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

Binary Adder-Subtractor

University
*of*Galway.ie

Introduction

Basic Arithmetic in Digital Computers:

- Computers perform various arithmetic operations; the most fundamental is binary addition.

Binary Addition Operations:

- **Four basic calculations:**
    - 0 + 0 = 0
    - 0 + 1 = 1
    - 1 + 0 = 1
    - 1 + 1 = 10 (produces a 'carry' bit)

Introduction

Carry Bit:

- In binary addition, when both digits are 1, the result is a two-digit number (10), with the '1' becoming the carry bit for the next significant digit pair.

Half Adder:

- A combinational circuit that adds two single binary digits.
- Does not account for carries from previous digits.

Introduction

Full Adder:

- Handles the addition of three bits: two significant bits and one carry bit from the addition of the previous lower significant bits.
- Two half adders can be combined to create a full adder.

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

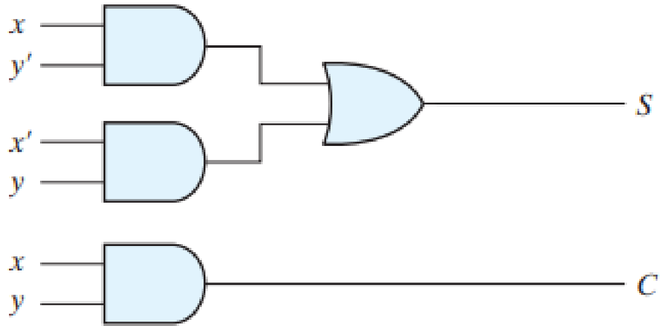
Half Adder

Boolean Functions:

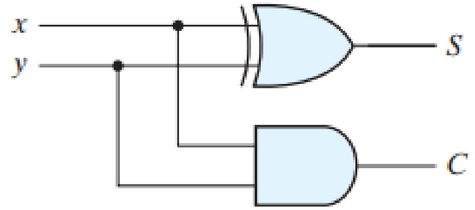
- Sum **(S)=x'y+xy'** (using XOR logic)
- Carry **(C)=xy** (using AND logic)

Logic Diagrams:

- Two representations:
  - Sum of products with AND, OR, and NOT gates (Fig. (a)).
  - With an exclusive-OR (XOR) for the sum and an AND gate for the carry (Fig. (b)).
- Illustrates how two half-adders can be integrated to form a full adder.



(a) $S = xy' + x'y$
$C = xy$



(b) $S = x \oplus y$
$C = xy$

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

Full Adder

Outputs Interpretation:

- In terms of binary addition and Boolean function variables.
- Outputs *S* and *C* have different interpretations depending on context (binary addition vs. Boolean logic).

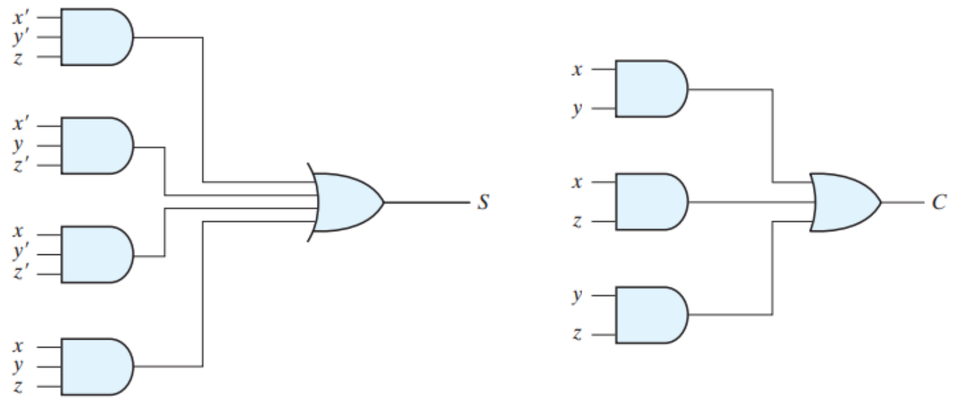
Boolean Expressions:

- Sum (*S*): $S = x'y'z + x'yz' + xy'z' + xyz$
- Carry (*C*): $C = xy + xz + yz$

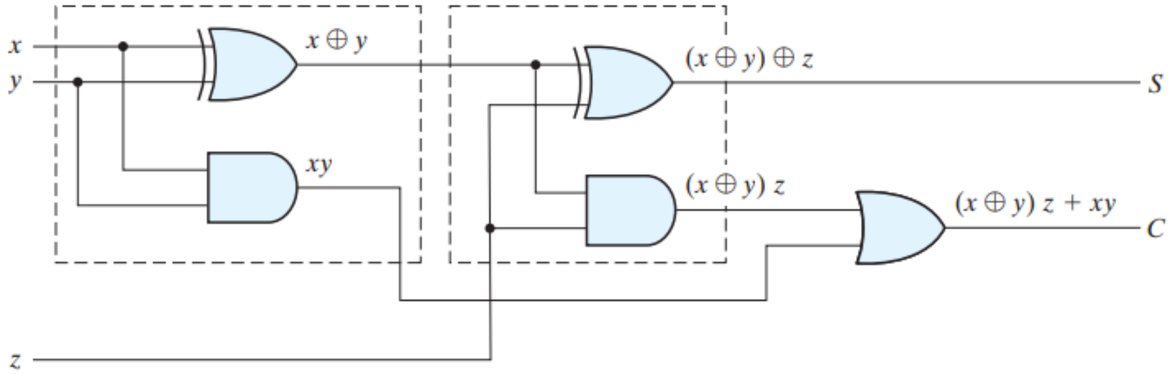
Ollscoil na Gaillimhe
University of Galway

Full Adder

- **Implementation Diagrams:**
  - Sum-of-products with logic gates (Fig. 4.7).



  - Using two half adders and an OR gate (Fig. 4.8).



Full Adder

Exclusive-OR for Output *S*:

- *S* is the XOR of *z* and the sum output of the first half adder.
- Expressed as

$$
\begin{aligned}
S &= z \oplus (x \oplus y) \\
&= z'(xy' + x'y) + z(xy' + x'y)' \\
&= z'(xy' + x'y) + z(xy + x'y') \\
&= xy'z' + x'yz' + xyz + x'y'z
\end{aligned}
$$

Carry Output (*C*):

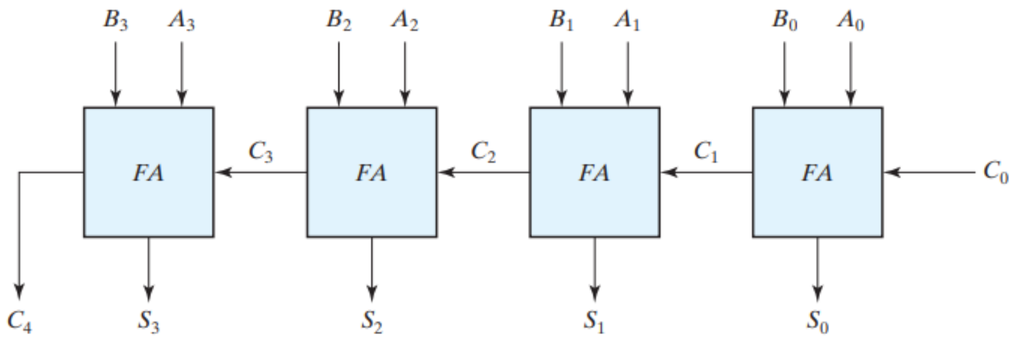
- *C* is the OR of the carry outputs from both half adders and the AND of *x* and *y*.
- Simplified to **C = z(xy' + x'y) + xy**
  = xy'z + x'yz + xy

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}

Binary Adder

Carry Operation:

- **Initial Carry:** The carry-in for the LSB (C_0) is set to 0.
- **Sequential Carry:** The carry-out from each full adder (C_i) becomes the carry-in for the next significant bit's full adder (C_{i+1}).
- **Propagation:** Carry values ripple through from right to left, which is why it's called a ripple carry adder.

Sum Generation:

- The sum bits (S) are produced in sequence from the rightmost to the leftmost position.
- The correctness of each sum bit relies on the carry generated from the addition of the previous bits.

Binary Adder

Efficiency:

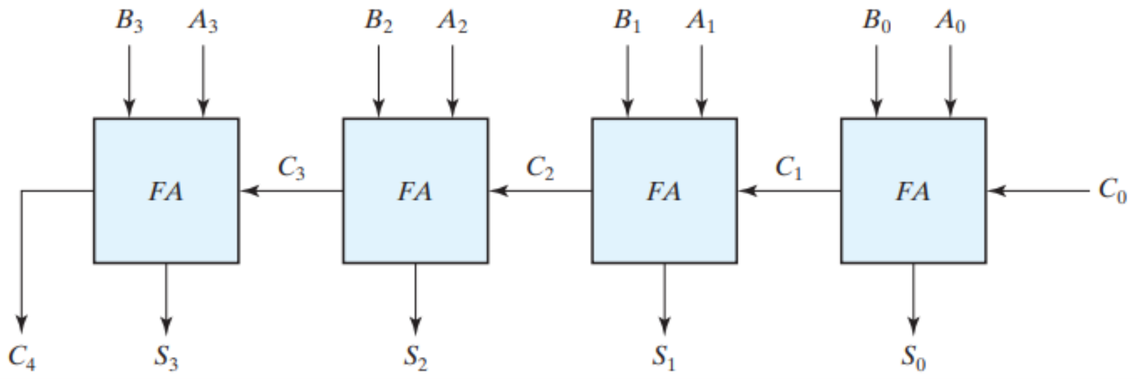
- **Iterative Design:** Using a standard full adder repeatedly avoids the complexity of designing a new circuit for each application.
- **Simplifies Implementation:** Cascading standard full adders significantly reduces design complexity compared to creating a truth table for all input combinations, which would be impractical for large bit sizes (e.g., a truth table with 512 entries for a 4-bit adder).

Applications:

- The 4-bit adder can be employed in various arithmetic operations and is a staple in digital system design.
- Its standard component nature makes it a building block in more complex arithmetic circuits.

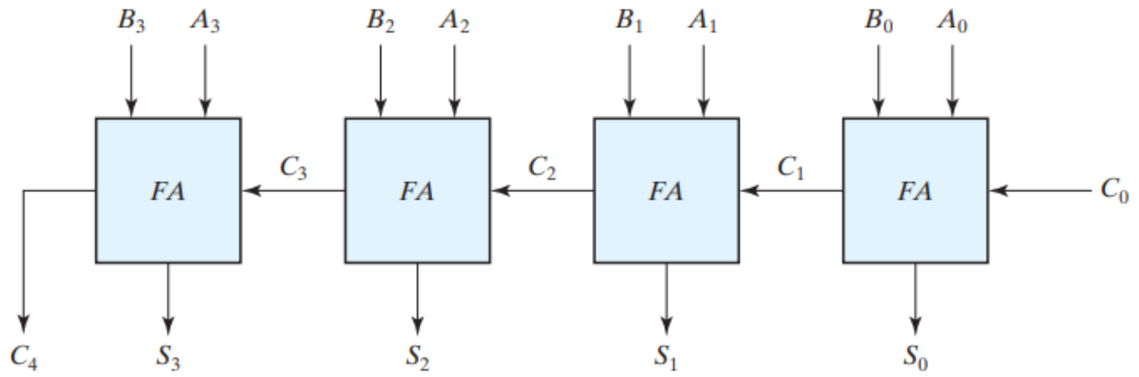
Carry Propagation

- **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.



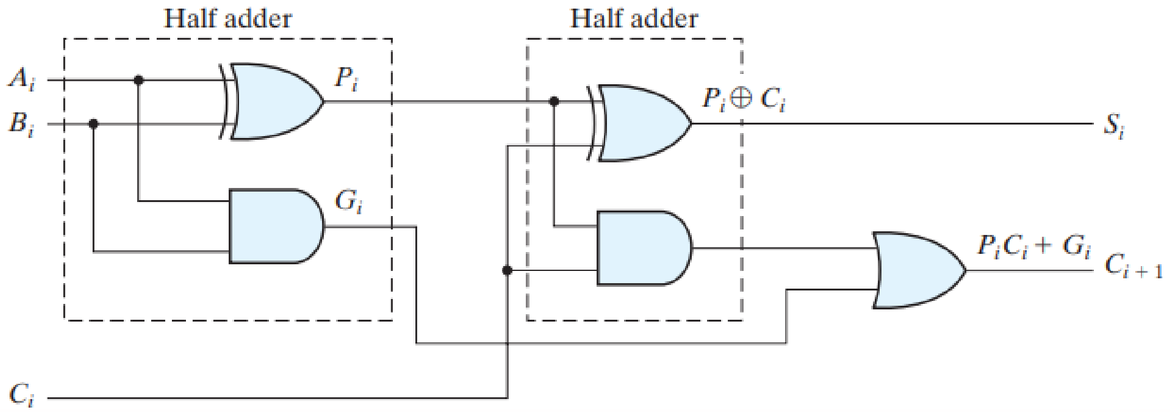
Carry Propagation

- **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).



Carry Propagation

- Circuit in Figure below shows carry signal progression through gate levels.
- Each stage, labeled with subscript *i*, adds two gate levels (AND and OR gates) for carry progression.
- P_i and G_i signals stabilize after passing through their respective gates.
- In a 4-bit adder, C_4 results after 8 gate levels (2 gate levels per adder stage).
- General formula: For an *n*-bit adder, carry signal propagates through 2_n gate levels from C_0 to C_n.



Carry Propagation

Speed Limitations Due to Carry Propagation:

- Carry propagation time is crucial for adder speed—directly impacts how fast two numbers can be added.
- Correct outputs require enough time for signal propagation through the gates (from inputs to outputs).
- This is vital since all arithmetic operations build on addition; thus, addition speed is a bottleneck.

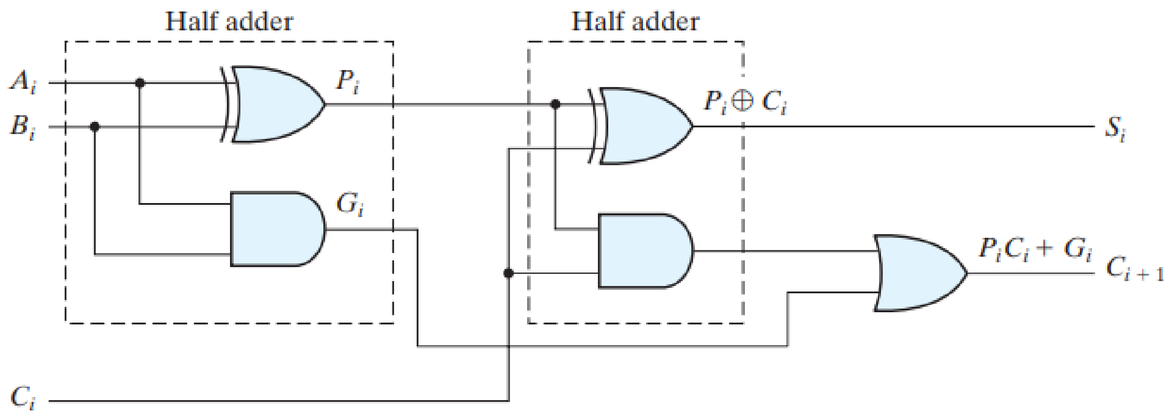
Carry Propagation

- **Solutions to reduce delay include:**
    - Using faster gates with lower delay times (though there's a physical limit to gate speed).
    - Increasing circuit complexity to decrease carry delay time.
- The carry lookahead logic is a prevalent method for minimizing carry delay in parallel adders.

Carry Propagation

- Consider the circuit of the full adder shown in Figure below. If we define two new binary variables

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$



Carry Propagation

- The output sum and carry can respectively be expressed as

$$\mathbf{Si = P_i \oplus C_i}$$
$$\mathbf{C_{i+1} = G_i + P_iC_i}$$

  - G_i (carry generate) produces a carry of 1 when both A_i and B_i are 1, regardless of the input carry C_i.
  - P_i (carry propagate) determines whether a carry into stage i will propagate into stage i + 1 (i.e., whether an assertion of C_i will propagate to an assertion of C_{i+1}).

Carry Propagation

- We now write the Boolean functions for the carry outputs of each stage and substitute the value of each C_i from the previous equations:

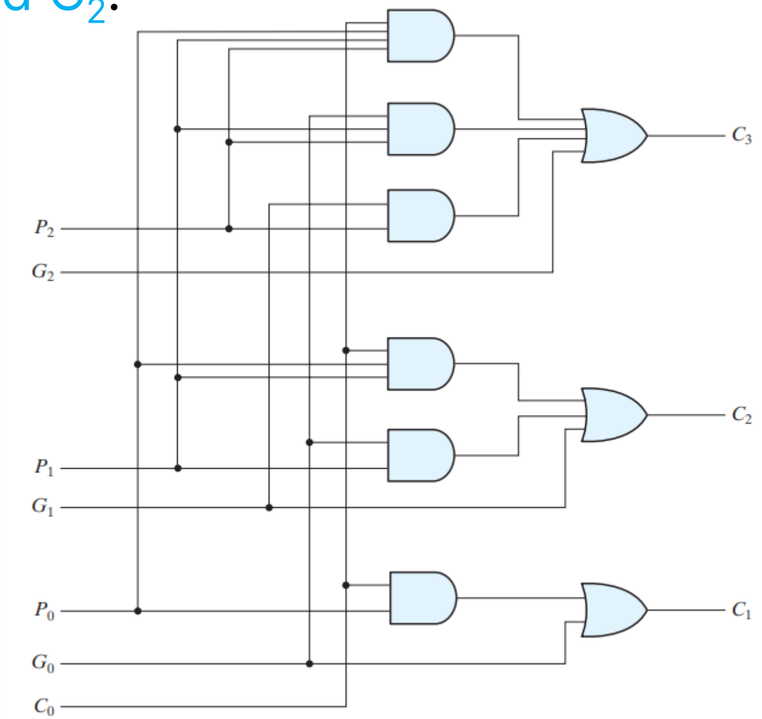
$$C_0 = \text{input carry}$$
$$C_1 = G_0 + P_0C_0$$

$$C_2 = G_1 + P_1C_1 = G_1 + P_1 (G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$$
$$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 = P_2P_1P_0C_0$$

- The Boolean function for each output carry is represented in sum-of-products form.
- Each function can be implemented with one level of AND gates followed by an OR gate, or alternatively, a two-level NAND gate.

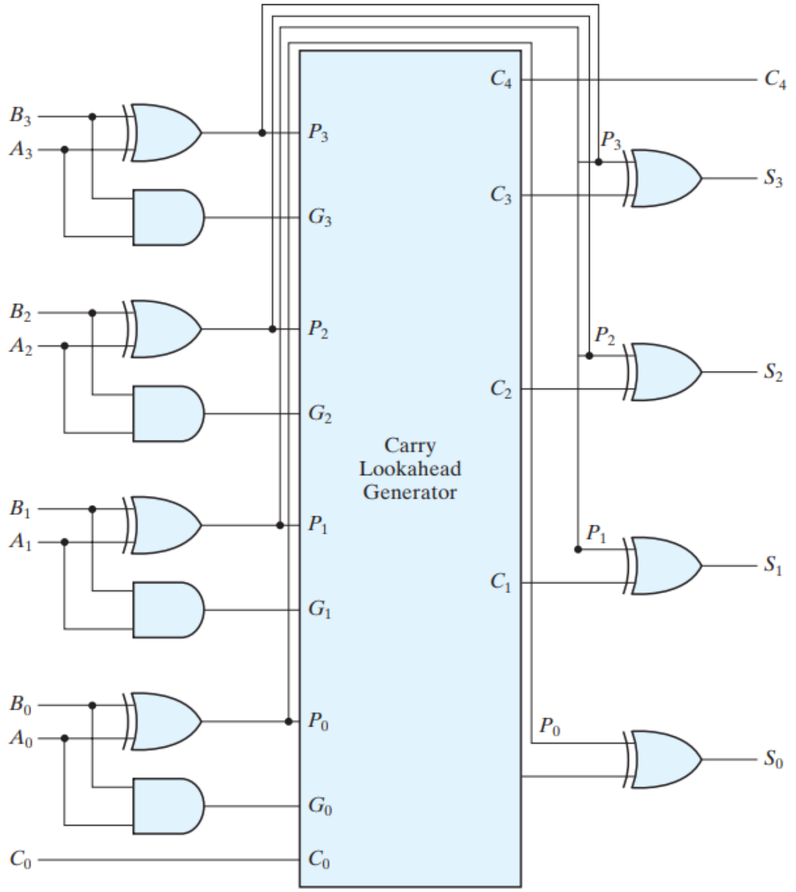
Carry Propagation

- The carry lookahead generator, depicted in Fig. 4.11, is used to implement the three Boolean functions for C_1, C_2, and C_3.
- Notably, this circuit speeds up addition as C_3 no longer needs to wait for C_2 and C_1 to propagate. Instead, C_3 is propagated simultaneously with C_1 and C_2.
- This improved speed comes at the cost of increased complexity in hardware.



Carry Propagation

- The construction of a four-bit adder with a carry-lookahead scheme is depicted in Figure.
- For each sum output, two exclusive-OR gates are used.
- The first exclusive-OR gate generates the P_i variable, while the AND gate produces the Gi variable.
- Carries are propagated through a carry-lookahead generator, similar to the one shown in Figure in the previous slide.



Carry Propagation

- These generated carries serve as inputs to the second exclusive-OR gate.
- All output carries experience a delay through two levels of gates, resulting in equal propagation delay times for outputs S1 through S3.
- While not shown, the two-level circuit for output carry C4 can be derived using the equation-substitution method.

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.

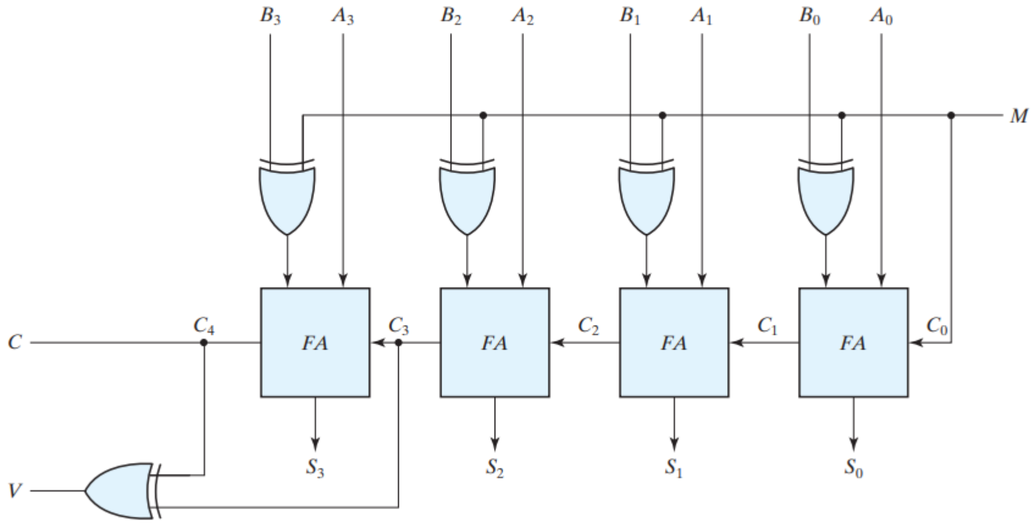
Binary Subtractor

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.



Binary Subtractor

Exclusive-OR Gate Role:

- Each exclusive-OR gate takes in the mode input M and one of the B inputs.
- When **M=0: B ⊕ 0=B**. The full adders get the B value with input carry as 0. The operation becomes **A+B**.
- When **M=1: B⊕1=B′** (complement of B) and **C_0=1**. All B inputs are inverted, and 1 is added via the input carry. This results in the operation A plus the 2's complement of B.

Binary Subtractor

- **Overflow Detection:**
  An exclusive-OR gate with an output V is included to identify overflow situations.

- **Signed-Complement System and Binary Arithmetic:**

  - Binary numbers in the signed-complement system use the same basic rules for addition and subtraction as unsigned numbers.
  - Computers need only one unified hardware circuit to perform both types of arithmetic.
  - The interpretation of results varies based on the assumed number type (signed or unsigned).

Overflow

- **Definition of Overflow:**
  - Overflow happens when adding two numbers with n digits each results in a sum occupying n+1 digits.
  - This applies to both binary and decimal numbers, whether they're signed or unsigned.

- **Paper vs. Digital Overflow:**

  - While doing arithmetic manually (e.g., using paper and pencil), overflow isn't a concern because there's no fixed width constraint like in digital systems.
  - In contrast, digital computers have a fixed bit-width to store numbers, making overflow a significant issue.

Overflow

- **Overflow Detection in Computers:**
  - Computers can detect overflow, typically setting a specific flip-flop (or flag) to signal it.
  - Users or programs can then check this flag to ascertain if an overflow has occurred during computation.

- **Overflow in Unsigned vs. Signed Numbers:**
  - **Unsigned Numbers:** Overflow is detected from the end carry-out of the most significant position.
  - **Signed Numbers:**
    - The leftmost bit represents the sign, and negative numbers are typically in 2's-complement form.
    - The end carry doesn't indicate an overflow for signed numbers because the leftmost bit is part of the number itself.

Overflow

Overflow Scenarios with Signed Numbers:

- **Different Signs:**
  - Overflow can't happen if one number is positive and the other is negative.
  - The result's magnitude will always be less than the magnitude of the larger original number.
- **Same Signs:**
  - Overflow can occur if both numbers being added are positive or both are negative.
  - For instance, adding two signed numbers, +70 and +80, stored in two eight-bit registers can lead to an overflow because the sum (+150) exceeds the register's capacity, which ranges from +127 to -128.

Overflow

The two additions in binary are shown next, together with the last two carries:

carries:	0 1	carries:	1 0
+70	0 1000110	−70	1 0111010
+80	0 1010000	−80	1 0110000
+150	1 0010110	−150	0 1101010

Note that

- the eight-bit result that should have been positive has a negative sign bit (i.e., the eighth bit) and the eight-bit result that should have been negative has a positive sign bit.
- If, however, the carry out of the sign bit position is taken as the sign bit of the result, then the nine-bit answer so obtained will be correct.
- But since the answer cannot be accommodated within eight bits, we say that an overflow has occurred.

Overflow

- The presence of overflow in binary addition of signed numbers can be determined by comparing the carry into the sign bit position with the carry out of the sign bit position.

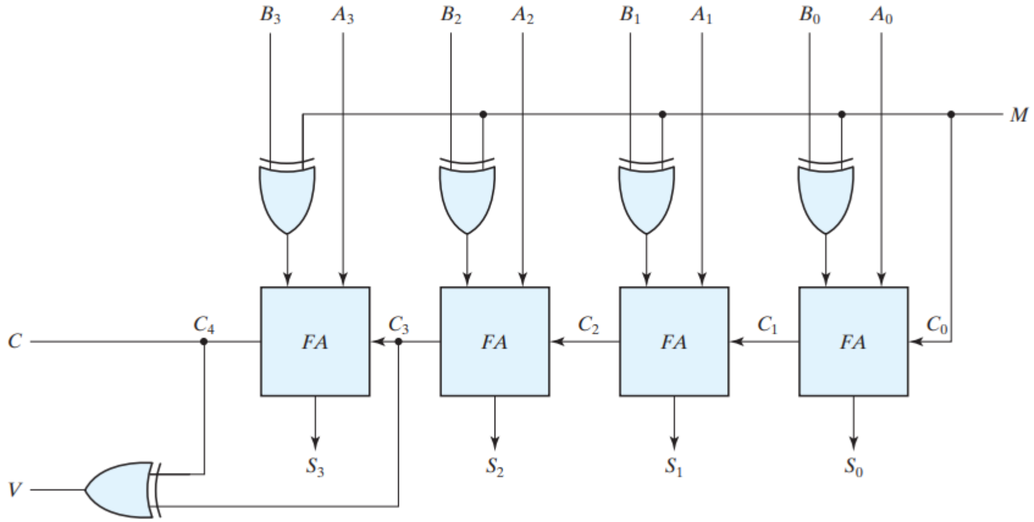
- If they are different, it indicates an overflow condition. Conversely, if they are the same, no overflow occurred.

- **Exclusive-OR Gate for Detection**:
  - By using an exclusive-OR (XOR) gate on the carry into and carry out of the sign bit, overflow can be easily detected.
  - If the XOR gate outputs a '1', it's a clear indication of overflow. But if it outputs a '0', no overflow has taken place.
  - This method is reliable only when 2’s-complement notation is used for representing negative numbers, ensuring proper calculation of negative number complements

Overflow

Binary Adder–Subtractor Circuit:

- The reference to Figure likely depicts a circuit diagram of a binary adder-subtractor, showcasing its key outputs: C (Carry) and V (Overflow).
- The 'C' bit is primarily used for detecting carry (in addition) or borrow (in subtraction) for unsigned numbers.
- The 'V' bit is dedicated to detecting overflow conditions when dealing with signed numbers.
- If after a computation, V = 0, it means the result is valid without any overflow. However, if V = 1, it indicates an overflow condition.



Overflow

Result Implication:

- When V = 1 after an operation, it means that the resultant value needs $n+1$ bits for accurate representation.

- However, given that only n bits are available for storage, the extra bit (which in the case of signed numbers represents the actual sign) is effectively shifted out, causing an overflow.

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