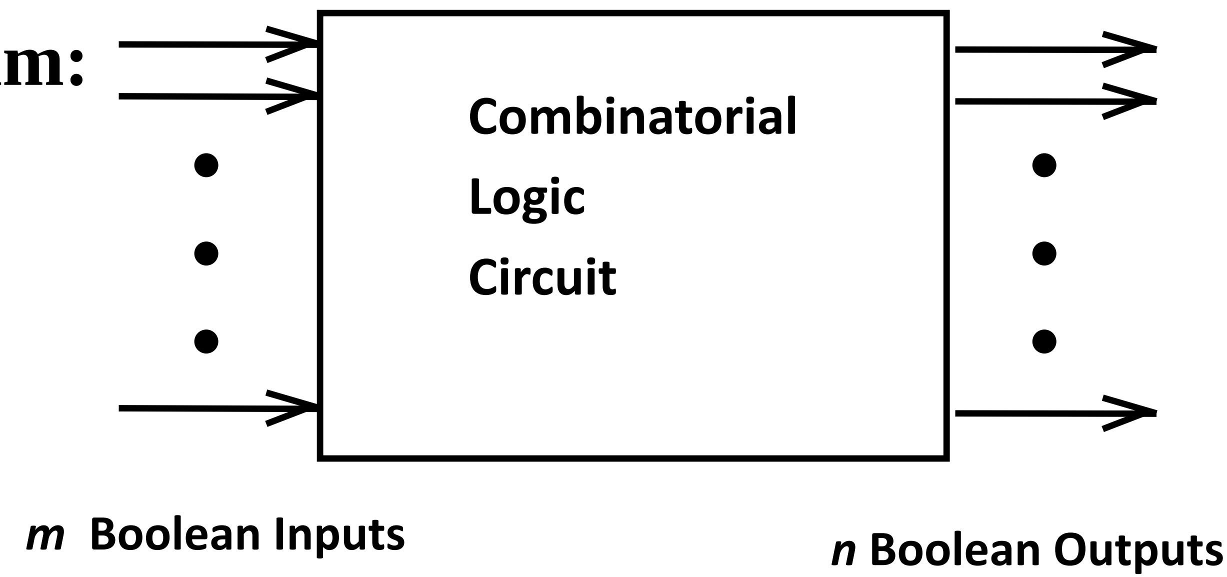
Chapter 3 – Combinational Digital Circuit Design

Part 1 – Implementation Technology and Logic Design

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:**



Addition of Positive Numbers

Carry	00000		01100	
X	**01100**	**12**	**10110**	**22**
Y	**+10001**	**+17**	**+10111**	**+23**
Sum	11101	**29**	101101	**45**

Note:

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

Addition Circuit

- **Addition of two 2-bit numbers**
  - $S = X + Y$
  - $X = (x_1\ x_0)$ and $Y = (y_1\ y_0)$
  - $S = (s_2\ s_1\ s_0)$

- **Addition of bits**
  1. $s_0 = x_0 \oplus y_0$
     $c_1 = x_0\, y_0$ (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

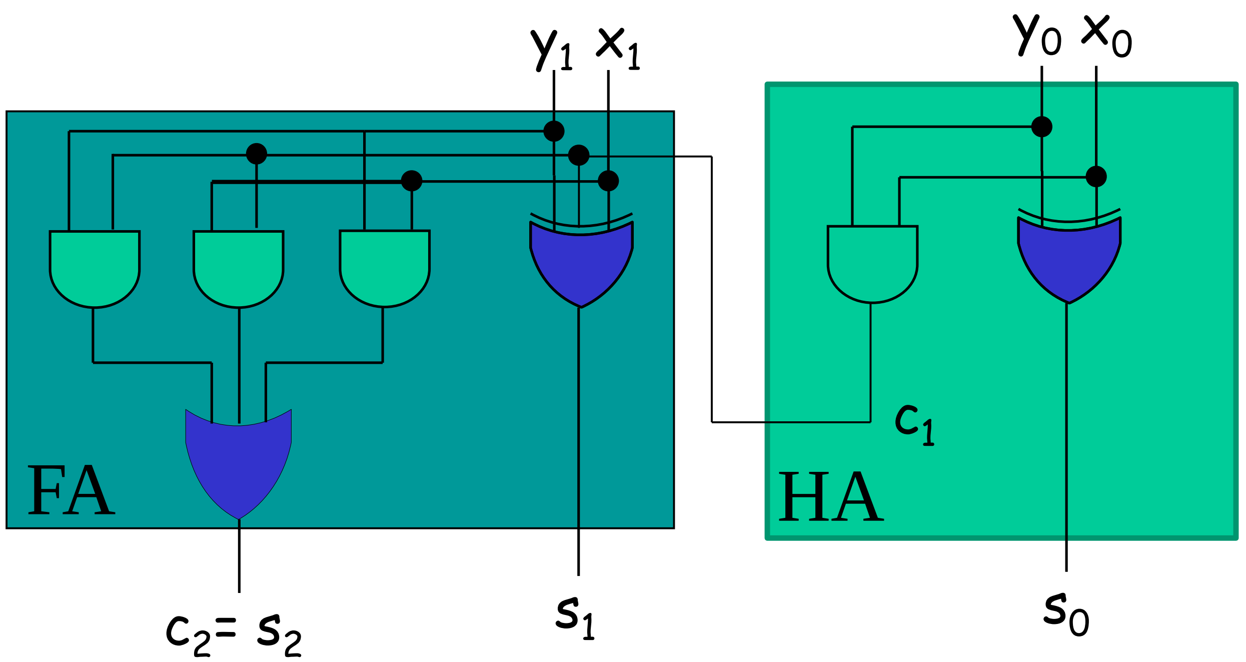
Addition Circuit

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

$$s_0 = x_0 \oplus y_0$$

$$s_2 = c_2 \qquad c_2 = x_1\, y_1 + x_1\, c_1 + y_1\, c_1$$

$$c_1 = x_0\, y_0$$

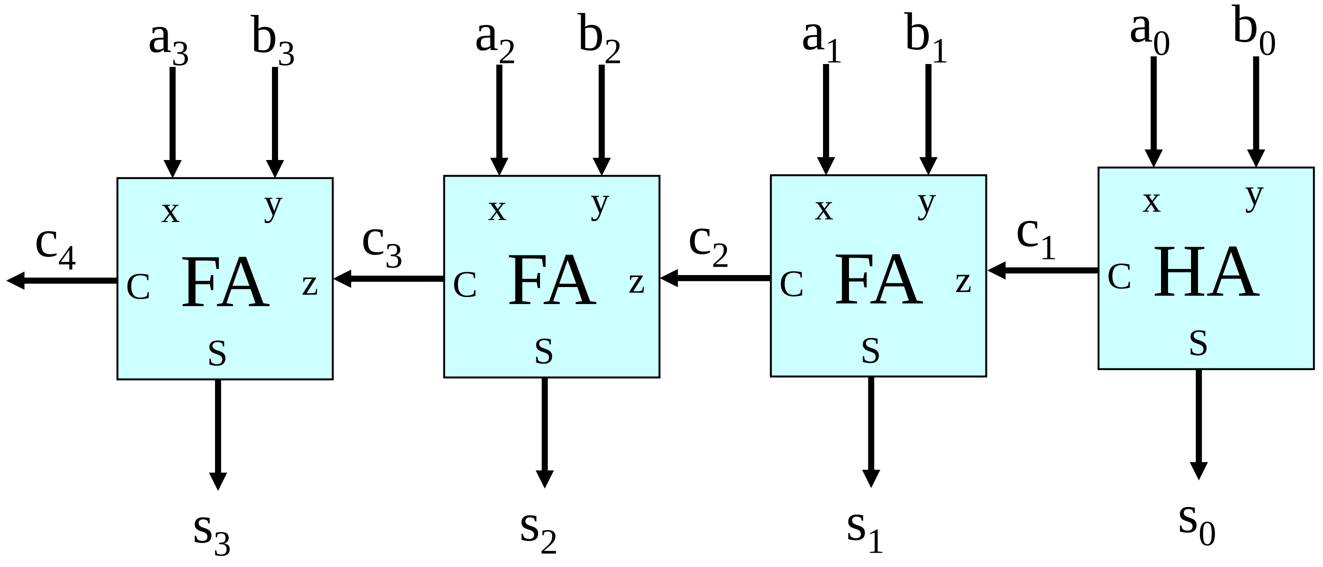


Full Adder

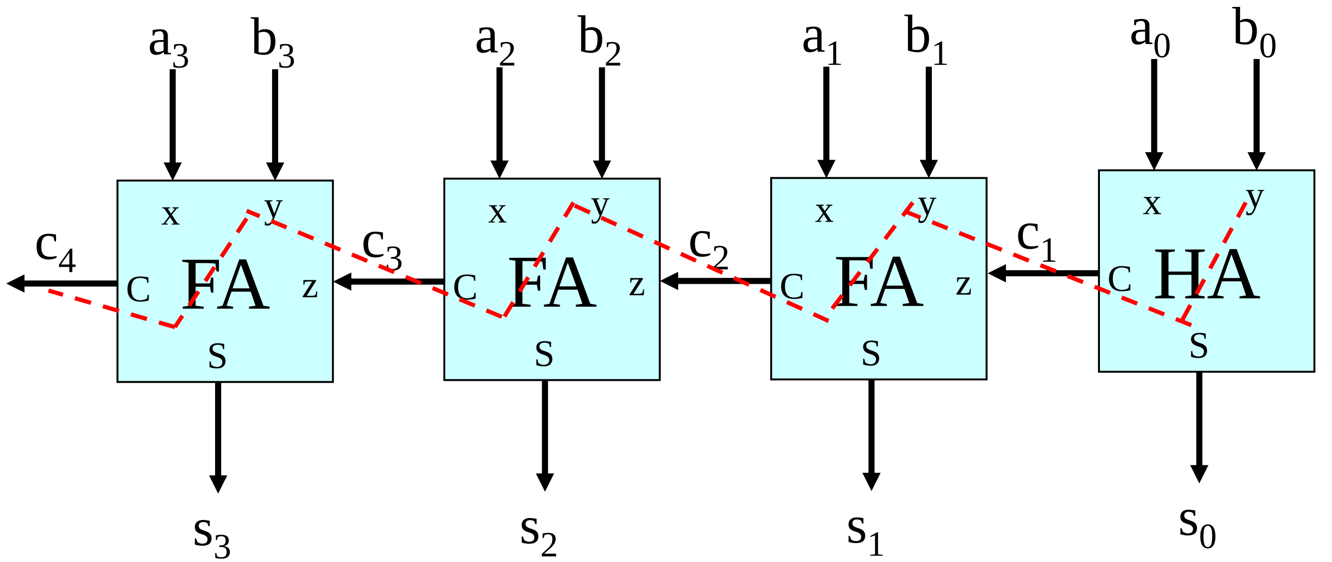
Half Adder

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 <u>formulation</u> and are not the <u>original specification</u>, 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**