DIGITAL DESIGN AND COMPUTER ORGANIZATION

Binary Adder Subtractor, Decimal Adder

Team DDCO
Department of Computer Science and Engineering

DIGITAL DESIGN AND COMPUTER ORGANIZATION

Adder Subtractor Decimal Adder

Department of Computer Science and Engineering

Combinational logic
Adder Subtractor

Binary Adder-Subtractor

- A binary adder–subtractor is a combinational circuit that performs the arithmetic operations of addition and subtraction with binary numbers.
- A combinational circuit that performs the **addition of two bits is called a half adder.**
- One that performs the **addition of three bits (two significant bits and a previous carry) is a full adder**
- The names of the circuits stem from the fact that two half adders can be employed to implement a full adder.

M. Morris Mano, Michael D. Ciletti, *Digital Design: With an Introduction to the Verilog HDL*, 5th ed., Prentice Hall, 2012, Section 4.5.

Combinational logic
Adder Subtractor

Binary Adder

- Simple addition consists of four possible elementary operations:
  0+0 = 0
  0+1 = 1
  1+0 = 1
  1+1 = 10 ← Sum has Carry

Half Adder (two ‘2’ input bits)
Full Adder (three ‘3’ input bits)

Half Adder

- The simplified Boolean functions for the two outputs can be obtained directly from the truth table. The simplified sum-of-products expressions are

$$S = x'y + xy'$$
$$C = xy$$

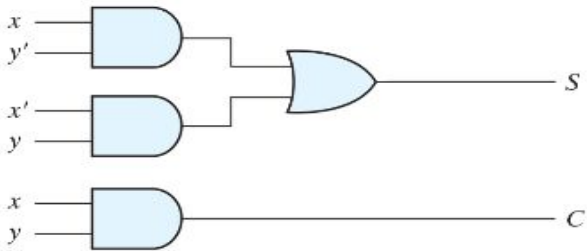
Combinational logic
Adder Subtractor

Truth Table of Half Adder

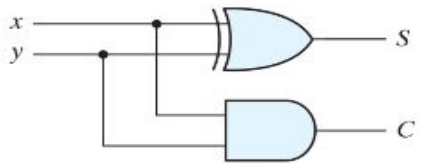
x	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Combinational logic
Adder Subtractor

Implementation of half adder.



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



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

Combinational logic
Adder Subtractor

Full Adder

- A full adder is a combinational circuit that forms the arithmetic sum of three bits. It consists of three inputs and two outputs. Two of the input variables, denoted by x and y ,represent the two significant bits to be added.

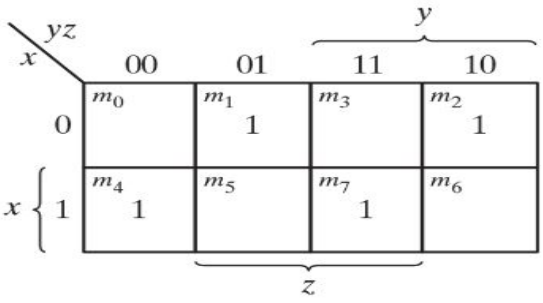
Combinational logic
Adder Subtractor

Truth Table of Full Adder

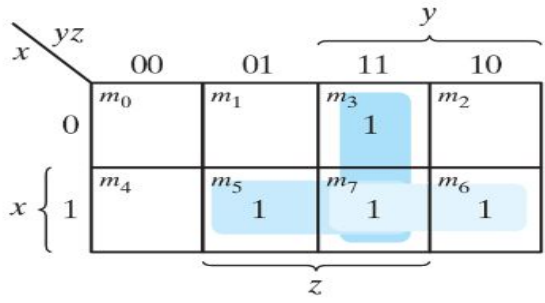
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Combinational logic
Adder Subtractor

K-map for full adder.



(a) $S = x'y'z + x'yz' + xy'z' + xyz$



(b) $C = xy + xz + yz$

Full Adder

- The S output from the second half adder is the exclusive-OR of z and the output of the first half adder, giving

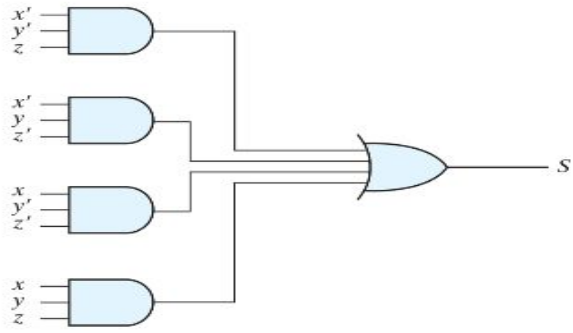
$$\begin{aligned} S &= z \oplus (x \oplus y) \\ &= z'(xy' + x'y) + z(xy' + x'y)' \\ &= z'(xy' + x'y) + z(xy + x'y') \\ &= xy'z' + x'yz' + xyz + x'y'z \end{aligned}$$

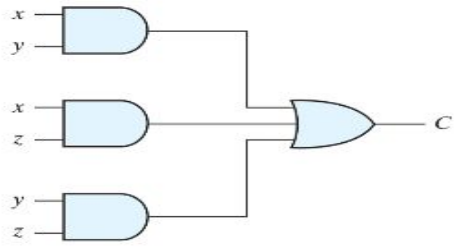
The carry output is

$$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$$

Combinational logic
Adder Subtractor

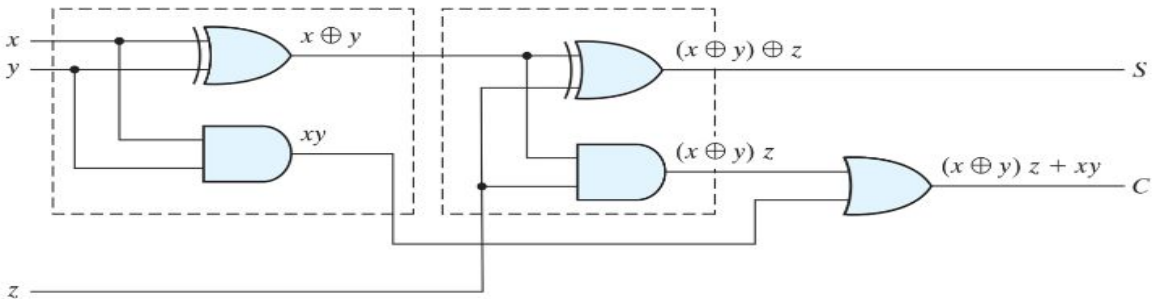
Implementation of full adder in sum-of-products form.





Combinational logic
Adder Subtractor

Implementation of full adder with two half adders and an OR



Full Adder

- Full adders are used in the Arithmetic Logic Unit (ALU) of processors to perform binary addition in devices like computers, smartphones, and calculators.

Combinational logic

Adder Subtractor

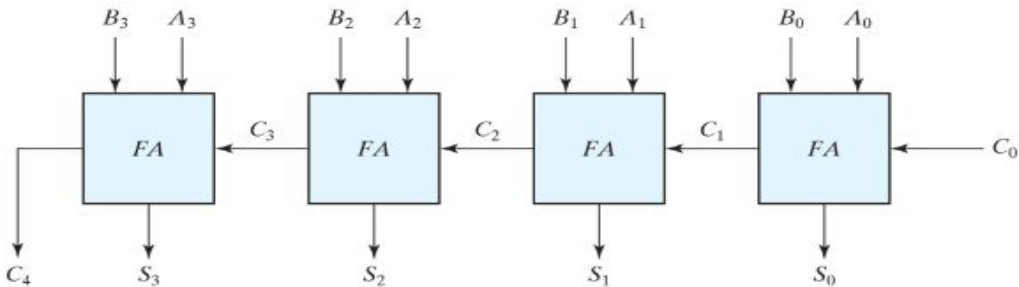
Binary Adder

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, 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 one-half adder and n - 1 full adders.

Combinational logic
Adder Subtractor

Implementation of Four-bit binary adder can be implemented by using 4 full adders 2^9 = 512 entries (classical method, by using adders it can be avoided)



Combinational logic

Adder Subtractor

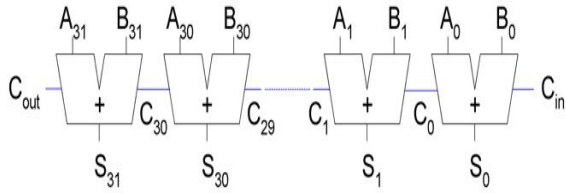
Binary Adder

- Consider a 4-bit adder. Let's denote Ai and Bi as the input bits of the two binary numbers and Ci is the carry from previous bit addition of the numbers.
- We also let Si be the sum and Ci+1 the carry output in the current addition.

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}

Combinational logic
Ripple Carry Adder

- Chain 1-bit adders together
- Carry ripples through entire chain
- Disadvantage: **slow**
- In ripple carry adders, for each adder block, the two bits that are to be added are available instantly.
- However, each adder block waits for the carry to arrive from its previous block.



Combinational logic
Ripple Carry Adder

- The *C*out of one stage acts as the *C*in of the next stage
- The ith block waits for the (i-1) th block to produce its carry.
- So there will be a considerable time delay which is carry propagation delay.
- The propagation time is equal to the propagation delay of each adder block, multiplied by the number of adder blocks in the circuit.
- Time complexity:

$t_{ripple} = Nt_{FA}$
where t_{FA} is the delay of a 1-bit full adder

Combinational logic

Ripple Carry Adder

For example, if each full adder stage has a propagation delay of 20 nanoseconds, then will reach its final correct value after 80 (20 × 4) nanoseconds.

The situation gets worse, if we extend the number of stages for adding more number of bits.

The ripple carry adder has the disadvantage of being slow when *N* is large

Ripple-carry addition requires $\Theta(n)$ time because of the rippling of carry bits through the circuit.

Combinational logic
Carry Propagation

Instead of generating and propagating carry bit-by-bit, **can we generate all of them in parallel** and break the sequential chain?

CLA is another type of carry propagate adder that solves this problem by **dividing the adder into *blocks*** and providing circuitry to quickly determine the carry out of a block as soon as the carry in is known.

Thus it is said **to *look ahead* across the blocks rather than waiting to ripple through all the full adders inside a block**

Combinational logic
Carry Propagation

In Ripple Carry Adder, each full adder has to wait for its carry-in from its previous stage full adder.

Thus, **n^{th}** full adder has to wait until all **(n-1) full adders** have completed their operations.

This causes a delay and makes ripple carry adder extremely slow.

The situation becomes worst when the value of n becomes very large.
To overcome this disadvantage, Carry Look Ahead Adder comes into play
Carry Look Ahead Adder is an improved version of the ripple carry adder.
It generates the carry-in of each full adder simultaneously without causing any delay.

Combinational logic
Carry Propagation

- **In 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.**
- The total propagation time is equal to the propagation delay of a typical gate, times the number of gate levels in the circuit.
- Carry propagation time is an important attribute of the adder because it limit the speed with which two numbers are added. Consider the circuit of the full adder

Combinational logic
Carry Propagation

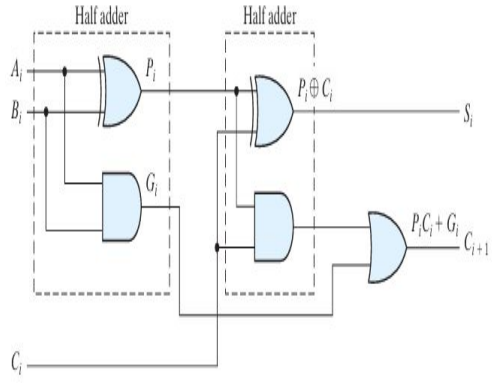


FIGURE 4.10
Full adder with *P* and *G* shown

- A block is said to generate a carry if it produces a carry out independent of the carry in to the block
- **Column i will generate a carry out if Ai AND Bi are both 1.**

  Gi = Ai Bi
- Gi produces the carry when both Ai, Bi are 1 regardless of the input carry.
- The block is said to propagate a carry if it produces a carry out whenever there is a carry in to the block
- The *i*th column will propagate a carry Ci, if either *Ai* or *Bi* is 1.

  Thus, *Pi* = *Ai* XOR *Bi*.

Combinational logic
Carry Propagation

Full adder with P and G shown.

These two signals are common to all half adders and depend on only the input augend and addend bits. The signal from the input carry Ci to the output carry Ci+1 propagates through an AND gate and an OR gate, which constitute two gate levels. If there are four full adders in the adder, the output carry C4 would have 2 * 4 = 8 gate levels from C0 to C4. For an n -bit adder, **there are 2 n gate levels for the carry to propagate from input to output.**

Combinational logic
Carry Propagation

- Consider the circuit of the full adder. If we define two new binary variables

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

output sum and carry can respectively be expressed as

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

- Pi=carry propagate
- Gi=Carry generate
- Carry lookahead logic-most widely used technique to reduce the carry delay time

Carry Propagation

$$C_0 = \text{input carry}$$

$$C_1 = G_0 + P_0C_0$$

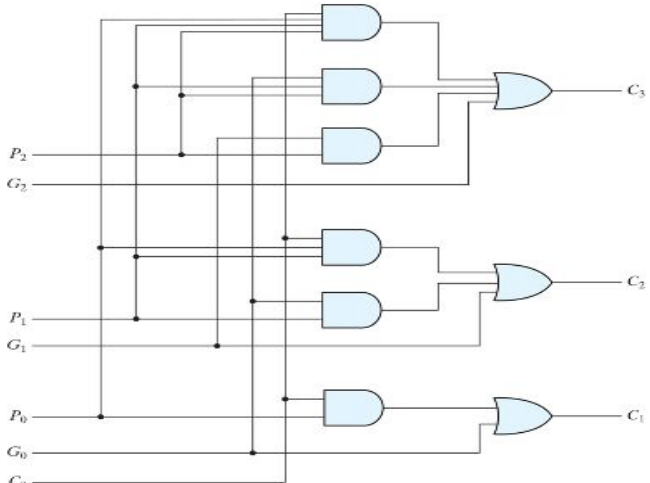
$$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$$

$$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 = P_2P_1P_0C_0$$

Combinational logic

Carry Propagation

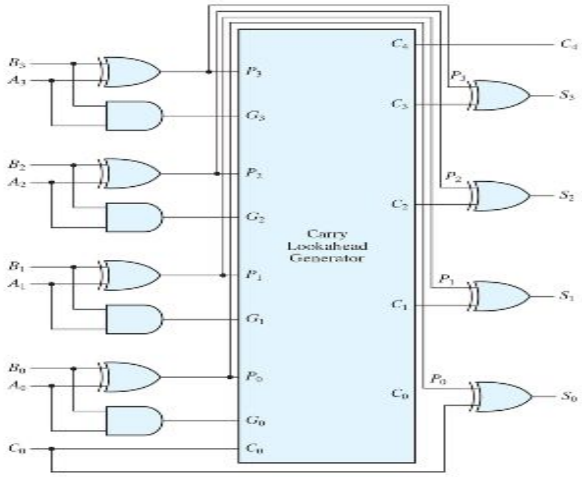
Logic diagram of carry lookahead generator



Combinational logic

Carry Propagation

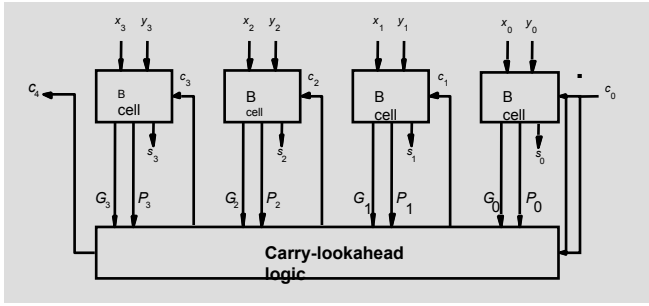
Four-bit adder with carry lookahead.



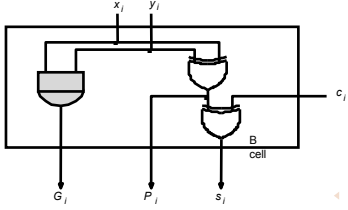
Combinational logic

Carry Look Ahead Adder

4 bit Carry-lookahead adder



4-bit carry-lookahead adder

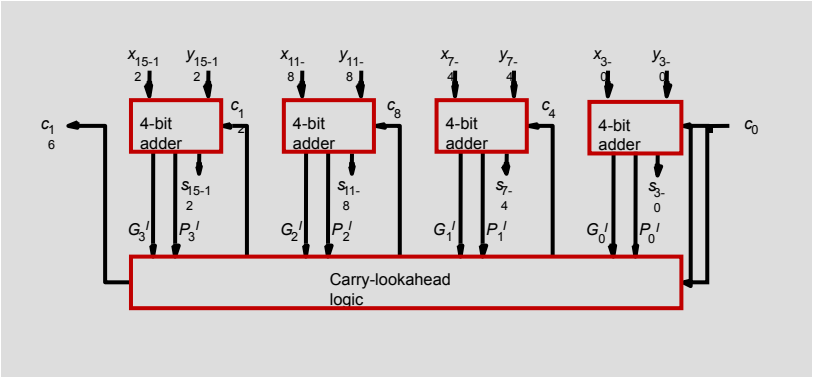


B-cell for a single stage

Combinational logic

Carry Look Ahead Adder

16- bit Block Carry-Look ahead adder



Combinational logic
Adder Subtractor

Binary Subtractor

- Binary Subtraction is the process of finding the difference between two binary numbers. Instead of directly subtracting, it is implemented by adding the 2's complement of the subtrahend to the minuend

- This method allows the same hardware circuit to perform both addition and subtraction, making it efficient and widely used in digital computers and processors.

Combinational logic
Adder Subtractor

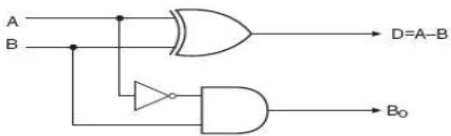
Binary Subtractor

$$D = \overline{A}.B + A.\overline{B}$$

$$B_o = \overline{A}.B$$

A	B	D	B_O
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

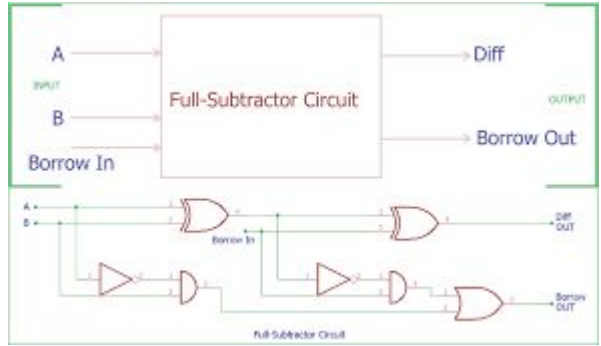
Half Subtractor



Combinational logic
Adder Subtractor

Full subtractor

Inputs			Outputs	
A	B	$Borrow_{in}$	Diff	Borrow
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



Combinational logic
Adder Subtractor

Adder-subtractor circuit

A-B is equal to A+ 2's complement of B= a+(1's complement of B+1)

For unsigned numbers, that gives $A - B$ if $A \geq B$ or the 2's complement of $(B - A)$ if $A < B$. For signed numbers, the result is $A - B$, provided that there is no overflow.

ADDER, SUBTRACTOR, OVERFLOW - 3

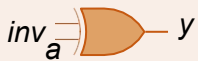
Two's Complement Adder / Subtractor

XOR as Controlled Inverter

- Truth table:

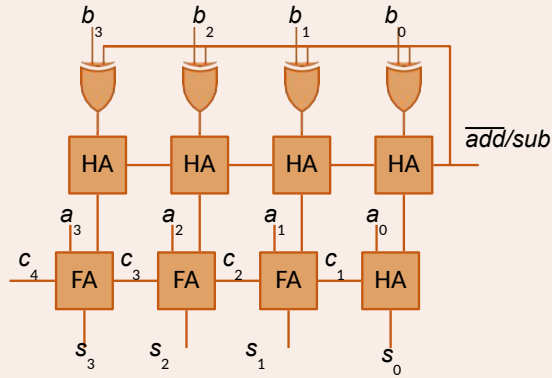
inv	*a*	*y*
0	0	0
0	1	1
1	0	1
1	1	0

- Symbol:



- When *inv* = 0, *y* = *a*
- When *inv* = 1, *y* = ~~*a*~~

Two's Complement Adder / Subtractor



- *add/sub* denotes: ___
  - Addition when *add/sub* = 0
  - Subtraction when ~~*add/sub*~~ = 1

Combinational logic
Adder Subtractor

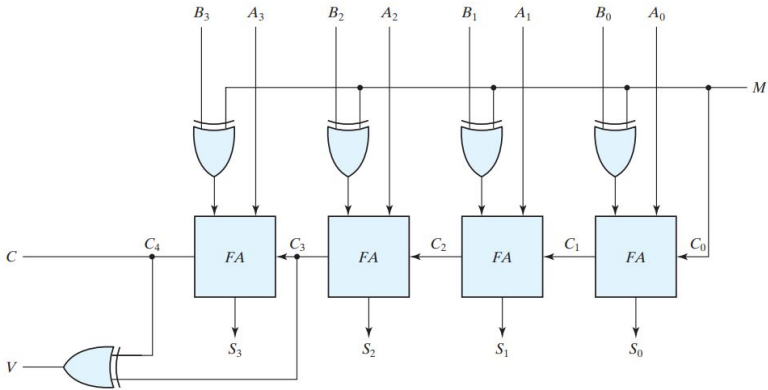


FIGURE 4.13
Four-bit adder–subtractor (with overflow detection)

Combinational logic
Adder Subtractor

Overflow condition

When two numbers with n digits each are added and the sum is a number occupying n + 1 digits, we say that an overflow occurred

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.**
EX A-B= 1001 -0110= 1_0011

Signed numbers
MSB(leftmost bit)- sign
Negative numbers are represented in 2's complement.

Overflow will not occur if opposite sign
Overflow will occur if same sign

Combinational logic
Overflow Condition

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

+70 and +80 are eight bit numbers Range: +127 to -128

70 + 80=150 – not in range
-70-80= -150 not in range

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

ADDER, SUBTRACTOR, OVERFLOW - 4

Two's complement addition

No overflow when

numbers have opposite signs

(or one/both are zero)

Overflow can occur when

When both positive

msb is 1 (c_{msb} is 1)

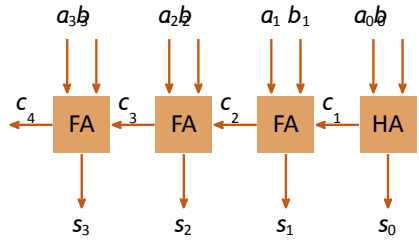
c_{msb-1} is 1 and c_{msb} is 0

When both negative

msb is 0 (c_{msb-1} is 0)

c_{msb-1} is 0 and c_{msb} is 1

overflow = $c_{msb} \oplus c_{msb-1}$



Gate-Level Minimization and Combinational logic

Adder Subtractor with Overflow Detection

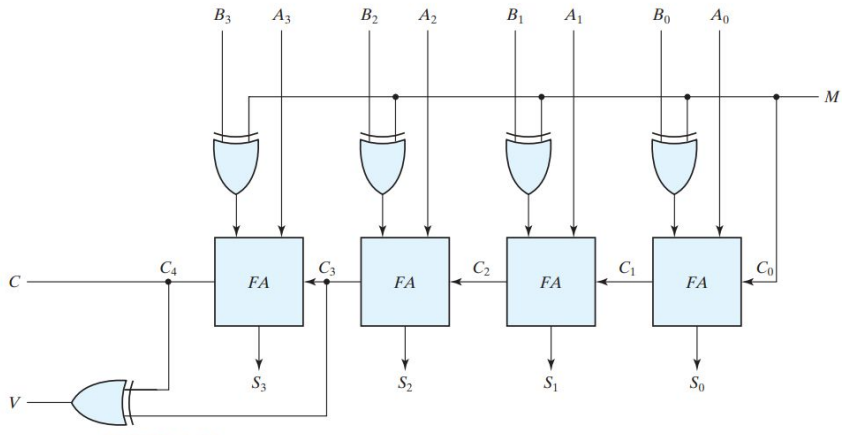


FIGURE 4.13
Four-bit adder–subtractor (with overflow detection)

Decimal Adder- BCD adder

Binary to BCD conversion

BCD-0 to 9

Add 6 to convert the number from Binary to BCD(8421 representation) if it exceeds 9

The number 6 is chosen because it is the smallest value that, when added to a binary number between 10 (1010) and 15 (1111), produces a valid BCD representation.

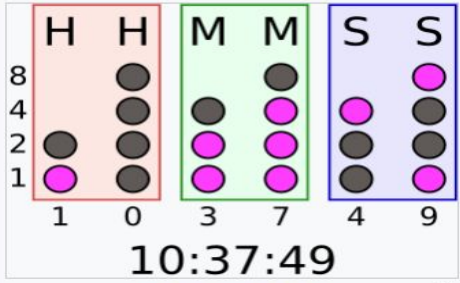
BCD addition
45+28

M. Morris Mano, Michael D. Ciletti, *Digital Design: With an Introduction to the Verilog HDL*, 5th ed., Prentice Hall, 2012, Section 4.6.

Combinational logic

Decimal Adder-BCD Adder

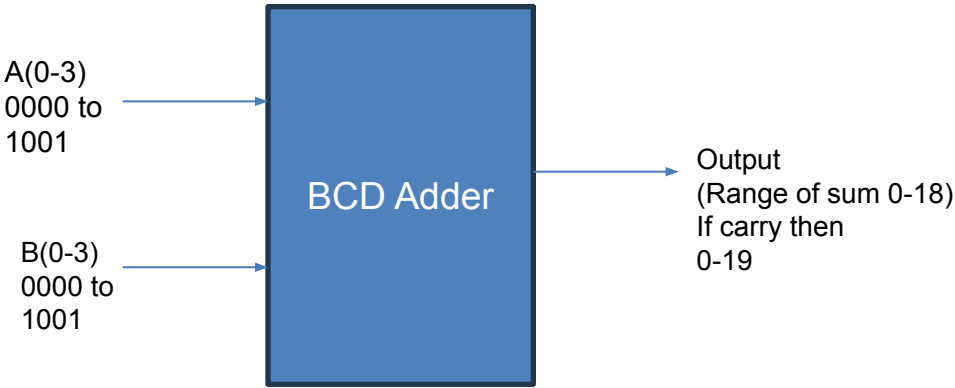
Why BCD?



Reading a binary-coded decimal clock: Add the values of each column of LEDs to get six decimal digits. There are two columns each for hours, minutes and seconds.

Combinational logic
Decimal Adder-BCD Adder

BCD adder



Combinational logic

Decimal Adder-BCD Adder

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

Table 4.5
Derivation of BCD Adder

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

Combinational logic

Decimal Adder-BCD Adder

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

Table 4.5
Derivation of BCD Adder

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

Combinational logic

Decimal Adder-BCD Adder

In examining the contents of the table, it becomes apparent that when the binary sum is equal to or less than 1001, the corresponding BCD number is identical, and therefore no conversion is needed. 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.

Combinational logic

Decimal Adder-BCD Adder

The logic circuit that detects the necessary correction can be derived from the entries in the table. It is obvious that a correction is needed when the binary sum has an output carry K=1. The other six combinations from 1010 through 1111 that need a correction have a 1 in position Z8. To distinguish them from binary 1000 and 1001, which also have a 1 in position Z8, we specify further that either Z4 or Z2 must have a 1. The condition for a correction and an output carry can be expressed by the Bool ean function

$$C=K+Z_8Z_4+Z_8Z_2$$

Combinational logic

Decimal Adder-BCD Adder

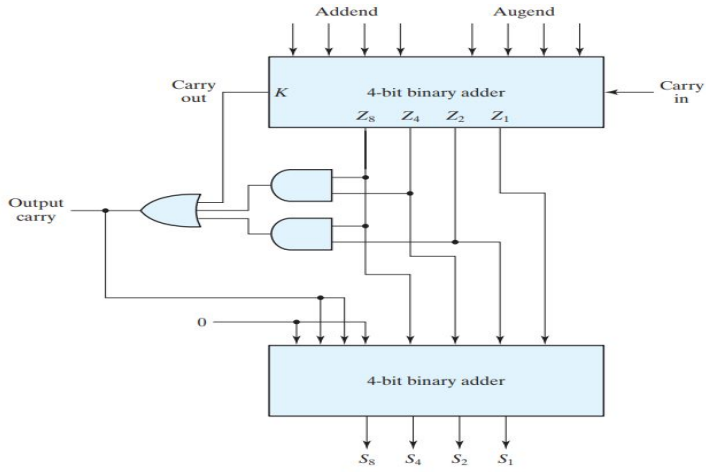


FIGURE 4.14
Block diagram of a BCD adder

Combinational logic

Decimal Adder-BCD Adder

A BCD adder that adds two BCD digits and produces a sum digit in BCD . The two decimal digits, together with the input carry, are first added in the top four-bit adder to produce the binary sum. **When the output carry is equal to 0, nothing is added to the binary sum. When it is equal to 1, binary 0110 is added to the binary sum through the bottom four-bit adder.**

The output carry generated from the bottom adder can be ignored, since it supplies information already available at the output carry terminal. A decimal parallel adder that adds n decimal digits needs n BCD adder stages. The output carry from one stage must be connected to the input carry of the next higher order stage.

Combinational logic

Think about it

. What is the Boolean expression used to detect correction in a BCD adder?

A) C=K+Z8Z4+Z8Z2

B) C=K+Z4Z2

C) C=K · Z8C

D) C=Z8+Z4+Z2

In a 4-bit ripple carry adder, the worst-case carry propagation delay occurs when:

A) All input bits are 0

B) Only LSB has 1 and rest are 0

C) All inputs are 1

D) Inputs cause a carry to propagate through all stages

Combinational logic

Think about it

. What is the Boolean expression used to detect correction in a BCD adder?

A) C=K+Z8Z4+Z8Z2

B) C=K+Z4Z2

C) C=K · Z8C

D) C=Z8+Z4+Z2

Ans: A

In a 4-bit ripple carry adder, the worst-case carry propagation delay occurs when:

A) All input bits are 0

B) Only LSB has 1 and rest are 0

C) All inputs are 1

D) Inputs cause a carry to propagate through all stages

Answer: D) Inputs cause a carry to propagate through all stages

Combinational logic

Think about it

In 2's complement representation, what is the result of 1001−0110?

A) 0011
B) 1111
C) 1011
D) 0111

In a BCD adder, a correction factor is added when:

A) The binary sum exceeds 1111
B) The binary sum exceeds 1001
C) The output carry is 0
D) The binary sum is less than 1001

Combinational logic

Think about it

In 2's complement representation, what is the result of 1001−0110?

A) 0011
B) 1111
C) 1011
D) 0111

Answer: A) 0011

In a BCD adder, a correction factor is added when:

A) The binary sum exceeds 1111
B) The binary sum exceeds 1001
C) The output carry is 0
D) The binary sum is less than 1001

Answer: B) The binary sum exceeds 1001

THANK YOU

Team DDCO
Department of Computer Science