Lecture 7: Arithmetic Circuits

CS207: Digital Logic

Jialin Liu

Department of Computer Science and Engineering (CSE)
Southern University of Science and Technology (SUSTech)

21 October 2022

Acknowledgement

These slides were prepared based on the slides by Dr. Jianqiao Yu and the ones by Prof. Georgios Theodoropoulos of the Department of CSE at the SUSTech, as well as the contents of the following book:
M. M. Mano and M. Ciletti, *Digital design: with an introduction to the Verilog HDL*. Pearson, 2013

Recap: Types of Logic Circuits

- "A **combinational circuit** consists of logic gates whose outputs **at any time** are determined from only the present combination of inputs."
  → **Combinational logic** (Lecture 5 and Lecture 6).
- "**Sequential circuits** employ memory elements in addition to logic gates. Their outputs are a function of the inputs and the state of the memory elements."
  - State of the memory elements: a function of previous inputs.
  - Outputs of a sequential circuit depend on values of present inputs and past inputs.

  → **Memory elements** (Lecture 7).
  → **Sequential logic** (from Lecture 8).

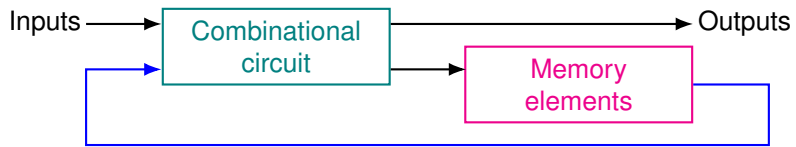


Figure: Block diagram of sequential circuit.

Recap: Combinational Circuit II

- Combinational circuit: consists of logic gates whose outputs at any time are determined from only the present combination of inputs.
  - Procedure of analysing combinational circuits.
  - Procedure of designing combinational circuits.
- Standard components: combinational circuits that are employed extensively in the design of digital systems.
  - Available in integrated circuits as medium-scale integration (MSI) circuits.
  - Also used as standard cells in complex very large-scale integrated (VLSI) circuits such as application-specific integrated circuits (ASICs).
  - Encoders, decoders, demultiplexers, multiplexers, comparators.

Recap: Encoder (编码器)

- An **encoder** performs the inverse operation to that of a decoder.
- Example: Octal-to-binary encoder.
  - Eight inputs and three outputs connected with `OR`.
  - Only one input can be active for one time.

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Recap: Decoder (译码器)

- Digital systems
  - Discrete quantities of information are represented by binary codes.
  - A binary code of n bits → up to 2^n distinct elements of coded information.
- Decoder
  - A **decoder** is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines.
  - The selected output is identified either by a 1, when all other outputs are 0, or by a 0 when all other outputs are 1.
  - The basic function of an MSI decoder having n inputs is to select 1-out-of-2^n output lines.

Recap: Main Applications of Decoders

- Minterm generator (函数最小项发生器): Generate the 2^n (or fewer) minterms of n input variables.
- Demultiplexer (数据分配器): A decoder with enable input can function as a **demultiplexer** – a circuit that receives information from a single line and directs it to one of 2^n possible output lines.
- Address decoder (地址解码器): Identify a memory cell, disk sector, or other memory or storage device, to ensure one device can communicate with the processor at one time.

Recap: Decoder as Demultiplexer (数据分配器)

- A decoder with enable input can function as a **demultiplexer** – a circuit that receives information from a single line and directs it to one of 2^n possible output lines.
- Because decoder and demultiplexer operations are obtained from the same circuit, a decoder with an enable input is referred to as a **decoder-demultiplexer**.

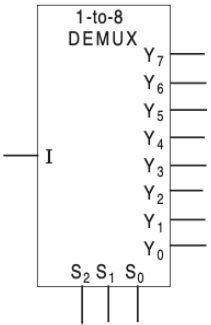
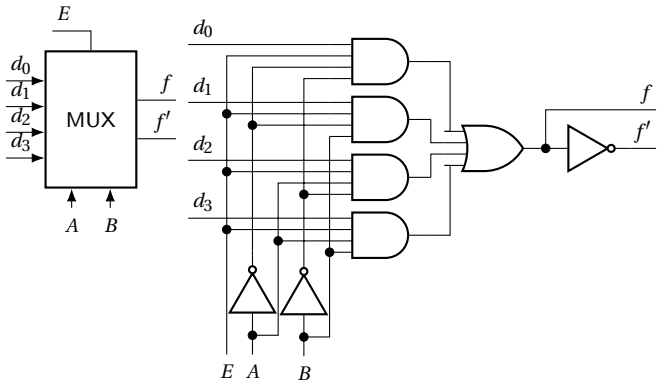


Figure: Figure 5.95 in [2].

Recap: Multiplexer

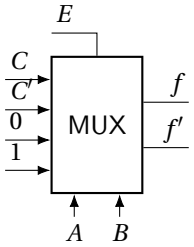
- A **multiplexer (MUX)** (数据选择器/多路复用器) is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.
- It selects 1-out-of-n lines where n is usually `2`, `4`, `8`, or `16`.
- Example: A block diagram of a multiplexer having 4 input data lines `d0`, `d1`, `d2`, and `d3` and complementary outputs `f` and `f'`.



Recap: Multiplexer as a Boolean Function Generator

- For a 4-to-1 MUX the characteristic equation is $f = A'B'd_0 + A'Bd_1 + AB'd_2 + ABd_3$.
  - A and B are Boolean variables, applied at the select inputs, which can be factored out of any Boolean function of n variables.
  - The remaining $n-2$ variables, referred to as the **residue variables**, can be formed into residue functions which can then be applied at the data inputs.
- Example: $f(A,B,C) = \sum(1,2,6,7)$.

A	B	C	f	
0	0	0	0	$f = C$
0	0	1	1	
0	1	0	1	$f = C'$
0	1	1	0	
1	0	0	0	$f = 0$
1	0	1	0	
1	1	0	1	$f = 1$
1	1	1	1	



Example: Implementing A Four-input Function with A Multiplexer

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

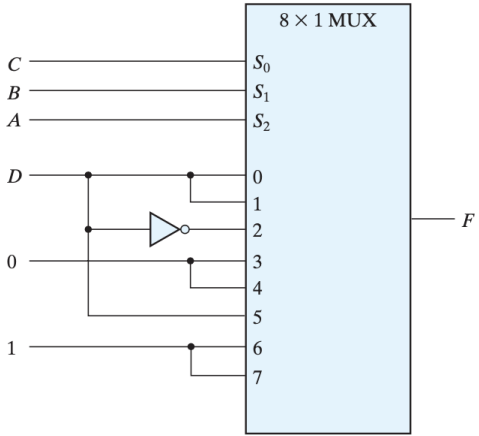


Figure: Screenshot of Figure 4.28 in [1].

Outline of This Lecture

More Standard Components: Arithmetic Circuits

- Binary Adder
- Binary Subtraction
- Binary Multiplication
- Decimal Adder

Summary

Outline

Binary Add

- Similar to the addition operation of decimal numbers.
  - Example:
    $0+0=0,\ 0+1=1,\ 1+0=1,\ 1+1=10$ ← The higher significant bit is called a **carry** (进位).
  - When the augend (被加数) and addend (加数) numbers contain more significant digits, the carry is added to the next higher order pair of significant bits.

augend:	101101	minuend:	101101	multiplicand:	1011
addend:	+100111	subtrahend:	−100111	multiplier:	× 101
sum:	1010100	difference:	000110		1011
					0000
				partial product:	1011
				product:	110111

Question: Do we need to design a new circuit for every n-bit addition? (i.e., design a circuit for 1-bit addition, a circuit for 2-bit addition, a circuit for 3-bit addition, . . .)

Binary Add

Question: Do we need to design a new circuit for every n-bit addition? (i.e., design a circuit for 1-bit addition, a circuit for 2-bit addition, a circuit for 3-bit addition, ...)

- Considering the decimal addition,
  - if we need to calculate $38+76$:
    - we can calculate $8+6=\underline{1}4$,
    - then $3+7+\underline{1}=\underline{1}1$,
    - finally $38+76=114$.
  - Then, we know how to calculate $138+976$:
    - $138+976=(100+900)+(38+76)$
    - First, $38+76=\underline{1}14$,
    - then, $1+9+\underline{1}=\underline{1}1$,
    - Finally, $138+976=1114$

Binary Add

Question: Do we need to design a new circuit for every n-bit addition? (i.e., design a circuit for 1-bit addition, a circuit for 2-bit addition, a circuit for 3-bit addition, ...)

- Considering binary addition,
  - if we need to calculate $11 + 01$:
    - we can calculate $1 + 1 = \underline{1}0$,
    - then $1 + 1 + \underline{1} = \underline{1}0$,
    - then $0 + 1 = 1$
    - finally $11 + 01 = 100$
  - We can also calculate $111 + 101$ similarly:
    - $111 + 101 = (100 + 100) + (011 + 001)$
    - First, $11 + 01 = \underline{1}00$
    - then $1 + 1 + \underline{1} = \underline{1}1$
    - finally $111 + 101 = 1100$

→ **If we can sum up two bits and a carry, then we can perform any n-bit addition.**

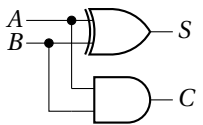
Half Adder and Full Adder

- A combinational circuit that performs the addition of two bits is called a **half adder** (半加器).
- A combinational circuit that performs the addition of three bits (two significant bits and a previous carry) is a **full adder** (全加器).
- Two half adders can be employed to implement a full adder.

Half Adder

- A combinational circuit that performs the addition of two bits is called a **half adder**.
- Design of half adder:
  - The circuit needs two binary inputs and two binary outputs.
    - Input variables: the augend (被加数) bit A and addend (加数) bit B.
    - Output variables: the sum S and carry C.
  - The simplified SOP expressions can be obtained directly from the truth table: $S = A \oplus B$ and $C = AB$.
  - Logic diagram can be implemented in SOPs.

Input		Output	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



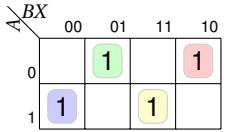
Full Adder

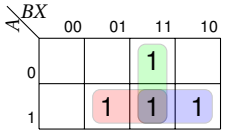
- Addition of n-bit binary numbers requires the use of a full adder.
- A **full adder** is a combinational circuit that forms the arithmetic sum of three bits
  - Input variables:
    - A and B: the two significant bits to be added;
    - X: the carry from the previous lower significant position.
  - Output variables: sum S and carry C.

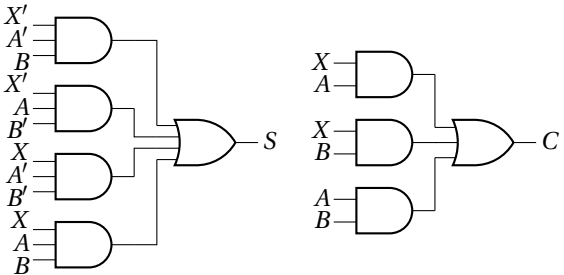
Input			Output	
A	B	X	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Circuit for Full Adder

$$S = A'B'X + A'BX' + AB'X' + ABX \qquad C = ABX + ABX' + A'BX + AB'X = AB + BX + AX$$

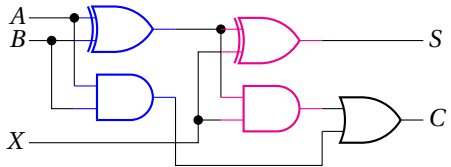






Full Adder Implemented with Half Adders

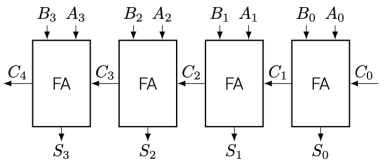
- A full adder can also be implemented with **two half adders and one** `OR` **gate**, as shown below.



$$S = (A \oplus B) \oplus X = (AB' + A'B)X' + (AB' + A'B)'X$$
$$= (AB' + A'B)X' + (AB + A'B')X = AB'X' + A'BX' + ABX + A'B'X.$$
$$C = X(A \oplus B) + AB = X(AB' + A'B) + AB = AB'X + A'BX + AB = AX + BX + AB$$

Binary Adder

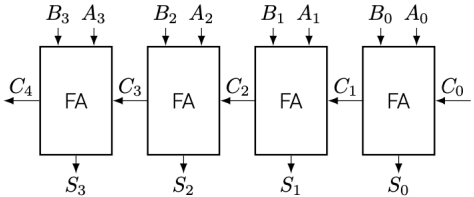
- A **binary adder** is a digital circuit that produces the arithmetic sum of two binary numbers.
- It can be constructed with full adders connected in cascade, i.e., **ripple-carry-adder** (串行进位加法器), with the output carry from each full adder connected to the input carry of the next full adder in the chain.
- Addition of n-bit numbers requires a chain of n full adders or a chain of 1 half adder and $n-1$ full adders.
  - Below shows the interconnection of 4 full adder (FA) circuits to provide a 4-bit binary ripple-carry-adder.



Example

- $1011 + 0011 = 1110$.

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

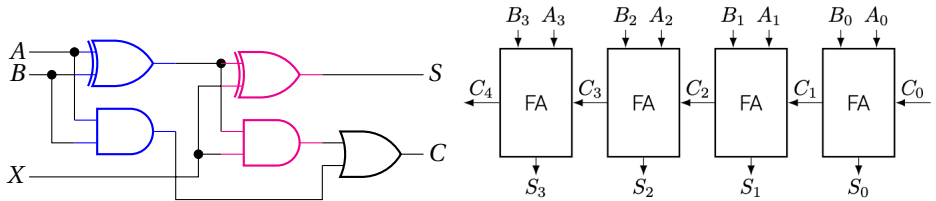


Four-bit Adder as A Standard Component

- The 4-bit adder is a typical example of a standard component.
- It can be used in many applications involving arithmetic operations.
- Observe that the design of this circuit by the classical method would require a truth table with $2^9 = 512$ entries, since there are 9 inputs to the circuit.
- By using an iterative method of cascading a standard function, it is possible to obtain a simple and straightforward implementation.
  → **Drawback**: Simple circuit but low speed.

Propagation Time (传播时间)

- As in any combinational circuit, the signal must propagate through the gates before the correct output sum is available in the output terminals.
  Propagation time = propagation delay of a typical gate × number of gate levels
- For an n-bit adder, there are $2n$ gate levels for the carry to propagate from input to output.
  - Considering S_3, inputs A_3 and B_3 are available as soon as input signals are applied to the adder.
  - However, C_3 does not settle to its final value until C_2 is available from the previous stage.

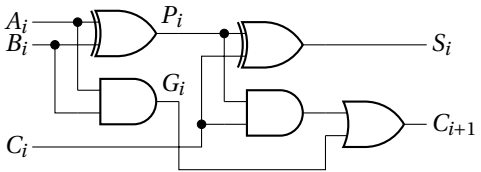


Carry Propagation

- The **carry propagation time** is an important attribute of the adder because it limits the speed with which two numbers are added.
  - Since all other arithmetic operations are implemented by successive additions, the time consumed during the addition process is critical.
- Solutions:
  - Employ faster gates with reduced delays. However, physical circuits have a limit to their capability.
  - Increase the complexity of the equipment in such a way that the carry delay time is reduced.
    - For instance, adding two binary numbers in parallel.
    - The most widely used technique employs the principle of **carry lookahead logic**.

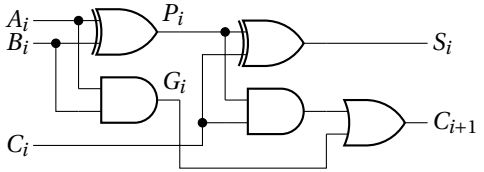
Carry Lookahead Logic (超前进位逻辑)

- The addition of two binary numbers in parallel implies that all the bits of the augend and addend are available for computation at the same time.



- Consider the above full-adder circuit:
  - $G_i = A_i B_i$ ← Called a carry propagator
  - $P_i = A_i \oplus B_i$ ← Called a carry generator
- The output sum and carry can respectively be expressed as
  - $S_i = P_i \oplus C_i$,
  - $C_{i+1} = G_i + P_i C_i$.

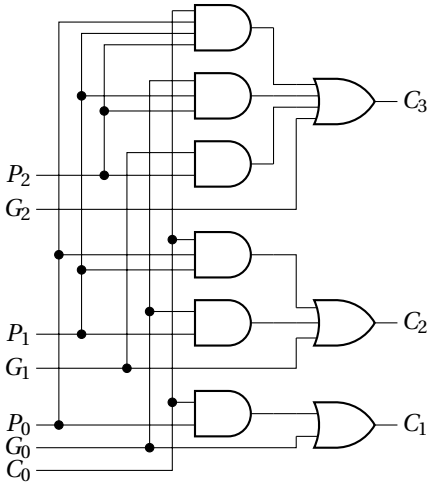
Carry Lookahead Logic

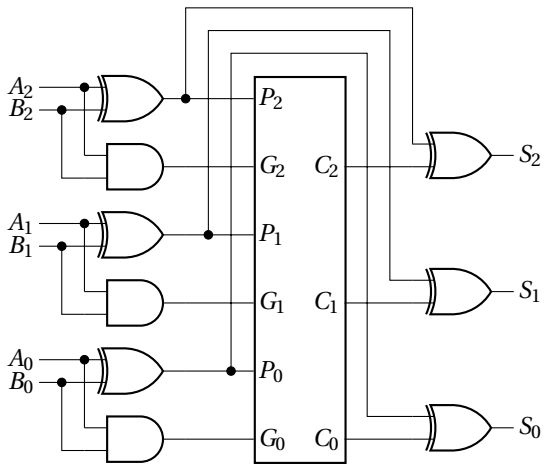


- The output sum and carry can respectively be expressed as $S_i = P_i \oplus C_i$, $C_{i+1} = G_i + P_i C_i$, $G_i = A_i B_i$, $P_i = A_i \oplus B_i$.
- We now write the Boolean functions for the carry outputs of each stage and substitute the value of each C_i from the previous equations.

$$C_0 = \text{input carry},$$
$$C_1 = G_0 + P_0 C_0,$$
$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_0,$$
$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0.$$

Implementing A Three-bit Adder with Carry Lookahead Generator





$$S_i = P_i \oplus C_i,\ C_{i+1} = G_i + P_i C_i,\ G_i = A_i B_i,\ P_i = A_i \oplus B_i$$

Outline

Subtraction (减法)

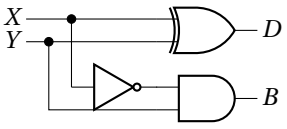
- **Subtraction** is another basic function of arithmetic operations of information-processing tasks of digital computers.
  - $0 - 0 = 0$,
  - $0 - 1 = 1$ with borrow of 1,
  - $1 - 0 = 1$,
  - $1 - 1 = 0$.
  - The first, third, and fourth operations produce a subtraction of one digit, but the second operation produces a difference bit as well as a **borrow** bit.
- A combinational circuit that performs the subtraction of 2 bits as described above is called a **half-subtractor** (半减器).
- The subtraction operation involves 3 bits — the **minuend** (被减数) bit, **subtrahend** (减数) bit, and the **borrow** (借位数) bit, and produces a different result as well as a borrow. The combinational circuit that performs this type of subtraction operation is called a **full-subtractor** (全减器) .

Half-subtractor

- As described above, a half-subtractor has two inputs and two outputs.
  - Input variables: the minuend X and subtrahend Y
  - Output variables: D for difference and B for borrow.

Input variables		Output variables	
X	Y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

- $D = X \oplus Y$,
- $B = X'Y$.



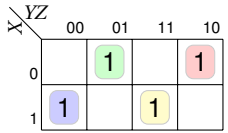
Full-subtractor

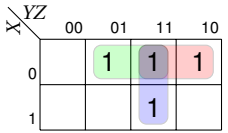
- A combinational circuit of full-subtractor performs the operation of subtraction of three bits:
  - Input variables: the minuend X, subtrahend Y, and borrow Z generated from the subtraction operation of previous significant digits.
  - Output variables: D for difference and B for borrow.

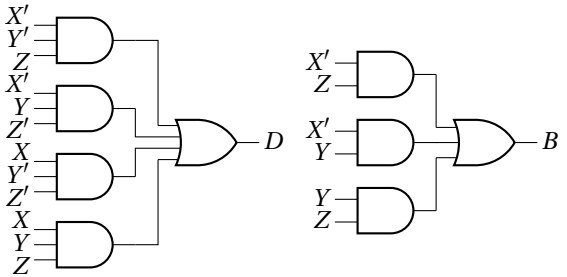
Input variables			Output variables	
X	Y	Z	D	B
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Full-subtractor

$$D = X'Y'Z + X'YZ' + XY'Z' + XYZ \qquad B = X'Z + X'Y + YZ$$





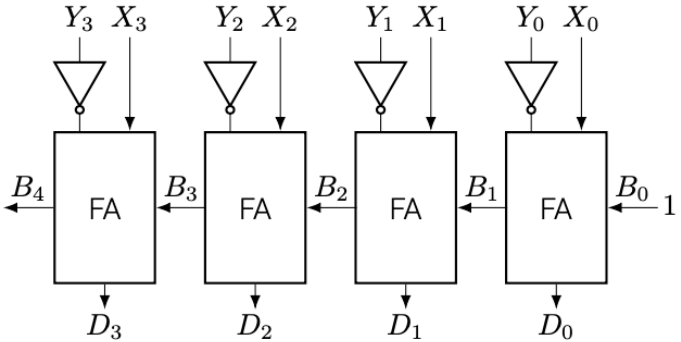


Binary Subtractor

- The subtraction of unsigned binary numbers can be done most conveniently by means of **complements** (补码). (cf. Lecture 1)
  - The subtraction $X - Y$ can be done by taking the 2's complement of Y and adding it to A. Thus, $X - Y = X + 2's\ complement\ of\ Y$ (Subtraction → Addition)
  - The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits.
  - The 1's complement can be implemented with inverters, and a 1 can be added to the sum through the input carry.

Binary Subtractor

- The circuit for subtracting $X - Y$ consists of an adder with inverters placed between each data input Y and the corresponding input of the full adder.



Binary Subtractor

- The addition and subtraction operations can be combined into one circuit with one common binary adder by including an `exclusive-OR` gate with each full adder.

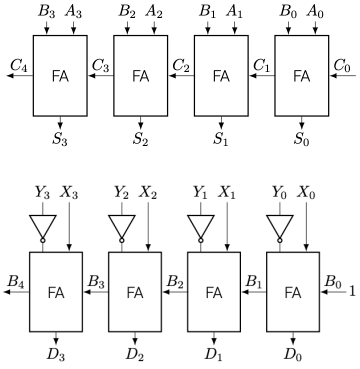


Figure: Top: adder. Bottom: subtractor.

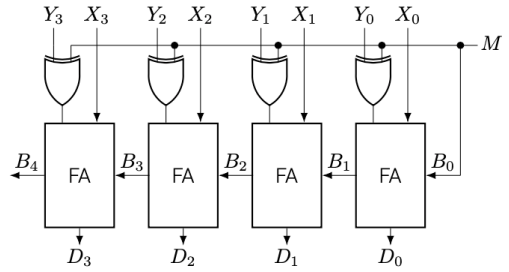


Figure: One circuit of addition and subtraction.

Binary Subtractor

- It is worth noting that binary numbers in the signed-complement system are added and subtracted by the same basic addition and subtraction rules as are unsigned numbers.
- Therefore, computers need only one common hardware circuit to handle both types of arithmetic.

We have mentioned it in Lecture 1 :)

Recap: Lecture 1

- **Sign-magnitude representation** ← Separate handling of the sign and the magnitude makes it awkward when employed in computer arithmetic.
- **Signed-complement system**
  - Signed-1's complement representation ← Useful as a logical operation.
  - Signed-2's complement representation ← Widely used for signed binary arithmetic.

Question

- **Why prefer signed-2's complement representation over the other two ?**

Answer: It allows to perform both addition and subtraction operations using a single circuit because:
"*Compared to other systems for representing signed numbers (e.g., ones' complement), two's complement has the advantage that the fundamental arithmetic operations of addition, subtraction, and multiplication are identical to those for unsigned binary numbers (as long as the inputs are represented in the same number of bits as the output, and any overflow beyond those bits is discarded from the result). This property makes the system simpler to implement, especially for higher-precision arithmetic. Unlike ones' complement systems, two's complement has no representation for negative zero, and thus does not suffer from its associated difficulties.*"
(`https://en.wikipedia.org/wiki/Two's_complement`)

Overflow

- When two numbers with n digits each are added and the sum is a number occupying $n+1$ digits, we say that an **overflow** (溢出) occurred.
- An overflow may occur if the two numbers are both positive or negative.

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

- If the carry out of the sign bit position is taken as the sign bit of the result, then the nine-bit answer obtained will be correct.
- But since the answer cannot be accommodated within eight bits, we say that an overflow has occurred.

Overflow

- Overflow is a problem in digital computers because the number of bits that hold the number is finite and a result that contains $n+1$ bits cannot be accommodated by an n-bit word.
  - For this reason, many computers detect the occurrence of an overflow, and when it occurs, a corresponding **flip-flop** (触发器) is set that can then be checked by the user.
- The detection of an overflow after the addition of two binary numbers depends on whether the numbers are considered to be signed or unsigned.
  - When two unsigned numbers are added, an overflow is detected from the end carry out of the most significant position.
  - When two signed numbers are added, the sign bit is treated as part of the number and the end carry does not indicate an overflow.

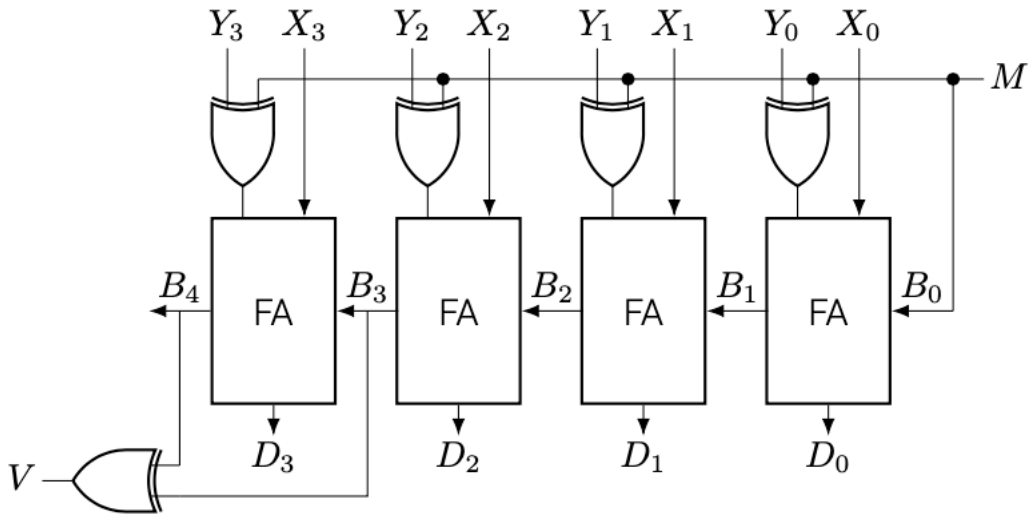
Overflow

- An overflow condition can be detected by observing the carry into the sign bit position and the carry out of the sign bit position.
- If these two carries are not equal, an overflow has occurred.

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

- If the two carries are applied to an `exclusive-OR` gate, an overflow is detected when the output of the gate is equal to 1.

4-bit Adder–Subtractor with Overflow Detection



Outline

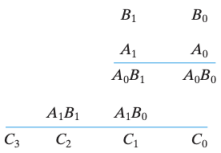
Binary Multiplication (乘法器)

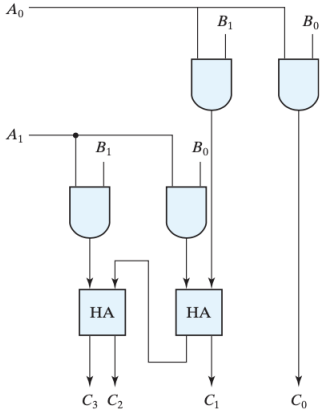
- Same as multiplication of decimal numbers.
  - Starting from the least significant bit (right → left),
    - the multiplicand is multiplied by each bit of the multiplier;
    - each such multiplication forms a partial product;
    - successive partial products are shifted one position to the left.
  - The final product is obtained from the sum of the partial products.
- Example: $B_1B_0 \times A_1A_0$

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

Binary Multiplier Implemented with A Combinational Circuit

- Example: $B_1B_0 \times A_1A_0$
  - The two partial products are added with two half-adder (HA) circuits.





Binary Multiplier

- A combinational circuit binary multiplier with more bits can be constructed in a similar fashion.
  - A bit of the multiplier is AND with each bit of the multiplicand in as many levels as there are bits in the multiplier. → $(J \times K)$ AND gates
  - The binary output in each level of AND gates is added with the partial product of the previous level to form a new partial product. → $(J-1)K$-bit adders
  - The last level produces the product.

- For J multiplier bits and K multiplicand bits, we need $(J \times K)$ AND gates and $(J-1)K$-bit adders to produce a product of $(J+K)$ bits.

Question: How to perform binary division?

Outline

Decimal Adder

- 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.
- Consider the arithmetic addition of two decimal digits in BCD, together with an input carry from a previous stage.
  - Since each input digit does not exceed 9, the output sum cannot be greater than $9+9+1=19$, the 1 in the sum being an input carry.
  - Suppose we apply two BCD digits to a four-bit binary adder. The adder will form the sum in binary and produce a result that ranges from 0 through 19.

Decimal Adder

- When the binary sum is equal to or less than 1001, the result is a valid BCD code. (cf. Lecture 1)
- When the binary sum is greater than 1001, we obtain an invalid BCD representation. The addition of binary 6 (0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required.

K	$Z8$	$Z4$	$Z2$	$Z1$	C	$S8$	$S4$	$S2$	$S1$
0	1	0	1	0	1	0	0	0	0
0	1	0	1	1	1	0	0	0	1
0	1	1	0	0	1	0	0	1	0
0	1	1	0	1	1	0	0	1	1
0	1	1	1	0	1	0	1	0	0
0	1	1	1	1	1	0	1	0	1
1	0	0	0	0	1	0	1	1	0
1	0	0	0	1	1	0	1	1	1
1	0	0	1	0	1	1	0	0	0
1	0	0	1	1	1	1	0	0	1

Recap: Binary-Coded-Decimal (BCD, 8421)

- Four bits are required to code each decimal number.
  - Also known as **8-4-2-1 code**, as 8, 4, 2, and 1 are the weights of the four bits of BCD.
  - Only use 0000 – 1001 for representing 0 – 9 (decimal), the other 6 are not used.
- It is convenient to use BCD for input and output in digital systems.
- Example: Give the BCD equivalent for the decimal number 69.27.

Decimal number	6	9	.	2	7
BCD code	0110	1001	.	0010	0111

Therefore, $(69.27)_{10} = (01101001.00100111)_{\text{BCD}}$.

Recap: BCD Addition

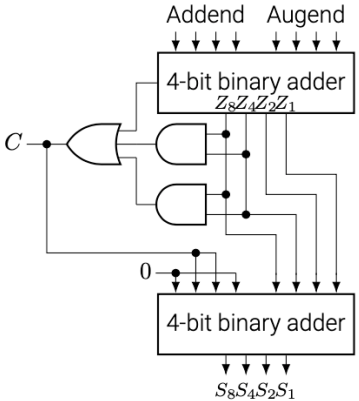
Rules

- First add the two numbers using normal rules for binary addition.
- If the 4-bit sum is equal to or less than 9, it becomes a valid BCD number.
- If the 4-bit sum is greater than 9, or if a carry (进位) out of the group is generated, it is an invalid result.
  - In such a case, add $(0110)_2$ or $(6)_{10}$ to the 4-bit sum in order to skip the six invalid states and return the code to BCD.
  - If a carry results when 6 is added, add the carry to the next 4-bit group.
- Example: $0111_{\text{BCD}} + 1001_{\text{BCD}}$:

$$
\begin{array}{rl}
 & 0111 \\
+ & 1001 \\
\hline
 & 10000 \rightarrow \text{Invalid BCD number} \\
+ & 0110 \rightarrow \text{Add } 6 \\
\hline
0001 & \quad 0110 \rightarrow \text{Valid BCD number}
\end{array}
$$

Decimal Adder

- $C = K + Z_8Z_4 + Z_8Z_2$.
- When $C = 1$, it is necessary to add 0110 to the binary sum and provide an output carry for the next stage.



Outline of This Lecture

Summary

- Standard components: combinational circuits that are employed extensively in the design of digital systems.
  - Adders, subtractors, multipliers, comparators, decoders, encoders, and multiplexers.
- Adders can be used to implement subtractors and multipliers.
- Encoders and multiplexers can be used to implement any combinatorial circuits.

Essential Reading

- Essential reading for this lecture: pages 133-148 of the textbook.
- Essential reading for next lecture: pages 190-196 of the textbook.

[1] M. M. Mano and M. Ciletti, *Digital design: with an introduction to the Verilog HDL*. Pearson, 2013

Today's lab: multiplexer & demultiplexer