Chapter 3 – Combinational Digital Circuit Design

Part 1 – Implementation Technology and Logic Design

Overview

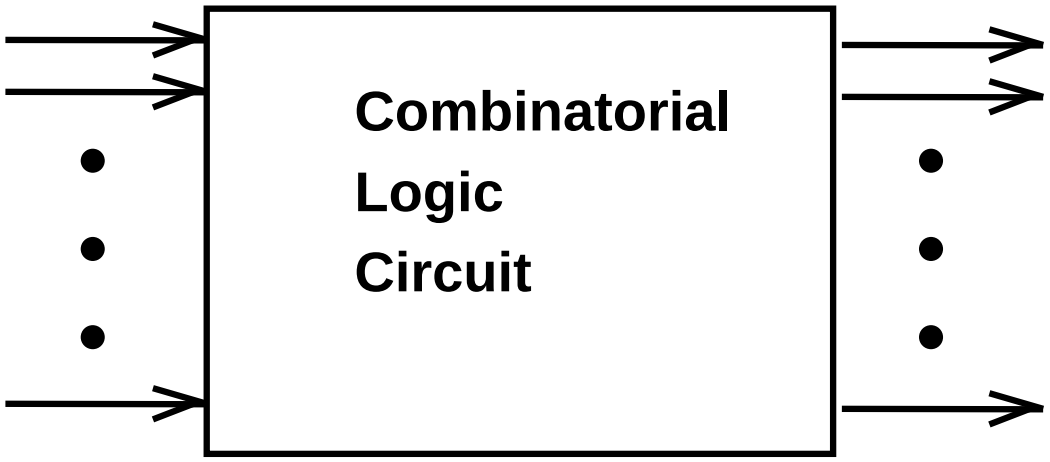
- **Part 1 – Design Procedure**
  - **Steps**
    - **Specification**
    - **Formulation**
    - **Optimization**
    - **Technology Mapping**
  - **Beginning Hierarchical Design**
  - **Technology Mapping - AND, OR, and NOT to NAND or NOR**
  - **Verification**
    - **Manual**
    - **Simulation**

Overview (continued)

- **Part 2 – Combinational Logic**
  - **Functions and functional blocks**
  - **Rudimentary logic functions**
  - **Decoding using Decoders**
    - **Implementing Combinational Functions with Decoders**
  - **Encoding using Encoders**
  - **Selecting using Multiplexers**
    - **Implementing Combinational Functions with Multiplexers**

Combinational Circuits

- **A combinational logic circuit has:**
  - **A set of *m* Boolean inputs,**
  - **A set of *n* Boolean outputs, and**
  - ***n* switching functions, each mapping the 2^m input combinations to an output such that the current output depends only on the current input values**
- **A block diagram:**

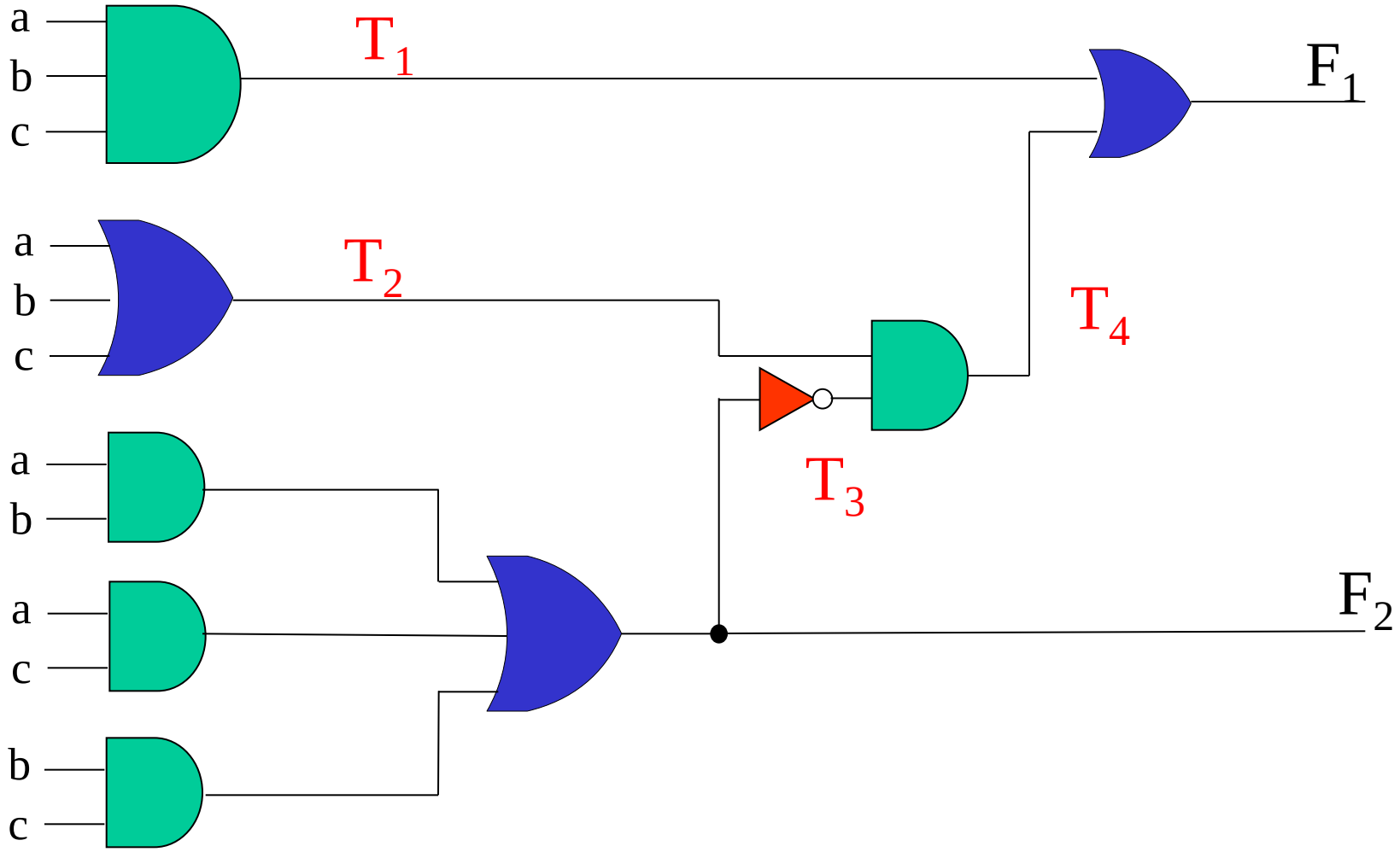


Analysis of Combinational Digital Circuits

- **Analysis: Finding the Boolean function that a digital circuit implements**
  - **A digital circuits is given.**
  - **We should find**
    1. **Boolean function**
    2. **Truth table**
    3. **Some information about the operation of the digital circuit.**

Finding the Boolean Function

Example



Example: Finding the Boolean Function

- **The Boolean functions of the shown points on the circuit**
  - $T_1 = abc$
  - $T_2 = a + b + c$
  - $F_2 = ab + ac + bc$
  - $T_3 = F_2' = (ab + ac + bc)'$
  - $T_4 = T_3T_2 = (ab + ac + bc)' (a + b + c)$
  - $F_1 = T_1 + T_4$

$$= abc + (ab + ac + bc)' (a + b + c)$$
$$= abc + ((a' + b')(a' + c')(b' + c')) (a + b + c)$$
$$= abc + ((a' + a'c' + a'b' + b'c')(b' + c')) (a + b + c)$$
$$= abc + (a'b' + a'c' + a'b'c' + a'b' + a'b'c' + b'c' + b'c') (a + b + c)$$

Example: Finding the truth table

$F_1 = a \oplus b \oplus c$

$F_2 = ab + ac + bc$

elde toplam

a	b	c	T_1	T_2	T_3	T_4	F_2	F_1
0	0	0	0	0	1	0	0	0
0	0	1	0	1	1	1	0	1
0	1	0	0	1	1	1	0	1
0	1	1	0	1	0	0	1	0
1	0	0	0	1	1	1	0	1
1	0	1	0	1	0	0	1	0
1	1	0	0	1	0	0	1	0
1	1	1	1	1	0	0	1	1

Full Adder (FA)

Design Procedure

1. **Specification**
   - **Write a specification for the circuit if one is not already available**
2. **Formulation**
   - **Derive a truth table or initial Boolean equations that define the required relationships between the inputs and outputs, if not in the specification**
   - **Apply hierarchical design if appropriate**
3. **Optimization**
   - **Apply 2-level and multiple-level optimization**
   - **Draw a logic diagram or provide a netlist for the resulting circuit using ANDs, ORs, and inverters**

Design Procedure

4. **Technology Mapping**
   - **Map the logic diagram or netlist to the implementation technology selected**
5. **Verification**
   - **Verify the correctness of the final design manually or using simulation**

Addition Circuit

- **Addition of two 2-bit numbers**
  - $Z = X + Y$
  - $X = (x_1\ x_0)$ and $Y = (y_1\ y_0)$
  - $Z = (z_2\ z_1\ z_0)$
- **Addition of bits**

$s_0 = x_0 \oplus y_0$

$c_1 = x_0\ y_0$ (carry)

2. $s_1 = x_1 \oplus y_1 \oplus c_1$

   $c_2 = x_1\ y_1 + x_1\ c_1 + y_1\ c_1$

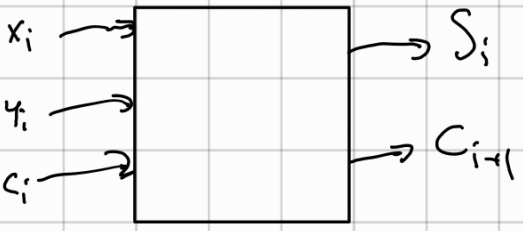
3. $s_2 = c_2$

x_i	y_i	c_i	s_i	c_{i+1}
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

Carry:	0	0	0	0	0			Carry:	0	1	1	0	0		
x:	0	1	1	0	0	12		x:	1	0	1	1	0	22	
y:	+ 1	0	0	0	1	+ 17		y:	+ 1	0	1	1	1	+ 23	
Sum:	1	1	1	0	1	29		Sum:	1	0	1	1	0	1	45

Note:

1. The carry input to the LSB is always '0'
2. The sum of two n-bit numbers has n+1 bits



$X = (x_2, x_1, x_0)_2$

$Y = (y_2, y_1, y_0)_2$

$S = X + Y\ (s_3, s_2, s_1, s_0)_2$

$C = (c_2, c_1, c_0)_2$

x_i	y_i	c_i	c_{i+1}	S_i
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

c_i \ $x_i y_i$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$C_{i+1} = x_i y_i' + y_i c_i' + x_i c_i'$

c_i \ $x_i y_i$	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$S_i = x_i' y_i' c_i' + x_i y_i' c_i' + x_i' y_i c_i' + x_i y_i c_i$

$= c_i' (x_i' y_i + x_i y_i') + c_i (x_i' y_i' + x_i y_i)$

$= c_i' (x \oplus y) + c_i (x \oplus y)'$

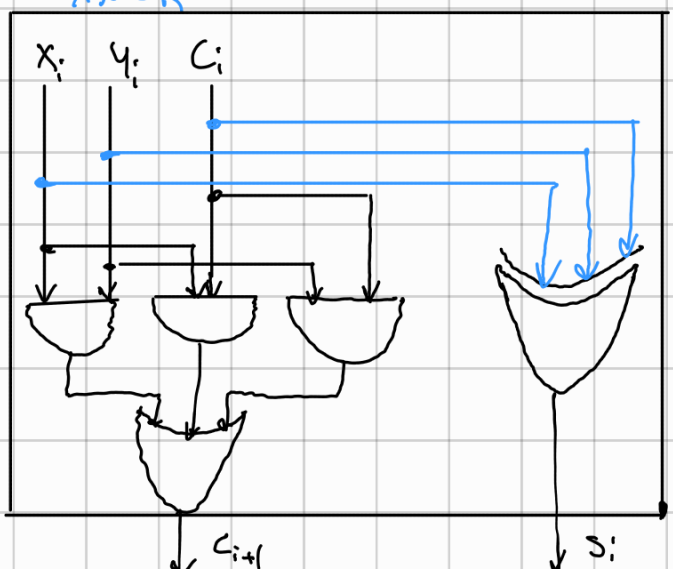
$= c_i \oplus x_i \oplus y_i$

XOR

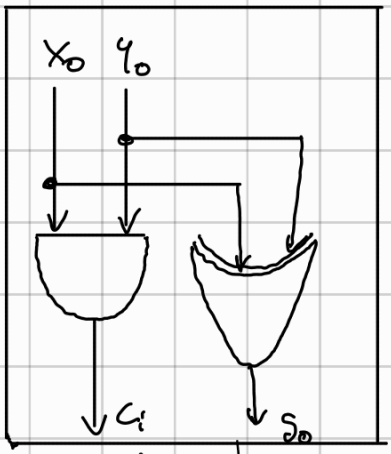
$X \oplus Y = x'y + xy'$

$(X \oplus Y)' = x'y' + xy$

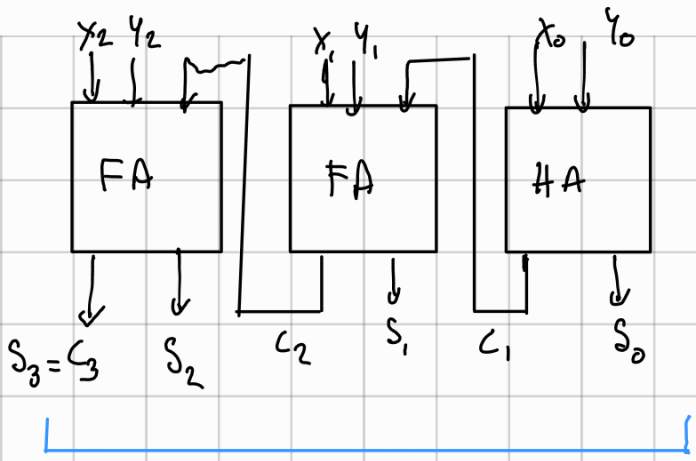
↳ XNOR



→ Full Adder FA



Half adder HA



Ripple Carry Adder

RCA

Addition Circuit for 2 bits addition

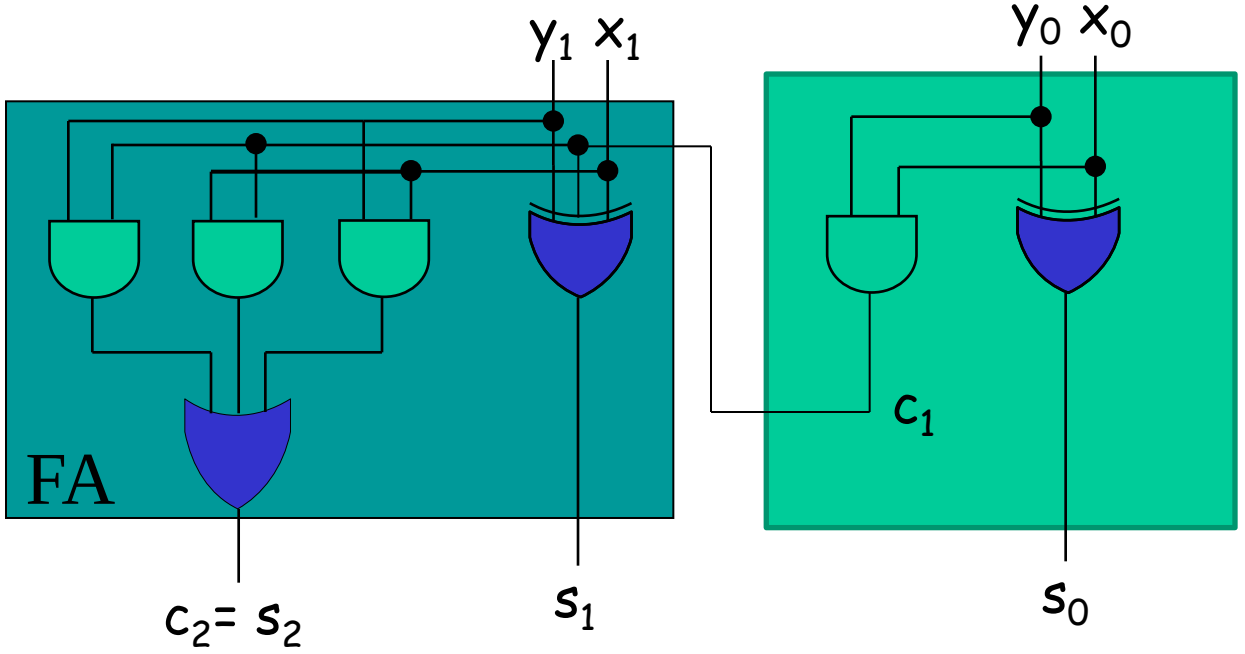
$s_2 = c_2$

$s_1 = x_1 \oplus y_1 \oplus e_1$

$c_2 = x_1 y_1 + x_1 c_1 + y_1 c_1$

$s_0 = x_0 \oplus y_0$

$c_1 = x_0 y_0$

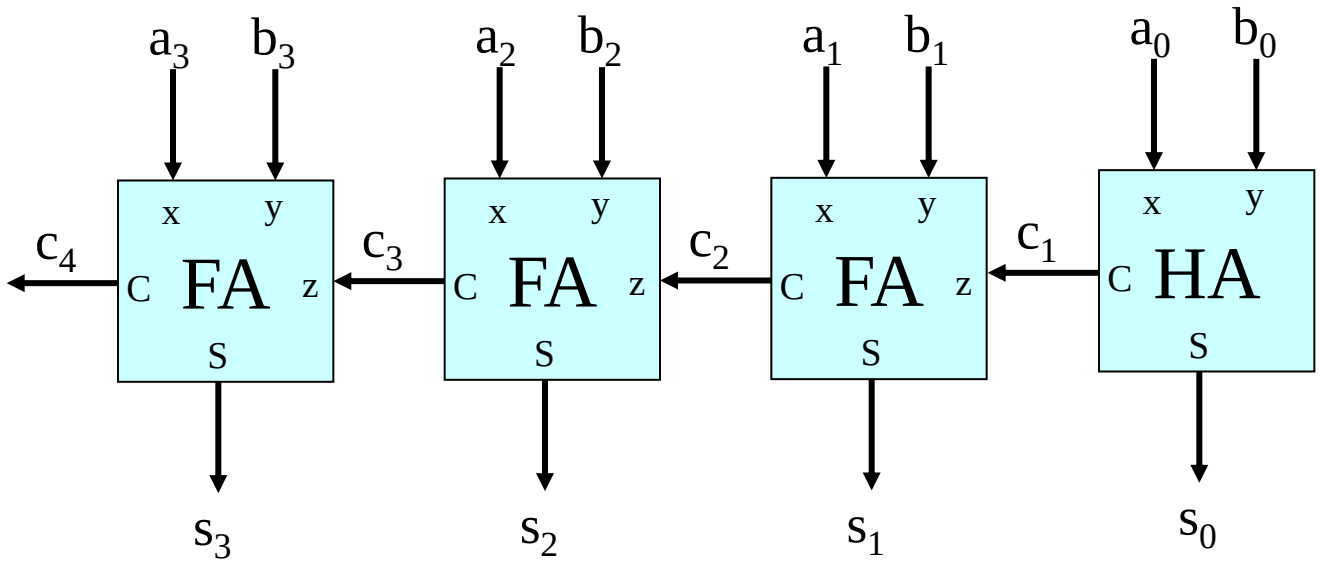


Full Adder

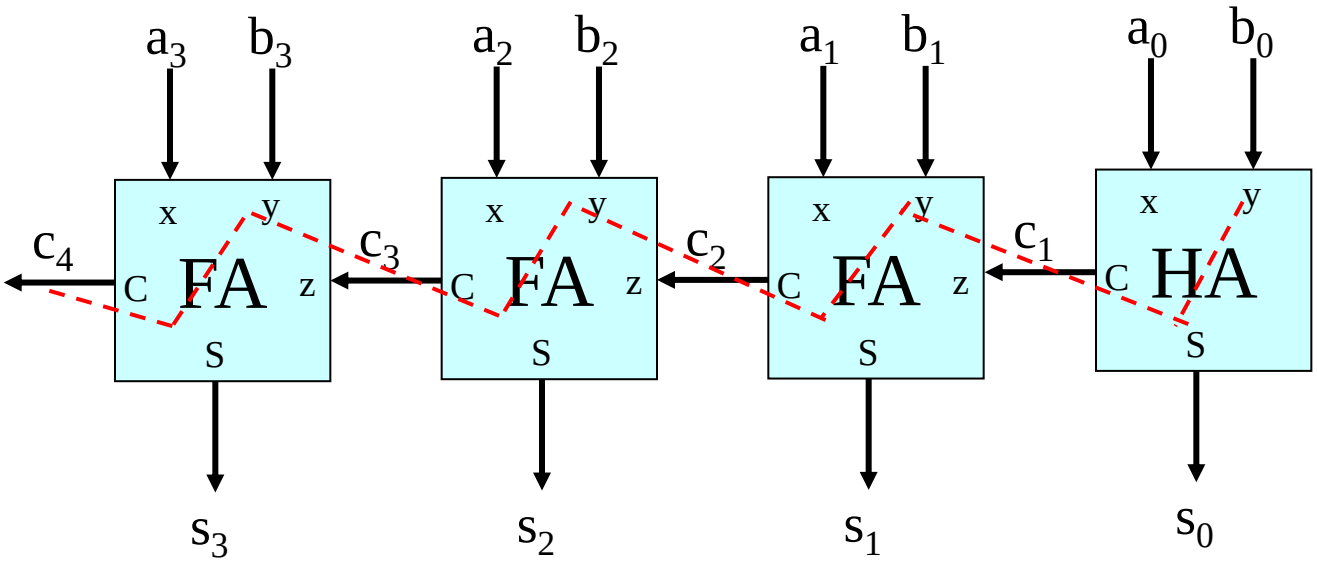
Half Adder

Integer Adder 1/2

- **n-bit Binary Adder:**
  - $A = (a_{n-1}, a_{n-2}, \ldots, a_1, a_0)$
  - $B = (b_{n-1}, b_{n-2}, \ldots, b_1, b_0)$
  - $A + B = S = (s_n, s_{n-1}, s_{n-2}, \ldots, s_1, s_0)$
- **4-bit Binary Adder:**



Integer Adder 2/2



Ripple-carry adder

Reusable Functions

- **We used top-down design method in ripple carry adder design.**
- **If we would use classical design method:**
  - **Number of inputs: 8**
  - **Number of outputs: 5**
  - **A truth table with $2^9 = 512$ is needed**
  - **We have to optimize 5 Boolean functions with 9 variables**
- **When reuse functions**
  - **We decompose the design to simpler operation blocks.**
  - **We design the complex function by the use of sub-blocks.**

Reusable Functions

- **Whenever possible, we try to decompose a complex design into common, *reusable* function blocks**
- **These blocks are**
  - **verified and well-documented**
  - **placed in libraries for future use**

Top-Down versus Bottom-Up

- **A *top-down design* proceeds from an abstract, high-level specification to a more and more detailed design by decomposition and successive refinement**
- **A *bottom-up design* starts with detailed primitive blocks and combines them into larger and more complex functional blocks**
- **Design usually proceeds top-down to known building blocks ranging from complete CPUs to primitive logic gates or electronic components.**
- **Much of the material in this chapter is devoted to learning about combinational blocks used in top-down design.**

Verification

- **Verification - show that the final circuit designed implements the original specification**
- **Simple specifications are:**
  - **truth tables**
  - **Boolean equations**
  - **HDL code**
- **If the above result from formulation and are not the original specification, it is critical that the formulation process be flawless for the verification to be valid!**

Basic Verification Methods

- **Manual Logic Analysis**
  - **Find the truth table or Boolean equations for the final circuit**
  - **Compare the final circuit truth table with the specified truth table, or**
  - **Show that the Boolean equations for the final circuit are equal to the specified Boolean equations**
- **Simulation**
  - **Simulate the final circuit (or its netlist, possibly written as an HDL) and the specified truth table, equations, or HDL description using test input values that fully validate correctness.**
  - **The obvious test for a combinational circuit is application of all possible "care" input combinations from the specification**