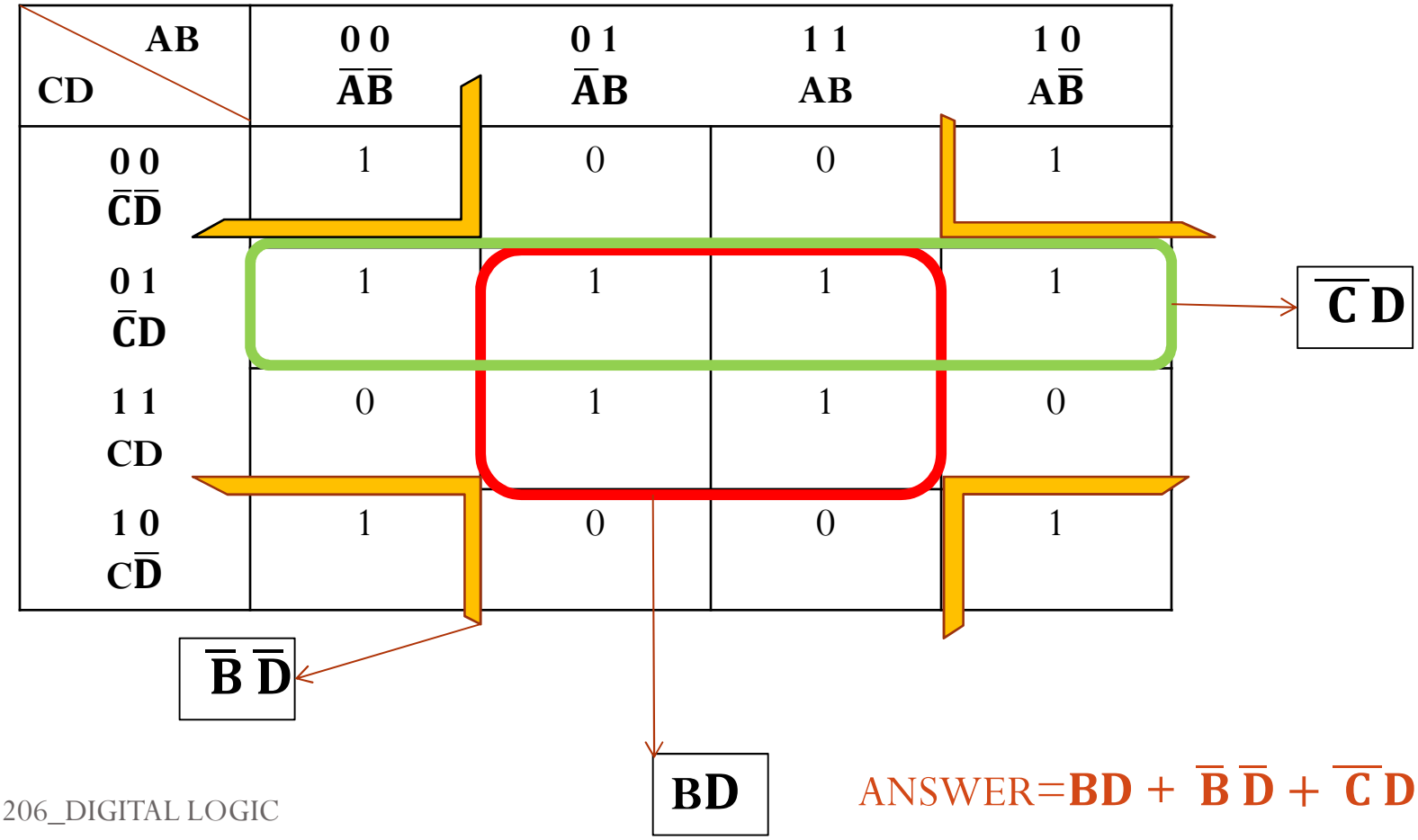
Solution

Since the expression has 4 variables, we draw a $2^n = 2^4$ K-map. Fill in the 1's as per the truth table.

CD \ AB	0 0 $\bar{A}\bar{B}$	0 1 $\bar{A}B$	1 1 AB	1 0 $A\bar{B}$
0 0 $\bar{C}\bar{D}$	1	0	0	1
0 1 $\bar{C}D$	1	1	1	1
1 1 CD	0	1	1	0
1 0 $C\bar{D}$	1	0	0	1

Solution

Looping…



ANSWER=$BD + \bar{B}\bar{D} + \bar{C}D$

Example 2

- Minimize the following Boolean function:

$F(A,B,C,D)=\{m(0,1,3,5,7,8,9,11,13,15)\}$

A	B	C	D	OUTPUT
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Solution

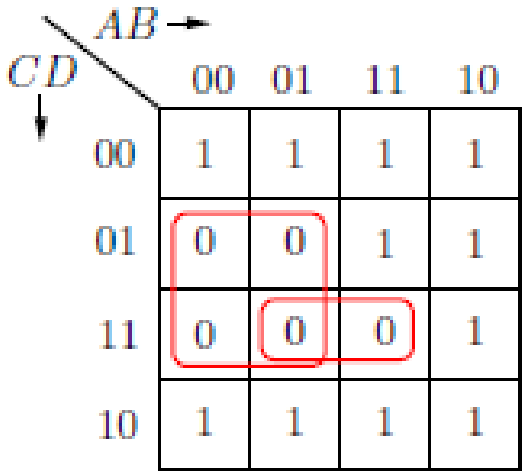
Since the expression has 4 variables, we draw a $2^n = 2^4$ K-map. Fill in the 1's as per the truth table.

CD \ AB	0 0 $\overline{A}\overline{B}$	0 1 $\overline{A}B$	1 1 AB	1 0 $A\overline{B}$
0 0 $\overline{C}\overline{D}$				
0 1 $\overline{C}D$				
1 1 CD				
1 0 $C\overline{D}$				

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

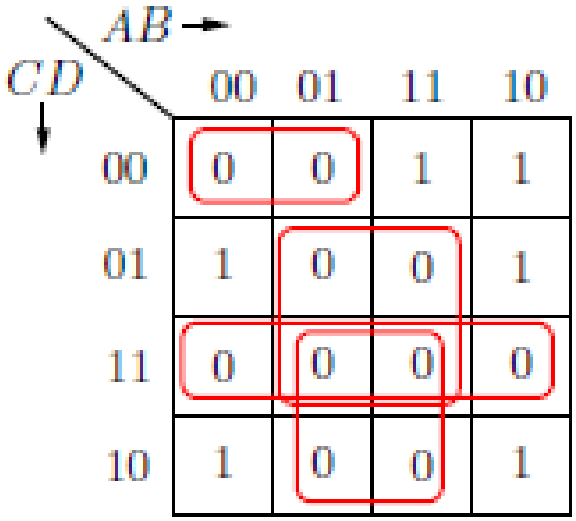
3.1.5 Obtaining Product of Sums Expressions from K-maps

- So far, we have talked about obtaining Sum of Products expressions from K-maps. It is also possible to obtain Product of Sums expressions . only this time we loop the zeros together, not the ones. The examples below illustrate the procedure.



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

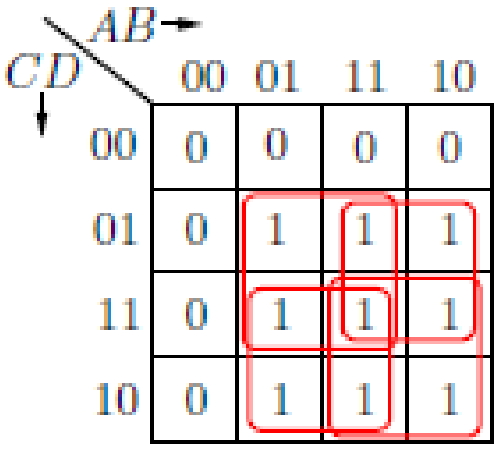
3.1.5 Obtaining Product of Sums Expressions from K-maps



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

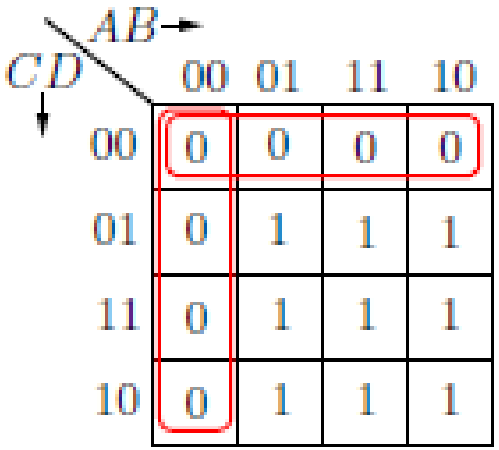
3.1.5 Obtaining Product of Sums Expressions from K-maps

- Note that for a given K-map, looping the 1s and looping the zeros gives the same results, only that the results are expressed in different ways. Consider the example shown below where in one case the 1s are looped and in the other case the 0s are looped.



CD \ AB	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	0	1	1	1
10	0	1	1	1

ANSWER=AD+BD+BC+AC



CD \ AB	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	0	1	1	1
10	0	1	1	1

ANSWER=(C+D)(A+B)

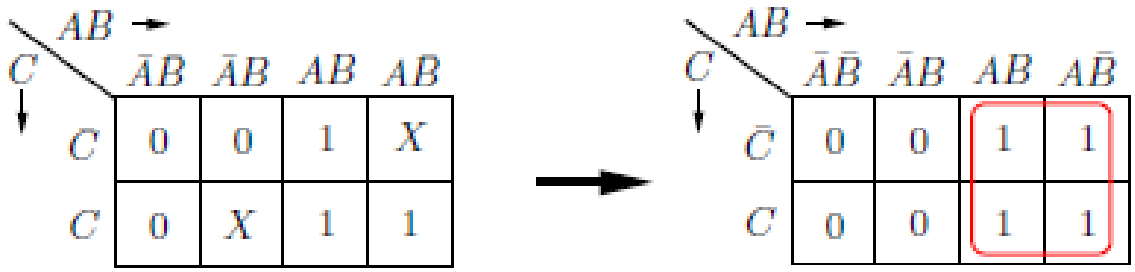
3.2 Don't Care Terms

- These are also referred to as Unused terms, Forbidden terms or Redundant terms. These terms describe combinations of variables which never occur in practice.
- In a truth-table or a K-map, these inputs are represented by an X, an R or a d.
- Example, suppose that we have a digital circuit with three inputs and one output, and the input combinations 000, 001 and 010 give an output 0, input combinations 101, 110 and 111 give an output 1, and input combinations 011 and 100 never occur in practice.
- Such a truth table is represented as:

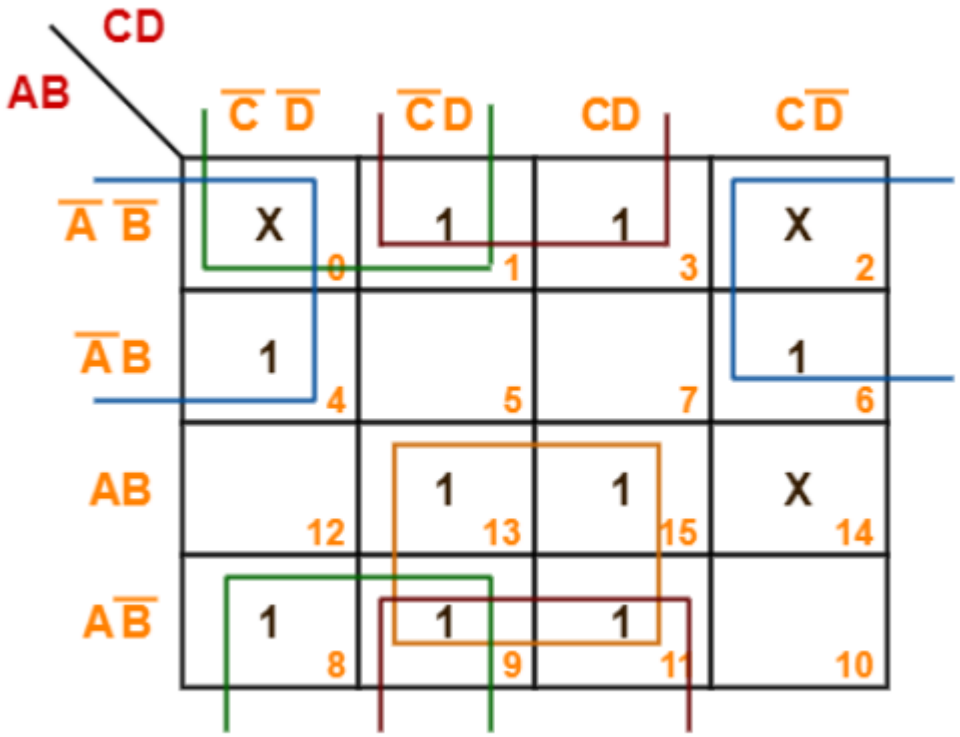
A	B	C	*output*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	X
1	0	0	X
1	0	1	1
1	1	0	1
1	1	1	1

3.2 Don't Care Terms

- When designing with K-maps containing don't care variables, the designer can make the output of any don't care condition either a 1 or a 0 in order to produce the simplest output expression.
- Example:



3.2 Don't Care Terms



F(A, B, C, D) = AD + B'D + B'C' + A'D'

3.2 Don't Care Terms

- Minimize the below Boolean expression:

$F(A, B, C) = \Sigma m(0, 1, 6, 7) + \Sigma d(3, 5)$

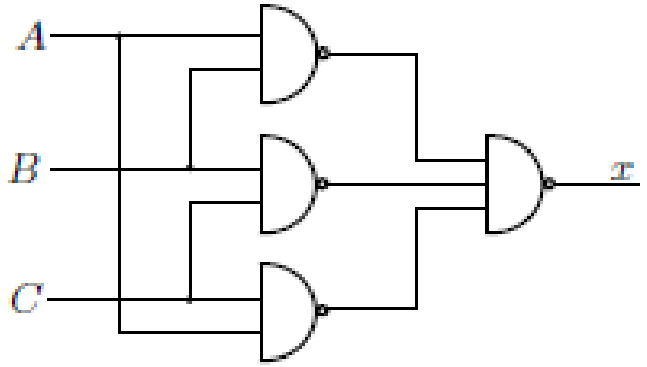
3.3 NAND/NOR gate circuit implementation

- To implement a logic circuit using NAND gates only:

1. Derive the minimized expression for the function in *sum of products form* (obtained by minimizing Boolean expressions algebraically or by looping 1s in a K-map).

2. Apply double negation and De Morgan's theorem to convert the expression in a form suitable for NAND gate implementation.

- Suppose a design problem results in a *minimized sum of products* expression: **AB+BC+AC**
- $X=AB+BC+AC=\overline{\overline{AB+BC+AC}}=\overline{\overline{AB}\cdot\overline{BC}\cdot\overline{AC}}$



Example: implementation using NAND gates only

3.3 NAND/NOR gate circuit implementation

- To implement a logic circuit using NOR gates only:

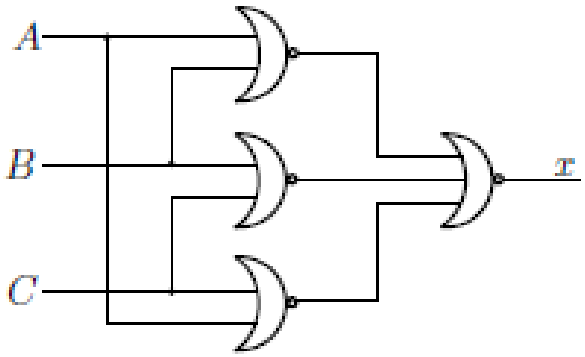
1. Derive the minimized expression for the function in product of sums form (obtained by looping the 0s in a K-map).

2. Apply double negation and De Morgan's theorem to convert the expression in a form suitable for NOR gate implementation.

- As an example, suppose a design problem resulted in a minimized product of sums expression: **(A+B) (B+C) (A+C)**

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

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



Example: implementation using NOR gates only

End of Part I

Questions….?

DIGITAL ELECTRONICS

PART II

CHAPTER FOUR: Sequential Logic Circuits (Part I)

By J. Mathenge

4.1 Introduction

- So far, we have dealt with **Combinational Logic Circuits (CLCs)**, whose outputs are functions of the current inputs only. This chapter deals with sequential logic circuits.
- A Sequential Logic Circuit (SLC) is a circuit whose **present outputs** are functions of the present inputs, as well as previous inputs.
- In it, there's a unit called the memory which stores the effect of the previous sequence of inputs. The **stored** effects of the previous inputs are called the STATE of the circuit.
- The block diagram of a SLC is as shown next:

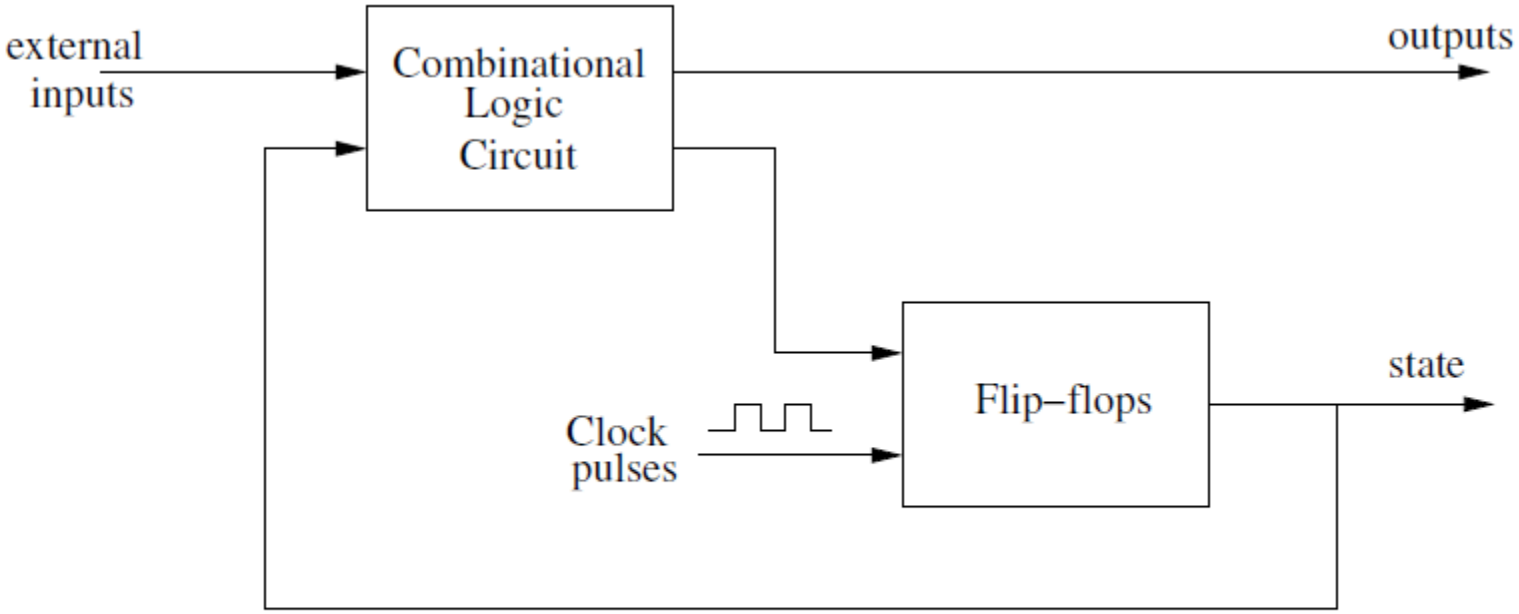


Figure 4.1: Block diagram of a sequential logic circuit

4.2 Flip Flops/Bi-stable Multi-vibrator

- A flip-flop is a logic circuit that is capable of storing one bit of information (0 or 1).
- It stores the one bit of information as long as power is supplied to the circuit. It is the simplest memory element.
- Flip-flops are the basic building blocks for sequential logic circuits. The block diagram of a flip-flop is shown below:

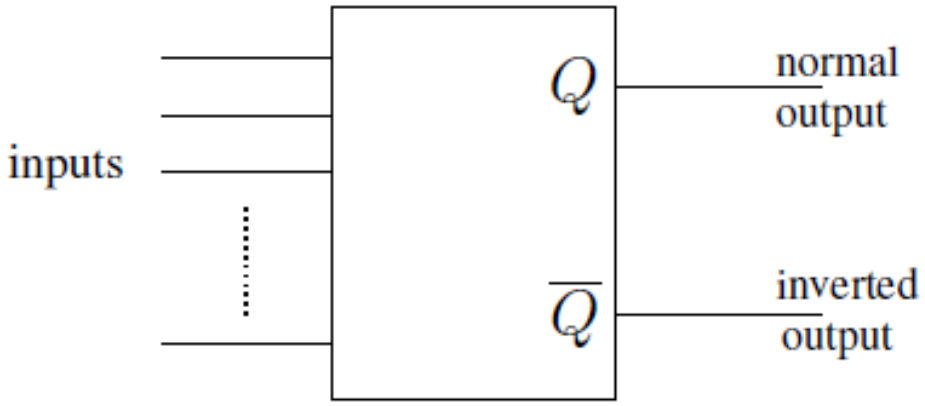


Figure 4.2: A flip-flop

4.2 Flip Flops/Bi-stable Multi-vibrator

- A Flip-flop can have one or more inputs, but it has only **two outputs**; the normal output Q and the inverted (or complemented) output $\bar{Q}$.
- Under normal operating conditions Q and $\bar{Q}$ are always complements of each other, i.e. either $Q = 0$ and $\bar{Q} = 1$, or $Q = 1$ and $\bar{Q} = 0$.
- When we refer to the STATE of a flip-flop, we are referring to the state of its normal (Q) output i.e. if we say that the state of a flip-flop is 1, we mean Q = 1.

4.2.1 The NAND-gate latch

- The NAND-gate latch is the simplest flip-flop.
- Also referred to as a bi-stable latch.

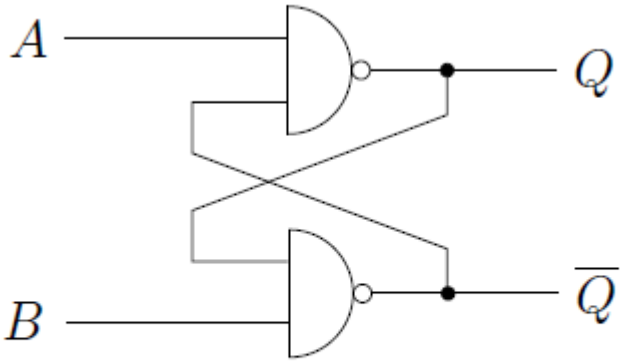


Figure 4.3: A NAND gate latch

A	B	Q_n	Q_{n+1}	$\bar{Q}_{n+1}$
0	0	0	1	1
0	0	1	1	1
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	0

- The truth-table for the NAND-gate latch is shown above (Q_n stands for present state, while Q_{n+1} stands for next state).

4.2.1 The NAND-gate latch

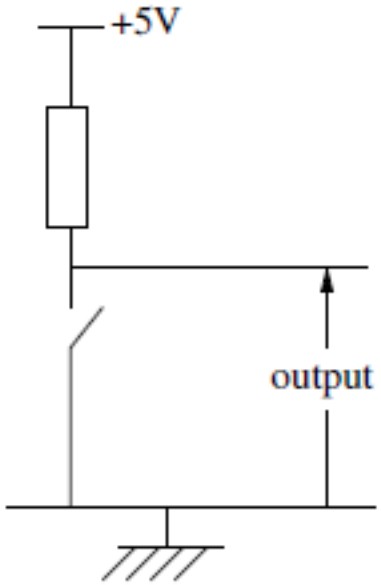
- The truth table is summarized as:

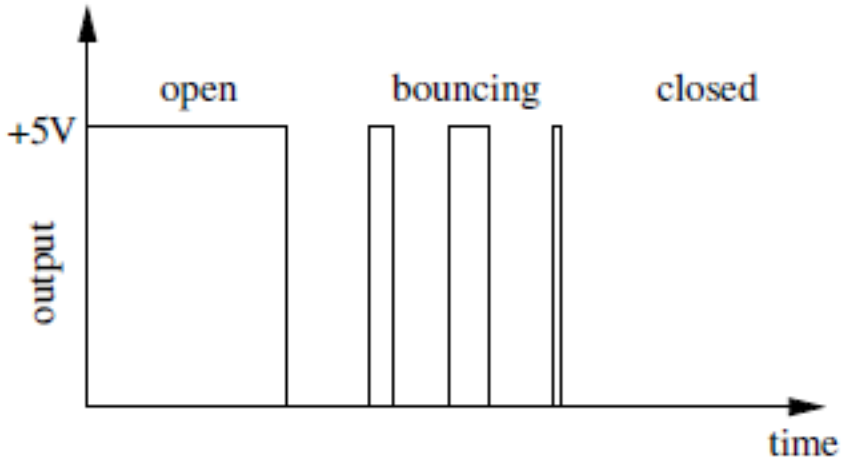
A	B	Q_{n+1}
0	0	Disallowed as it causes $Q = \bar{Q} = 1$
0	1	1
1	0	0
1	1	Q_n - 'remembers' previous state (Memory State)

- A NAND-gate latch is used as a building block for other more complicated flip-flops, and for de-bouncing switches.
- The next illustration shows how the NAND gate latch can be used to de-bounce a switch:

4.2.1 The NAND-gate latch application

- Normal operation:





4.2.1 The NAND-gate latch application

- After connecting the NAND gate latch

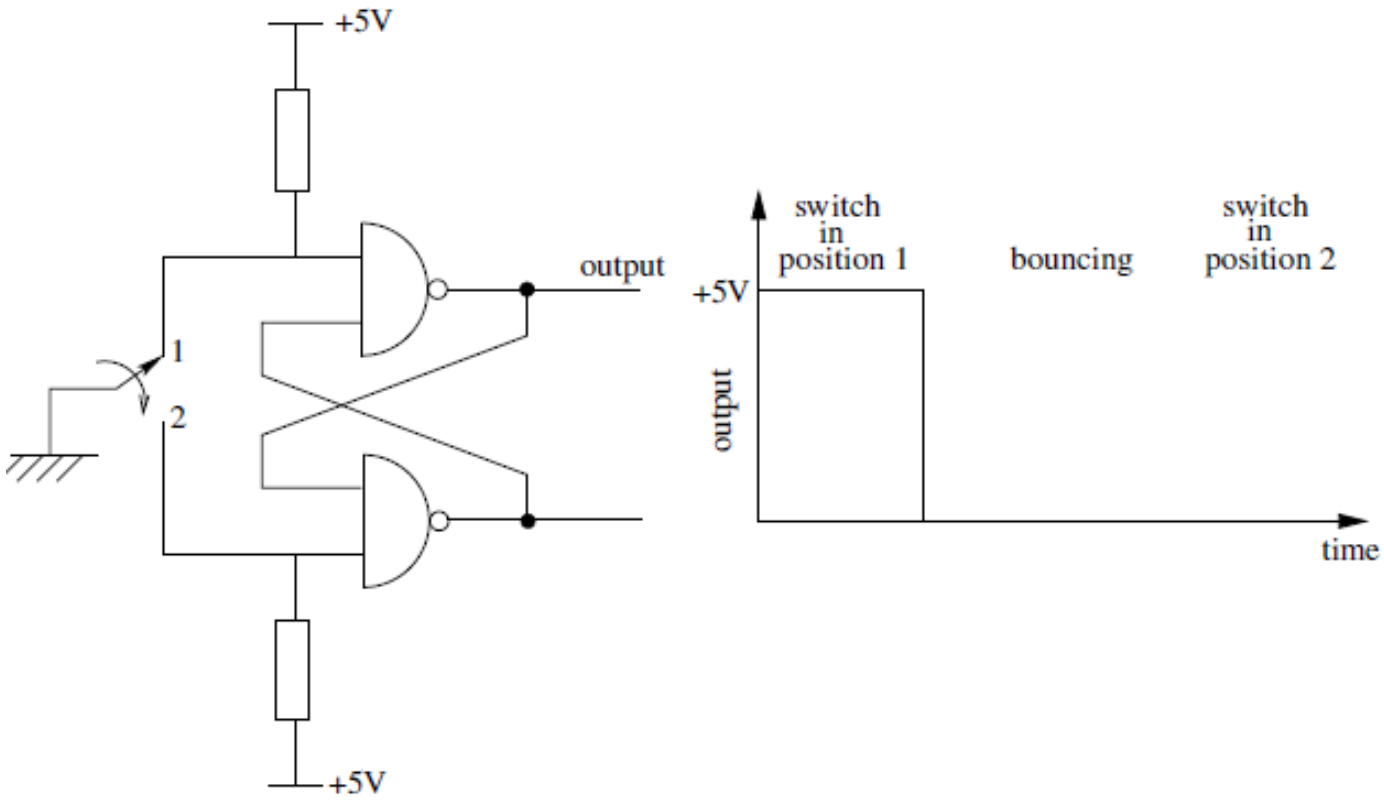


Figure 4.4: Switch debouncing

4.2.2 The SET-RESET flip-flop

- Illustration:

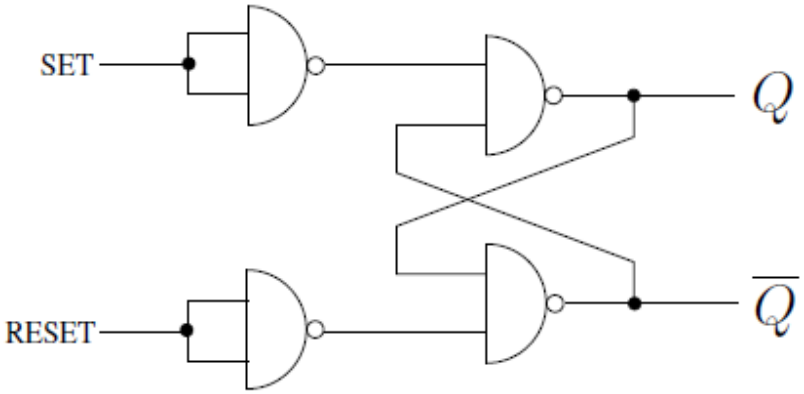


Figure 4.5: SET-RESET (SR) flip-flop

SET	$RESET$	Q_{n+1}
0	0	Q_n - 'remembers' previous state (Memory State)
0	1	0 - Flip-Flop RESET
1	0	1 - Flip-Flop SET
1	1	Disallowed as it causes $Q = \bar{Q} = 1$

4.2.2 The SET-RESET flip-flop

- When we have SET=0 and RESET=1, the flip flop is RESET ($Q = 0$), and when SET=1 and RESET=0, the flip flop is SET ($Q = 1$).
- If SET=1 and RESET=1, this is similar to setting and resetting the flip flop at the same time and this mode is **disallowed** since it causes $Q = \bar{Q} = 1$.

4.2.3 The D Flip-Flop

- It is a modified version of the SET-RESET flip-flop where **SET** $= \overline{\textbf{RESET}}$. It only has one input, D. The illustration and truth-table for this flip-flop is shown next:

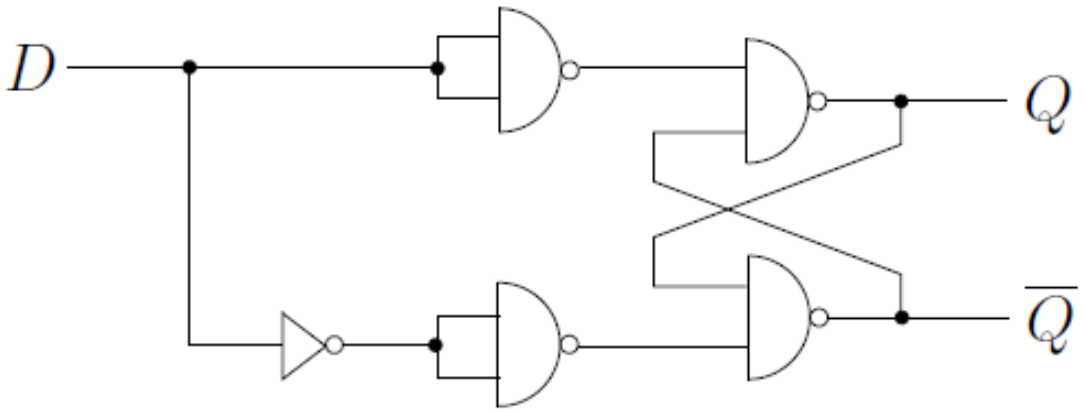


Figure 4.6: D flip-flop

D	Q_{n+1}
0	0
1	1

4.2.4 The T flip-flop

- A T-flip-flop (also known as a toggle flip-flop) is illustrated

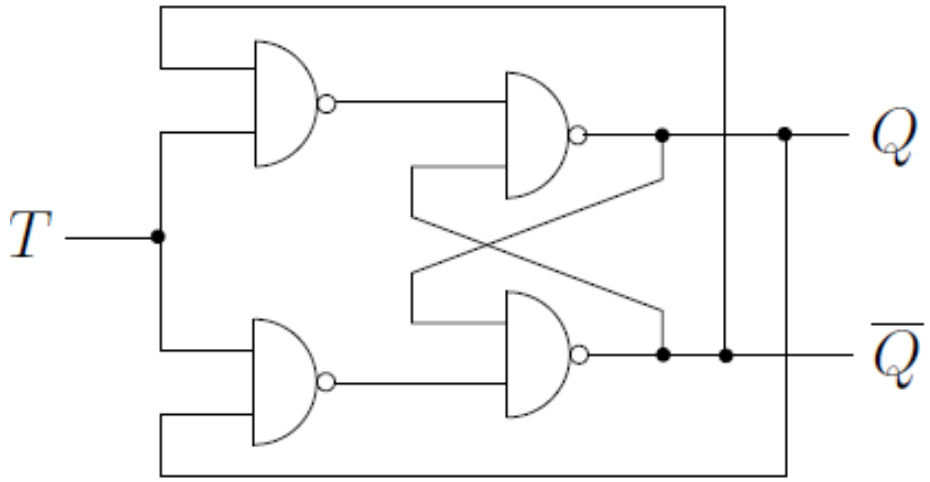


Figure 4.7: T flip-flop

Truth Table for the T-Flip Flop

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

Summarized as:

T	Q_{n+1}
0	Q_n
1	$\bar{Q}_n$

4.2.5 The JK flip-flop

- The JK flip-flop is shown below:

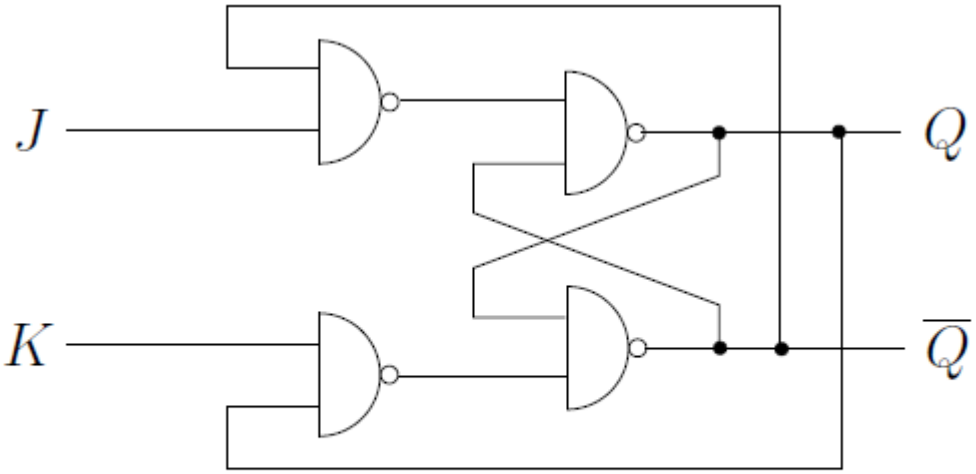


Figure 4.8: JK flip-flop

J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\bar{Q}_n$

- From the truth-table above, we can see that the JK flip-flop does not have invalid inputs and it can complement. These properties make the JK flip-flop **very versatile**, and it is used in many sequential logic circuits.

Next Lecture…

4.3 Clock signals and clocked Flip-Flops

End of session

Questions….?