**COMBINATIONAL LOGIC DESIGN / ANALYSIS**

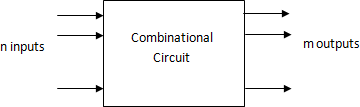
In general, logic circuits can be classified as:

1. Combinational logic circuits
2. Sequential logic circuits

# COMBINATIONAL LOGIC CIRCUITS

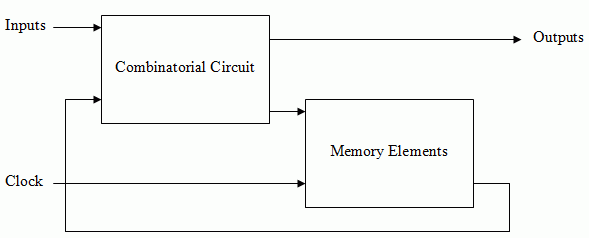
The circuit consists of logic gates whose outputs are functions of current inputs only. That is, output values are determined by the values of current inputs.

This type of circuits does not contain memory elements.



# SEQUENTIAL LOGIC CIRCUITS

The circuit consists of memory elements (flip-flops) in addition to logic gates. The outputs are functions of current inputs and some of previous output values.



# DESIGN PROCEDURE OF COMBINATIONAL LOGIC CIRCUITS

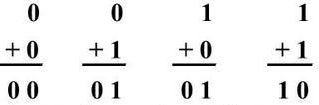
The design procedure for a combinational logic circuit starts with the problem specification and comprises the following steps:

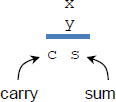
1. Determine the required number of inputs and outputs from the specifications.
2. Assign letter symbols for inputs and outputs.
3. Derive the truth table for each of the outputs based on their relationships to the inputs.
4. Simplify the Boolean expression for each output. Use Karnaugh Map or Boolean algebra.
5. Draw the logic diagram of the simplified Boolean expression.

# ADDERS

***Half-Adder:*** a combinational circuit that performs the addition of two bits and produces the corresponding output(s). It has two inputs and two outputs.



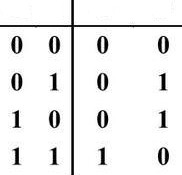
inputs : *x*, *y*



+

outputs: *s*, *c*

K-Map for sum: K-Map for carry:



**x y c**

**s**



1

1

* Truth Table -

*s*  *x* ' *y*  *xy* '

* 1. *s*
  2. *c*



1

*c*  *xy*

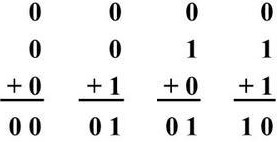
*s*  *x*  *y*

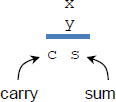
- Simplified Functions -

- Logic Diagram - - Half-Adder (HA) Block Diagram -

***Full-Adder:*** a combinational circuit that performs the addition of three bits and produces the corresponding output(s). It has three inputs and two outputs.

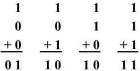
inputs : *x*, *y*, *z*



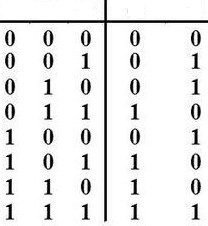
+

carry-out sum

outputs: *s*, *c*

* 1. carry-in
* Two of the input variables, x and y, represent the two bits to be added
* The third input, z, represents the carry-in from the previous lower significant position

K-Map for sum: K-Map for carry:

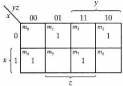


**x y z c**

**s**

- Truth Table -

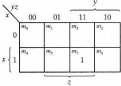
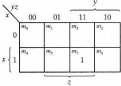
*s*  *x*' *y*' *z*  *x*' *yz*'*xy*' *z*' *xyz*

*s*  *x*'( *y*' *z*  *yz*')  *x*( *y*' *z*' *yz*) *s*  *x*'( *y*  *z*)  *x*( *y*  *z*)'

*s*  *x*  *y*  *z*

*c*  *yz*  *xz*  *xy*

The simplified function of carry can be manipulated to include the XOR of *x* and *y*.



1

1

1

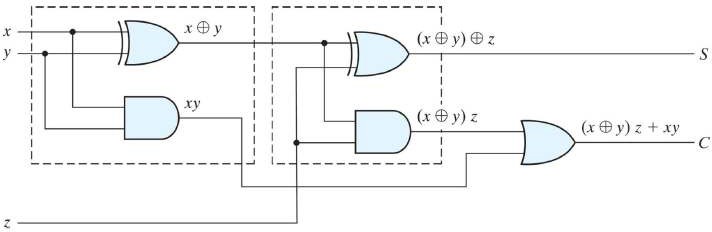
1

1

1

*c*  *x*' *yz*  *xy*' *z*  *xy c*  *z*(*x*' *y*  *xy*')  *xy c*  *z*(*x*  *y*)  *xy*

* Simplified Functions -

Half-Adder (HA) Half-Adder (HA)

* Logic Diagram -

A Full-Adder (FA) can be implemented with two Half-Adders (HA) and one OR gate.

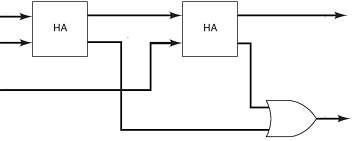
## *x* y

***z***

**-** Full-Adder (FA) Block -

## *s*

***x s***



Full Adder

***y***

***z c***

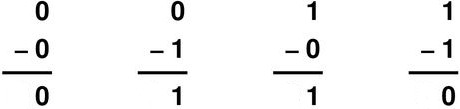
## *c*

- Full-Adder (FA)

Block Diagram -

# SUBTRACTORS

***Half-Subtractor:*** a combinational circuit that performs the subtraction of two bits and produces the corresponding output(s). It has two inputs and two outputs.



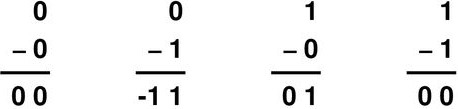
Borrow

inputs : *x*, *y*



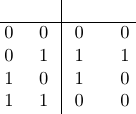
\_

B D



outputs: *B, D*

K-Map for difference:



**x y B D**



1

1

-1 represents a borrow is taken, *B*=1, otherwise *B*=0

K-Map for borrow:

borrow difference

- Truth Table –

*D*  *x* ' *y*  *xy* '

1. *D*
2. *B*



1



*D*

*B*



Half Subtractor

*B*  *x*' *y*

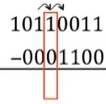
*D*  *x*  *y*

- Logic Diagram - - Half-Subtractor (HS)

- Simplified Functions - Block Diagram -

***Full-Subtractor:*** a combinational circuit that performs the subtraction of three bits and produces the corresponding output(s). It has three inputs and two outputs.

-1-1 -1-1 -z



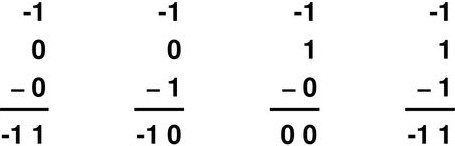
borrow given

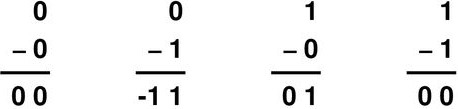


\_

B D

borrow taken difference

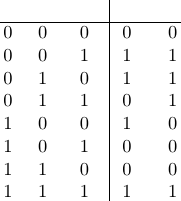
inputs : *x*, *y*, *z*

outputs: *D*, *B*

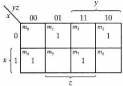


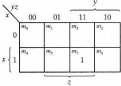
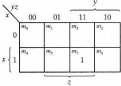
-1 represents a borrow is taken, *B*=1, otherwise *B*=0

K-Map for difference: K-Map for borrow:



**x y z B D**





1

1

1

1

1 1

- Truth Table -

*D*  *x*' *y*' *z*  *x*' *yz*'*xy*' *z*' *xyz D*  *x*'( *y*' *z*  *yz*')  *x*( *y*' *z*' *yz*) *D*  *x*'( *y*  *z*)  *x*( *y*  *z*)'

*D*  *x*  *y*  *z*

*B*  *x*' *z*  *yz*  *x*' *y*

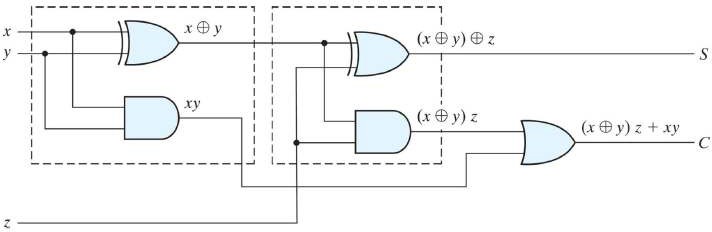
The simplified function of borrow can be manipulated to include the XOR of *x* and *y*.

*B*  *x*' *y*' *z*  *xyz*  *x*' *y B*  *z*(*x*' *y*'*xy*)  *x*' *y*

*B*  *z*(*x*  *y*)'*x*' *y*

- Simplified Functions -

Half-Subtractor (HS) Half-Subtractor (HS)



*D*

*x'y*

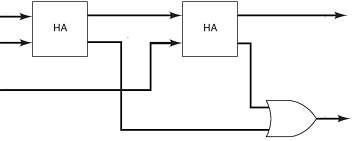
*'*

*B*

- Logic Diagram -

A Full-Subtractor (FS) can be implemented with two Half-Subtractors (HS) and one OR gate.

## *x* y



**HS**

**HS**

***z***

**-** Full-Subtractor (FS) Block -

***D***

***x D***



Full Subtractor

## *y*

1. ***B***

***B*** - Full-Subtractor (FS) Block Diagram -