Chapter 11 - Digital Logic

Luis Tarrataca
luis.tarrataca@gmail.com

CEFET-RJ

Table of Contents I

Table of Contents II

Table of Contents I

Table of Contents II

Introduction

Today's class will be a revision of digital logic elements:

- Boolean algebra.
- Gates:
  - AND, OR, NOT, XOR, ...
- Combinatorial logic:
  - Multiplexers, Decoders, Adders, ...
- Sequential logic:
  - Flip-flops, registers, counters, ...

Computer architecture builds on these concepts to develop new ones.

Lets see how to review an entire semestre of concepts in a single class =)

Boolean Algebra

Boolean algebra makes use of logical variables and operations:

- A variable may take on the value 1 (TRUE) or 0 (FALSE).

What are some of the logical operations that you know?

What are some of the logical operations that you know?

- NOT
- AND
- OR
- ...

Lets see how well you remember these operations: volunteers?

P	Q	NOT P	P AND Q	P OR Q	P NAND Q	P NOR Q	P XOR Q	P XNOR Q
0	0							
0	1							
1	0							
1	1							

P	Q	NOT P	P AND Q	P OR Q	P NAND Q	P NOR Q	P XOR Q	P XNOR Q
0	0	1	0	0	1	1	0	1
0	1	1	0	1	1	0	1	0
1	0	0	0	1	1	0	1	0
1	1	0	1	1	0	0	0	1

What if we have more than two variables? Any ideas?

What if we have more than two variables? Any ideas?

Operation	Expression	Output = 1 if
AND	$A \cdot B \cdot \ldots$	All of the set {A, B, ...} are 1.
OR	$A + B + \ldots$	Any of the set {A, B, ...} are 1.
NAND	$\overline{A \cdot B \cdot \ldots}$	Any of the set {A, B, ...} are 0.
NOR	$\overline{A + B + \ldots}$	All of the set {A, B, ...} are 0.
XOR	$A \oplus B \oplus \ldots$	The set {A, B, ...} contains an odd number of ones.

Figure: Boolean operators extended to more than two inputs (Source: (Stallings, 2015))

Basic Postulates		
$A \cdot B = B \cdot A$	$A + B = B + A$	Commutative Laws
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	$A + (B \cdot C) = (A + B) \cdot (A + C)$	Distributive Laws
$1 \cdot A = A$	$0 + A = A$	Identity Elements
$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$	Inverse Elements
Other Identities		
$0 \cdot A = 0$	$1 + A = 1$	
$A \cdot A = A$	$A + A = A$	
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$	Associative Laws
$\overline{A \cdot B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \cdot \overline{B}$	DeMorgan's Theorem

Figure: Basic identities of boolean algebra (Source: (Stallings, 2015))

Exercises (1/6)

Use the previous table to simplify the following expressions:

1. $X + XY$
2. $XY + X\bar{Y}$
3. $X + \bar{X}Y$
4. $X(X + Y)$
5. $(X + Y)(X + \bar{Y})$
6. $X(\bar{X} + Y)$
7. $\bar{X}YZ + \bar{X}Y\bar{Z} + XZ$
8. $XY + \bar{X}Z + YZ$
9. $(A + B)(\bar{A} + C)$

Exercises (2/6)

$$X + XY = X(1 + Y) = X$$

$$XY + X\bar{Y} = X(Y + \bar{Y}) = X$$

$$X + \bar{X}Y = (X + \bar{X})(X + Y) = X + Y$$

Exercises (3/6)

$$X(X+Y) = X + XY = X(1+Y) = X$$

$$(X+Y)(X+\bar{Y}) = XX + X\bar{Y} + XY + Y\bar{Y} = XX + X(Y+\bar{Y}) = X(1+X) = X$$

$$X(\bar{X}+Y) = X\bar{X} + XY = XY$$

Exercises (4/6)

$$
\begin{aligned}
XY + \bar{X}Z + YZ &= XY + \bar{X}Z + YZ(X + \bar{X}) \\
&= XY + \bar{X}Z + XYZ + \bar{X}YZ \\
&= XY + XYZ + \bar{X}Z + \bar{X}YZ \\
&= XY(1 + Z) + \bar{X}Z(1 + Y) \\
&= XY + \bar{X}Z
\end{aligned}
$$

Exercises (5/6)

$$
\begin{aligned}
(A+B)(\bar{A}+C) &= A\bar{A} + AC + \bar{A}B + BC \\
&= AC + \bar{A}B + BC \\
&= AC + \bar{A}B + BC(A+\bar{A}) \\
&= AC + \bar{A}B + ABC + \bar{A}BC \\
&= AC + ABC + \bar{A}B + \bar{A}BC \\
&= AC(1+B) + \bar{A}B(1+C) \\
&= AC + \bar{A}B
\end{aligned}
$$

Exercises (6/6)

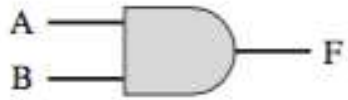
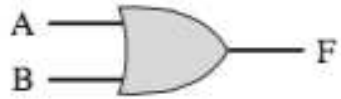
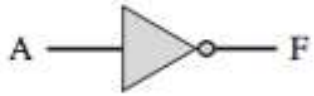
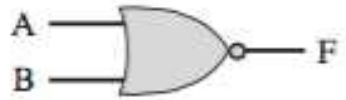
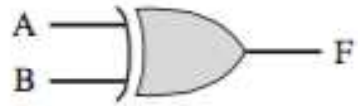
Prove the following Boolean equations using algebraic manipulation:

1. $\bar{X}\bar{Y} + \bar{X}Y + XY = \bar{X} + Y$
2. $\bar{A}B + \bar{B}\bar{C} + AB + \bar{B}C = 1$
3. $Y + \bar{X}Z + X\bar{Y} = X + Y + Z$
4. $\bar{X}\bar{Y} + \bar{Y}Z + XZ + XY + Y\bar{Z} = \bar{X}\bar{Y} + XZ + Y\bar{Z}$

Gates

Fundamental building block of all digital logic circuits is the gate.

- Logical functions are implemented by the interconnection of gates.
- Basic gates used are AND, OR, NOT, NAND, NOR, and XOR.

Name	Graphical Symbol	Algebraic Function	Truth Table
AND		$F = A \cdot B$ or $F = AB$	A B | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = A + B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT		$F = \overline{A}$ or $F = A'$	A | F 0 | 1 1 | 0
NAND		$F = \overline{AB}$	A B | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = \overline{A + B}$	A B | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
XOR		$F = A \oplus B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0

There are a lot of gates. Do we really need all of these gates? Any ideas?

There are a lot of gates. Do we really need all of these gates? Any ideas?

- Ever heard of universal gates?

Universal Gates

The NAND and NOR gates are also known as universal gates:

- Any Boolean function can be implemented using only them;
- Lets have a look at some examples:
  - NOT gate;
  - AND gate;
  - OR gate;
  - NOR gate;

Universal Gates :: Obtaining the NOT Operator (1/2)

How can we use a NAND gate to obtain the NOT operator? Any ideas?

Universal Gates :: Obtaining the NOT Operator (1/2)

How can we use a NAND gate to obtain the NOT operator? Any ideas?

- What happens when we duplicate the same input on a NAND gate?

Universal Gates :: Obtaining the NOT Operator (1/2)

How can we use a NAND gate to obtain the NOT operator? Any ideas?

- What happens when we duplicate the same input on a NAND gate?

P	Q	P NAND Q
0	0	1
1	1	0

Universal Gates :: Obtaining the NOT Operator (2/2)

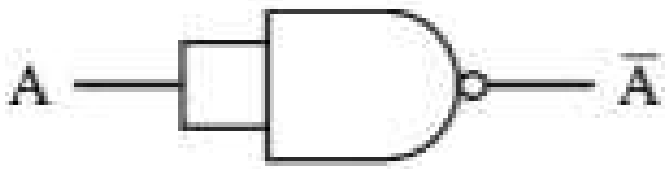


Figure: NOT operation achieved through a NAND gate (Source: (Stallings, 2015))

Universal Gates :: Obtaining the AND Operator (1/2)

How can we use a NAND gate to obtain the AND operator? Any ideas?

Universal Gates :: Obtaining the AND Operator (1/2)

How can we use a NAND gate to obtain the AND operator? Any ideas?

Remember the algebraic properties?

$$AB = \overline{\overline{AB}}$$

Universal Gates :: Obtaining the AND Operator (2/2)

Remember the algebraic properties?

$$AB = \overline{\overline{AB}}$$

Figure: AND operation achieved through a NAND gate (Source: (Stallings, 2015))

Universal Gates :: Obtaining the OR Operator (1/2)

How can we use a NAND gate to obtain the OR operator? Any ideas?

Universal Gates :: Obtaining the OR Operator (1/2)

How can we use a NAND gate to obtain the OR operator? Any ideas?

Remember the algebraic properties?

$$\begin{aligned} A + B &= \overline{\overline{A + B}} \\ &= \overline{\bar{A}.\bar{B}} \end{aligned}$$

Universal Gates :: Obtaining the OR Operator (2/2)

Remember the algebraic properties?

$$A + B = \overline{\overline{A + B}}$$
$$= \overline{\overline{A}.\overline{B}}$$

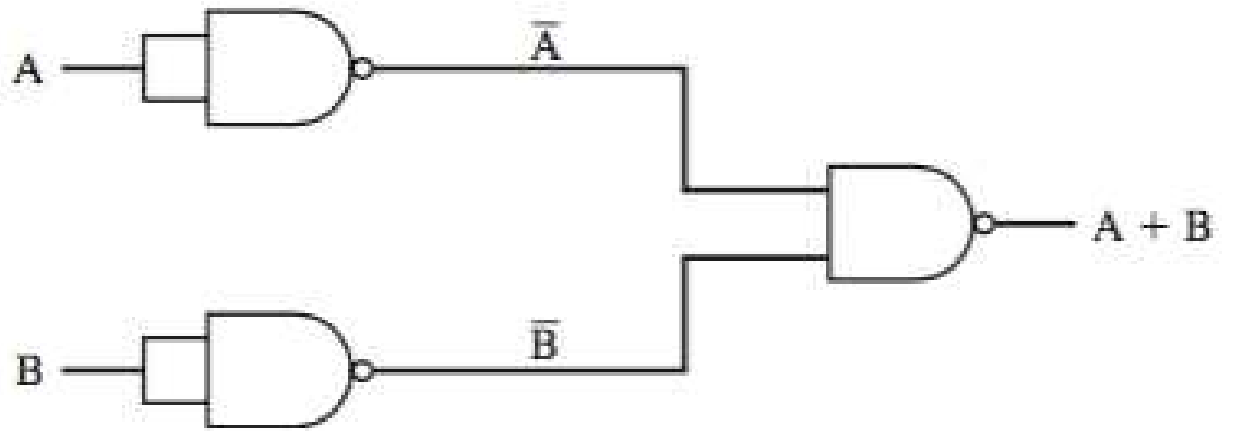


Figure: OR operation achieved through a NAND gate (Source: (Stallings, 2015))

Universal Gates :: Obtaining the NOR Operator (1/2)

How can we use a NAND gate to obtain the NOR operator? Any ideas?

- Recall that the NOR operator is also a universal gate;
- Therefore there is a mapping between NAND and NOR;

Universal Gates :: Obtaining the NOR Operator (1/2)

How can we use a NAND gate to obtain the NOR operator? Any ideas?

- Recall that the NOR operator is also a universal gate;
- Therefore there is a mapping between NAND and NOR;

$$\begin{aligned}\overline{A+B} &= \overline{\overline{A+B}} \\ &= \overline{\overline{A}.\overline{B}}\end{aligned}$$

$$\overline{A+B} = \overline{\overline{A+B}}$$
$$= \overline{\overline{A}.\overline{B}}$$

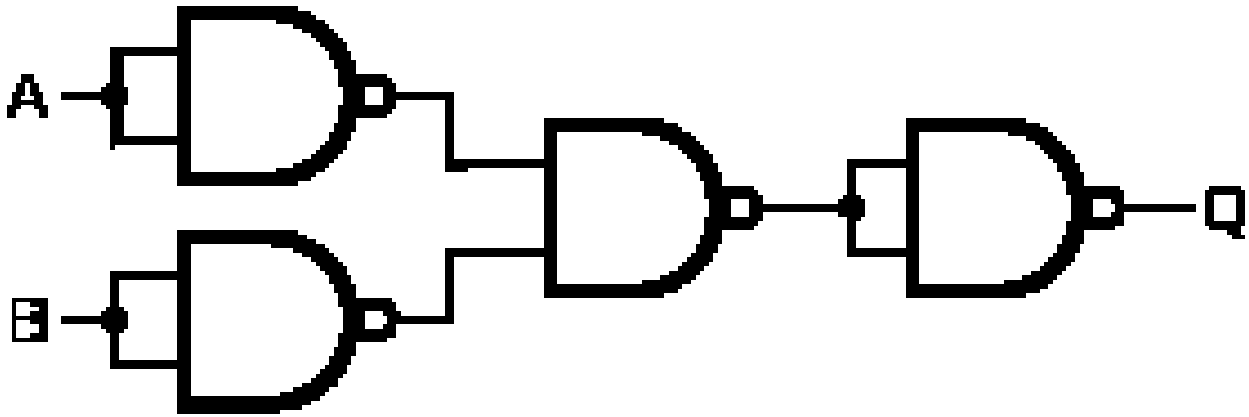


Figure: NOR operation achieved through a NAND gate (Source: wikipedia)

Now that we have a basic understanding of the logical operations:

- Lets see how we can combine these elements to calculate functions;

Combinatorial Circuit

A set of interconnected gates

- Output at any time is a function only of the input at that time;
- Consists of n binary inputs and m binary outputs
- Can be defined in three ways:
  - Truth table;
  - Graphical symbols;
  - Boolean equations

Example

Consider the following truth table for a boolean function:

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

Figure: A boolean function of three variables (Source: (Stallings, 2015))

F can be expressed by itemizing the combinations of either:

- Sum of minterms that have value 1;

$$\sum m(2, 3, 6)$$

- Product of maxterms that have value 0;

$$\Pi M(0, 1, 4, 5, 7)$$

F can be expressed by itemizing the combinations of either:

- sum of minterms that have value 1;

$$\sum m(2,3,6) = \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C}$$

- product of maxterms that have value 0;

$$\Pi M(0,1,4,5,7) = (A+B+C)(A+B+\bar{C})(\bar{A}+B+C)(\bar{A}+B+\bar{C})(\bar{A}+\bar{B}+\bar{C})$$

Lets focus on the minterms:

$$F = \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C}$$

How can we obtain the equivalent logical circuit? Any ideas?

How can we obtain the equivalent logical circuit? Any ideas?

$$F = \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C}$$

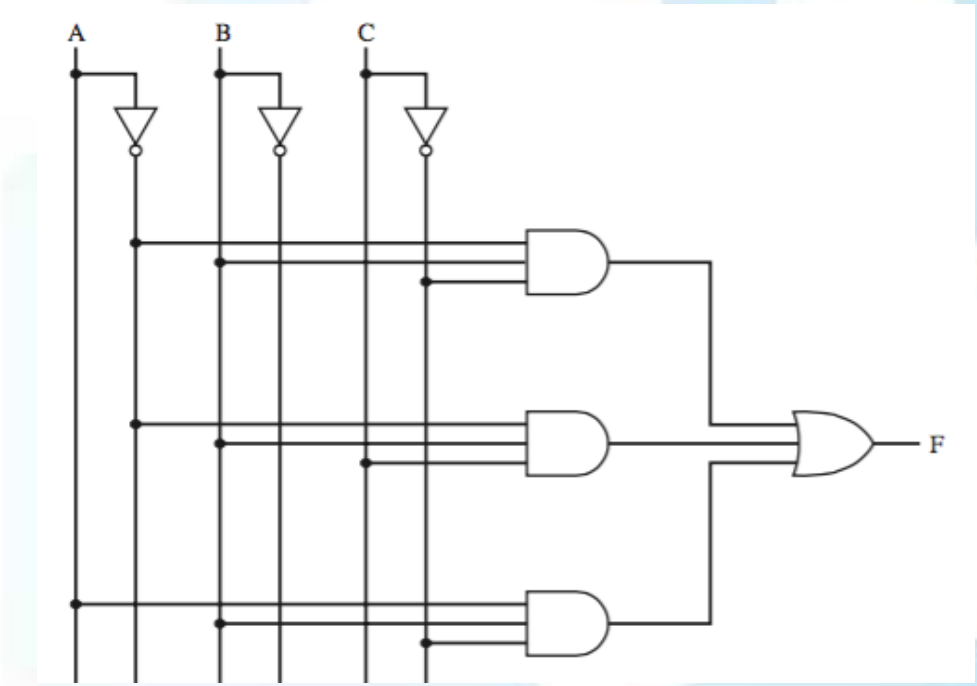


Figure: Circuit implementation (Source: (Stallings, 2015))

Can we do better than the previous logical circuit? Any ideas?

Can we do better than the previous logical circuit? Any ideas?

- Convenient to obtain the simplified form...

Can we do better than the previous logical circuit? Any ideas?

- Convenient to obtain the simplified form...

How can we obtain the simplified form? Any ideas?

How can we obtain the simplified form? Any ideas?

$$F = \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C}$$

How can we obtain the simplified form? Any ideas?

$$
\begin{aligned}
F &= \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C} \\
&= \bar{A}B(\bar{C} + C) + AB\bar{C} \\
&= \bar{A}B.1 + AB\bar{C} \\
&= B(\bar{A} + A\bar{C}) \\
&= B((\bar{A} + A)(\bar{A} + \bar{C})) \\
&= B(1(\bar{A} + \bar{C})) \\
&= \bar{A}B + B\bar{C} \\
&= B(\bar{A} + \bar{C})
\end{aligned}
$$

Once we have obtained the simplified form it is easy to obtain the equivalent circuit... Any ideas?

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

Once we have obtained the simplified form it is easy to obtain the equivalent circuit... Any ideas?

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

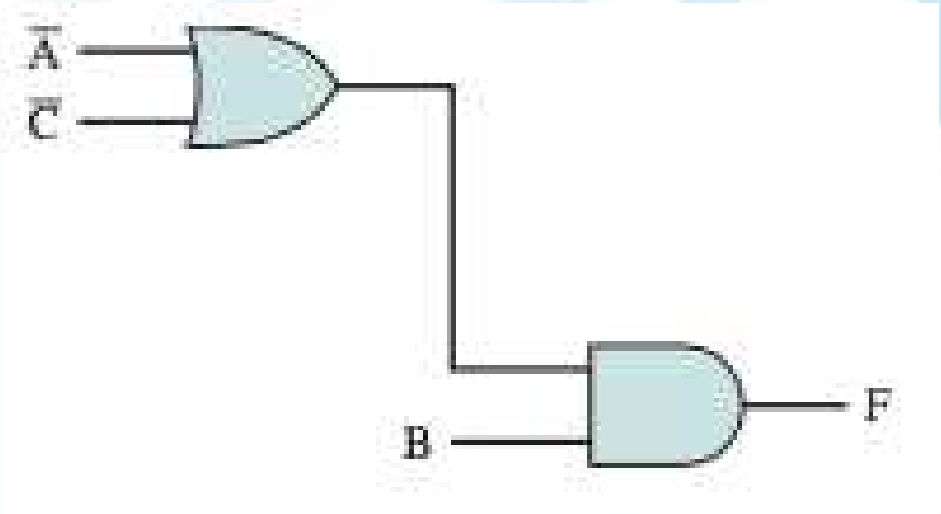


Figure: Simplified Circuit implementation (Source: (Stallings, 2015))

Karnaugh Maps

Algebraic simplification procedure is awkward:

- Lacks specific rules to predict each succeeding step;
- Difficult to determine if the simplest expression has been obtained;

Karnaugh map provides a way for simplifying boolean expressions:

- Up to four variables;
  - More than this becomes difficult to use.
- Takes advantage of humans' pattern-recognition capability.

This section is based on (Mano and Kime, 2002).

The map is a diagram made up of squares:

- Each square represents one **minterm** of the function;
- Visual diagram of all possible ways a function may be expressed;

Lets take a look at a three-variable map.

- Only one bit changes in value from one column to the other;
- Any two adjacent squares differ in only a single variable;

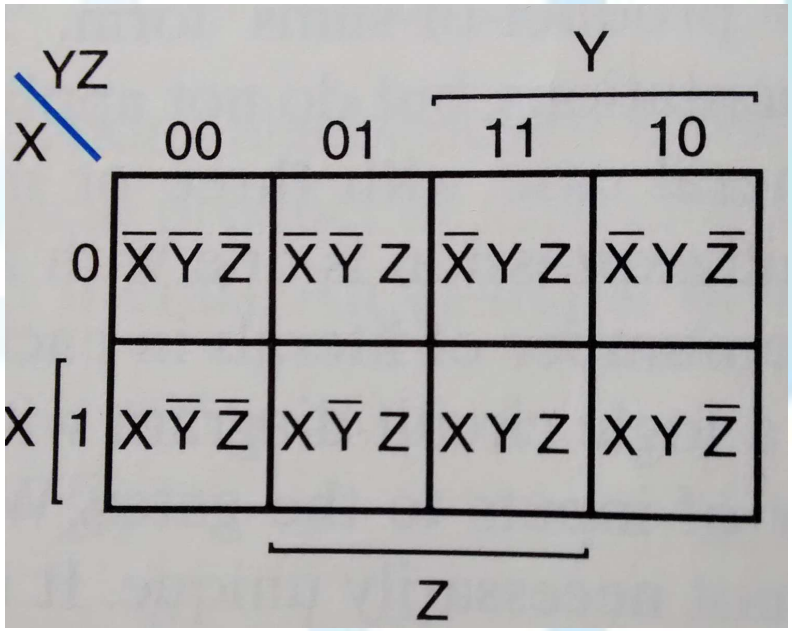


Figure: Three Variable Map (Source: (Mano and Kime, 2002))

- Any two minterms in adjacent squares produce a product of two variables.
  - Why?

- Lets see why...

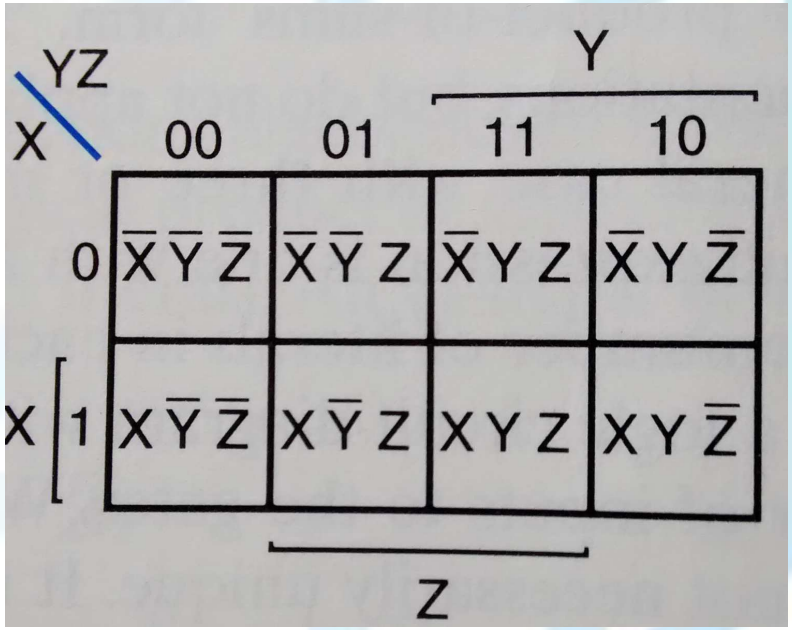


Figure: Three Variable Map (Source: (Mano and Kime, 2002))

- *E.g.:* $m_5 + m_7 = X\bar{Y}Z + XYZ$
  - *I.e.* $m_5 + m_7 = X\bar{Y}Z + XYZ = XZ(\bar{Y} + Y) = XZ$
  - The two squares differ in variable Y, which can be removed.

Karnaugh Map Example 1

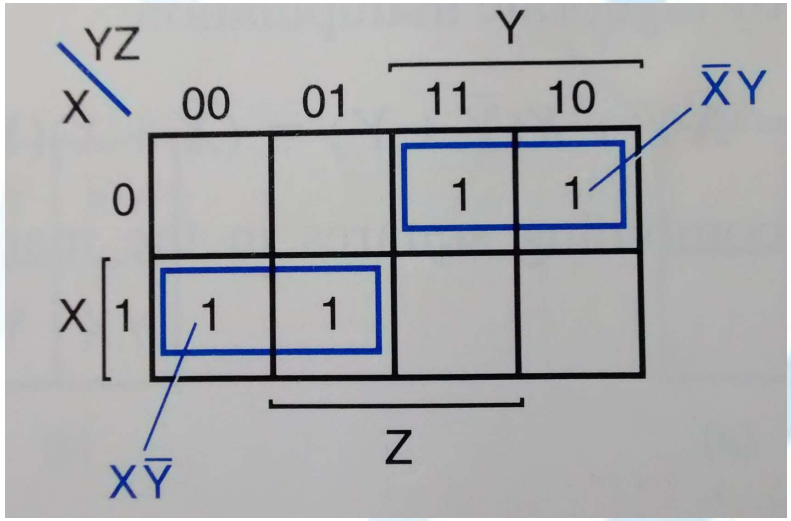


Figure: $\sum m(2,3,4,5) = \bar{X}Y + X\bar{Y}$ (Source: (Mano and Kime, 2002))

Karnaugh Map Example 2

Two squares can also be adjacent without touching each other:

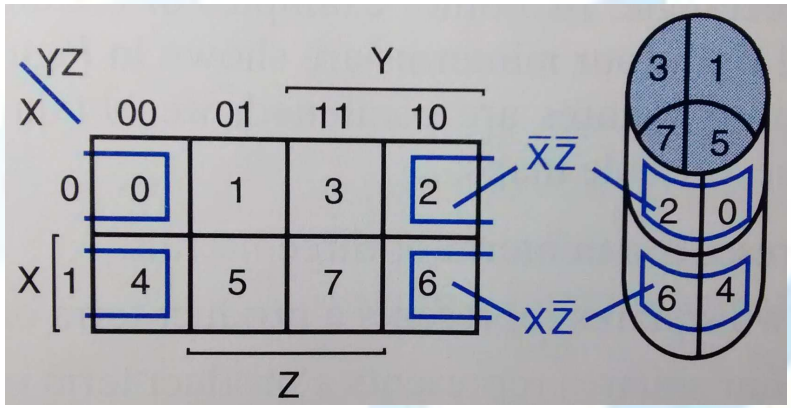


Figure: $\sum m(0, 2, 4, 6) = \bar{X}\bar{Z} + X\bar{Z}$ (Source: (Mano and Kime, 2002))

- The minterms continue to differ by one variable;

Karnaugh Map Example 3

It is also possible to combine 4 squares:

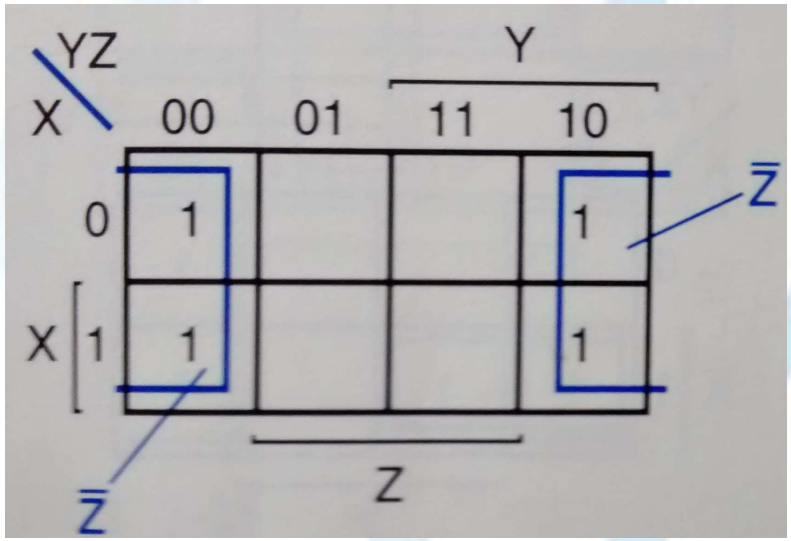


Figure: $\sum m(0,2,4,6) = \bar{X}\bar{Z} + X\bar{Z} = (\bar{X} + X)\bar{Z} = \bar{Z}$ (Source: (Mano and Kime, 2002))

Karnaugh Map Example 4

It is also possible to have several combinations:

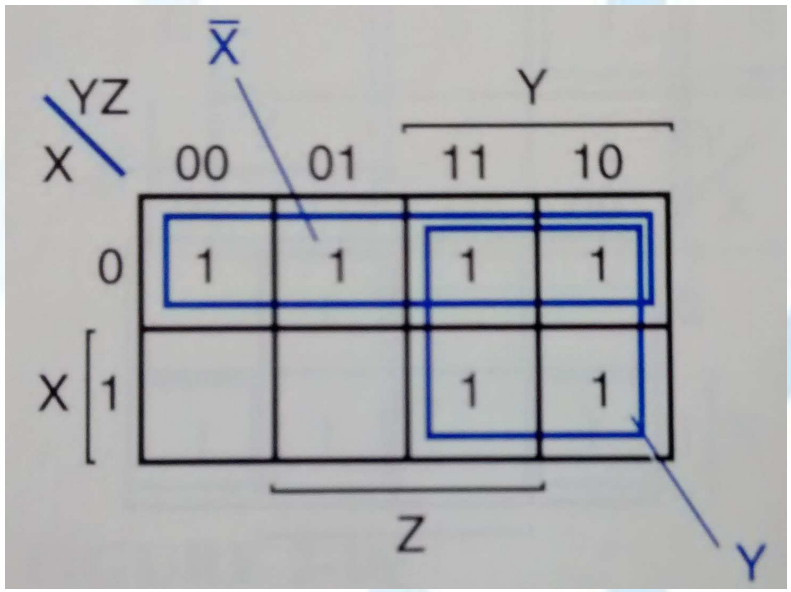


Figure: $\sum m(0, 1, 2, 3, 6, 7) = \bar{X} + Y$ (Source: (Mano and Kime, 2002))

The more squares are combined the fewer literals are used:

- One square represents three literals;
- Two squares represents a product term of two literals;
- Four squares represents a product term of one literal;
- Eight squares (entire map) is equal to value 1.

Now that we have a basic understanding of boolean algebra:

- Lets have a look at other types of gates...

Multiplexers

The multiplexer connects multiple inputs to a single output.

- At any time, one of the inputs is selected to be passed to the output.

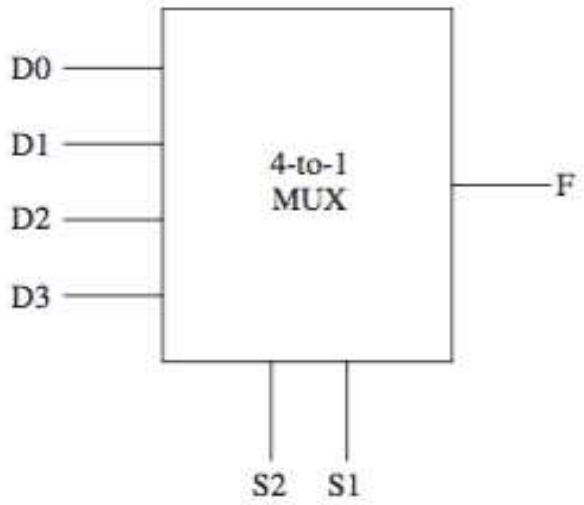


Figure: A 4-to-1 Multiplexer representation (Source: (Stallings, 2015))

S2	S1	F
0	0	D0
0	1	D1
1	0	D2
1	1	D3

Figure: A 4-to-1 Multiplexer truth table (Source: (Stallings, 2015))

- Four input lines (D0, D1, D2, and D3);
- Two selection lines (S0 and S1);

How can we implement a multiplexer using the logical gates we know? Any ideas?

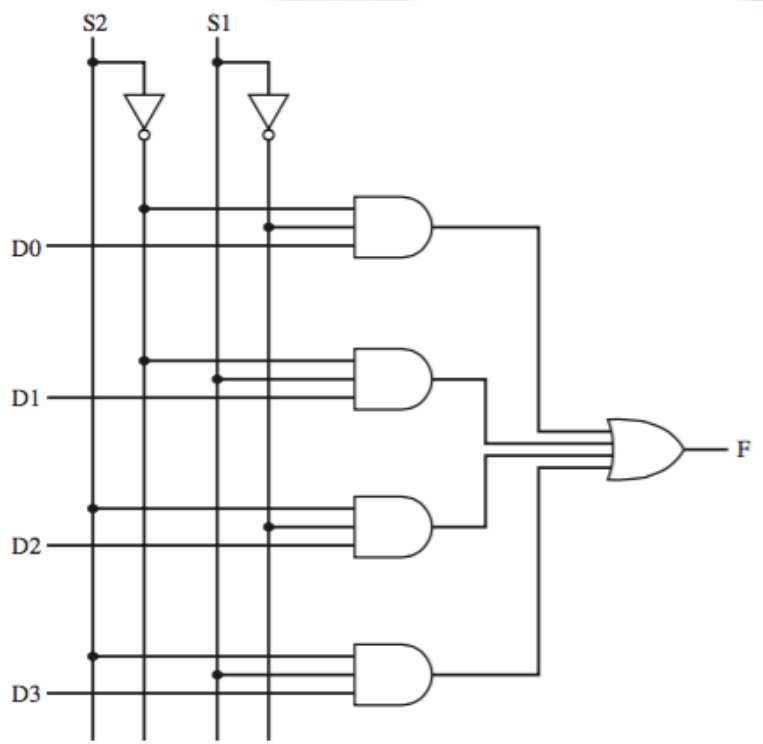


Figure: Multiplexer Implementation (Source: (Stallings, 2015))

Multiplexeres are useful for:

- To control signal and data routing;
  - *E.g.* loading PC from different sources;

Decoders

Combinational circuit with a number of output lines:

- Only one of which is asserted at any time, dependent on input;
- n inputs, 2^n output lines;

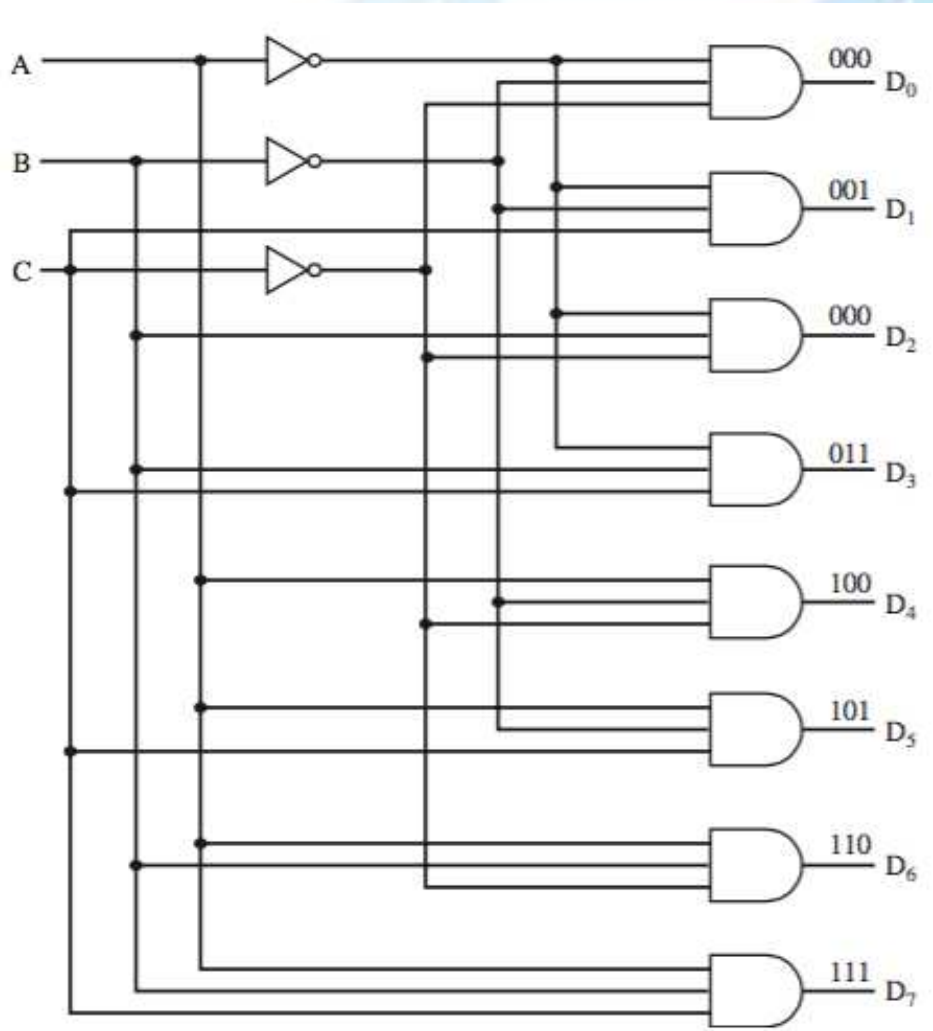


Figure: Decoder with 3 inputs and $2^3 = 8$ outputs (Source: (Stallings, 2015))

Adders

Binary addition differs from Boolean algebra in that the result includes a carry term.

$\begin{array}{r} 0 \\ +\,0 \\ \hline 0 \end{array}$	$\begin{array}{r} 0 \\ +\,1 \\ \hline 1 \end{array}$	$\begin{array}{r} 1 \\ +\,0 \\ \hline 1 \end{array}$	$\begin{array}{r} 1 \\ +\,1 \\ \hline 10 \end{array}$

(a) Single-Bit Addition			
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(b) Addition with Carry Input				
C_{in}	A	B	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure: Binary addition truth tables (Source: (Stallings, 2015))

The two outputs can be expressed:

- $Sum = \overline{C_{in}}\overline{A}B + \overline{C_{in}}A\overline{B} + C_{in}\overline{A}\overline{B} + C_{in}AB$
- $C_{out} = \overline{C_{in}}AB + C_{in}\overline{A}B + C_{in}A\overline{B} + C_{in}AB$

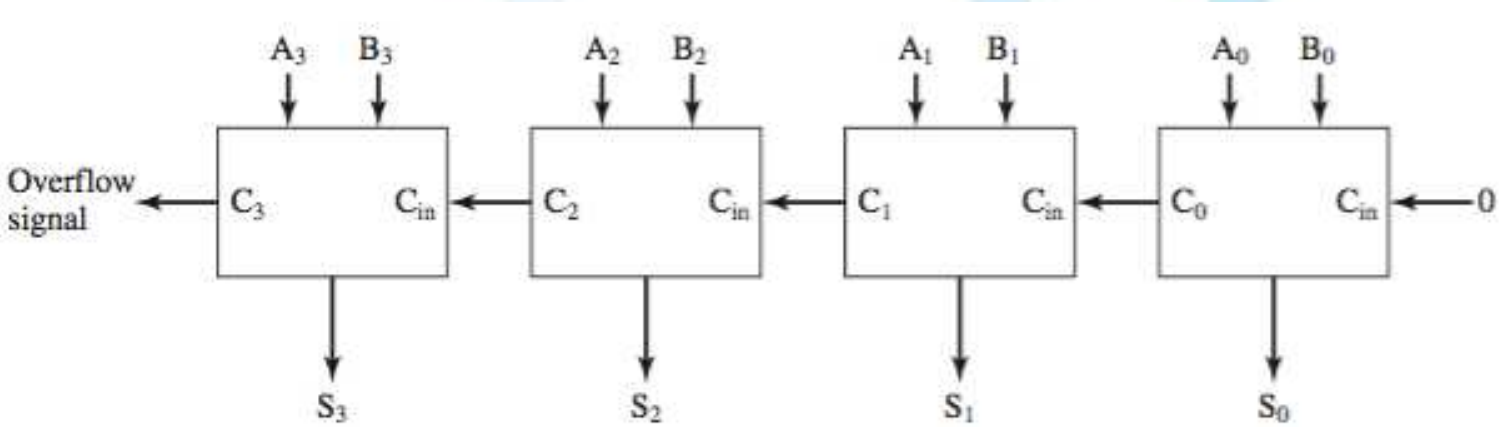


Figure: 4 bit adder (Source: (Stallings, 2015))

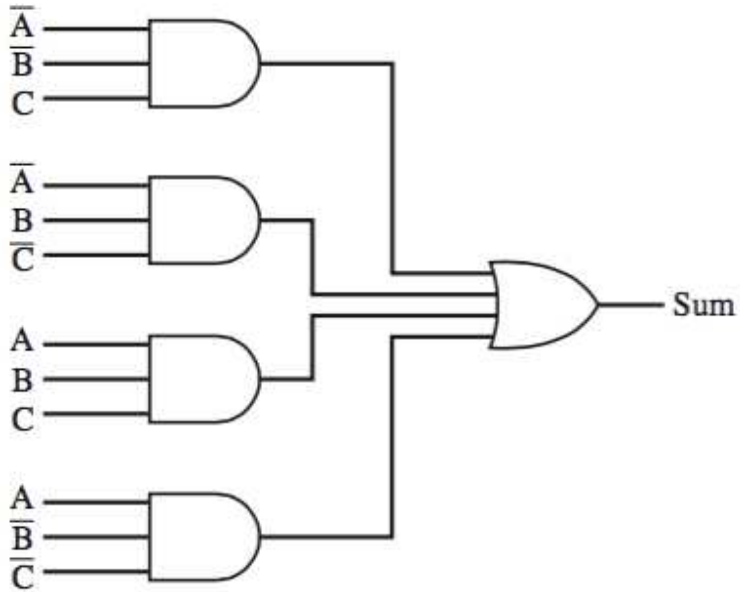


Figure: Implementation of an adder using AND, OR and NOT gates (Source: (Stallings, 2015))

Sequential Circuits

Combinational circuits implement the essential functions of a computer.

- However, they provide no memory or state information,

In sequential circuits the output depends:

- Not only on the current input...
- But also of the current circuit state;

Useful examples of sequential circuits:

- Flip-flops;
- Registers;
- Counters.

Useful examples of sequential circuits:

- Flip-flops;
- Registers;
- Counters.

Guess what we will be seeing today! Any ideas?

Flip-flops

- Simplest form of the sequential circuit;
- A variety of flip-flops exist, all of which share two properties:
  - Can maintain a binary state indefinitely*:
    - Until directed by an input signal to switch states;
    - The flip-flop can function as a 1-bit memory;
    - *As long as powered is deliver to the circuit.
  - Has two outputs, these are generally labeled Q and $\overline{Q}$.

Before we go any further into our presentation:

Does anyone have any idea how flip-flops are implemented?

Before we go any further into our presentation:

Does anyone have any idea how flip-flops are implemented?

Several possibilities:

- Science / Engineering;
- Magic ;)

SR flip-flops

SR circuit has:

- Two inputs S (Set) and R (Reset);
- Two outputs Q and $\overline{Q}$;
- Two NOR gates connected in a **feedback** arrangement;

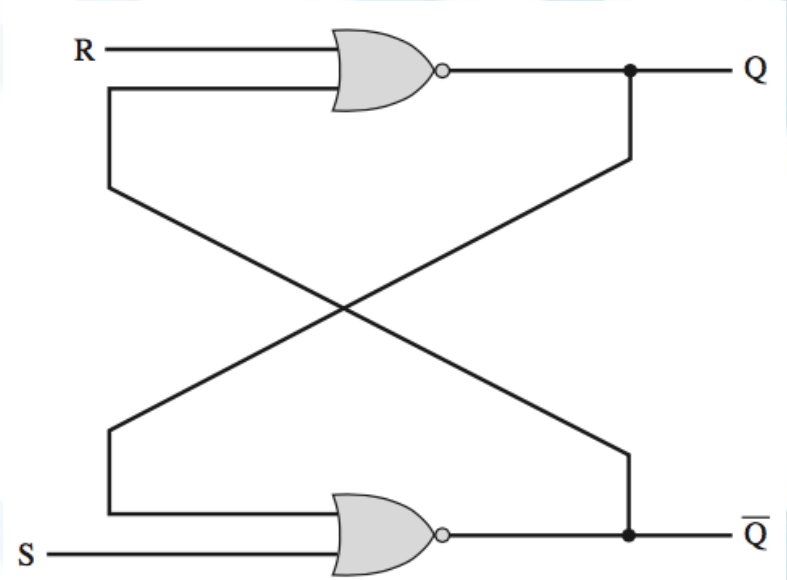


Figure: The S-R Latch implemented with NOR Gates (Source: (Stallings, 2015))

Circuit functions as a 1-bit memory:

- Inputs **S**et and **R**eset serve to write the values 1 and 0 to Q;
- Consider the state $Q = 0, \overline{Q} = 1, S = 0, R = 0$

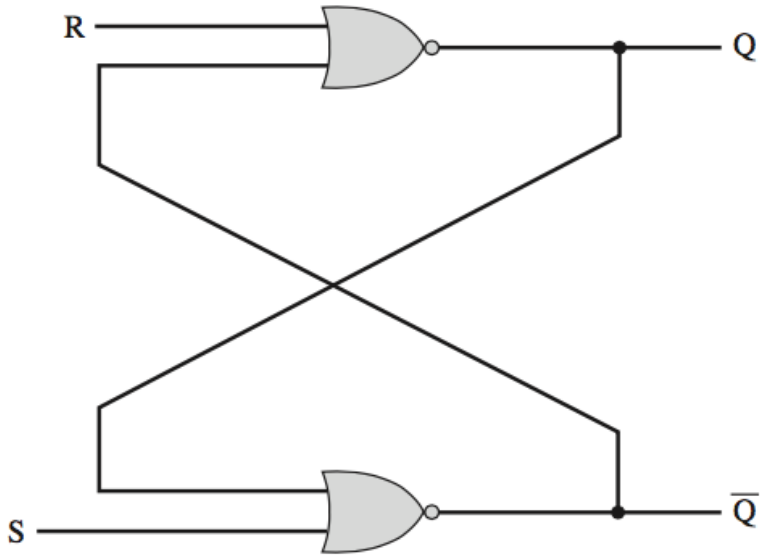


Figure: S-R Latch implemented with NOR Gates (Source: (Stallings, 2015))

Suppose that S changes to the value 1.

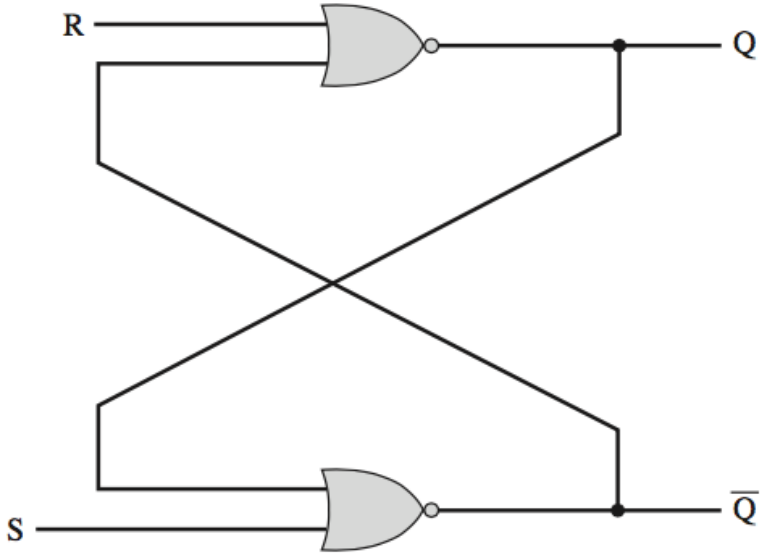


Figure: S-R Latch implemented with NOR Gates (Source: (Stallings, 2015))

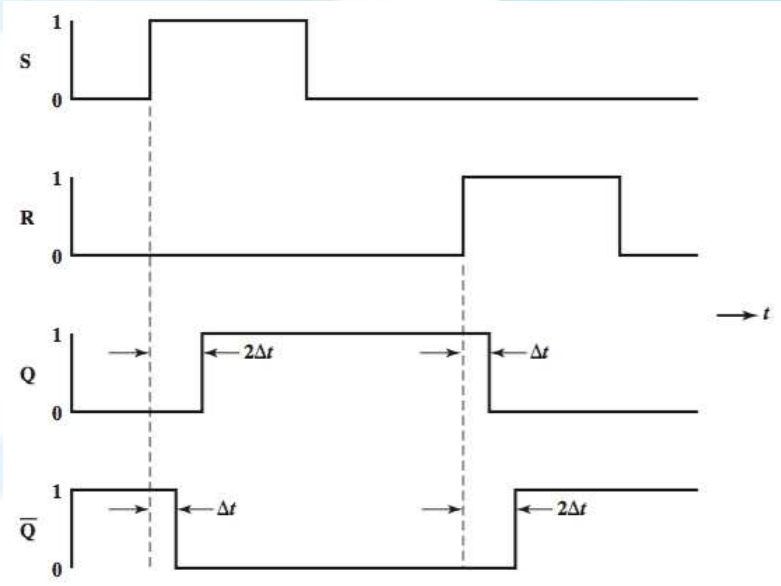


Figure: NOR S-R Latch timing Diagram (Source: (Stallings, 2015))

1. Now the inputs to the lower NOR gate are S = 1, Q = 0.
2. After some time delay Δt, the output of the lower NOR gate will be $\bar{Q} = 0$.

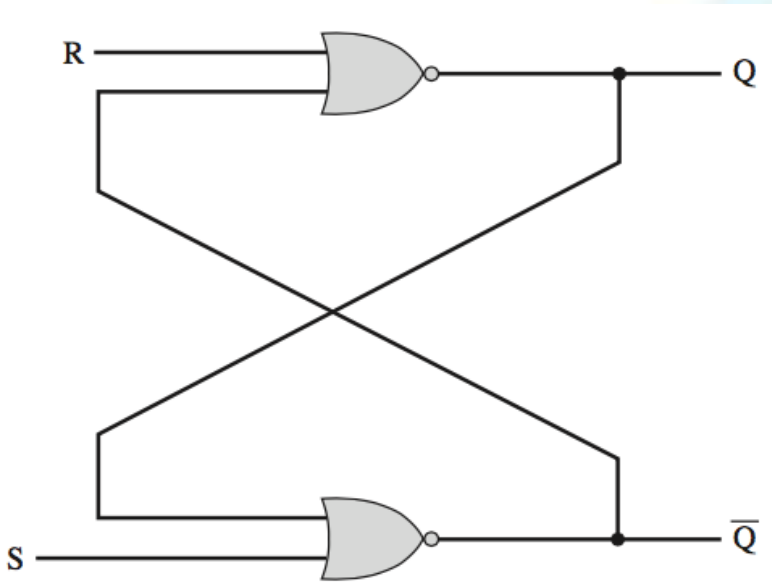


Figure: S-R Latch implemented with NOR Gates (Source: (Stallings, 2015))

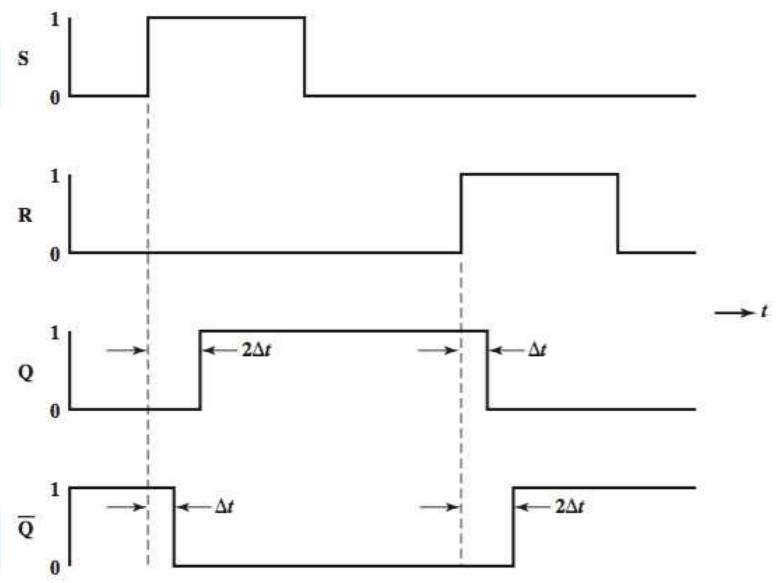


Figure: NOR S-R Latch timing Diagram (Source: (Stallings, 2015))

3. The inputs to the upper NOR gate become $R = 0, \overline{Q} = 0$.
4. After another gate delay of Δt, the output Q becomes 1.

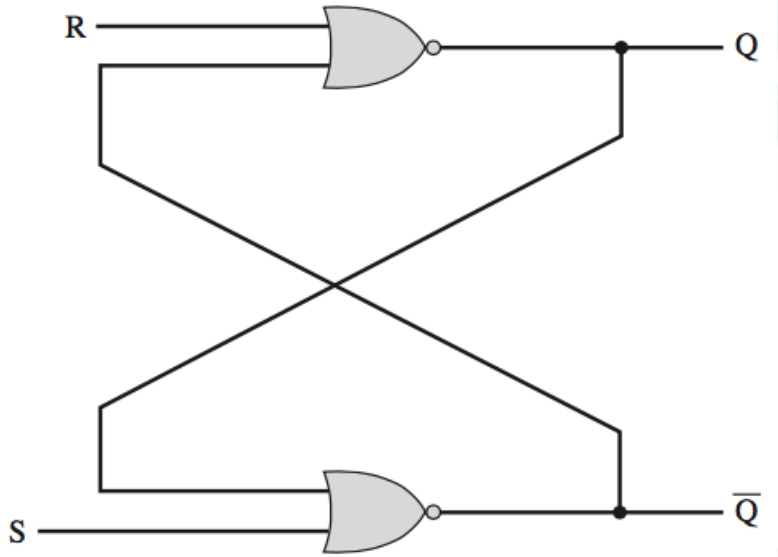


Figure: S-R Latch implemented with NOR Gates (Source: (Stallings, 2015))

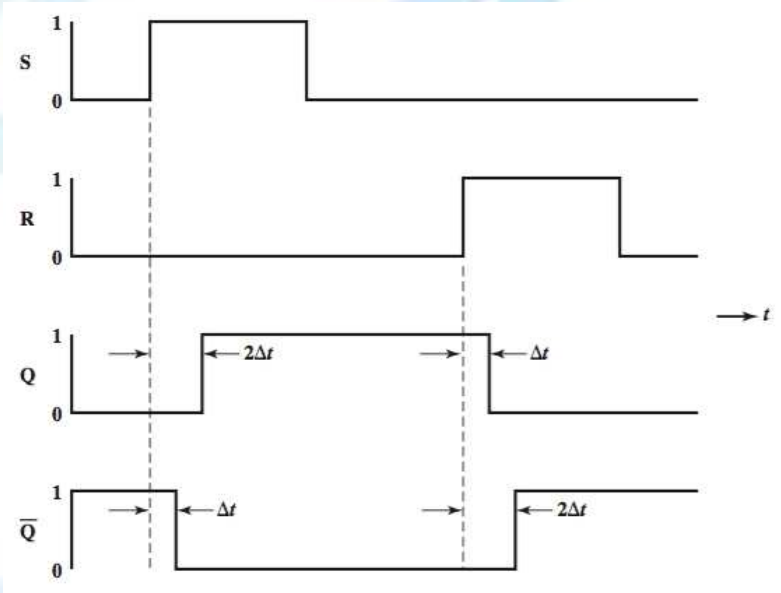


Figure: NOR S-R Latch timing Diagram (Source: (Stallings, 2015))

5. This is a stable state. The inputs to the lower gate are now S = 1, Q = 1, which maintain the output $\bar{Q} = 0$.
   - As long as $S = 1$ and $R = 0$, the outputs will remain $Q = 1$, $Q = 0$.
   - Furthermore, if S returns to 0, the outputs will remain unchanged.

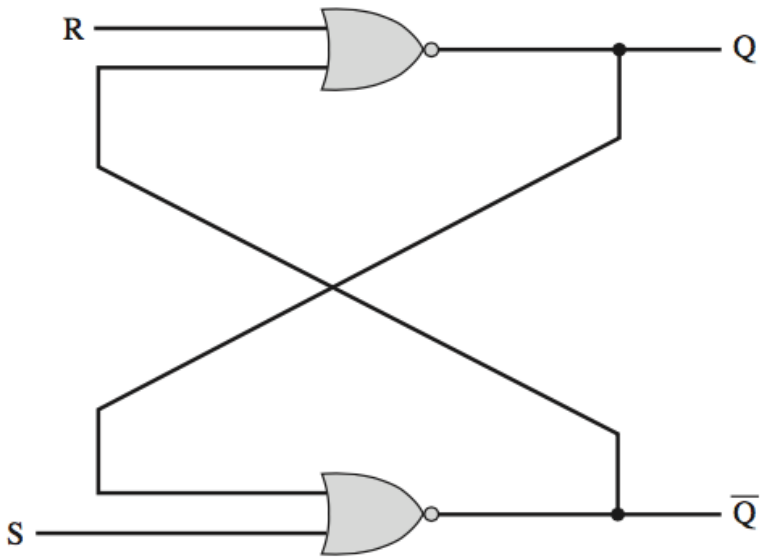


Figure: S-R Latch implemented with NOR Gates (Source: (Stallings, 2015))

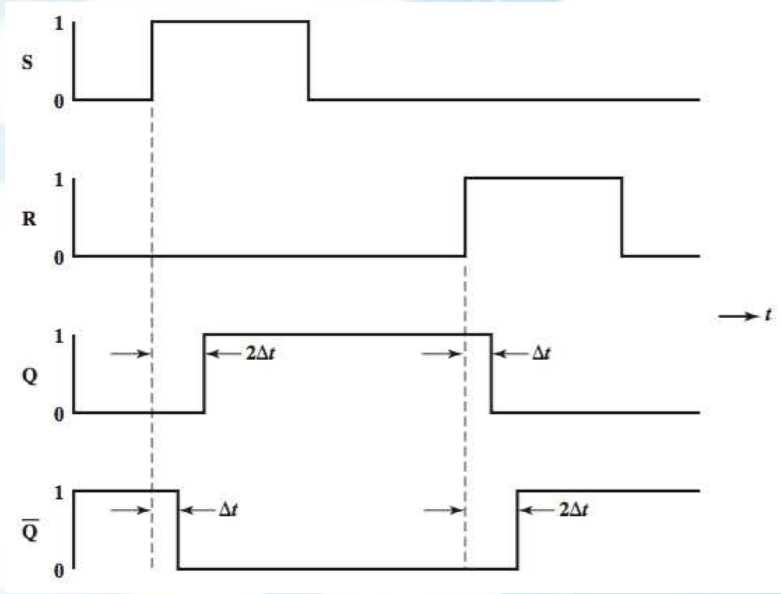


Figure: NOR S-R Latch timing Diagram (Source: (Stallings, 2015))

6. The R output performs the opposite function.
   - When R goes to 1, it forces $Q = 0, \bar{Q} = 1$
     - Regardless of the previous state of Q and $\bar{Q}$.
   - Again, a time delay of $2\Delta t$ occurs before the final state is established

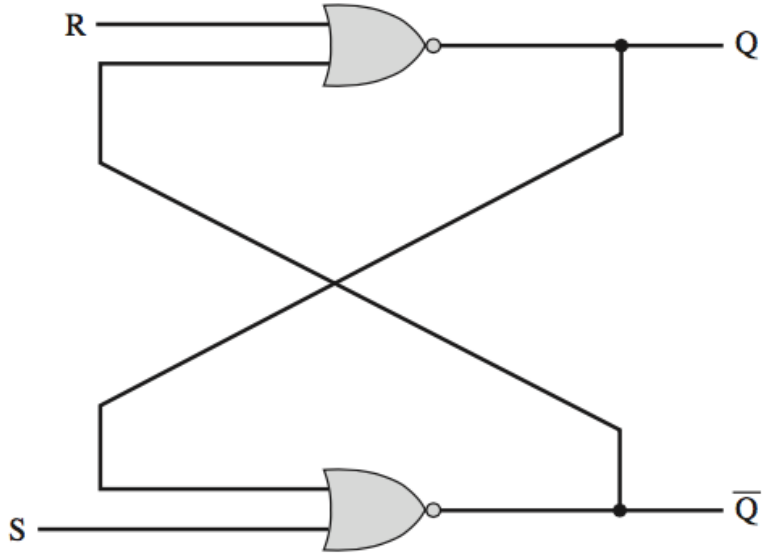


Figure: S-R Latch implemented with NOR Gates (Source: (Stallings, 2015))

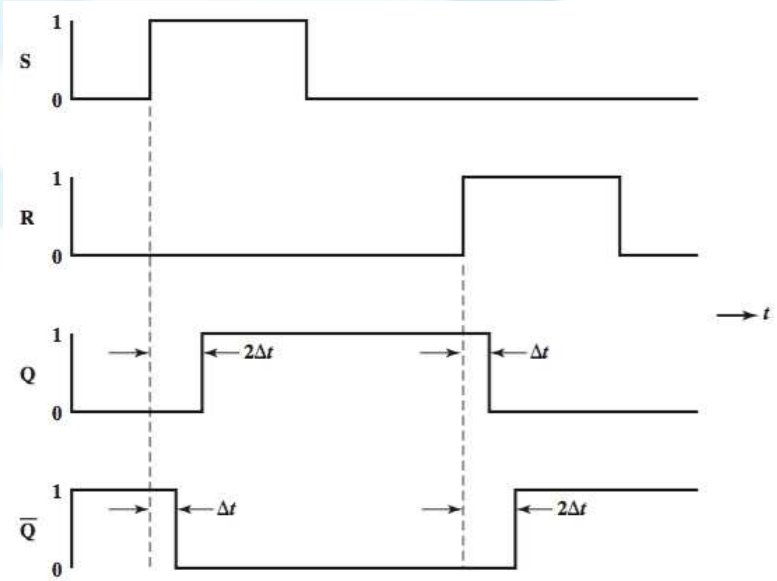


Figure: NOR S-R Latch timing Diagram (Source: (Stallings, 2015))

In essence:

- $S = 1$ makes $\bar{Q} = 0$
  - If $S = 1$ then $R = 0$ which makes $Q = 1$
- $R = 1$ makes $Q = 0$
  - If $R = 1$ then $S = 0$ which makes $\bar{Q} = 1$

This behaviour can be described by a characteristic table:

(a) Characteristic Table		
Current Inputs	**Current State**	**Next State**
SR	Q_n	Q_{n+1}
00	0	0
00	1	1
01	0	0
01	1	0
10	0	1
10	1	1
11	0	—
11	1	—

Figure: (Source: (Stallings, 2015))

This behaviour can be described by a characteristic table:

(a) Characteristic Table		
Current Inputs	**Current State**	**Next State**
SR	Q_n	Q_{n+1}
00	0	0
00	1	1
01	0	0
01	1	0
10	0	1
10	1	1
11	0	—
11	1	—

Figure: (Source: (Stallings, 2015))

But what happens when the inputs are set to $S = 1, R = 1$? Any ideas?

This behaviour can be described by a characteristic table:

(a) Characteristic Table		
Current Inputs	**Current State**	**Next State**
SR	Q_n	Q_{n+1}
00	0	0
00	1	1
01	0	0
01	1	0
10	0	1
10	1	1
11	0	—
11	1	—

Figure: (Source: (Stallings, 2015))

Inputs $S = 1, R = 1$ are **not allowed**:

- Would produce the inconsistent output $Q = \bar{Q} = 0$

It is also possible to derive a simplified version:

(b) Simplified Characteristic Table		
S	**R**	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	—

Figure: (Source: (Stallings, 2015))

Example

Lets look at a particular example:

Response to Series of Inputs										
t	0	1	2	3	4	5	6	7	8	9
S	1	0	0	0	0	0	0	0	1	0
R	0	0	0	1	0	0	1	0	0	0
Q_{n+1}										

Example

Lets look at a particular example:

Response to Series of Inputs										
t	0	1	2	3	4	5	6	7	8	9
S	1	0	0	0	0	0	0	0	1	0
R	0	0	0	1	0	0	1	0	0	0
$\mathbf{Q_{n+1}}$	1	1	1	0	0	0	0	0	1	1

Typically events in the digital computer are synchronized to a clock pulse,

- Changes occur only when a clock pulse occurs;
- R and S inputs are passed to the NOR gates only during the clock pulse.

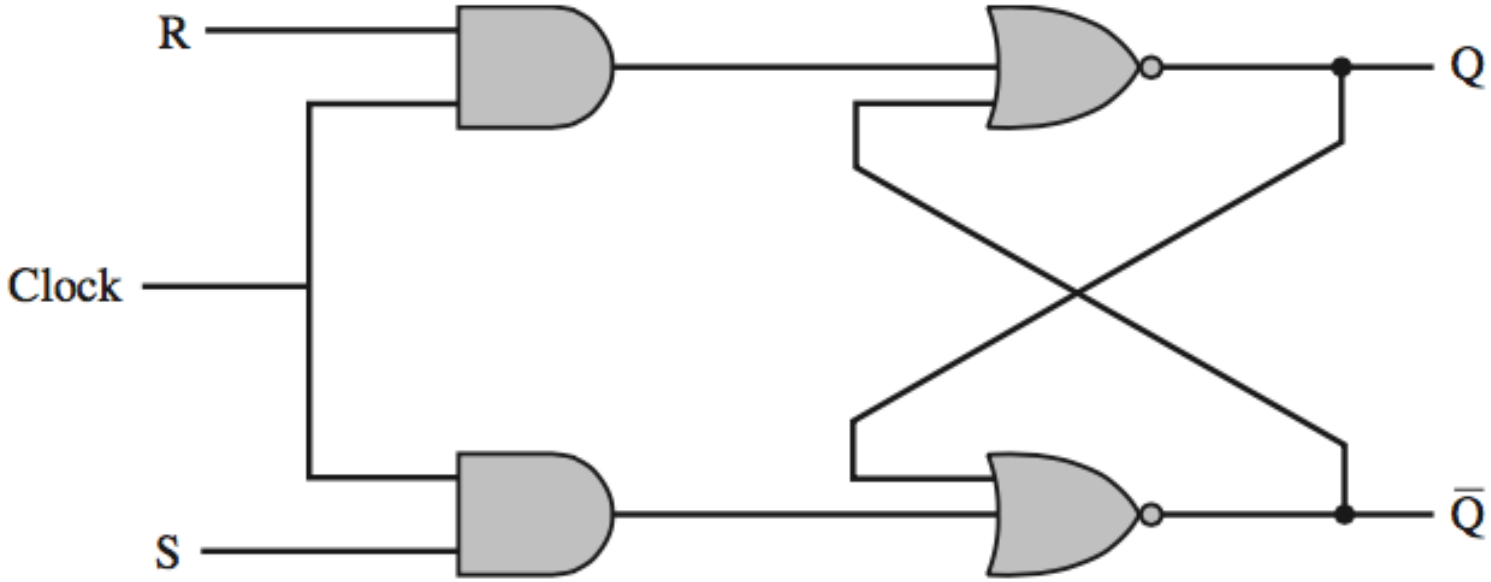


Figure: Clocked SR flip-flops (Source: (Stallings, 2015))

D flip-flops

Problem with S-R flip-flop, the condition R = 1, S = 1 must be avoided.

How can we be sure that these inputs are not allowed? Any ideas?

D flip-flops

Problem with S-R flip-flop, the condition R = 1, S = 1 must be avoided.

- One way to do this is to allow just a single input.

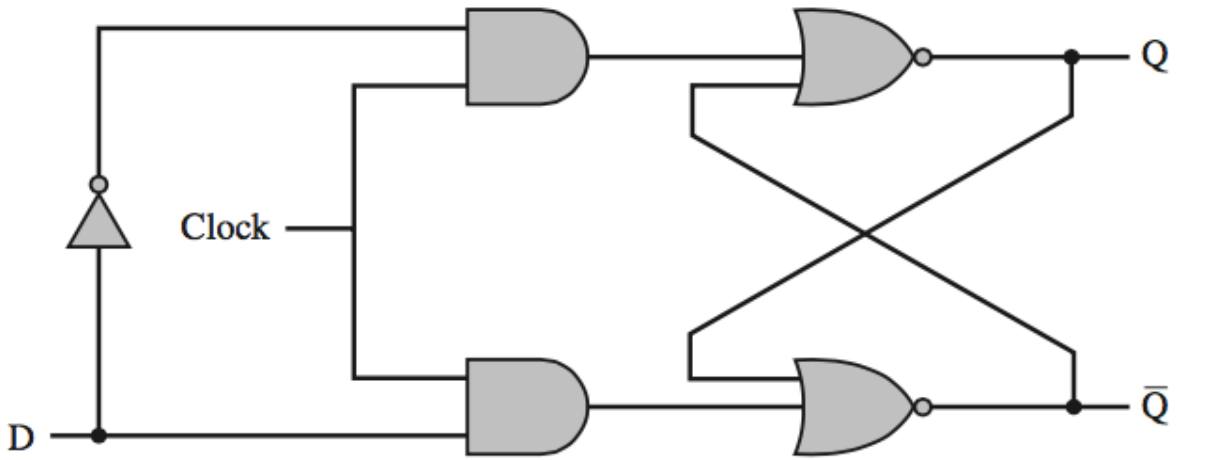


Figure: (D flip-flops (Stallings, 2015))

- By using a NOT gate:
  - Nonclock inputs are the opposite of each other.

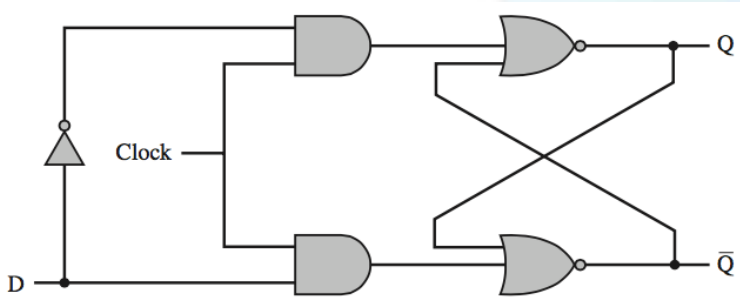


Figure: D flip-flops (Stallings, 2015)

Clock	D	Q_{n+1}
0	0	Q_n
0	1	Q_n
1	0	0
1	1	1

- Flip-flop captures the value of the D-input during the clock cycle;
- Captured value becomes the Q output.
- Other times, the output Q does not change.

J-K flip-flops (1/3)

Has two inputs, with all possible combinations of inputs values being valid:

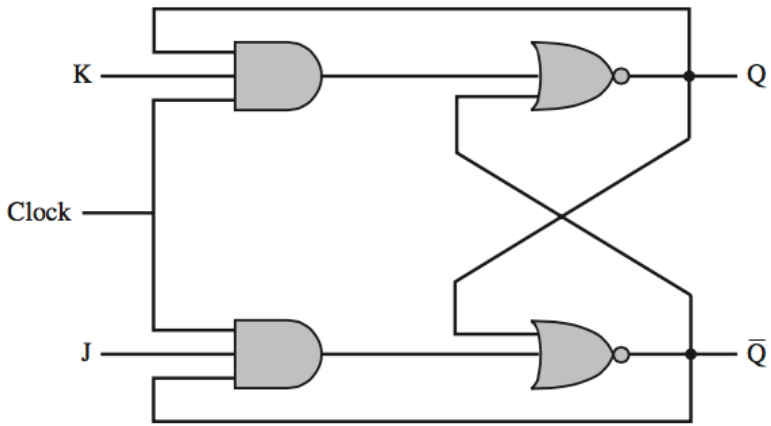


Figure: J-K flip-flops (Stallings, 2015)

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

- Note that the first three combinations are the same as for the SR flip-flop;
- With no input asserted (J=K=0): output is stable;

J-K flip-flops (2/3)

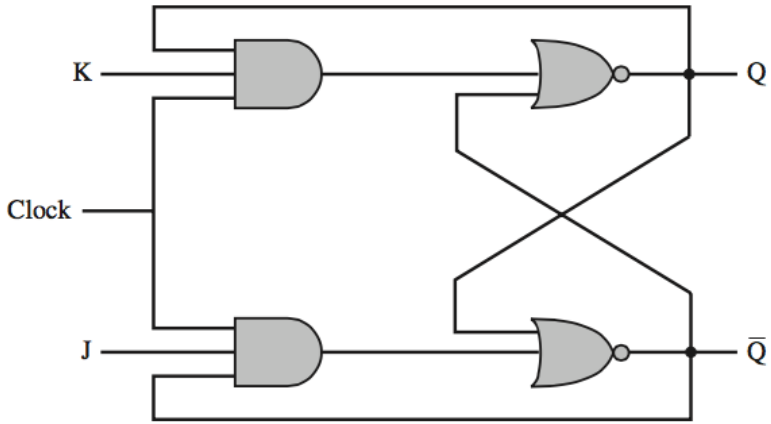


Figure: J-K flip-flops (Stallings, 2015)

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

- If only the J input is asserted, the output is **set** to 1;
- if only the K in put is asserted, the output is **reset** to 0.

J-K flip-flops (3/3)

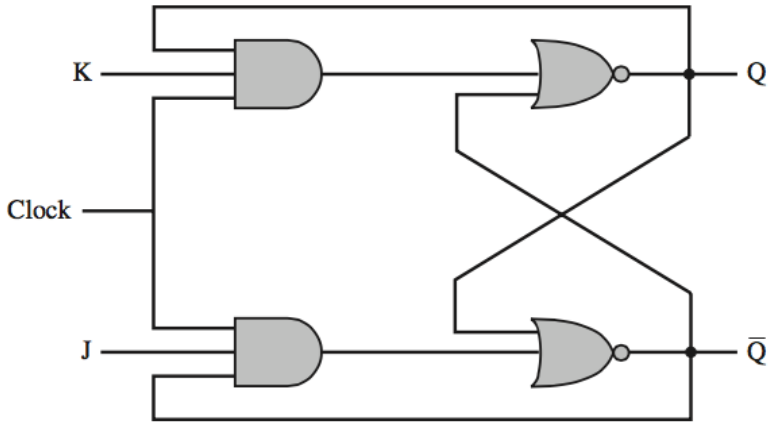


Figure: J-K flip-flops (Stallings, 2015)

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

- When both J and K are 1: output is reversed;
  - If $Q_n = 0$ then $Q_{n+1} = 1$
  - If $Q_n = 1$ then $Q_{n+1} = 0$

In summary:

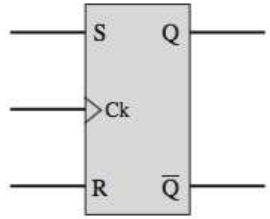
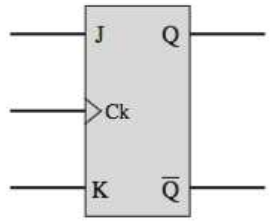
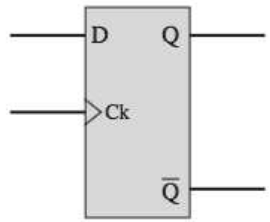
Name	Graphical Symbol	Truth Table
S–R		S R Q_{n+1} 0 0 Q_n 0 1 0 1 0 1 1 1 –
J–K		J K Q_{n+1} 0 0 Q_n 0 1 0 1 0 1 1 1 $\overline{Q_n}$
D		D Q_{n+1} 0 0 1 1

Figure: Basic flip-flops summary (Stallings, 2015)

Registers

Lets look at another type of sequential circuits: **registers**:

First, how many of you have heard of registers? Any ideas?

Registers

Lets look at another type of sequential circuits: **registers**:

- Circuit used within the CPU to store one or more bits of data
- Two basic types of registers are commonly used:
  - Parallel registers;
  - Shift registers.

Lets have a quick look at each one of these...

Parallel Registers

Consists of a set of 1-bit memories that can be read or written simultaneously.

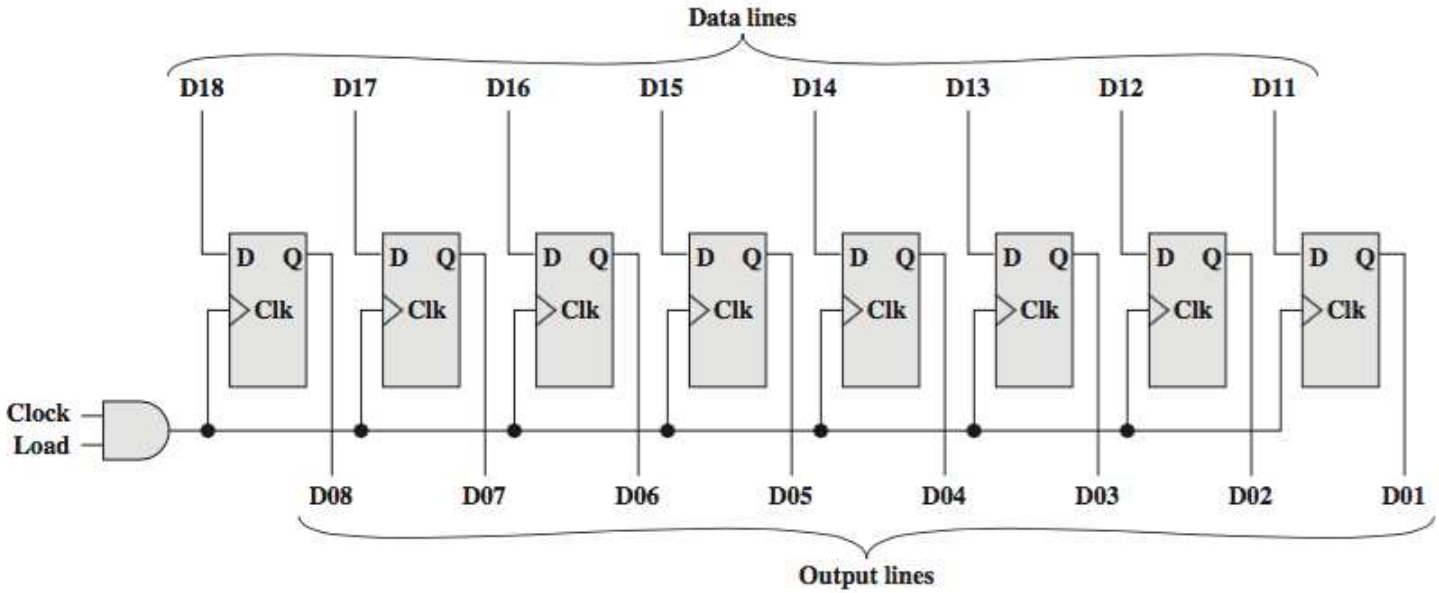


Figure: 8 Bit Parallel Register (Source: (Stallings, 2015))

- Makes use of D flip-flops
- Load control signal controls writing into the register from signal lines, D11 through D18.

Shift Registers (1/2)

A shift register accepts and/or transfers information serially:

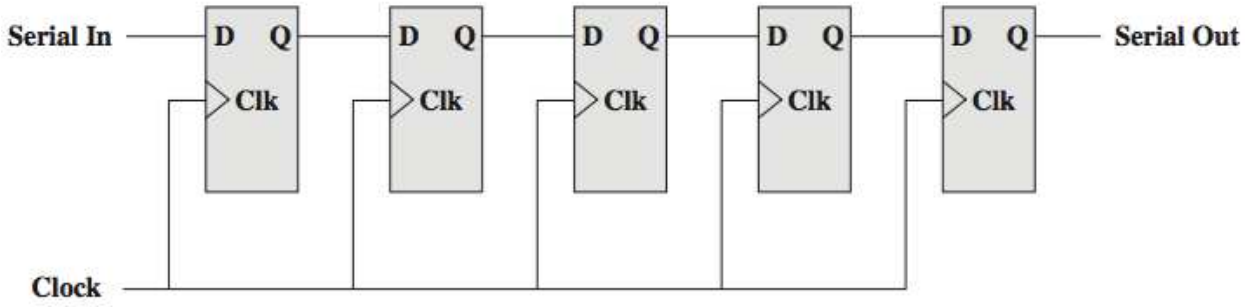


Figure: 5 Bit Shift Register (Source: (Stallings, 2015))

- A 5-bit shift register constructed from clocked D flip-flops;
- Data are input only to the leftmost flip-flop;
- With each clock pulse, data are shifted to the right one position;
- and the rightmost bit is transferred out.

Shift Registers (2/2)

A shift register accepts and/or transfers information serially:

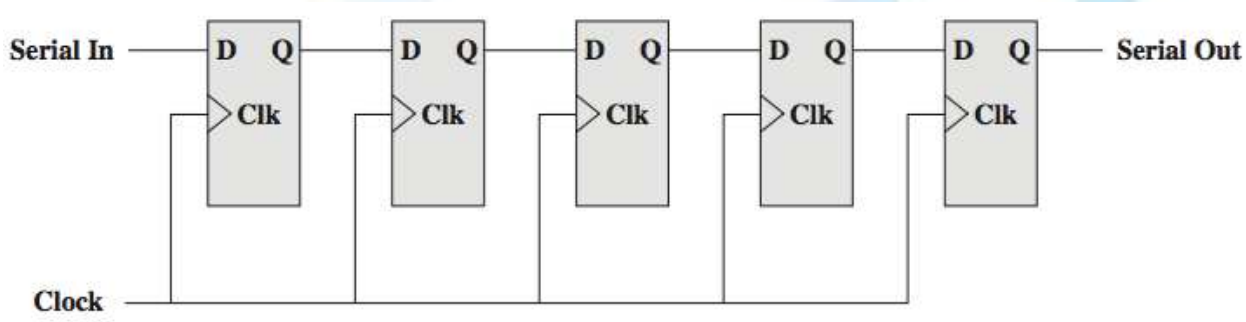


Figure: 5 Bit Shift Register (Source: (Stallings, 2015))

- Shift registers can be used to interface to serial I/O devices.
- In addition, they can be used within the ALU to perform logical shift and rotate functions.

Counters

Lets look at another type of sequential circuits: **counters**:

First, how many of you have heard of counters? Any ideas?

Counters

Register whose value is incremented by 1;

- Register made up of n flip-flops can count up to $2^n - 1$.
- After value $2^n - 1$ the next increment sets the counter value to 0.
- An example of a counter in the CPU is the program counter;

Counters can be designated as asynchronous or synchronous:

- **Asynchronous counter:**
  - Slow since output of one flip-flop triggers a change in next flip-flop.
- **Synchronous counter:**
  - All of the flip-flops change state at the same time.
  - The kind used in CPUs.

Lets have a look at each one of these...

Asynchronous counters (1/3)

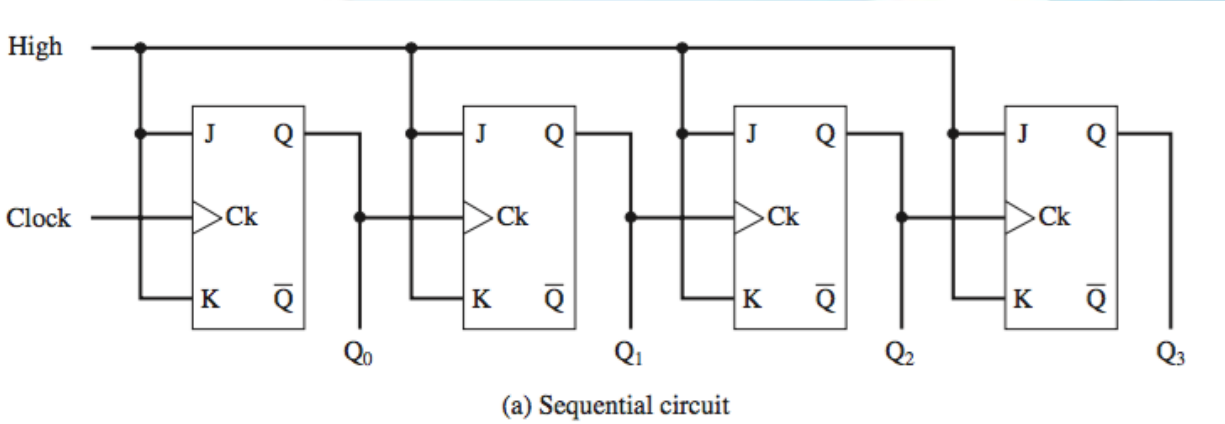


Figure: 4-Bit Counter (Source: (Stallings, 2015))

- Output of the leftmost flip-flop (Q_0) is the least significant bit;
- All output Q_i bits are initialized to zero;
- Extensible to an arbitrary number of bits by cascading more flip-flops;
- Counter is incremented with each clock pulse;

Asynchronous counters (2/3)

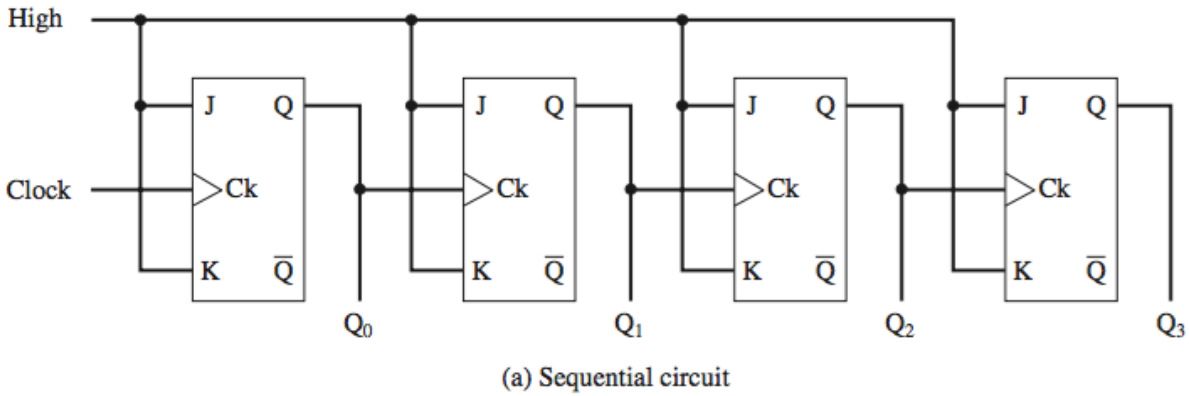


Figure: 4-Bit Counter (Source: (Stallings, 2015))

- J and K inputs to each flip-flop are held at a constant 1 (High).
- *I.e.* when there is a clock pulse, the output at Q will be inverted;
- Change in state occurs with the **falling edge** of the clock pulse
  - A.k.a. edge-triggered flip-flop
  - Timing is very important for the correct functioning of the counter.

Asynchronous counters (3/3)

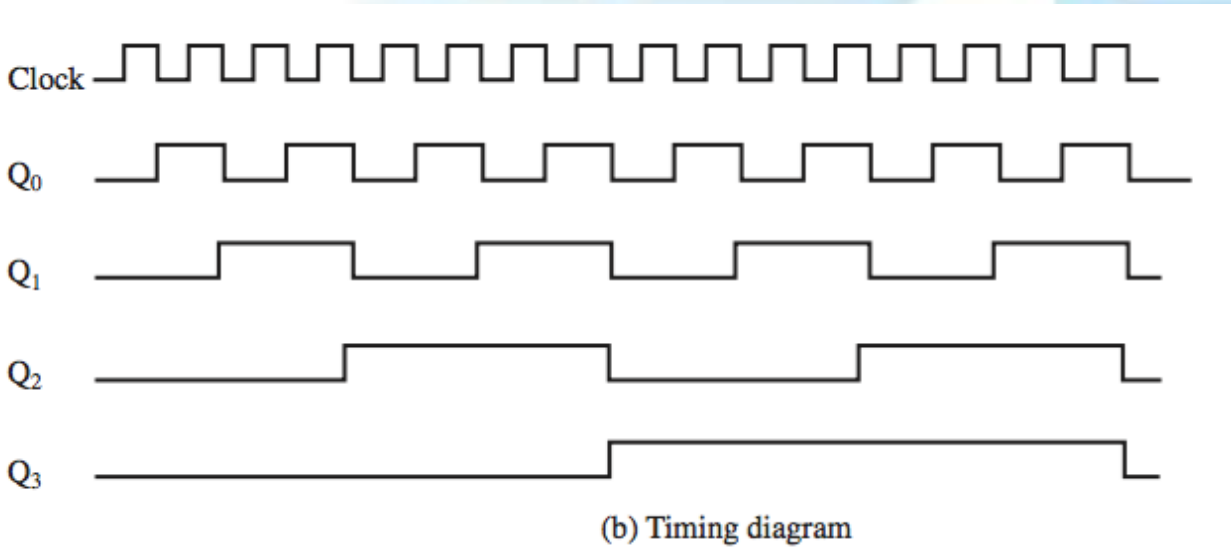


Figure: 4-Bit Counter (Source: (Stallings, 2015))

- State of each individual bit is initially set to zero
- If one looks at patterns of output for this counter, it can be seen that it cycles through 0000, 0001, . . ., 1110, 1111, 0000
- Note the **transitional delay** from each flip-flops

Synchronous counters (1/11)

Asynchronous counters have a disadvantageous built-in delay:

- Proportional to the length of the counter.

CPUs make use of **synchronous counters**:

- All of the flip-flops of the counter change at the same time.

Lets take a look at how to build a 3-bit synchronous counter.

Synchronous counters (2/11)

For a 3-bit counter, three flip-flops will be needed:

- Lets us use J-K flip-flops.
- Label the uncomplemented output of the three flip-flops A, B, C respectively, with C representing the least significant bit.
- It is helpful to **recast** the characteristic table for the J-K flip-flop:

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

Figure: Original

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

Synchronous counters (3/11)

For a 3-bit counter, three flip-flops will be needed:

- Lets us use J-K flip-flops.
- Label the uncomplemented output of the three flip-flops A, B, C respectively, with C representing the least significant bit.
- It is helpful to **recast** the characteristic table for the J-K flip-flop:

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

Figure: Original

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

Synchronous counters (4/11)

- It is helpful to recast the characteristic table for the J-K flip-flop:

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

Figure: Original

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

Figure: Recast

- If $Q_n = 0$ and we want to transition to $Q_{n+1} = 0$
  - Then $J = 0$ and $K = \{0, 1\}$
- If $Q_n = 0$ and we want to transition to $Q_{n+1} = 1$
  - Then $J = 1$ and $K = \{0, 1\}$

Synchronous counters (5/11)

- It is helpful to recast the characteristic table for the J-K flip-flop:

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

Figure: Original

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

Figure: Recast

- If $Q_n = 1$ and we want to transition to $Q_{n+1} = 0$
  - Then $K = 1$ and $J = \{0, 1\}$
- If $Q_n = 1$ and we want to transition to $Q_{n+1} = 1$
  - Then $K = 0$ and $J = \{0, 1\}$

Synchronous counters (6/11)

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

Figure: Synchronous Counter Truth Table (Source: (Stallings, 2015))

- Consider transition from ``000'' to ``001''
  - Value of A needs to remain 0;
    Value of B needs to remain 0;
    Value of C needs to go from 0 to 1;
  - Excitation table shows that to:
    - Maintain an output of 0: inputs must be $\{J = 0, K = d\}$;
    - To effect a transition from 0 to 1: inputs must be $\{J = 1, K = d\}$;

Synchronous counters (7/11)

- With this in mind we can construct a truth table that relates the J-K inputs and outputs

C	B	A	J_c	K_c	J_b	K_b	J_a	K_a
0	0	0						
0	0	1						
0	1	0						
0	1	1						
1	0	0						
1	0	1						
1	1	0						
1	1	1						

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

Synchronous counters (8/11)

- With this in mind we can construct a truth table that relates the J-K inputs and outputs

C	B	A	J_c	K_c	J_b	K_b	J_a	K_a
0	0	0	0	*d*	0	*d*	1	*d*
0	0	1	0	*d*	1	*d*	*d*	1
0	1	0	0	*d*	*d*	0	1	*d*
0	1	1	1	*d*	*d*	1	*d*	1
1	0	0	*d*	0	0	*d*	1	*d*
1	0	1	*d*	0	1	*d*	*d*	1
1	1	0	*d*	0	*d*	0	1	*d*
1	1	1	*d*	1	*d*	1	*d*	1

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

Synchronous counters (9/11)

We can develop Boolean expressions for these six functions:

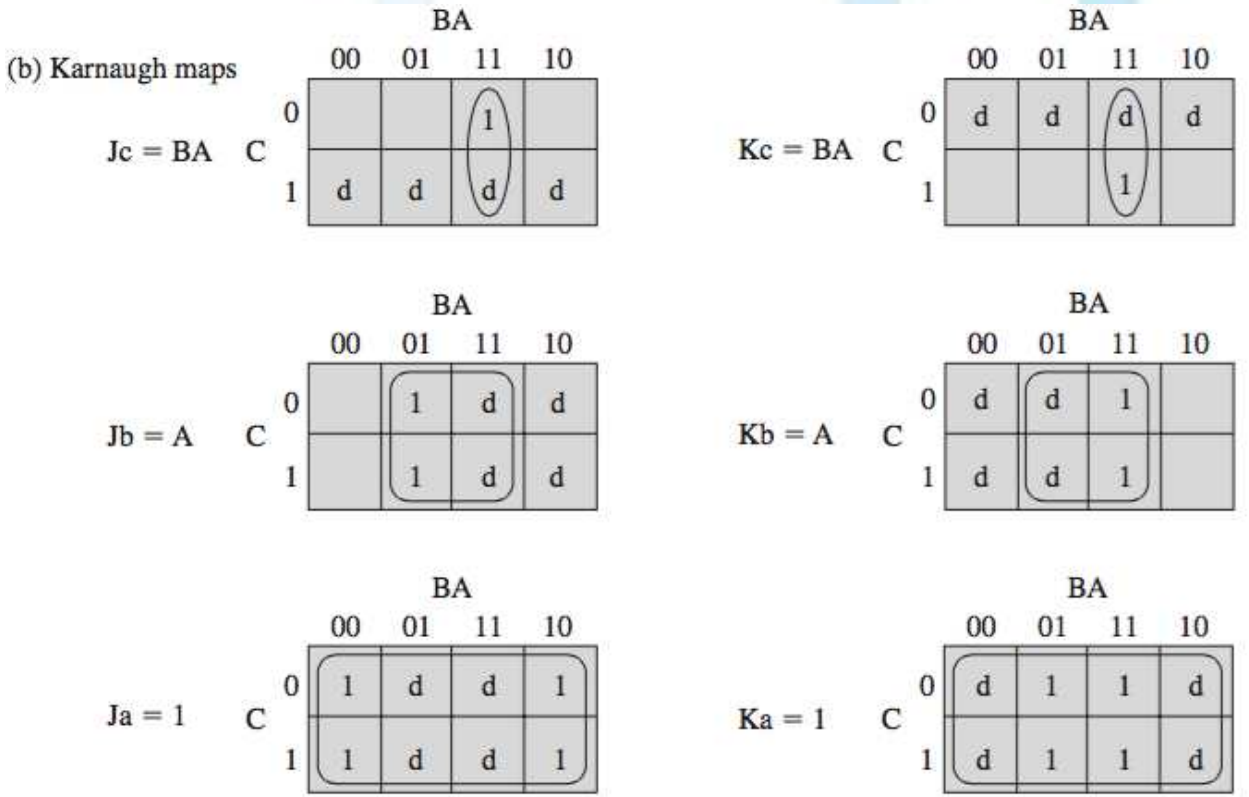


Figure: Synchronous Counter Karnaugh Maps (Source: (Stallings, 2015))

Synchronous counters (10/11)

For example, the Karnaugh map for the variable Jc:

- The J input to the flip-flop that produces the C output;
- yields the expression Jc = BA.

When all six expressions are derived,

- straightforward to design the circuit.

Synchronous counters (11/11)

Circuit example:

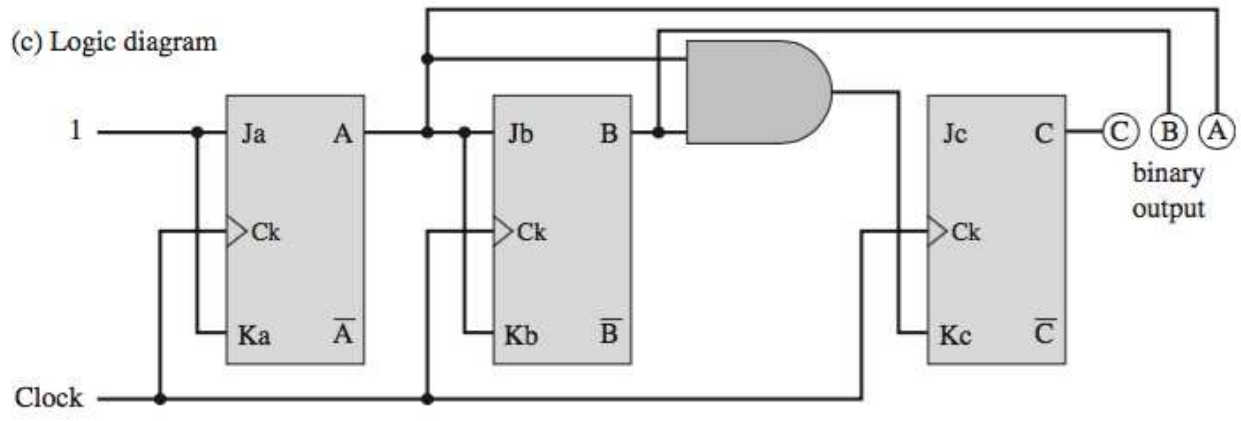


Figure: Synchronous Counter Design (Source: (Stallings, 2015))

References I

Mano, M. and Kime, C. R. (2002).

Logic and Computer Design Fundamentals: 2nd Edition.

Prentice Hall, Englewood Cliffs, NJ, USA.

Stallings, W. (2015).

Computer Organization and Architecture: Designing for Performance.

Pearson Education, 10th edition edition.