Ollscoil na Gaillimhe
University of Galway

CT101 Computing Systems

Dr. Bharathi Raja Chakravarthi
Lecturer-above-the-bar
Email: bharathi.raja@universityofgalway.ie

University *of* Galway.ie

Ollscoil na Gaillimhe
University of Galway

Recap:
Binary Adder-Subtractor

University
*of*Galway.ie

Introduction

- A binary adder–subtractor is a combinational circuit that performs the arithmetic operations of addition and subtraction with binary numbers.
- We will develop this circuit by means of a hierarchical design.
- The half adder design is carried out first, from which we develop the full adder.

Introduction

Creating an Adder for n-bit Numbers:

- n full adders are connected in a cascade configuration to add two n-bit binary numbers.

Subtraction Capability:

- Subtraction is facilitated by incorporating a complementing circuit that works with the adders.
- This design uses the concept of two's complement for binary subtraction.

Half Adder

- A digital circuit with two binary inputs and two binary outputs.
- Inputs represent two single binary digits (bits) to be added.
- Outputs produce the resulting sum and the carry bit.

Input/Output Designation:

- Inputs: x and y
- Outputs: S (sum) and C (carry)

Half Adder

Truth Table:

- Details the logic of the half-adder operation.
- Output *C* is high (1) only when both *x* and *y* are high (1).
- Output *S* represents the binary sum without the carry.

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

Full Adder

- A digital circuit that computes the arithmetic sum of three input bits.
- Used for bit-by-bit addition of *n*-bit binary numbers, considering carry.

Input/Output Designation:

- **Inputs:** *x*, *y* (significant bits), *z* (carry from previous addition)
- **Outputs:** *S* (sum), *C* (carry)

Full Adder

Functionality:

- Adds bits *x*, *y*, and *z* to produce a two-bit result (as binary sums can range from 0 to 3).
- Output *S* is the least significant bit of the sum.
- Output *C* represents the carry-out of the addition.

Truth Table:

- Eight rows representing all possible combinations of three inp
- Outputs are derived based on the binary sum of input bits.

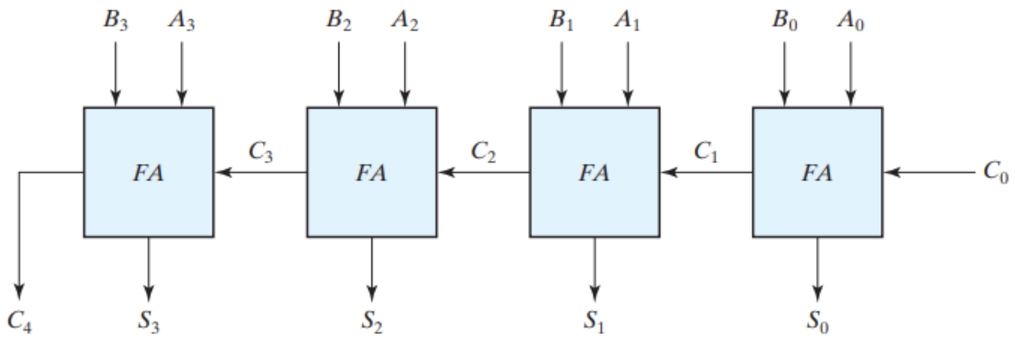
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Binary Adder

- A digital circuit is designed to compute the sum of two binary numbers.
- Built by connecting full adders in series, with each full adder's carry-out connected to the next full adder's carry-in.

Construction:

- Requires *n* full adders to add two *n*-bit numbers.
- For a 4-bit adder, four full adders are interconnected as shown in Figure below.
- The least significant full adder receives an input carry (denoted C_0), typically set to 0.



Binary Adder

Operation:

- Augend (A) and addend (B) bits are labeled from right to left, with the least significant bit as subscript 0.
- Carries ripple from one full adder to the next in sequence, from C_0 to C_4.

Example Calculation with A = 1011 and B = 0011:

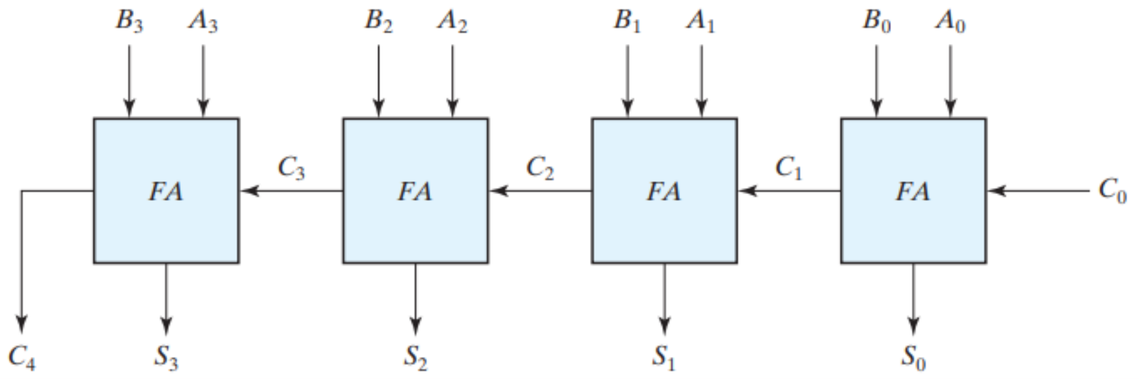
- Start with the least significant bit: $A_0 = 1$, $B_0 = 1$ plus the initial carry $C_0 = 0$.
- The first full adder computes S_0 and generates a carry C_1 for the next position.
- This process continues for each bit, with carries rippling to the next full adder.
- The final sum bits (S) are generated in sequence, resulting in $S = 1110$.

Sum S = 1110 is formed with the four-bit adder as shown right

Subscript *i*:	3	2	1	0	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

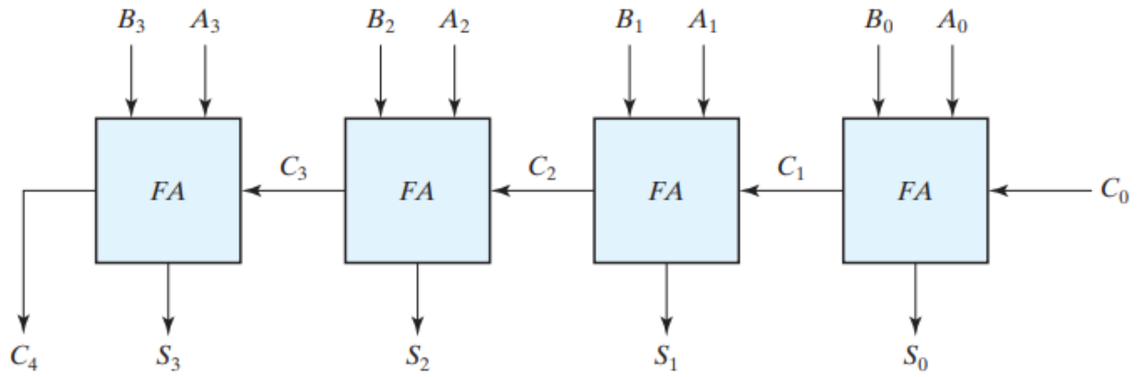
Carry Propagation

- **Parallel Binary Addition:** All bits (augend and addend) are processed simultaneously (see Fig below).
- **Signal Propagation:** Correct sums after signals pass through gates (total time = gate delay × number of gate levels).
- **Carry Propagation Delay:** Longest delay from carry bit passing through all full adders.



Carry Propagation

- **Output Sum Dependence:** Sum bit S_i final only after carry input C_i is stable.
- **Carry Ripple Effect:**
  - Output S_3 accuracy dependent on carry C_3, which awaits C_2 from previous stage (see Fig. below).
  - This dependency chain continues back to carry C_0.
- **Final Output:** S_3 and carry C_4 accurate after full carry ripple (carry propagation through 2_n gate levels for an n-bit adder).



Binary Subtractor

- Subtraction of two numbers, $A - B$, is performed by adding A to the 2's complement of B.

Finding the 2's Complement:

- Obtain the 1's complement of B by inverting all bits.
- Add 1 to the least significant bit (LSB) of the inverted B to get the 2's complement.

Binary Subtractor

Circuit Implementation:

- An adder circuit is used, where each bit of B is inverted before entering the adder.
- The input carry C_0 is set to 1 to account for the +1 needed in the 2's complement process.

Subtraction Operation:

- The circuit performs A plus the 1's complement of B plus 1, which results in A plus the 2's complement of B.

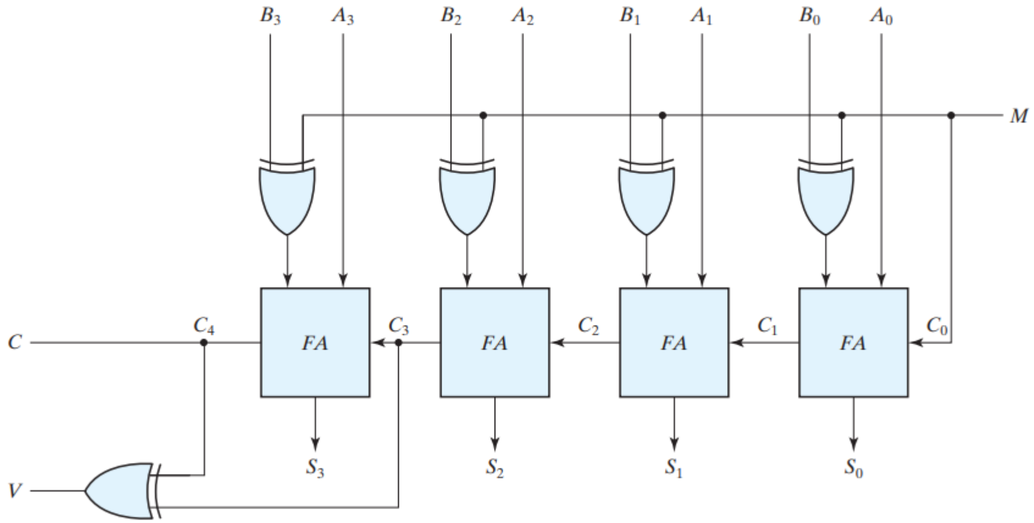
Binary Subtractor

Results Interpretation:

- For unsigned numbers: The result is $A-B$ if A is greater than or equal to B, or the 2's complement of $B-A$ if A is less than B.
- For signed numbers: The result is correct as $A-B$ provided there is no overflow in the operation.

Binary Subtractor

- **Common Circuit with Exclusive-OR Gate:**
  - Addition and subtraction can be integrated into one circuit using a binary adder with an exclusive-OR gate at each full adder.
- Four-Bit Adder-Subtractor Circuit referenced in Figure, the circuit operates based on a mode input *M*.
- **Mode Input *M* Functionality:**
  - If *M*=0: The circuit acts as an adder.
  - If *M*=1: The circuit functions as a subtractor.



Ollscoil na Gaillimhe
University of Galway

Decimal Adder

University
*of*Galway.ie

Introduction

- Computers or calculators that perform arithmetic operations directly in the decimal number system represent decimal numbers in binary coded form.
- An adder for such a computer must employ arithmetic circuits that accept coded decimal numbers and present results in the same code.
- For binary addition, it is sufficient to consider a pair of significant bits together with a previous carry.

Introduction

- A decimal adder, on the other hand, requires a minimum of nine inputs and five outputs.
- This is because four bits are necessary to code each decimal digit, and the circuit must accommodate both an input and an output carry.
- The specific design of a decimal adder circuit can vary depending on the code used to represent the decimal digits.
- Here, we focus on a decimal adder designed for the Binary Coded Decimal (BCD) code.

BCD Adder

- Adding two BCD digits, with each not exceeding 9, plus a possible carry, gives a sum range from 0 to 19. ((9+9+1) Note: 1 is carry)
- Using a four-bit binary adder, the sum is calculated in binary.
- The binary results are shown in the Table.

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

Ollscoil na Gaillimhe
University of Galway

BCD Adder

- The binary result is denoted as K, Z_8, Z_4, Z_2, Z_1, where K is the carry and the rest are weighted bits (8, 4, 2, 1).
- Binary sums up to 9 are already BCD. Sums 10 and above need to be converted.
- Conversion to BCD requires a rule to adjust binary sums over 9 to the proper BCD format.

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

BCD Adder

- If the binary sum is 1001 (9 in decimal) or less, it matches the BCD number directly; no change is needed.
- If the binary sum exceeds 1001, it's not a valid BCD number.
- Adding 0110 (6 in decimal) to this binary sum corrects it to a valid BCD number and generates the necessary carry.

BCD Adder

- The logic circuit required for correction after adding BCD numbers is based on certain binary sum conditions:
  - Correction is needed when there is an output carry K=1.
  - For sums between 1010 to 1111, correction is also needed. These have a 1 in the Z_8 position.
  - To differentiate from the binary sums of 1000 and 1001, which also have a 1 in Z_8, the sums requiring correction must also have a 1 in either Z_4 or Z_2.

- The condition for adding a correction and generating an output carry is encapsulated in the Boolean function:

$$C = K + Z_8 Z_4 + Z_8 Z_2$$

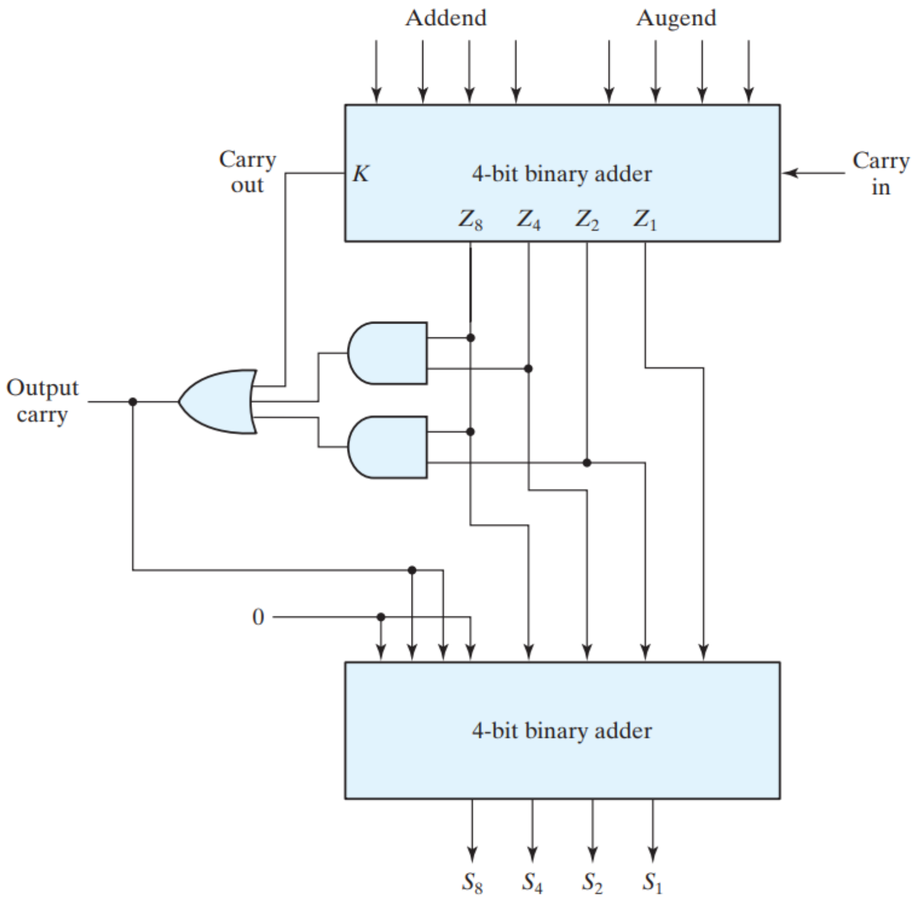
  - When C=1, the circuit should add 0110 to the binary sum and produce an output carry for the subsequent stage.

BCD Adder

To sum two BCD digits and obtain a BCD result, the process is as follows, referring to a specific diagram in a text (Figure in right):

1. The two BCD digits, along with any carry from a previous addition, are input to the initial four-bit binary adder.

2. This adder computes the binary sum. If there's no carry out (output carry = 0), the binary sum is already in the correct BCD format.

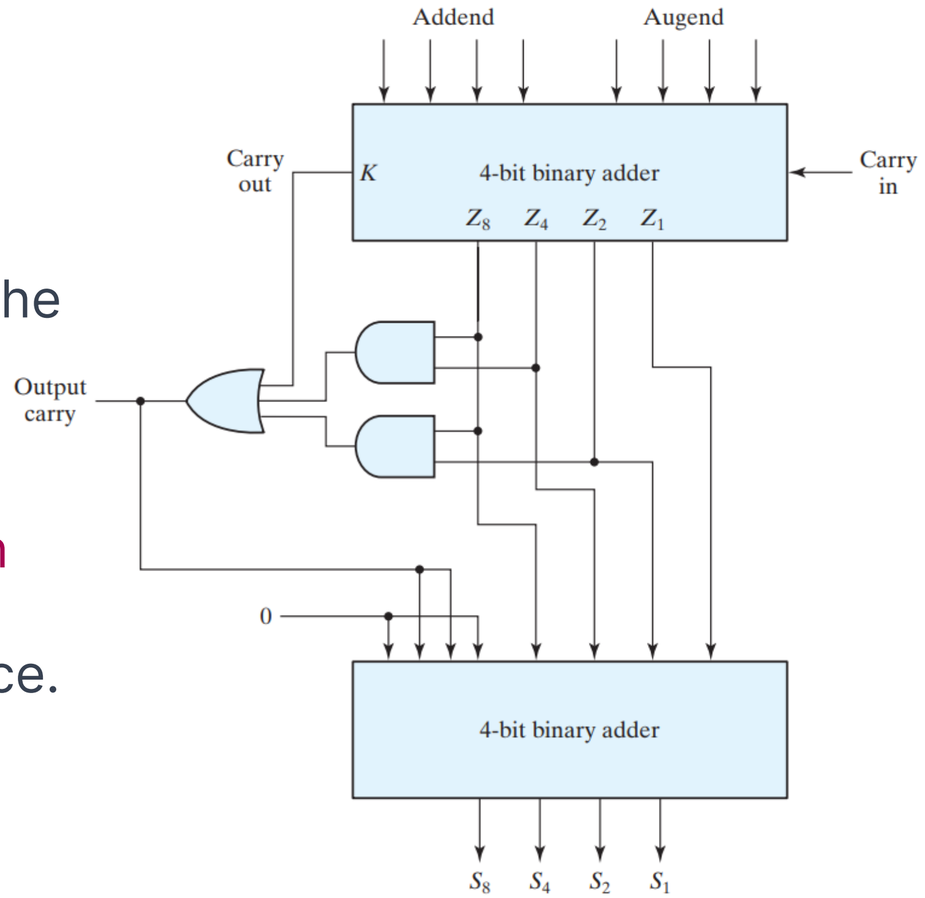


BCD Adder

3. If there is a carry out (output carry = 1), indicating an overflow, **a correction is needed**. The number 0110 is added to the sum via a second four-bit adder to convert the binary sum back to the correct BCD format.

4. The output carry from this second adder is not used, as the required carry information for the subsequent addition stage is already present at the carry output.

5. For adding multiple BCD digits (n digits), you would use n BCD adder stages, with the carry out from each stage connected to the carry input of the next stage in sequence.



Ollscoil na Gaillimhe
University of Galway

Binary Multiplier

University *of* Galway.ie

Binary Multiplier

- In binary multiplication, each bit of the multiplier is used to create a partial product with the multiplicand, starting with the least significant bit.
- These partial products are then shifted left appropriately and summed to form the final product.
- This process is analogous to the method used in decimal multiplication.

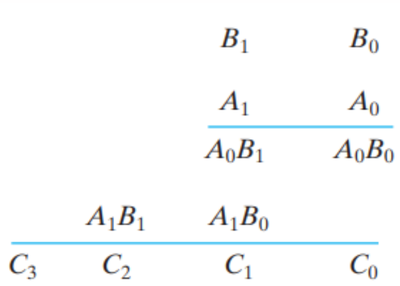
Binary Multiplier

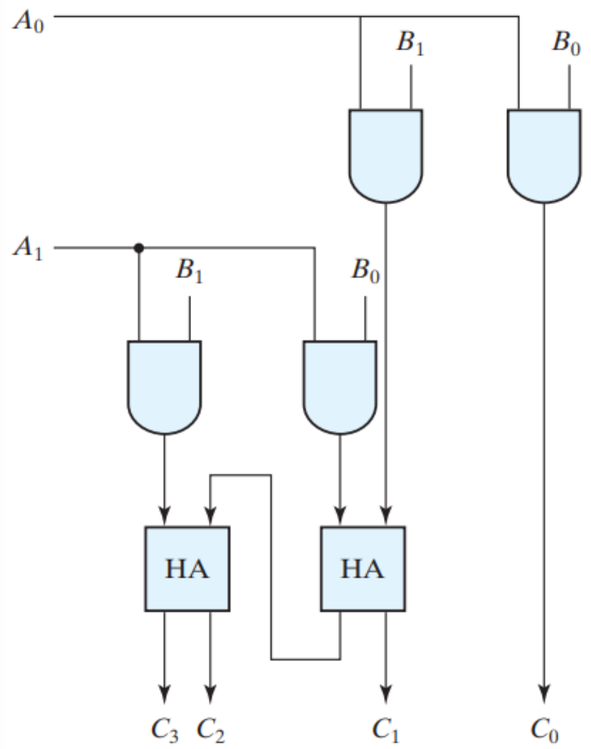
To create a binary multiplier circuit for two 2-bit numbers: (Refer Figure Below)

1. Label the multiplicand bits as **B_1 and B_0** and the multiplier bits as **A_1 and A_0**. The four bits of the product will be **$C_3C_2C_1C_0$**.

2. Generate the first partial product by multiplying the multiplicand **B_1B_0** with the least significant bit of the multiplier, **A_0**. Multiplication of binary bits is like the AND operation, yielding 1 only when both bits are 1.

3. Implement this AND operation using AND gates for each bit multiplication.



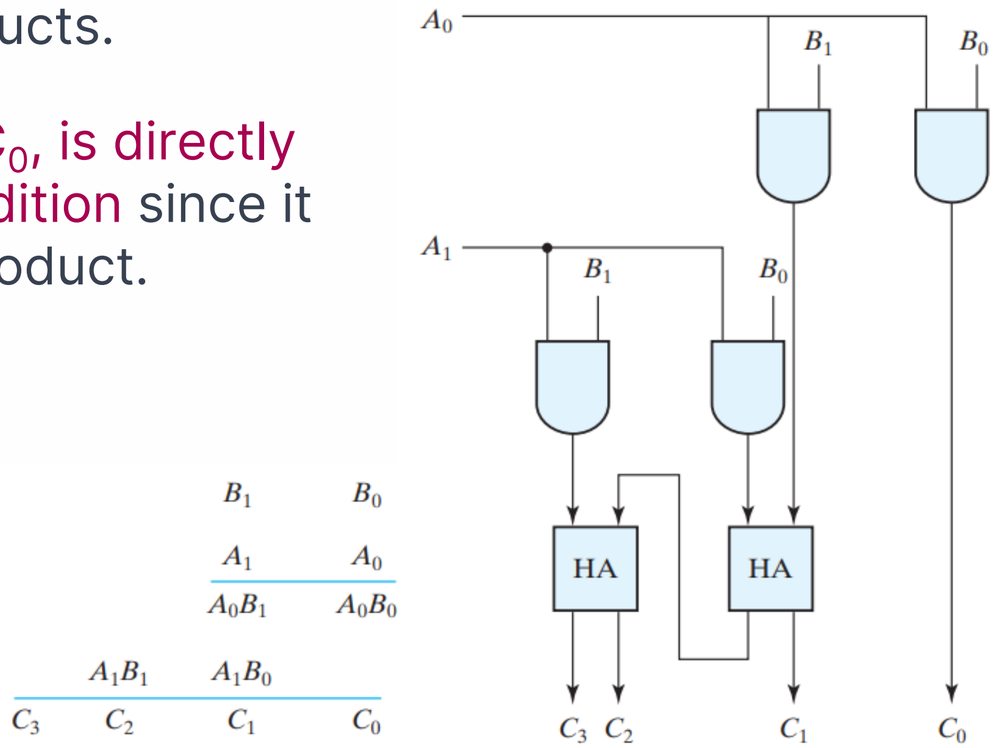


Binary Multiplier

4. For the second partial product, multiply the multiplicand $\mathbf{B_1B_0}$ with the next bit of the multiplier, A1, and shift the result one bit to the left.

5. Add the two partial products using two half-adder circuits. If there were more bits, full adders would be required for summing the partial products.

6. Note that the least significant bit of the final product, C_0, is directly obtained from the first AND gate without additional addition since it represents the least significant bit of the first partial product.



$$\begin{array}{cccc} & & B_1 & B_0 \\ & & A_1 & A_0 \\ \hline & & A_0B_1 & A_0B_0 \\ & A_1B_1 & A_1B_0 & \\ \hline C_3 & C_2 & C_1 & C_0 \end{array}$$

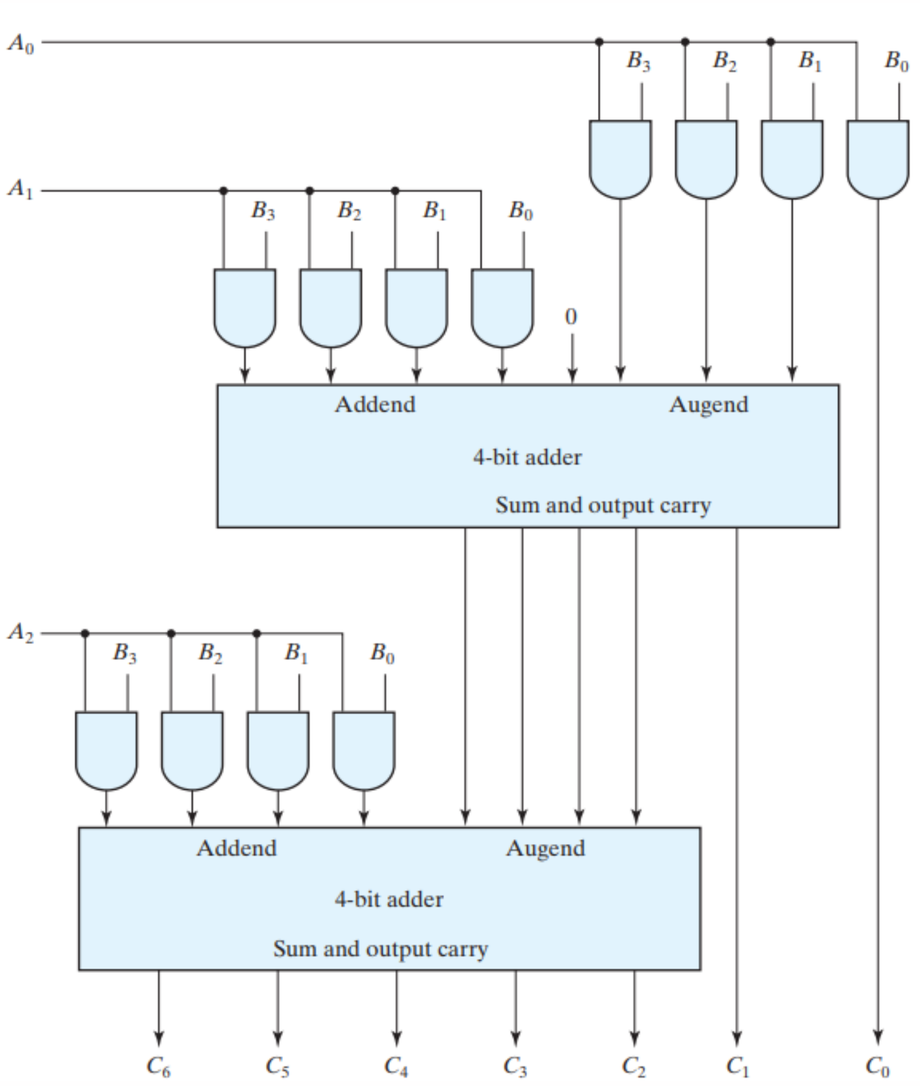
Binary Multiplier

- **AND Gates for Partial Products**:
  - Each bit of the multiplier is ANDed with each bit of the multiplicand.
  - For a multiplier with J bits and a multiplicand with K bits, **J*K** AND gates are needed.
  - This creates a series of partial products.
- **Adders for Summation**:
  - The binary outputs of the AND gates are summed to form new partial products.
  - For J bits in the multiplier, you need J-1 adders, each capable of handling K-bit sums.

Binary Multiplier

- **Overall Product**:
    - After adding up all the partial products, the final binary product is **J+K** bits long.
- **Example Case**:
    - Multiplying a 4-bit number ($B_3B_2B_1B_0$) by a 3-bit number ($A_2A_1A_0$) requires:
        - 12 AND gates (since 4*3 = 12).
        - Two 4-bit adders to sum the partial products.
        - The final product will be 7 bits long (since 4+3 = 7).

Logic Diagram



Ollscoil na Gaillimhe
University of Galway

Magnitude Comparator

University *of* Galway.ie

Magnitude Comparator

- The comparison of two numbers is an operation that determines whether one number is greater than, less than, or equal to the other number.
- A **magnitude comparator** is a combinational circuit that compares two numbers A and B and determines their relative magnitudes.
- The outcome of the comparison is specified by three binary variables that indicate whether **A > B, A = B, or A < B**.

Magnitude Comparator

- On the one hand, the circuit for comparing two n-bit numbers has 2^{2n} entries in the truth table and becomes too cumbersome, even with n = 3.
- On the other hand, as one may suspect, a comparator circuit possesses a certain amount of regularity.
- Digital functions that possess an inherent well-defined regularity can usually be designed by means of an algorithm—a procedure that specifies a finite set of steps that, if followed, give the solution to a problem.
- We illustrate this method here by deriving an algorithm for the design of a four-bit magnitude comparator.

Magnitude Comparator

- The algorithm is a direct application of the procedure a person uses to compare the relative magnitudes of two numbers.
- Consider two numbers, A and B , with four digits each.
- Write the coefficients of the numbers in descending order of significance:

Magnitude Comparator

- The algorithm used here is based on how a person compares the relative magnitudes of two numbers.

- Let's consider two numbers, A and B, each with four digits, and we'll write their coefficients in descending order of significance:

$$A = A_3\ A_2\ A_1\ A_0$$
$$B = B_3\ B_2\ B_1\ B_0$$

  - Each subscripted letter represents one of the digits in the number.

- The two numbers are considered equal if all pairs of significant digits are equal:

$$A_3 = B_3$$
$$A_2 = B_2$$
$$A_1 = B_1$$
$$A_0 = B_0$$

Magnitude Comparator

In the case of binary numbers, where the digits are either 1 or 0, the equality of each pair of bits can be expressed logically using an exclusive-NOR function:

- **$x_i = A_iB_i + A'_iB'_i$ for i = 0, 1, 2, 3**
- Here, x_i equals 1 only if the pair of bits in position i are equal (i.e., both are 1 or both are 0).

- To indicate the equality of two numbers, A and B, in a combinational circuit, an output binary variable is used, designated as (A = B).

- This binary variable, (A = B), equals 1 if the input numbers, A and B, are equal and equals 0 otherwise.

Ollscoil na Gaillimhe
University of Galway

Magnitude Comparator

- For equality to hold, all x_i variables (representing the comparison of pairs of digits) must be equal to 1.
- This condition requires an AND operation of all variables:

$$\mathbf{(A = B) = x_3 x_2 x_1 x_0}$$

- The binary variable (A = B) is only equal to 1 when all pairs of digits in the two numbers are equal.

Magnitude Comparator

- To ascertain whether A is greater or less than B, we examine the relative magnitudes of pairs of significant digits, beginning from the most significant position.
- The comparison starts with the most significant pair of digits, and if these two digits are equal, we move on to the next lower significant pair of digits.
- This comparison process continues until a pair of unequal digits is encountered.

Magnitude Comparator

- When we find an unequal pair, we make the following determinations:
  - If the corresponding digit of A is 1 and that of B is 0, we conclude that A > B (A is greater than B).
  - If the corresponding digit of A is 0 and that of B is 1, we conclude that A < B (A is less than B).
- The sequential comparison can be expressed logically by the two Boolean functions

$$(A > B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$$
$$(A < B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1' + x_3x_2x_1A'n_0B_0'$$

Ollscoil na Gaillimhe
University of Galway

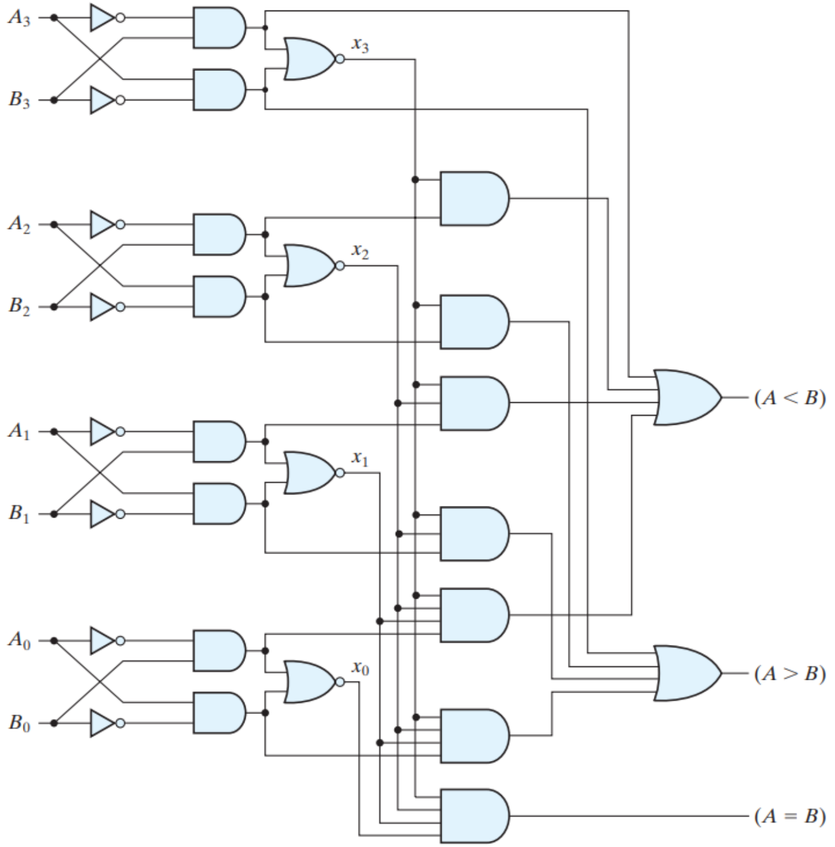
Note: The symbols (A > B) and (A < B) are binary output variables that are equal to 1 when A > B and A < B, respectively.

Magnitude Comparator

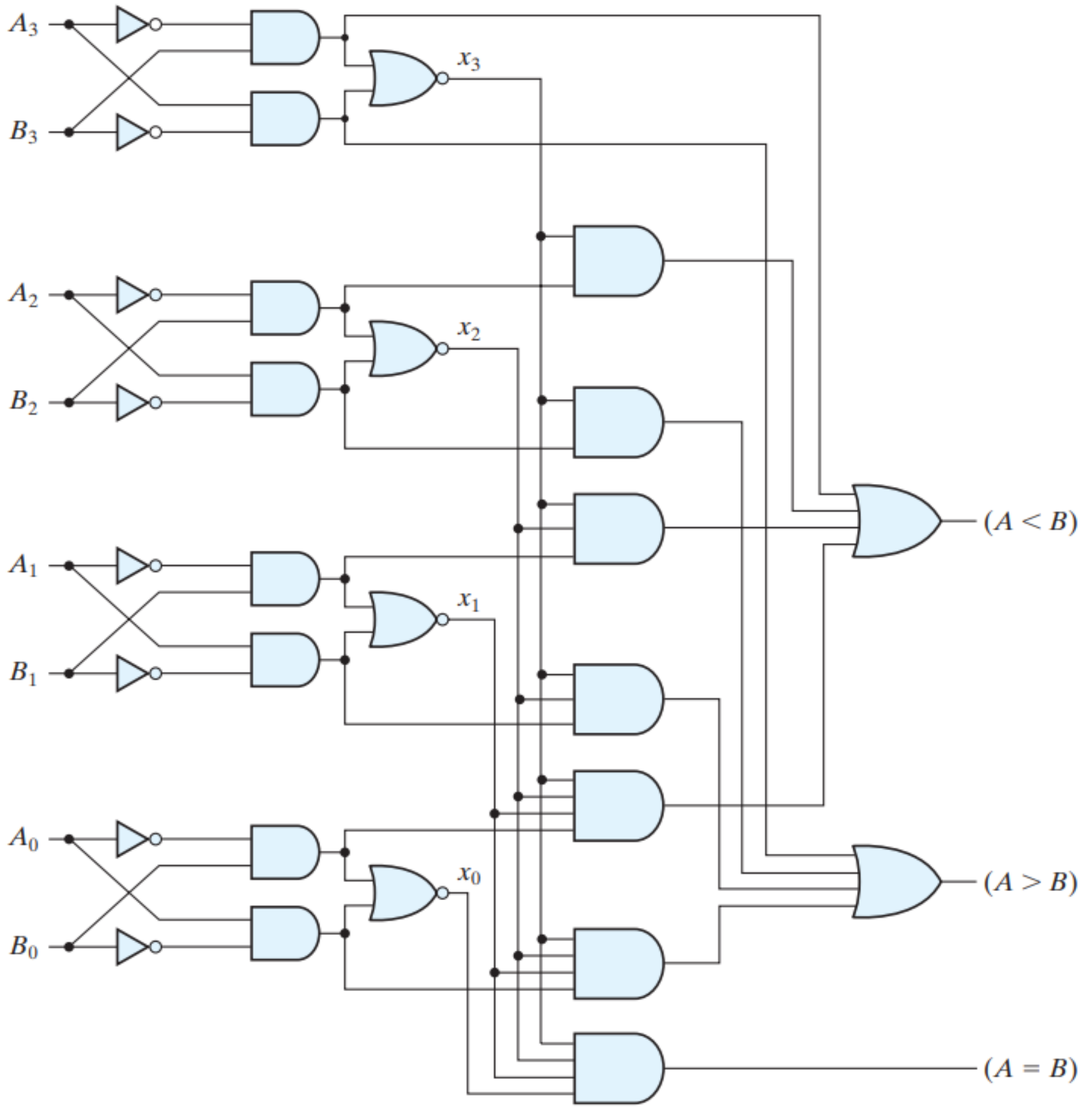
- The gate implementation of the three output variables for comparing the magnitude of two binary numbers is simpler than it may initially seem.

- The unequal outputs can utilize the same gates needed to generate the equal output, resulting in a more efficient design.

  - The logic diagram of a four-bit magnitude comparator is illustrated in Figure.



Magnitude Comparator



Magnitude Comparator

- The four x outputs are produced using exclusive-NOR (XNOR) circuits and are then fed into an AND gate to generate the output binary variable (A = B), indicating equality.
- The other two outputs are derived using the x variables to create the previously listed Boolean functions.
- This implementation follows a multilevel approach and exhibits a regular pattern, making it easier to extend to magnitude comparator circuits for binary numbers with more than four bits.

References

- Computer Organization and Architecture Designing for Performance Tenth Edition by William Stallings
- Digital Design With an Introduction to the Verilog HDL FIFTH EDITION by M Morris, M. and Michael, D., 2013.

Ollscoil na Gaillimhe
University of Galway

Thank *you*

University *of* Galway.ie