



Computer Architecture

2024 – 2025 Academic Year

CEIT – 51023 Computer Architecture I

(Fifth Year – First Semester)

Dr. Theingi Myint

Professor

Department of Computer Engineering and Information Technology

Mandalay Technological University

Contents

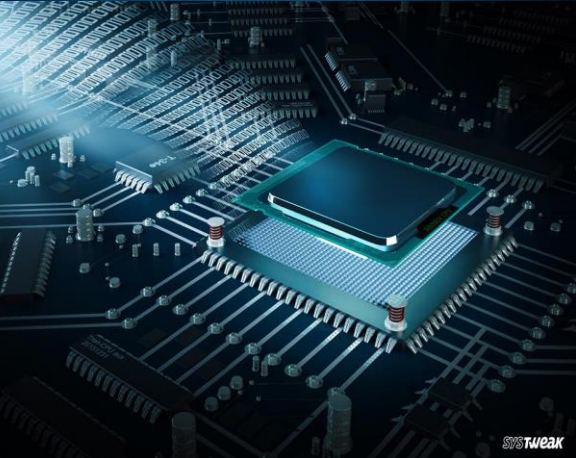
Chapter (1): Digital Logic Circuits

Chapter (2): Digital Components

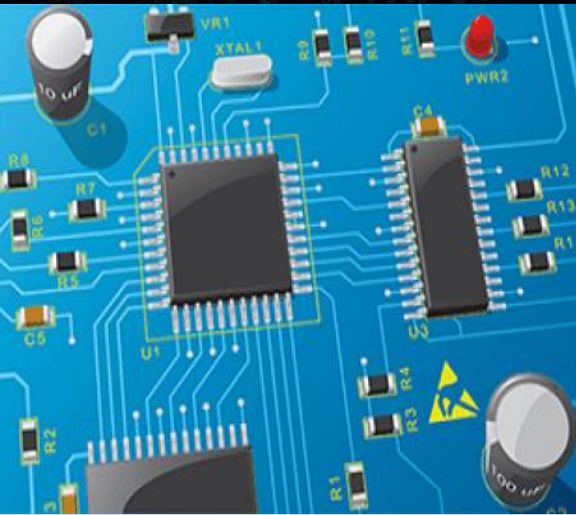
Chapter (4): Register Transfer and Microoperations

Chapter (6): Programming the Basic Computer

Chapter (8): Central Processing Unit



Introduction



- *Computer Architecture* is concerned with the structure and behavior of the various functional modules of the computer and how they interact to provide the processing needs of the user.
- *Computer Organization* is concerned with the way the hardware components are connected together to form a computer system.
- *Computer Design* is concerned with the development of the hardware for the computer taking into consideration a given set of specifications.



Chapter One: Digital Logic Circuits

- Introduces the fundamental knowledge needed for the design of digital systems constructed with individual gates and flip-flops.
- Covers Boolean algebra, combinational circuits, and sequential circuits.
- Provides the necessary background for understanding the digital circuits to be presented.

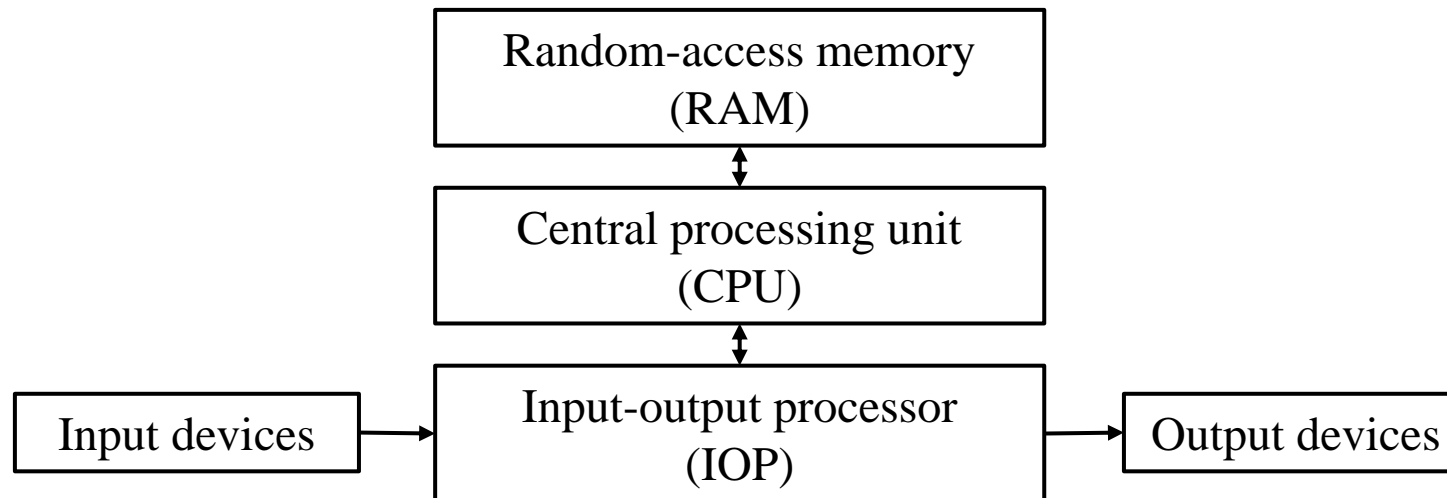
Digital Computers

- First electronic digital computers, developed in the late 1940s, were used primarily for numerical computations.
- It is a digital system that performs various computational tasks.
- Digital implies that the information in the computer is represented by variables that take a limited number of discrete values.
- These values are processed internally by components that can maintain a limited number of discrete states.
- Decimal digits 0, 1, 2, ..., 9 provide 10 discrete values.
- Digital computers function more reliably if only two states are used (binary).
- Digital computers use the binary number system, which has two digits: 0 and 1.

Digital Computers

- The decimal equivalent of a binary number can be found by expanding it into a power series with a base of 2.
- **Example:** The binary number 1001011 represents a quantity that can be converted to a decimal number by multiplying each bit by the base 2 raised to an integer power as follows:

$$1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 75$$



Digital Computers

- A computer system is subdivided into two functional entities: *hardware and software*.

Hardware

- All the electronic components and electromechanical devices that comprise the physical entity of the device
- Three major parts: *central processing unit (CPU), memory, input-output processor*

Software

- The instructions and data that the computer manipulates to perform various data-processing tasks.
- A sequence of instructions for the computer is called *a program*.
- The data that are manipulated by the program constitute the data base.

Logic Gates

- Binary information is represented in digital computer by physical quantities called **signals**.
- Electrical signals such as voltages exist through the computer in either one or two recognizable states.





Example: A particular digital computer may employ a signal of **3 volts** to represent **binary 1**, and **0.5 volt** to represent **binary 0**.





Logic Gates

- Binary logic deals with binary variables and with operations that assume a logical meaning.
- The manipulation of binary information is done by logic circuits called **gates**.
- Gates are **blocks of hardware** that produce signals of binary 1 or 0 when input logic requirements are satisfied.
- A variety of logic gates are commonly used in digital computer systems.
- Each gate has **a distinct graphic symbol**, and its operation can be described by means of **an algebraic expression**.
- Input-output relationship of the binary variables for each gate can be represented in tabular form by **a truth table**.

Logic Gates

- The names, graphic symbols, algebraic functions, and truth tables of eight logic gates are listed as follows:

Name	Graphic symbol	Algebraic function	Truth table															
AND		$x = A \cdot B$ or $x = AB$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	x	0	0	0	0	1	0	1	0	0	1	1	1
A	B	x																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$x = A + B$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	x	0	0	0	0	1	1	1	0	1	1	1	1
A	B	x																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
Inverter		$x = A'$	<table><tr><th>A</th><th>x</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	x	0	1	1	0									
A	x																	
0	1																	
1	0																	
Buffer		$x = A$	<table><tr><th>A</th><th>x</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	A	x	0	0	1	1									
A	x																	
0	0																	
1	1																	

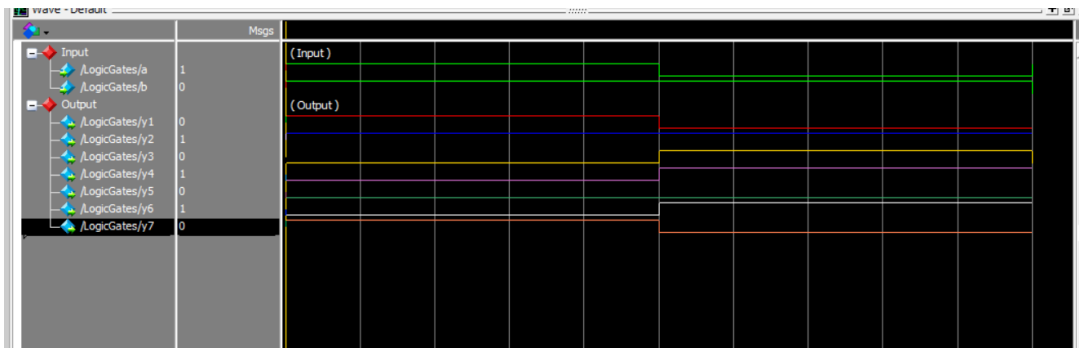
Name	Graphic symbol	Algebraic function	Truth table															
NAND		$x = (AB)'$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	x	0	0	1	0	1	1	1	0	1	1	1	0
A	B	x																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$x = (A + B)'$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	x	0	0	1	0	1	0	1	0	0	1	1	0
A	B	x																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
Exclusive-OR (XOR)		$x = A \oplus B$ or $x = A'B + AB'$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	x	0	0	0	0	1	1	1	0	1	1	1	0
A	B	x																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
Exclusive-NOR or equivalence		$x = (A \oplus B)'$ or $x = A'B' + AB$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	x	0	0	1	0	1	0	1	0	0	1	1	1
A	B	x																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

Logic Gates

Verilog for Logic Gates

```
module LogicGates(a,b,y1,y2,y3,y4,y5,y6,y7);  
  
    input a,b;  
  
    output y1,y2,y3,y4,y5,y6,y7;  
  
    and(y1,a,b);  
  
    or(y2,a,b);  
  
    not(y3,a);  
  
    nand(y4,a,b);  
  
    nor(y5,a,b);  
  
    xor(y6,a,b);  
  
    xnor(y7,a,b);  
  
endmodule
```

Simulation Results:



Testbench Code for Logic Gates

```
module TestModule;  
    //Inputs  
    reg a;  
    reg b;  
    //Outputs  
    wire y1;  
    wire y2;  
    wire y3;  
    wire y4;  
    wire y5;  
    wire y6;  
    wire y7;  
    // Instantiate the Unit Under Test (UUT)  
    LogicGates uut(  
        .a(a), .b(b), .y1(y1), .y2(y2), .y3(y3), .y4(y4), .y5(y5), .y6(y6), .y7(y7));  
    initial begin  
        // Initialize Inputs  
        a = 0;  
        b = 0;  
        // Wait 100ns for global reset to finish  
        #100  
        a = 1;  
        b = 0;  
    end  
endmodule
```


Boolean Algebra

Boolean Algebra

- The purpose of Boolean algebra is to facilitate the analysis and design of digital circuits:

Express in algebraic form a truth table relationship between binary variables.

Express in algebraic form the input-output relationship of logic diagrams.

Find simpler circuits for the same function.

- Truth table and logic diagram for $F = x + y'z$.

x	y	z	F
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	1

(a) Truth table



(b) Logic diagram

Boolean Algebra

Boolean Algebra

- The most basic identities of Boolean algebra are:

$$(1) \ x + 0 = x$$

$$(3) \ x + 1 = 1$$

$$(5) \ x + x = x$$

$$(7) \ x + x' = 1$$

$$(9) \ x + y = y + x$$

$$(11) \ x + (y + z) = (x + y) + z$$

$$(13) \ x(y + z) = xy + xz$$

$$(15) \ (x + y)' = x'y'$$

$$(17) \ (x')' = x$$

$$(2) \ x \cdot 0 = 0$$

$$(4) \ x \cdot 1 = x$$

$$(6) \ x \cdot x = x$$

$$(8) \ x \cdot x' = 0$$

$$(10) \ xy = yx$$

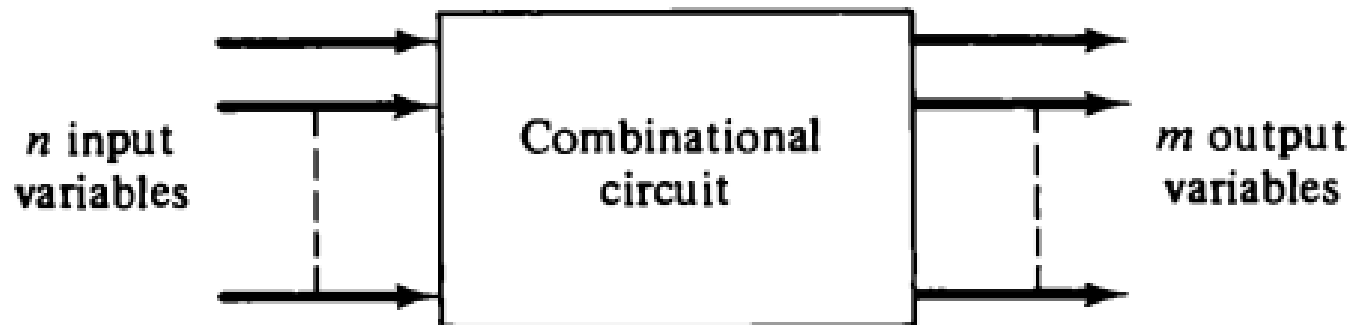
$$(12) \ x(yz) = (xy)z$$

$$(14) \ x + yx = (x + y)(x + z)$$

$$(16) \ (xy)' = x' + y'$$

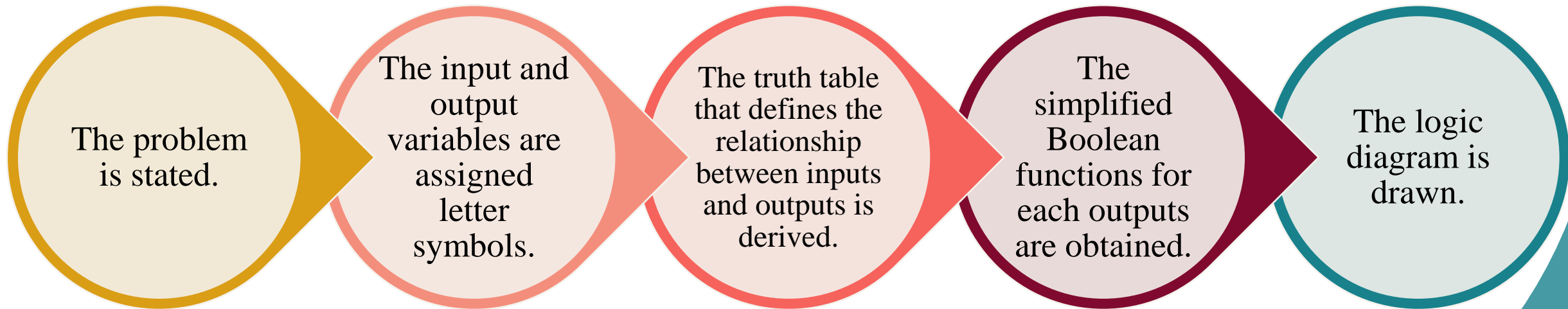
Combinational Circuits

- A combinational circuit is **a connected arrangement of logic gates** with a set of inputs and outputs.
- The binary values of the outputs are **a function of the binary combination** of the inputs.
- The n binary input variables come from an external source, the m binary output variables go to an external destination, and in between there is **an interconnection of logic gates**.
- A combinational circuit transforms binary information from the given input data to the required output data.
- Combinational circuits are employed in digital computers for generating binary control decisions and for providing digital components required for data processing.



Combinational Circuits

- A combinational circuit can be described by a truth table showing the binary relationship between the n input variables and the m output variables.
- Truth table lists the corresponding output binary values for each of the 2^n input combinations.
- The design of combinational circuits starts from the verbal outline of the problem and ends in a logic circuit diagram.

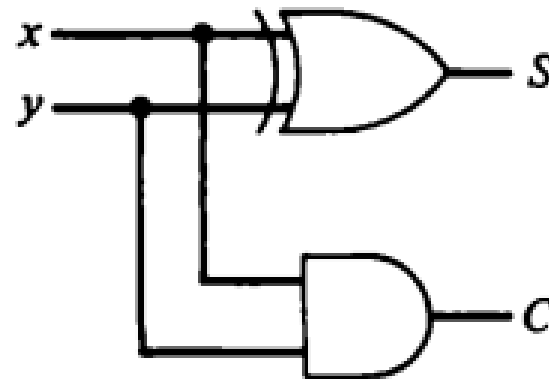


Combinational Circuits

- To demonstrate the design of combinational circuits, two examples of simple arithmetic circuits: **Half-adder** and **Full-adder**.
- A combinational circuit that performs the **arithmetic addition of two bits** is called a **half-adder**.
- One that performs the addition of three bits (two significant bits and a previous carry) is called a **full-adder**.
- To implement a full-adder, two half-adders are needed.

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth table



Logic diagram

Half-adder

Boolean functions for the two outputs can be obtained directly from the truth table:

$$(Sum) S = x'y + xy' = x \oplus y$$

$$(Carry) C = xy$$

It consists of an **exclusive-OR gate** and an **AND gate**.

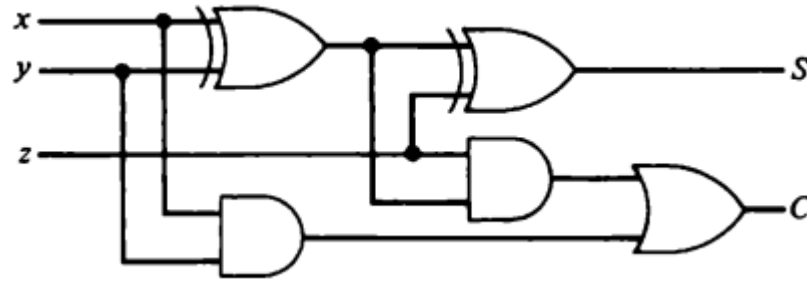
Combinational Circuits

- A full-adder is a combinational circuit that forms the arithmetic sum of three input bits.
- It consists of three inputs and two outputs.

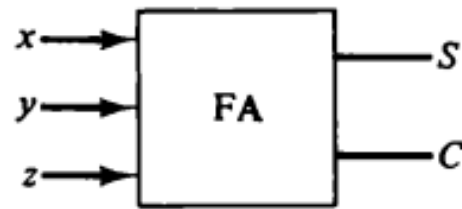
inputs			outputs	
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Truth table

Full-adder



Logic diagram



Block diagram

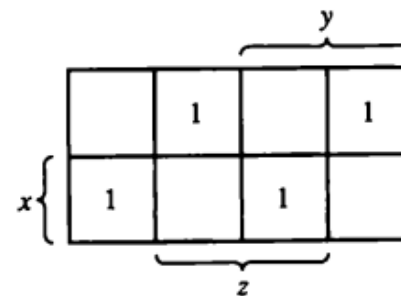
Boolean functions for the two outputs can be obtained directly from the truth table:

$$(Sum) S = x \oplus y \oplus z$$

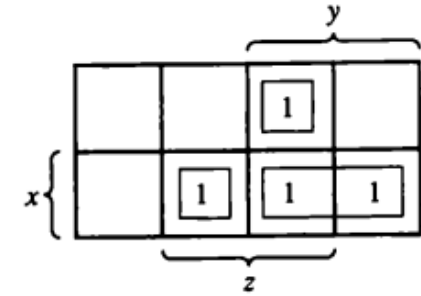
$$(Carry) C = xy + (x'y + xy')z$$

$$C = xy + (x \oplus y)z$$

It consists of **two half-adders** and **an OR gate**.



$$S = x'y'z + x'yz' + xy'z' + xyz \\ = x \oplus y \oplus z$$



$$C = xy + xz + yz \\ = xy + (x'y + xy')z$$

To do the Lab Report Assignment (II)

- HALF-ADDER
- FULL-ADDER

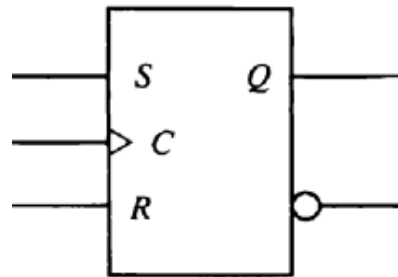
Flip-Flops

- The storage elements employed in clocked sequential circuits are called **flip-flops**.
- A flip-flop is a binary cell capable of **storing one bit** of information.
- It has **two outputs**, one for the **normal value** and one for the **complement value** of the bit stored in it.
- A flip-flop maintains **a binary state** until directed by a clock pulse to switch states.
- The difference among various types of flip-flops is in the number of inputs they possess and in the manner in which the inputs affect the binary state.

Flip-Flops

- **SR Flip-Flop**

- It has **three inputs**, labeled S (for set), R (for reset), and C (for clock).
- It has **an output Q** and the flip-flop has **a complemented output**, which is indicated with a small circle at the other output terminal.
- There is an arrowhead-shaped symbol in front of the letter C to designate **a dynamic input**.
- The dynamic indicator symbol denotes the fact that the flip-flop responds to a positive transition (from 0 to 1) of the input clock signal.



(a) Graphic symbol

Flip-Flops

- **Operation of SR Flip-Flop**

- If there is no signal at the clock input C , the output of the circuit cannot change irrespective of the values at inputs S and R .
- Only when the clock signal changes from 0 to 1 can the output be affected according to the values in inputs S and R .
- If $S = 1$ and $R = 0$ when C changes from 0 to 1, output Q is set to 1.
- If $S = 0$ and $R = 1$ when C changes from 0 to 1, output Q is cleared to 0.
- If both S and R are 0 during the clock transition, the output does not change ($Q(t+1) = Q(t)$).
- When both S and R are equal to 1, the output is unpredictable and may go to either 0 or 1, depending on internal timing delays that occur within the circuit.

S	R	$Q(t+1)$	
0	0	$Q(t)$	No change
0	1	0	Clear to 0
1	0	1	Set to 1
1	1	?	Indeterminate

(b) Characteristic table

- $Q(t)$ is the binary state of the Q output at a given time (**present state**).
- $Q(t+1)$ is the binary state of the Q output after the occurrence of a clock transition (**next state**).

Flip-Flops

Verilog for SR Flip-flop

```
module Main(input S,
  input R,
  input clk,
  output Q,
  output Qbar
);
  reg M,N;

  always @(posedge clk) begin
    M <= !(S & clk);
    N <= !(R & clk);
  end
  assign Q = !(M & Qbar);
  assign Qbar = !(N & Q);

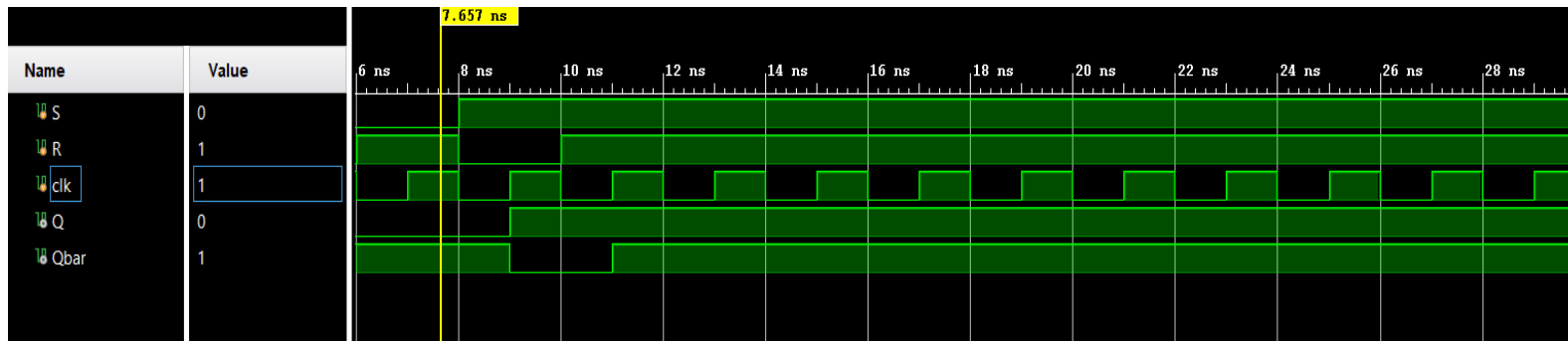
endmodule
```

Testbench for SR Flip-flop

```
module TestSR;
  // Inputs
  reg S;
  reg R;
  reg clk;
  // Outputs
  wire Q;
  wire Qbar;
  Main uut (.S(S), .R(R), .clk(clk),
    .Q(Q), .Qbar(Qbar));

  initial begin
    S = 0;
    R = 0;
    clk = 0;
    fork
      #2 S = 0;
      #2 R = 1;
      #4 S = 0;
      #4 R = 0;
      #6 S = 0;
      #6 R = 1;
      #8 S = 1;
      #8 R = 0;
      #10 S = 1;
      #10 R = 1;
    join
  end
  always #1 clk =! clk;
endmodule
```

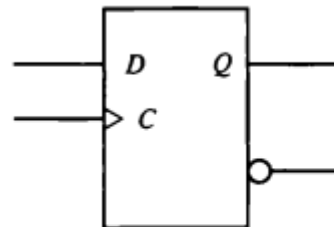
Simulation results of SR Flip-flop



Flip-Flops

- **D Flip-Flop**

- The D (data) flip-flop is a slight modification of the SR flip-flop.
- An SR flip-flop is converted to a D flip-flop by inserting an inverter between S and R and assigning the symbol D to the single input.
- The D input is sampled during the occurrence of a clock transition from 0 to 1.
- If $D = 1$, the output of the flip-flop goes to the 1 state.
- If $D = 0$, the output of the flip-flop goes to the 0 state.
- The next state $Q(t+1)$ is determined from the D input.
- The relationship can be expressed by a characteristic equation: $Q(t+1) = D$
- Q output of the flip-flop receives its value from the D input every time that the clock signal goes through a transition from 0 to 1.



(a) Graphic symbol

D	$Q(t+1)$
0	0
1	1

Clear to 0
Set to 1

(b) Characteristic table

Flip-Flops

- **Operation of D Flip-Flop**

- No input condition exists that will leave the state of the D flip-flop unchanged.
- Although a D flip-flop has the advantage of having only one input (excluding C), it has the disadvantage that its characteristics table does not have a “no change” condition $Q(t+1) = Q(t)$.
- The “no change” condition can be accomplished either by disabling the clock signal or by feeding the output back into the input, so that clock pulses keep the state of the flip-flop unchanged.

Flip-Flops

Verilog for D Flip-flop

```
module DFF (  
    input clk,    // Enable  
    input clr,    // Asynchronous clear  
    input D,      // Data input  
    output Q,  
    output Qbar);  
//wire X, Y;  
reg Q;  
reg Qbar;  
always @(posedge clk or negedge clr)  
begin  
    if(~clr)  
        begin  
            Q <= 0;  
            Qbar <= 1;  
        end  
    else  
        begin  
            Q <= D;  
            Qbar <= ~D;  
        end  
    end  
end  
endmodule
```

Testbench for D Flip-flop

```
module DFF_tb ();  
  
    reg clk;  
    reg clr;  
    reg D;  
    wire Q;  
    wire Qbar;  
  
    DFF myDFF(clk, clr, D, Q, Qbar);  
    // DFF myDFF(clk, D, Q, Qbar);  
  
    initial  
    begin  
        $monitor("clk = %b, clr = %b, D = %b, Q = %b", clk, clr, D, Q);  
        // $monitor("clk = %b, D = %b, Q = %b", clk, D, Q);  
        $dumpfile("DFF.vcd");  
        $dumpvars(0, DFF_tb);  
        clr <= 1;  
    end  
endmodule
```


Flip-Flops

Testbench for D Flip-flop

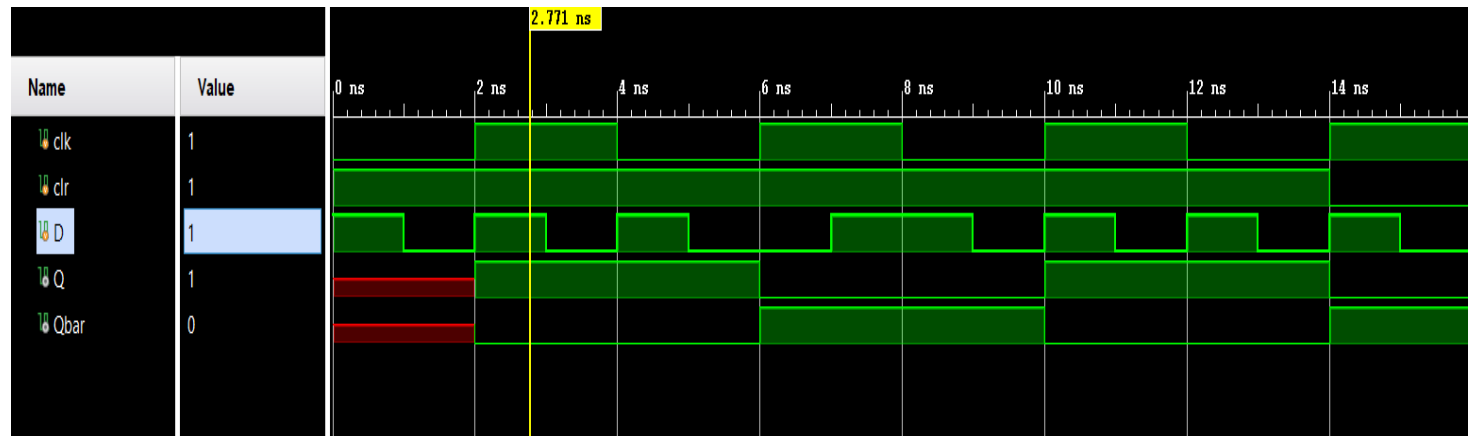
```
clk <= 0;
D <= 1; #1;
D <= 0; #1;
clk <= 1;
D <= 1; #1;
D <= 0; #1;
clk <= 0;
D <= 1; #1;
D <= 0; #1;
clk <= 1;
D <= 0; #1;
D <= 1; #1;
clk <= 0;
D <= 1; #1;
D <= 0; #1;
clk <= 1;
D <= 1; #1;
D <= 0; #1;
clk <= 0;
D <= 1; #1;
D <= 0; #1;
clr <= 0;
clk <= 1;
D <= 1; #1;
D <= 0; #1;
```

\$finish;

end

endmodule

Simulation results of D Flip-flop



Flip-Flops

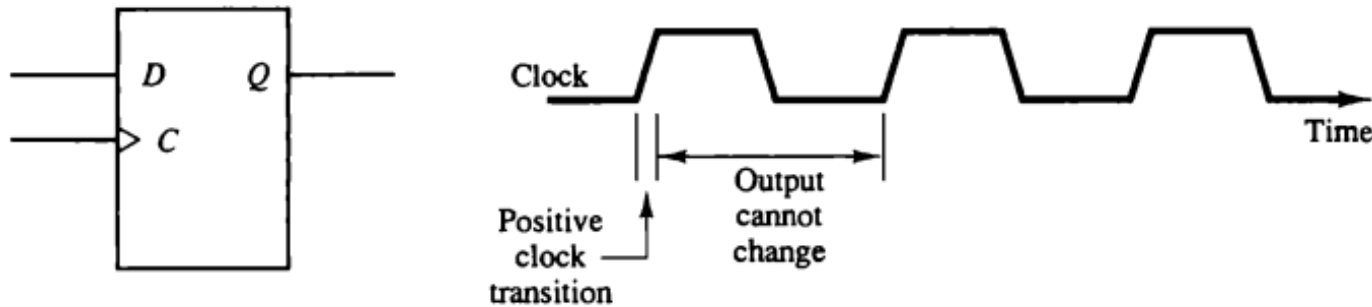
- **Edge-Triggered Flip-Flops**

- The most common type of flip-flop used to synchronize the state change during a clock pulse transition is the **edge-triggered flip-flop**.
- In this type of flip-flop, output transitions occur at a specific level of the clock pulse.
- When **the pulse input level exceeds this threshold level**, the inputs are **locked out** so that the flip-flop is unresponsive to further changes in inputs until the clock pulse returns to 0 and another pulse occurs.
- Some edge-triggered flip-flops cause a transition on the **rising edge** of the clock signal (**positive-edge transition**), and others cause a transition on the **falling edge** (**negative-edge transition**).

Flip-Flops

- **Edge-Triggered Flip-Flops**

- The value in the D input is transferred to the Q output when the clock makes a **positive transition**.
- The output cannot change when the clock is in the 1 level, in the 0 level, or in a transition from the 1 level to the 0 level.
- The effective positive clock transition includes a minimum time called the **setup time** in which the D input must remain at constant value before the transition, and a definite time called the **hold time** in which the D input must not change after the positive transition.
- The effective positive transition is a very small fraction of the total period of the clock pulse.

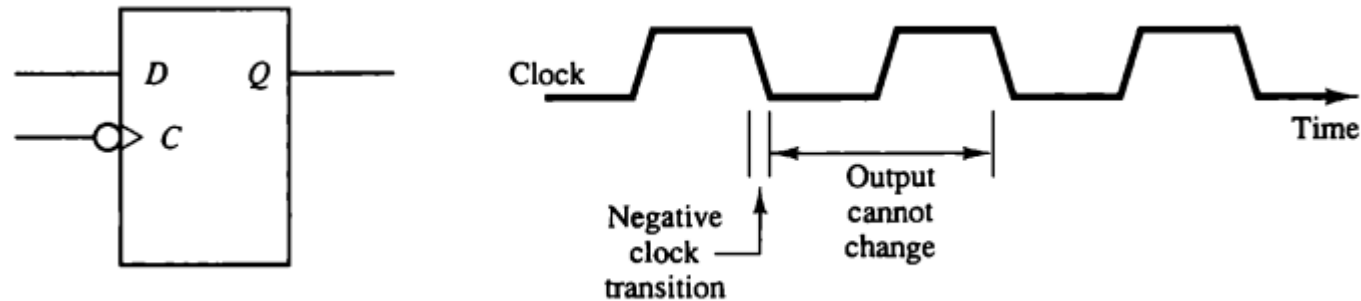


(a) Positive-edge-triggered D flip-flop.

Flip-Flops

- **Edge-Triggered Flip-Flops**

- The negative-edge triggered D flip-flop symbol includes a negation small circle in front of the dynamic indicator at the C input.
- In this case, the flip-flop responds to a transition from the 1 level to the 0 level of the clock signal.



(b) Negative-edge-triggered D flip-flop.

Flip-Flops

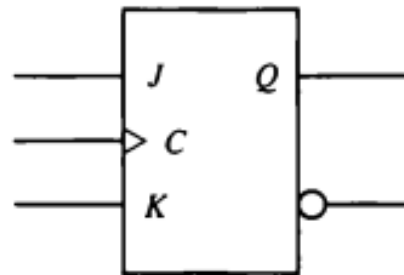
- **Master-slave Flip-Flops**

- This type of circuit consists of two flip-flops.
- The first is the **master**, which responds to **the positive level** of the clock, and the second is the **slave**, which responds to the **negative level** of the clock.
- The result is that the output changes during the 1-to-0 transition of the clock signal.

Flip-Flops

- **JK Flip-Flop**

- A *JK* flip-flop is a refinement of the *SR* flip-flop in that the indeterminate condition of the *SR* type is defined in the *JK* type.
- Inputs *J* and *K* behave like inputs *S* and *R* to set and clear the flip-flop, respectively.
- When **inputs *J* and *K* are both equal to 1**, a clock transition switches the outputs of the flip-flop to their **complement state**.
- The *J* input is equivalent to the *S* (set) input of the *SR* flip-flop, and the *K* input is equivalent to the *R* (clear) input.
- Instead of the indeterminate condition, the *JK* flip-flop has a complement condition $Q(t+1) = Q'(t)$ when both *J* and *K* are equal to 1.



(a) Graphic symbol

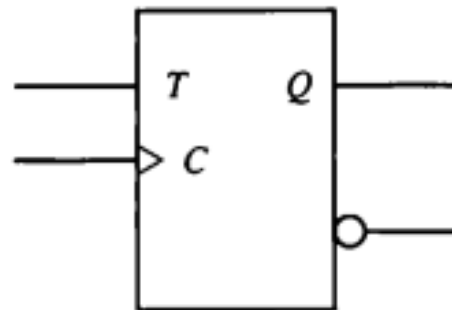
<i>J</i>	<i>K</i>	$Q(t+1)$	
0	0	$Q(t)$	No change
0	1	0	Clear to 0
1	0	1	Set to 1
1	1	$Q'(t)$	Complement

(b) Characteristic table

Flip-Flops

- ***T* Flip-Flop**

- *T* (toggle) flip-flop is obtained from a *JK* type when inputs *J* and *K* are connected to provide a single input designated by *T*.
- The *T* flip-flop has two conditions.
- When $T = 0$ ($J = K = 0$) a clock transition does not change the state of the flip-flop.
- When $T = 1$ ($J = K = 1$) a clock transition complements the state of the flip-flop.
- These conditions can be expressed by a characteristic equation: $Q(t+1) = Q(t) \oplus T$.



(a) Graphic symbol

T	$Q(t+1)$	
0	$Q(t)$	No change
1	$Q'(t)$	Complement

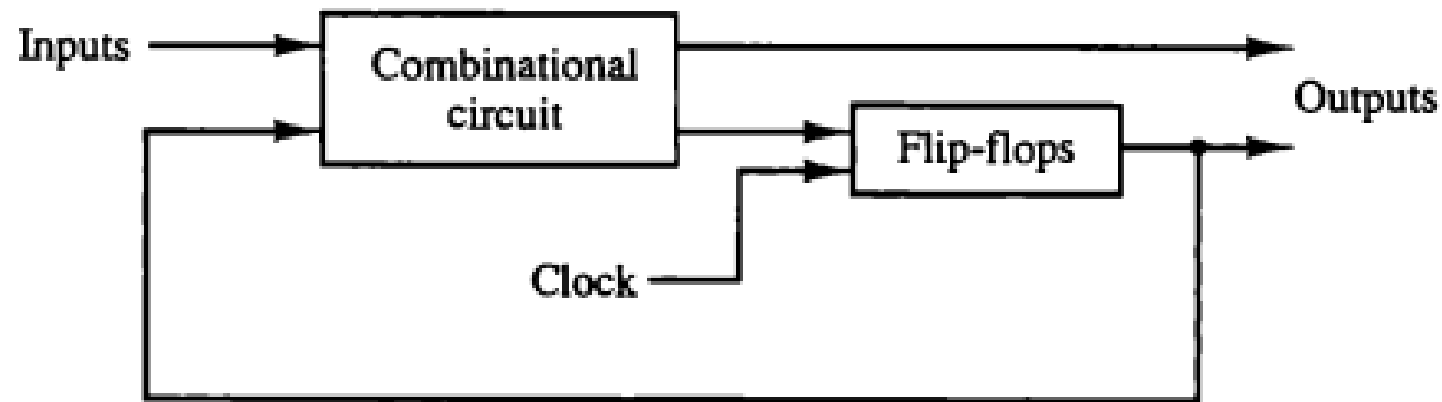
(b) Characteristic table

To do the Lab Report Assignment (III)

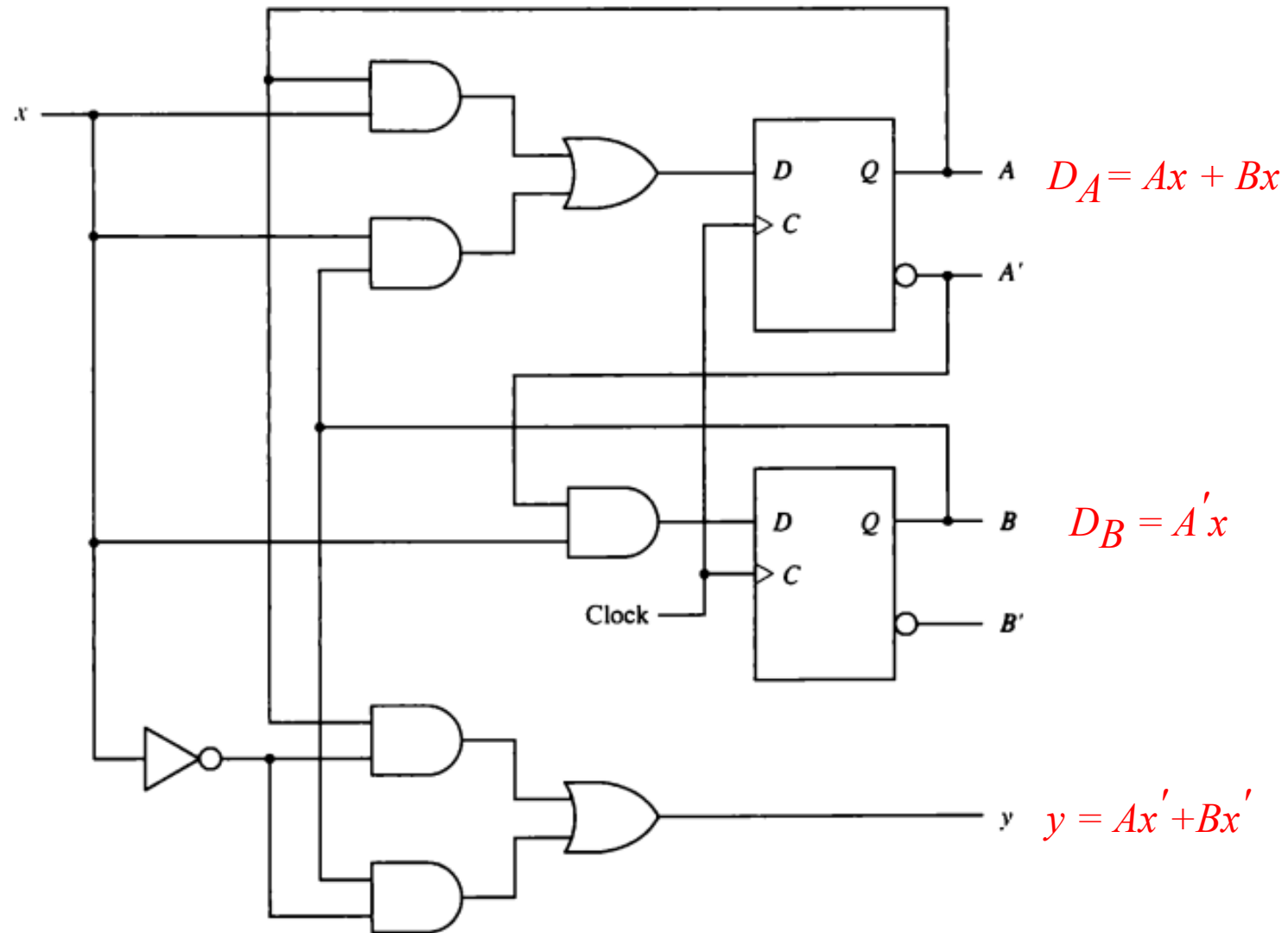
- JK Flip-Flop
- T Flip-Flop

Sequential Circuits

- A sequential circuit is **an interconnection of flip-flops and gates**.
- The block diagram of a clocked sequential circuit consists of **a combinational circuit** and a number of **clocked flip-flops**.
- A sequential circuit is specified by a time sequence of external inputs, external outputs, and internal flip-flop binary states.



Sequential Circuits



Sequential Circuits

- The behavior of a sequential circuit is determined from the inputs, the outputs, and the state of its flip-flops.
- Both the outputs and the next state are a function of the inputs and the present state.
- A sequential circuit is specified by a **state table** that relates outputs and next states as a function of inputs and present states.
- In clock sequential circuits, the transition from present state to next state is activated by the presence of a clock signal.

State Table for Circuit

Present state		Input	Next state		Output
A	B		A	B	
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

- The next-state and output columns are functions of the present state and input values and are derived directly from the circuit or the Boolean equations that describe the circuit.

$$D_A = Ax + Bx \quad D_B = A'x \quad y = Ax' + Bx'$$

A vibrant image of the Aurora Borealis (Northern Lights) in shades of green and blue, set against a dark, starry night sky. The aurora's light patterns are dynamic and flowing, creating a sense of movement and natural beauty.

Next Lecture:

Chapter (2): Digital Components