Combinational Logic Design

CSIM601251

Instructor: Tim Dosen DDAK

Slide By : Erdefi Rakun

Fasilkom UI

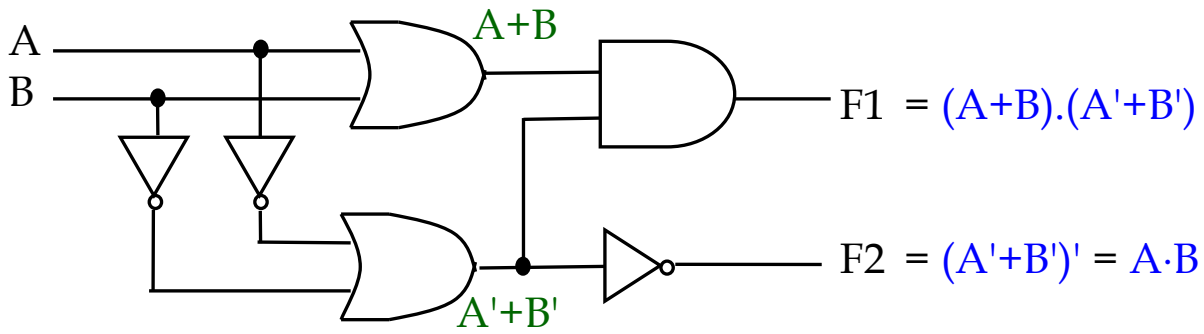
Outline:

- **Analysis Procedure**
- **Design Methods**
- Gate-level (SSI) Design
- Block-Level Design

Note: Portion of these materials are taken from Aaron Tan's slide and other portions of this material © 2008 by Pearson Education,Inc

Analysis Procedure

- Given a combinational circuit, how do you analyze its function?



- Steps:
  1. Label the inputs and outputs.
  2. Obtain the functions of intermediate points and the outputs.
  3. Draw the truth table.
  4. Deduce the functionality of the circuit ➲ Half adder.

A	B	(A+B)	(A'+B')	F1	F2
0	0	0	1	0	0
0	1	1	1	1	0
1	0	1	1	1	0
1	1	1	0	0	1

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

Design Example

1. Specification
   - BCD to Excess-3 code converter
   - Transforms BCD code for the decimal digits to Excess-3 code for the decimal digits
   - BCD code words for digits 0 through 9: 4-bit patterns 0000 to 1001, respectively
   - Excess-3 code words for digits 0 through 9: 4-bit patterns consisting of 3 (binary 0011) added to each BCD code word
   - Implementation:
     - multiple-level circuit
     - NAND gates (including inverters)

Design Example (continued)

2. Formulation

- Conversion of 4-bit codes can be most easily formulated by a truth table
- Variables
  - BCD: A,B,C,D
- Variables
  - Excess-3 W,X,Y,Z
- Don't Cares
  - BCD 1010 to 1111

Input BCD A B C D	Output Excess-3 W X Y Z
0 0 0 0	0 0 1 1
0 0 0 1	0 1 0 0
0 0 1 0	0 1 0 1
0 0 1 1	0 1 1 0
0 1 0 0	0 1 1 1
0 1 0 1	1 0 0 0
0 1 1 0	1 0 0 1
0 1 1 1	1 0 1 0
1 0 0 0	1 0 1 1
1 0 0 1	1 1 0 0

Design Example (continued)

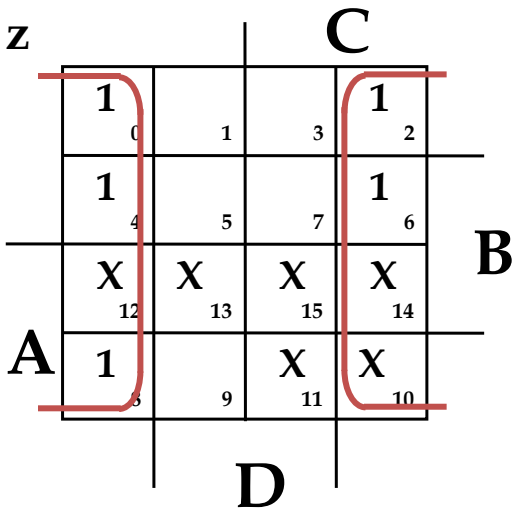
3. Optimization
   a. 2-level using K-maps

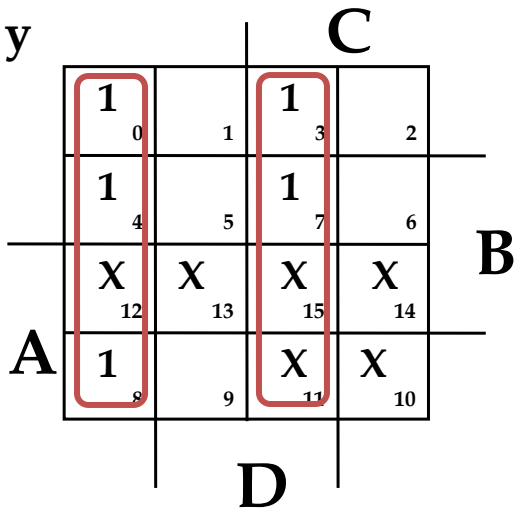
$W = A + BC + BD$

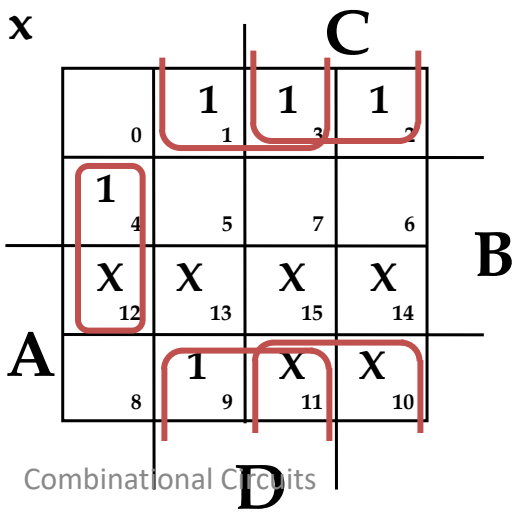
$X = \overline{B}C + \overline{B}D + B\overline{C}.\overline{D}$

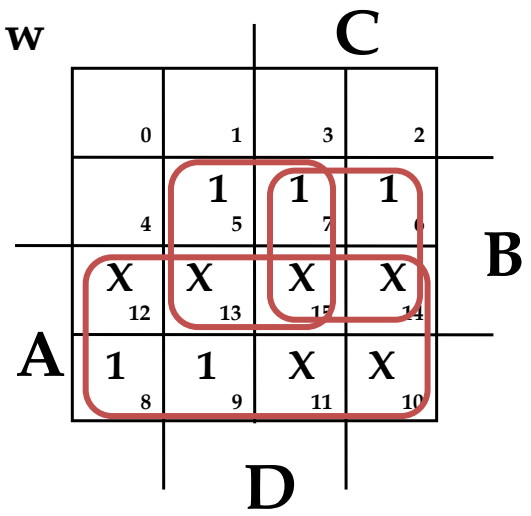
$Y = CD + \overline{C}.\overline{D}$

$Z = \overline{D}$









Design Example (continued)

3. Optimization (continued)

b. Multiple-level using transformations

$W = A + BC + BD$

$X = \overline{B}C + \overline{B}D + B\overline{C}.\overline{D}$

$Y = CD + \overline{C}.\overline{D}$

$Z = \overline{D}$

G = 7 + 10 + 6 + 0 = 23

– Perform extraction, finding factor:

$T_1 = C + D$

$W = A + BT_1$

$X = \overline{B}T_1 + B\overline{C}.\overline{D}$

$Y = CD + \overline{C}.\overline{D}$

$Z = \overline{D}$

G = 2 + 4 + 7 + 6 + 0 = 19

Design Example (continued)

3. Optimization (continued)

b. Multiple-level using transformations

$$T_1 = C + D$$

$$W = A + BT_1$$

$$X = \overline{B}T_1 + B\overline{C}.\overline{D}$$

$$Y = CD + \overline{C}.\overline{D}$$

$$Z = \overline{D}$$

G = 19

– An additional extraction not shown in the text since it uses a Boolean transformation: $(\overline{C}.\overline{D} = \overline{C + D} = \overline{T_1})$:

$$W = A + BT_1$$

$$X = \overline{B}T_1 + B\overline{T_1}$$

$$Y = CD + \overline{T_1}$$

$$Z = \overline{D}$$

G = 2 + 4 + 6 + 4 + 0 = 16!

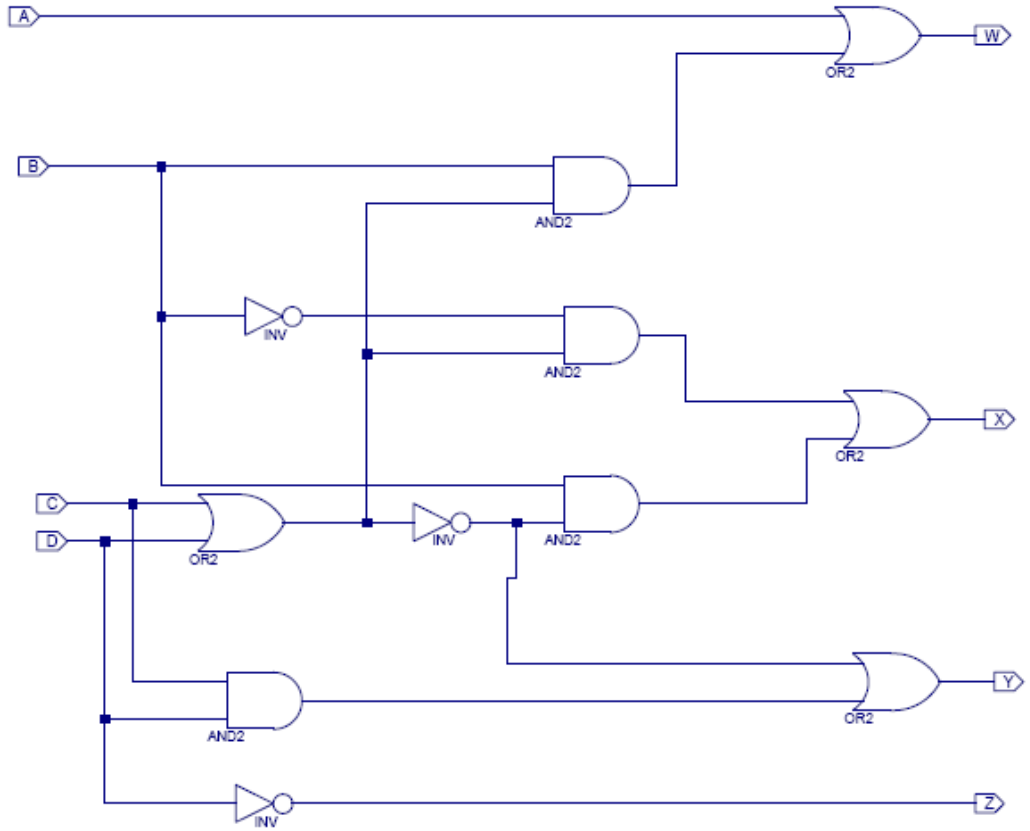
Design Example (continued)

4. Mapping Procedures
   - To NAND gates
   - To NOR gates
   - Mapping to multiple types of logic blocks is covered in the reading supplement: Advanced Technology Mapping.

Design Example (continued)

Technology Mapping

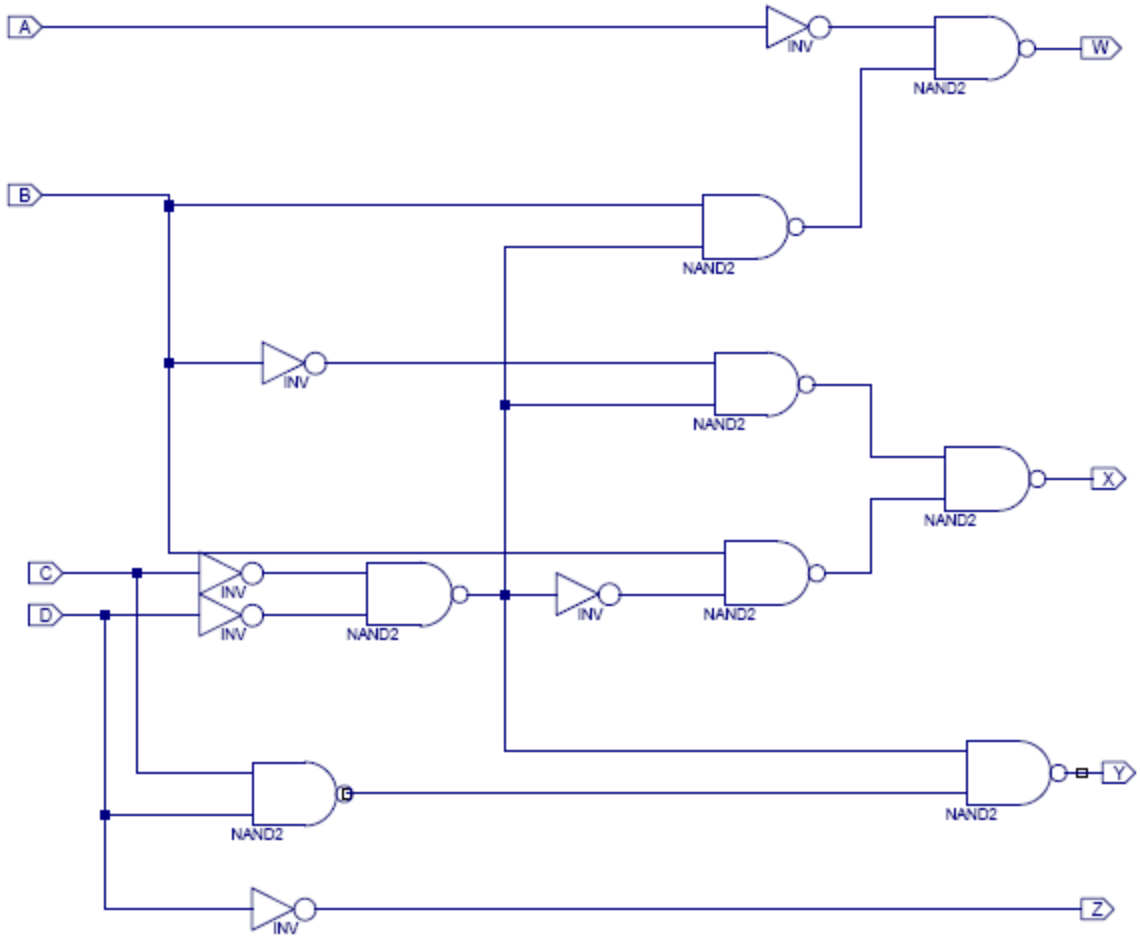
Mapping with a library containing AND, OR, NOT



Design Example (continued)

Technology Mapping

Mapping with a library containing inverters and 2-input NAND



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

Verification Example: Manual Analysis

- BCD-to-Excess 3 Code Converter
  - Find the SOP Boolean equations from the final circuit.
  - Find the truth table from these equations
  - Compare to the formulation truth table
- Finding the Boolean Equations:

$T_1 = C + D = C + D$

$W = A (T_1 B) = A + B T_1$

$X = (T_1 B) (B) = T_1 C + D B + B \overline{C}\,\overline{D}$

$Y = C + D = C + D + \overline{C}\,\overline{D}$

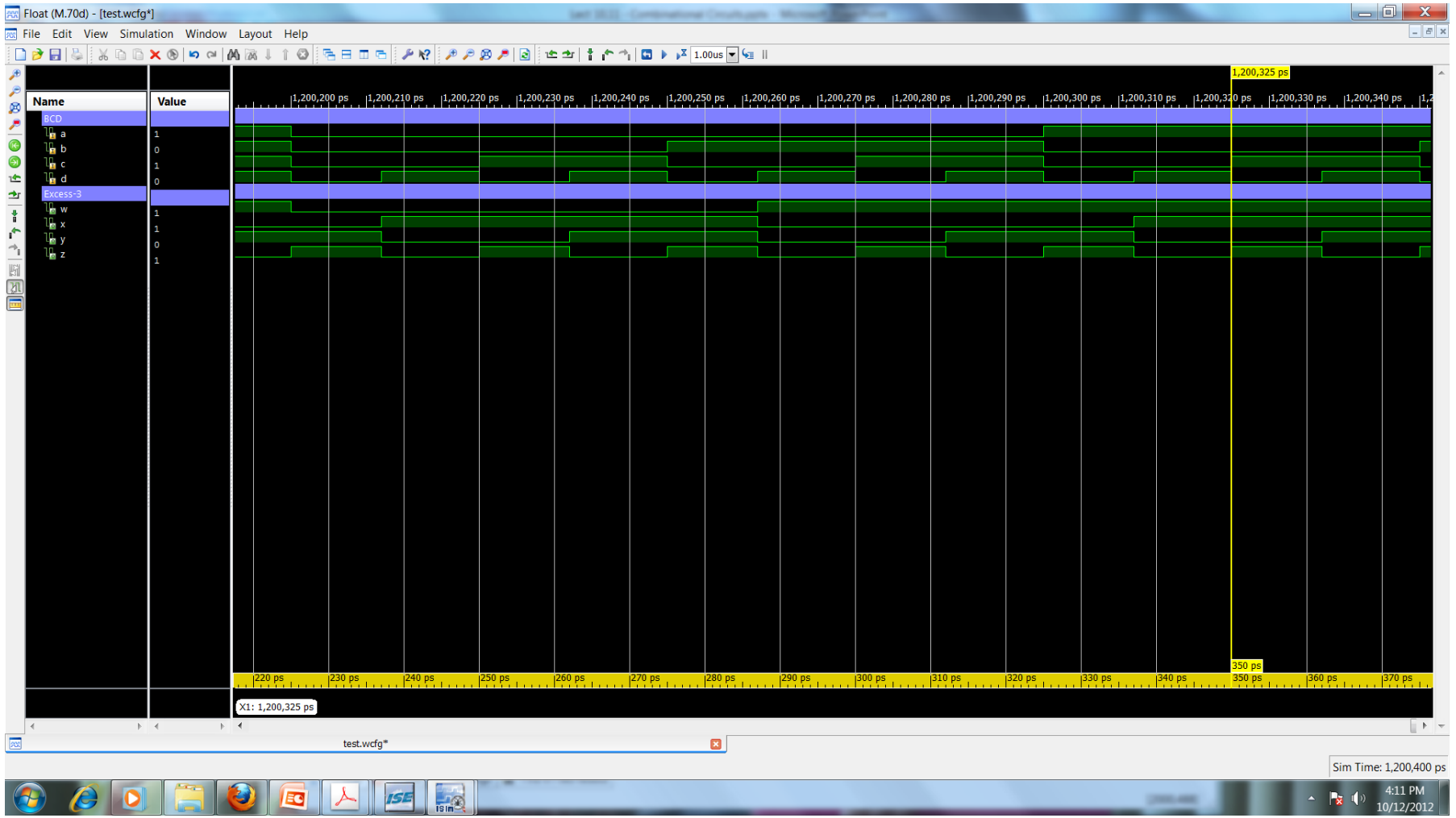
Verification Example: Manual Analysis

- Find the circuit truth table from the equations and compare to specification truth table:

Input BCD A B C D	Output Excess-3 WXYZ
0 0 0 0	0 0 1 1
0 0 0 1	0 1 0 0
0 0 1 0	0 1 0 1
0 0 1 1	0 1 1 0
0 1 0 0	0 1 1 1
0 1 0 1	1 0 0 0
0 1 1 0	1 0 0 1
0 1 1 1	1 0 1 0
1 0 0 0	1 0 1 1
1 0 0 1	1 1 0 0

The tables match!

Verification Example: Simulation Analysis



Outline:

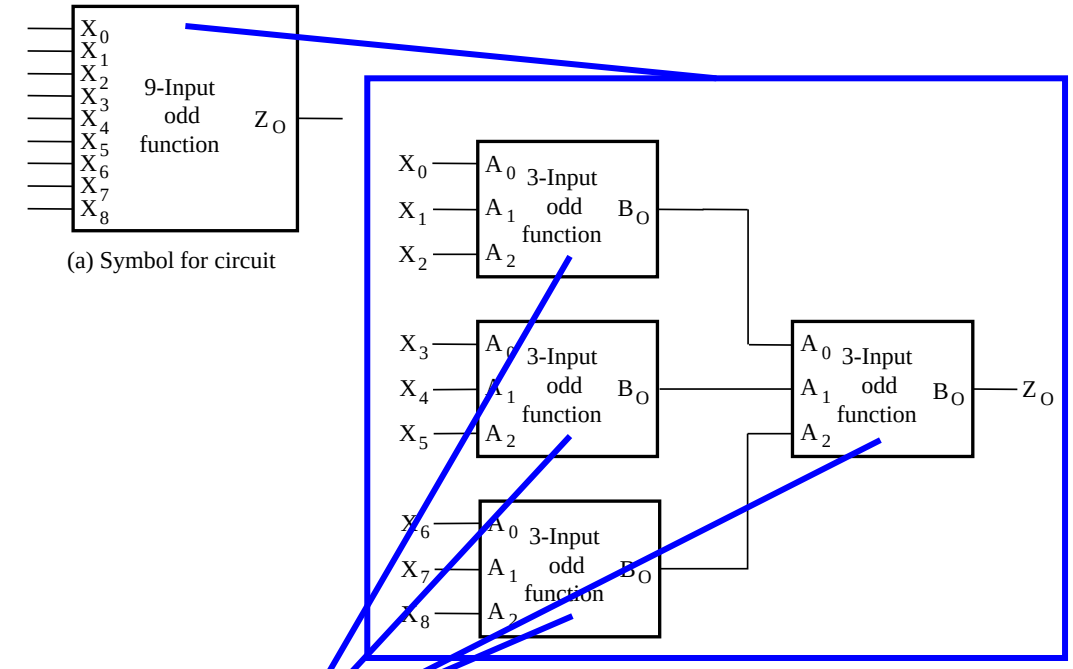
- Analysis Procedure
- Design Methods
- **Gate-level (SSI) Design**
- **Block-Level Design**

Note: Portion of these materials are taken from Aaron Tan's slide and other portions of this material © 2008 by Pearson Education,Inc

Beginning Hierarchical Design

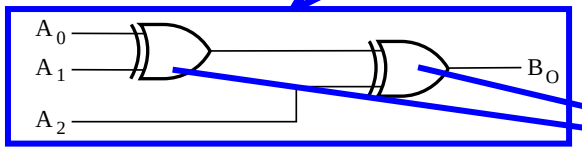
- To control the complexity of the function mapping inputs to outputs:
  - Decompose the function into smaller pieces called *blocks*
  - Decompose each block's function into smaller blocks, repeating as necessary until all blocks are small enough
  - Any block not decomposed is called a *primitive block*
  - The collection of all blocks including the decomposed ones is a *hierarchy*
- Example: 9-input parity tree (see next slide)
  - Top Level: 9 inputs, one output
  - 2nd Level: Four 3-bit odd parity trees in two levels
  - 3rd Level: Two 2-bit exclusive-OR functions
  - Primitives: Four 2-input NAND gates
  - Design requires 4 X 2 X 4 = 32 2-input NAND gates

Hierarchy for Parity Tree Example

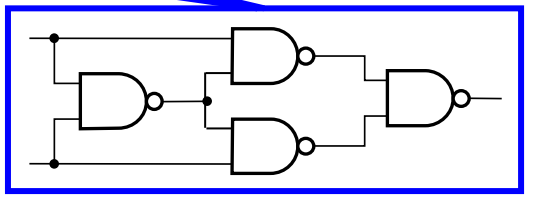


(a) Symbol for circuit

(b) Circuit as interconnected 3-input odd function blocks



(c) 3-input odd function circuit as interconnected exclusive-OR blocks



(d) Exclusive-OR block as interconnected NANDs

Reusable Functions

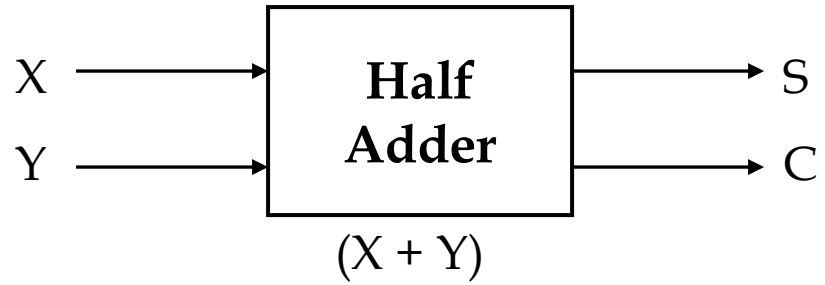
- Whenever possible, we try to decompose a complex design into common, *reusable* function blocks
- These blocks are
  - verified and well-documented
  - placed in libraries for future use

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.

Gate-Level (SSI) Design: Half Adder (1/2)

- Design procedure:
  1. State problem
     Example: Build a Half Adder.
  2. Determine and label the inputs and outputs of circuit.
     Example: Two inputs and two outputs labelled, as shown below.



  3. Draw the truth table.

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Gate-Level (SSI) Design: Half Adder (2/2)

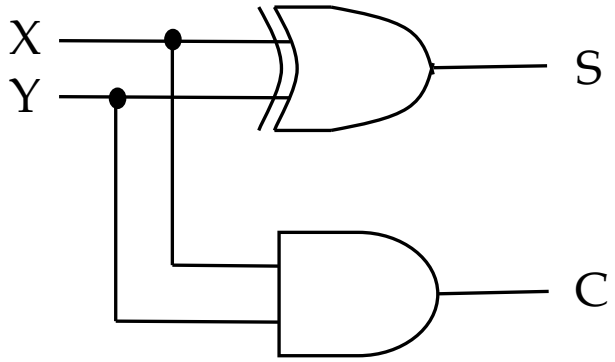
4. Obtain simplified Boolean functions.

   Example: $C = X \cdot Y$

   $S = X' \cdot Y + X \cdot Y' = X \oplus Y$

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

5. Draw logic diagram.



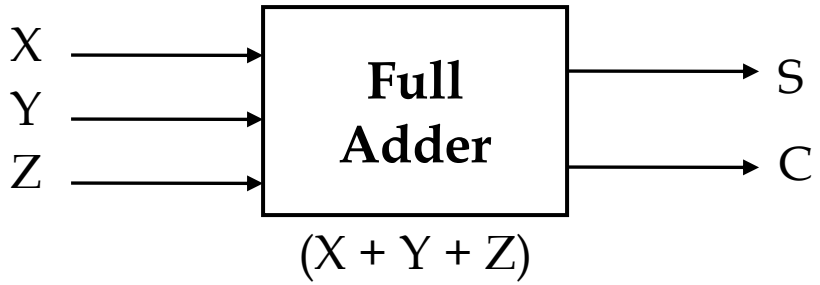
Half Adder

Gate-Level (SSI) Design: Full Adder (1/5)

- Half adder adds up only two bits.
- To add two binary numbers, we need to add 3 bits (including the *carry*).
  - Example:

	1	1	1		carry
	0	0	1	1	X
+	0	1	1	1	Y
	1	0	1	0	S

- Need Full Adder (so called as it can be made from two half adders).



Gate-Level (SSI) Design: Full Adder (2/5)

- Truth table:

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

Note:
Z - carry in (to the current position)
C - carry out (to the next position)

- Using K-map, simplified SOP form:

C = ?
S = ?

C

X \ YZ	00	01	11	10
0			1	
1		1	1	1

S

X \ YZ	00	01	11	10
0		1		1
1	1		1	

Gate-Level (SSI) Design: Full Adder (3/5)

- Alternative formulae using algebraic manipulation:

$$
\begin{aligned}
C &= X\cdot Y + X\cdot Z + Y\cdot Z \\
&= X\cdot Y + (X + Y)\cdot Z \\
&= X\cdot Y + ((X\oplus Y) + X\cdot Y)\cdot Z \\
&= X\cdot Y + (X\oplus Y)\cdot Z + X\cdot Y\cdot Z \\
&= X\cdot Y + X\cdot Y\cdot Z + (X\oplus Y)\cdot Z \\
&= X\cdot Y + (X\oplus Y)\cdot Z
\end{aligned}
$$

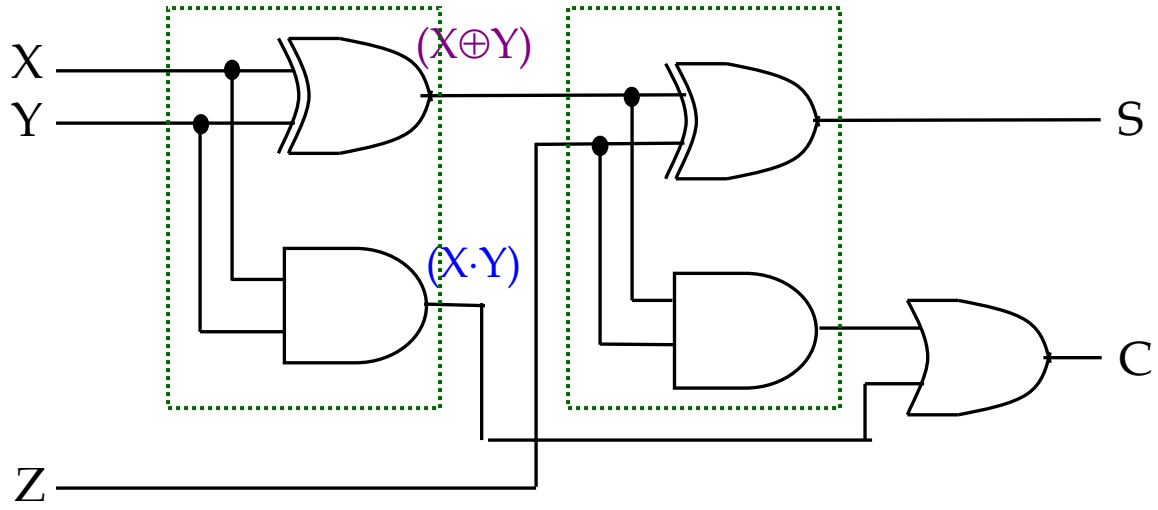
$$
\begin{aligned}
S &= X'\cdot Y'\cdot Z + X'\cdot Y\cdot Z' + X\cdot Y'\cdot Z' + X\cdot Y\cdot Z \\
&= X'\cdot(Y'\cdot Z + Y\cdot Z') + X\cdot(Y'\cdot Z' + Y\cdot Z) \\
&= X'\cdot(Y\oplus Z) + X\cdot(Y\oplus Z)' \\
&= X\oplus(Y\oplus Z)
\end{aligned}
$$

Gate-Level (SSI) Design: Full Adder (4/5)

- Circuit for above formulae:

  $C = X \cdot Y + (X \oplus Y) \cdot Z$

  $S = X \oplus (Y \oplus Z)$



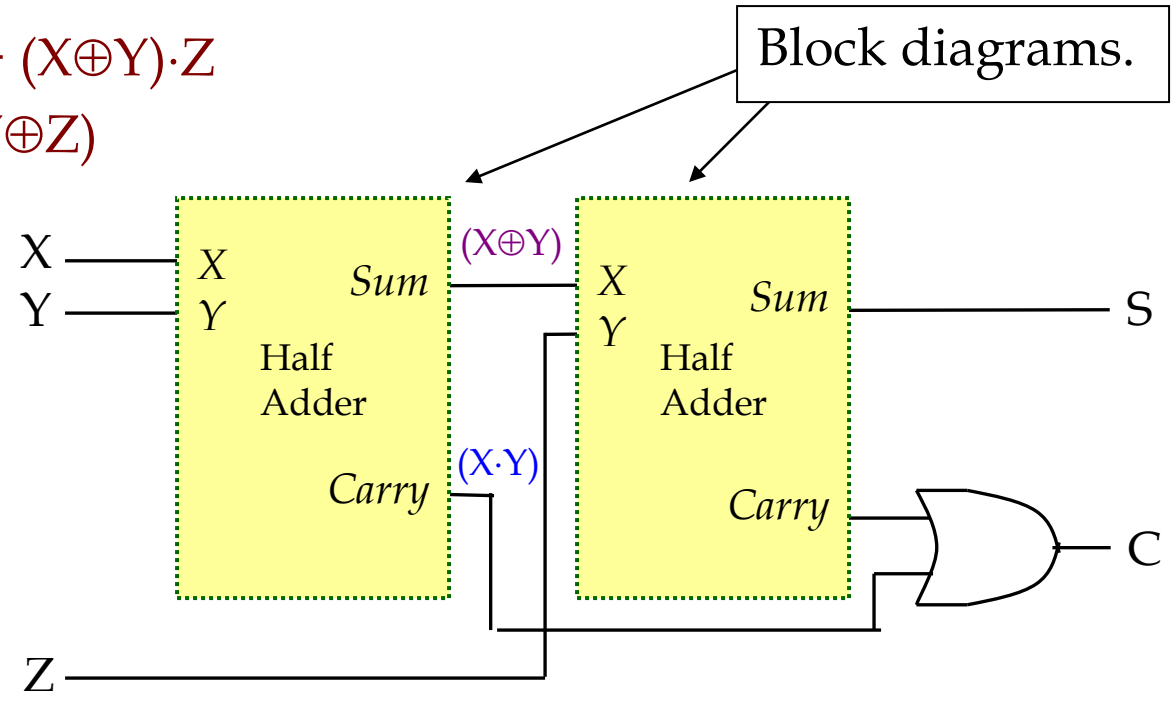
Full Adder made from two Half-Adders (+ an OR gate).

Gate-Level (SSI) Design: Full Adder (5/5)

- Circuit for above formulae:

  C= X·Y + (X⊕Y)·Z

  S = X⊕(Y⊕Z)



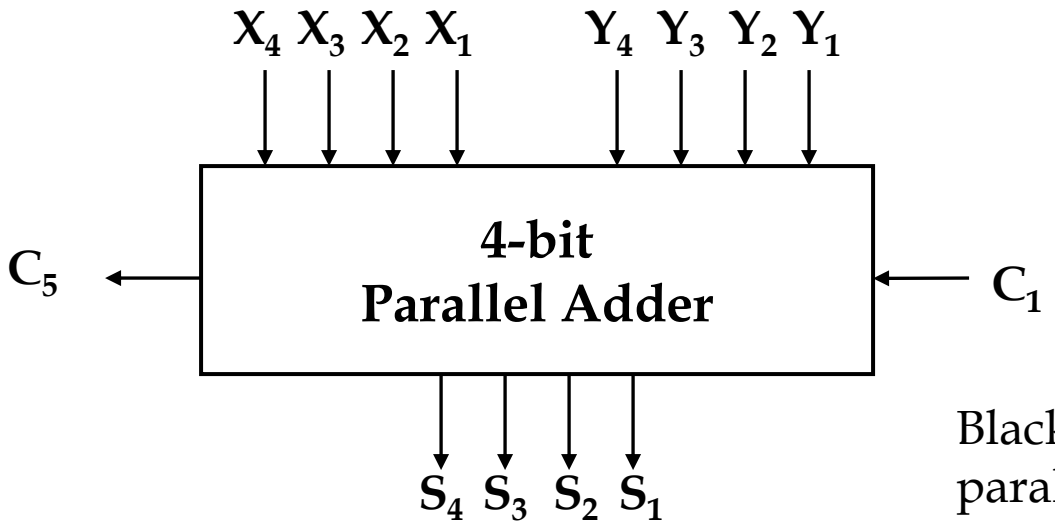
Full Adder made from two Half-Adders (+ an OR gate).

Block-Level Design

- More complex circuits can also be built using block-level method.
- In general, block-level design method (as opposed to gate-level design) relies on algorithms or formulae of the circuit, which are obtained by decomposing the main problem to sub-problems recursively (until small enough to be directly solved by blocks of circuits).
- Simple examples using 4-bit parallel adder as building blocks for 16-bit Parallel Adder

4-Bit Parallel Adder (1/4)

- Consider a circuit to add two 4-bit numbers together and a carry-in, to produce a 5-bit result.



Black-box view of 4-bit parallel adder

- 5-bit result is sufficient because the largest result is:

$$1111_2 + 1111_2 + 1_2 = 11111_2$$

4-Bit Parallel Adder (2/4)

- SSI design technique should not be used here.
- Truth table for 9 inputs is too big: $2^9 = 512$ rows!

$X_4X_3X_2X_1$	$Y_4Y_3Y_2Y_1$	C_1	C_5	$S_4S_3S_2S_1$
0 0 0 0	0 0 0 0	0	0	0 0 0 0
0 0 0 0	0 0 0 0	1	0	0 0 0 1
0 0 0 0	0 0 0 1	0	0	0 0 0 1
…	…	…	…	…
0 1 0 1	1 1 0 1	1	1	0 0 1 1
…	…	…	…	…
1 1 1 1	1 1 1 1	1	1	1 1 1 1

- Simplification becomes too complicated.

4-Bit Parallel Adder (3/4)

- Alternative design possible.
- Addition formula for each pair of bits (with carry in),

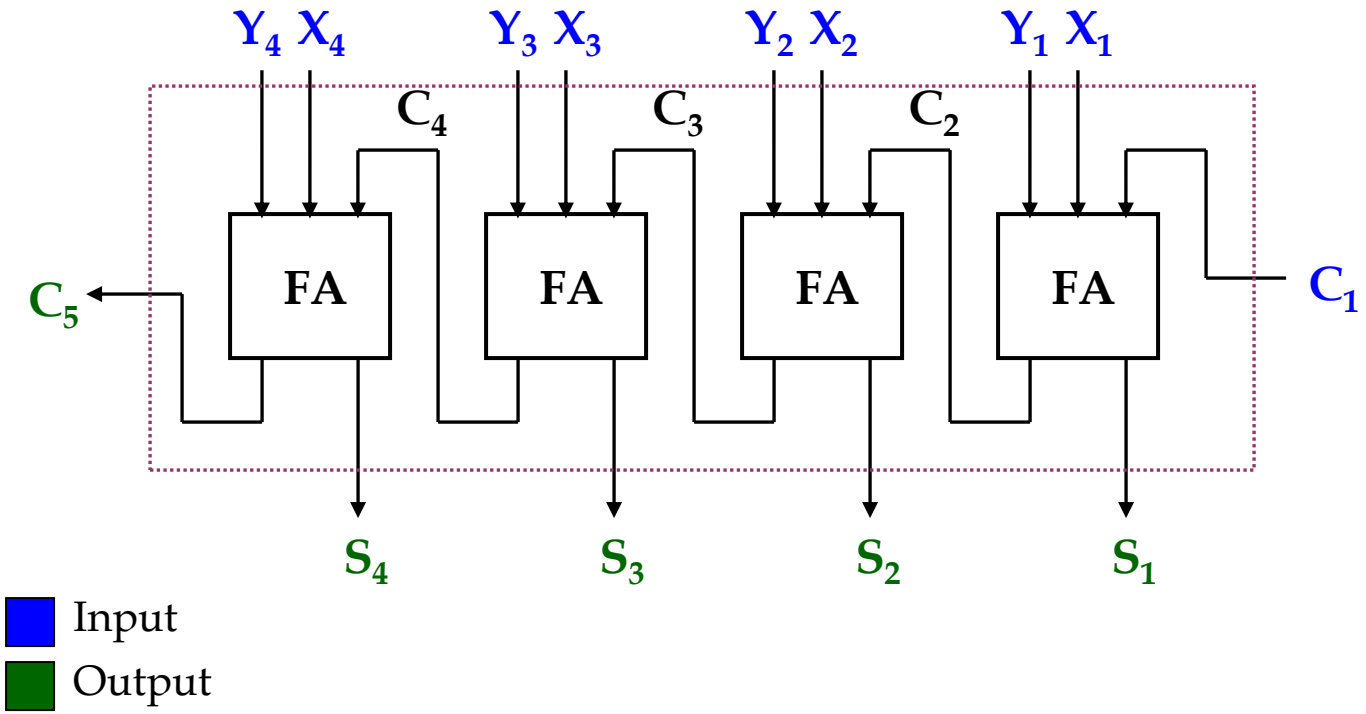
$$C_{i+1}S_i = X_i + Y_i + C_i$$

has the same function as a full adder:

$$C_{i+1} = X_i \cdot Y_i + (X_i \oplus Y_i) \cdot C_i$$
$$S_i = X_i \oplus Y_i \oplus C_i$$

4-Bit Parallel Adder (4/4)

- Cascading 4 full adders via their carries, we get:

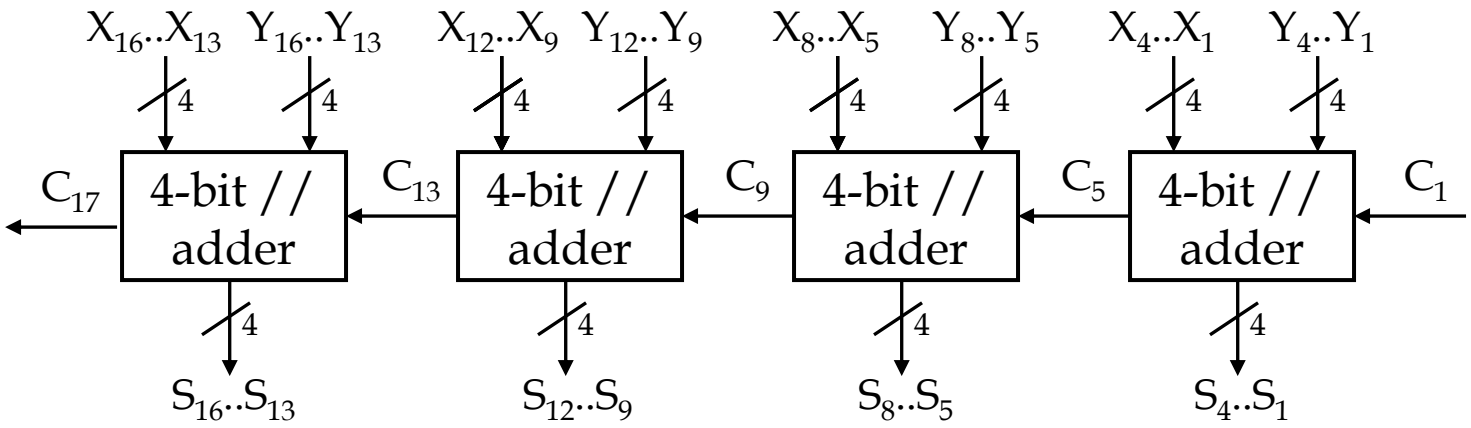


Parallel Adder

- Note that carry propagated by cascading the carry from one full adder to the next.
- Called Parallel Adder because inputs are presented simultaneously (in parallel). Also called Ripple-Carry Adder.

16-Bit Parallel Adder

- Larger parallel adders can be built from smaller ones.
- Example: A 16-bit parallel adder can be constructed from four 4-bit parallel adders:



A 16-bit parallel adder