DIGITAL TECHNICS II

Dr. Bálint Pődör

Óbuda University,
Microelectronics and Technology Institute

10. LECTURE: ARITHMETIC CIRCUITS

2nd (Spring) term 2017/2018

1

10. LECTURE: ARITMETHIC CIRCUITS, ALU

1. Basic arithmetic circuits and building blocks
2. Binary adders
3. BCD adders
4. Binary multipliers

2

ARITHMETIC ELEMENTS

• Arithemtic elements- perform various arithemetic operatios.

• Operations – performed between operands.

• Operands – from memory, from internal temporary storage elements (registers).

• Result – to internal temporary storage elements or to other type of memory.

ARITHMETIC CIRCUITS: BASIC BUILDING BLOCKS

We will discuss those combinational logic building blocks that can be used to perform addition and subtraction operations on binary numbers. Addition and subtraction are the two most commonly used arithmetic operations, as the other two, namely multiplication and division, are respectively the processes of repeated addition and repeated subtraction.

We will begin with the basic building blocks that form the basis of all hardware used to perform the aforesaid arithmetic operations on binary numbers. These include *half-adder*, *full adder*, *half-subtractor*, *full subtractor* and *controlled inverter*.

CIRCUITS FOR BINARY ADDITION (RECAPITULATION …)

- Half adder (add two 1-bit numbers)
  - Sum = Ai' Bi + Ai Bi' = Ai xor Bi
  - Cout = Ai Bi
- Full adder (carry-in to cascade for multi-bit adders)
  - Sum = Ci xor A xor B
  - Cout = B Ci + A Ci + A B = Ci (A + B) + A B

Ai	Bi	Sum	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Ai	Bi	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

FULL ADDER: BOOLEAN FUNCTIONS

Sum

$$S_i = \overline{A}_i\overline{B}_iC_{i-1} + \overline{A}_iB_i\overline{C}_{i-1} + A_i\overline{B}_i\overline{C}_{i-1} + A_iB_iC_{i-1}$$

Carry

$$C_i = \overline{A}_iB_iC_{i-1} + A_i\overline{B}_iC_{i-1} + A_iB_i\overline{C}_{i-1} + A_iB_iC_{i-1}$$

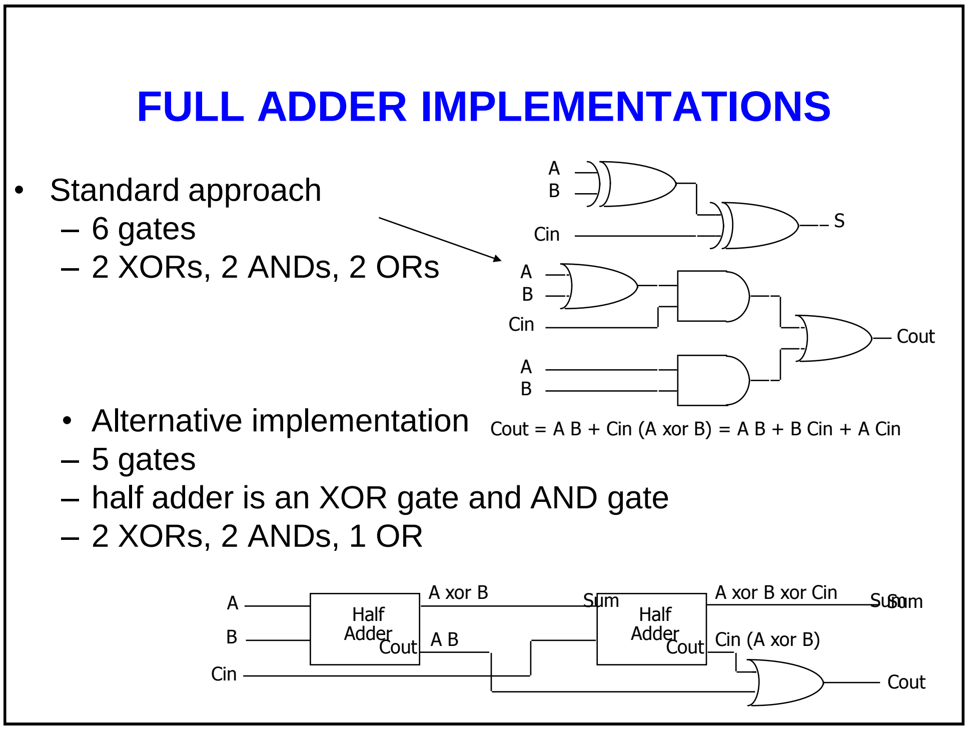
$$= A_iB_i + A_iC_{i-1} + B_iC_{i-1} = A_iB_i + (A_i + B_i)C_{i-1}$$

$$= A_iB_i + (A_i \oplus B_i)C_{i-1}$$

The sum can be expressed as a three-variable exclusive OR function ($S_i = A_i \oplus B_i \oplus C_i$).

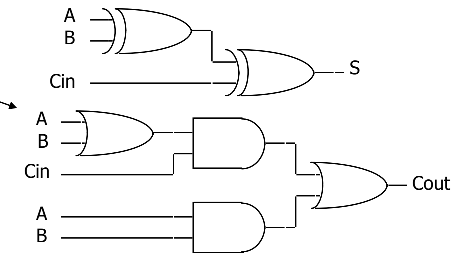
The carry is the three-variable majority function and can also be expressed in various other algebraic forms.

6



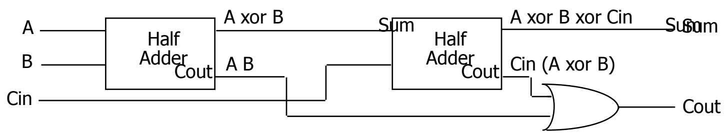
FULL ADDER IMPLEMENTATIONS

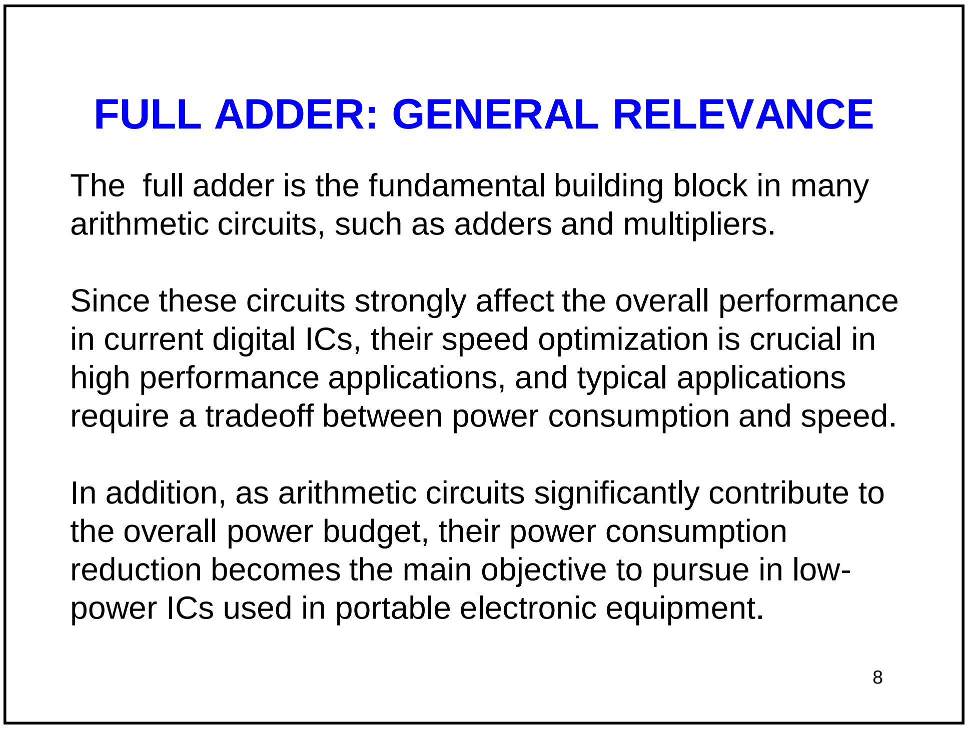
- Standard approach
  - 6 gates
  - 2 XORs, 2 ANDs, 2 ORs



Cout = A B + Cin (A xor B) = A B + B Cin + A Cin

- Alternative implementation
  - 5 gates
  - half adder is an XOR gate and AND gate
  - 2 XORs, 2 ANDs, 1 OR





FULL ADDER: GENERAL RELEVANCE

The full adder is the fundamental building block in many arithmetic circuits, such as adders and multipliers.

Since these circuits strongly affect the overall performance in current digital ICs, their speed optimization is crucial in high performance applications, and typical applications require a tradeoff between power consumption and speed.

In addition, as arithmetic circuits significantly contribute to the overall power budget, their power consumption reduction becomes the main objective to pursue in low-power ICs used in portable electronic equipment.

8

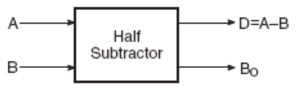
HALF- AND FULL SUBTRACTOR

The subtraction of two given binary numbers can be carried out by adding 2's complement of the subtrahend to the minuend. This allows us to do a subtraction operation with adder circuits.

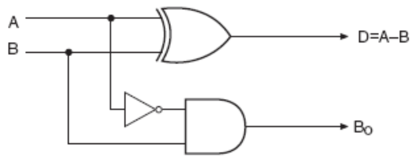
However, we will also briefly look at the counterparts of half-adder and full adder circuits in the *half-subtractor* and *full subtractor* for direct implementation of subtraction operations using logic gates.

HALF-SUBTRACTOR

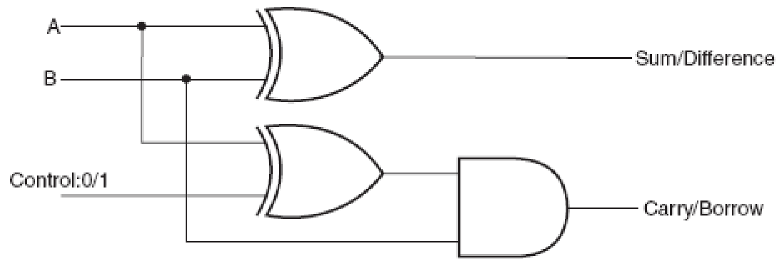
A *half-subtractor* is a combinational circuit that can be used to subtract one binary digit from another to produce a DIFFERENCE output and a BORROW output. The BORROW output here specifies whether a '1' has been borrowed to perform the subtraction.



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



COMBINED HALF ADDER/SUBTRACTOR

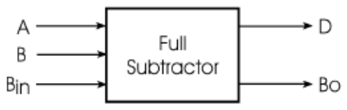


Control 0	ADD
Control 1	SUBTRACT

FULL SUBTRACTOR

A *full subtractor* performs subtraction operation on two bits, a *minuend* and a *subtrahend*, and also takes into consideration whether a ‘1’ has already been borrowed by the previous adjacent lower minuend bit or not. As a result, there are three bits to be handled at the input of a *full subtractor*, namely the two bits to be subtracted and a borrow bit designated as Bin . There are two outputs, namely the DIFFERENCE output D and the BORROW output Bo. The BORROW output bit tells whether the minuend bit needs to borrow a ‘1’ from the next possible higher minuend bit.

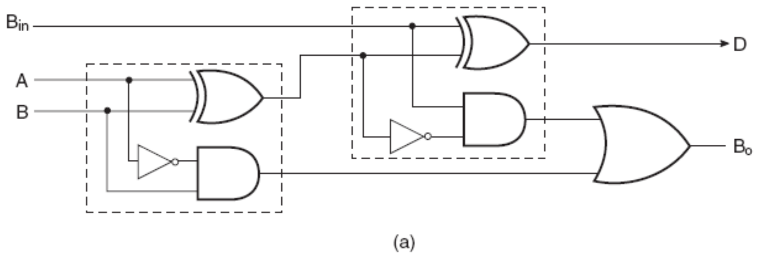
FULL SUBTRACTOR

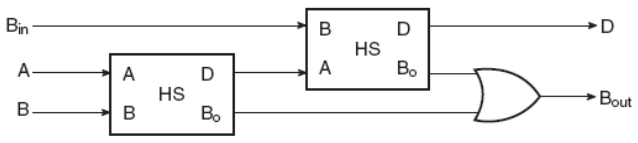


Minuend (A)	Subtrahend (B)	Borrow In (B_{in})	Difference (D)	Borrow Out (B_o)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Truth table of a full subtractor

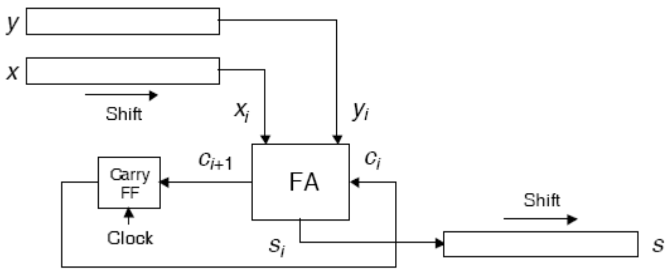
FULL SUBTRACTOR





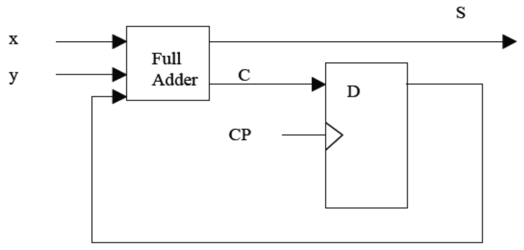
Logic implementation of a full subtractor with half-subtractors.

MULTIBIT ADDERS: BIT-SERIAL ADDER

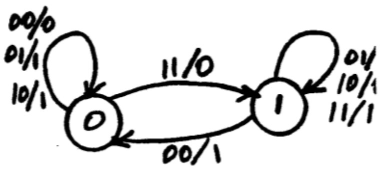


Functional diagram of the bit-serial adder.

OPERATION OF BIT-SERIAL ADDER

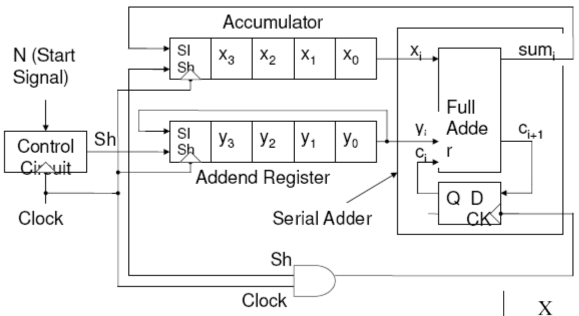


Present state	Inputs		Next State	output
Q	x	y	Q	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



$$S = x \oplus y \oplus Q$$
$$Q(t+1) = xy + xQ + yQ$$

SERIAL ADDER WITH ACCUMULATOR

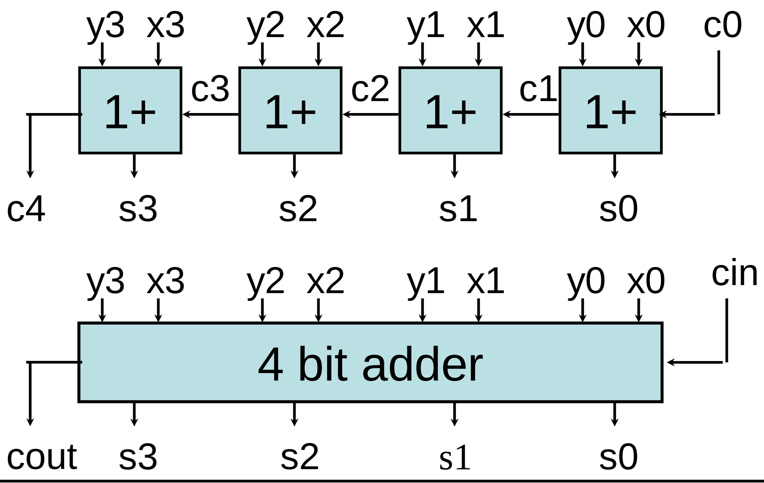


0 1 0 1
$\underline{0\ 1\ 1\ 1}$
1 1 0 0

	X	Y	c_i	sum_i	c_{i+1}
t_0	0101	0111	0	0	1
t_1	0010	1011	1	0	1
t_2	0001	1101	1	1	1
t_3	1000	1110	1	1	0
t_4	1100	0111	0	(1)	(0)

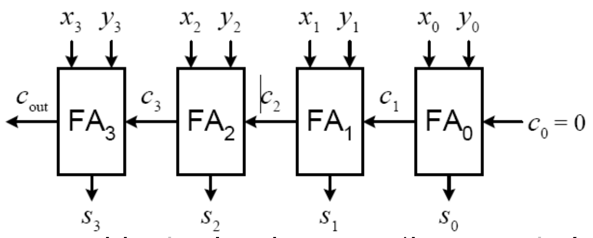
4-BIT PARALLEL ADDER (SERIES CARRY PROPAGATION, RIPPLE CARRY)

Carry is propagated serially!



RIPPLE CARRY ADDER

The full adder is for adding two operands that are only one bit wide. To add two operands that are, say four bits wide, we connect four full adders together in series. The resulting circuit is called a ripple carry adder for adding two 4-bit operands.



The ripple-carry adder is slow because the carry-in for each full adder is dependent on the carry-out signal from the previous FA. So before FA_i can output valid data, it must wait for FA_{i-1} to have valid data.

SUBTRACTION: 2S COMPLEMENT

5	0101
+2	+0010
7	0111

addition

$\rightarrow$

2	0010
-2	1101 1s complement
-2	1110 2s complement

$\downarrow$

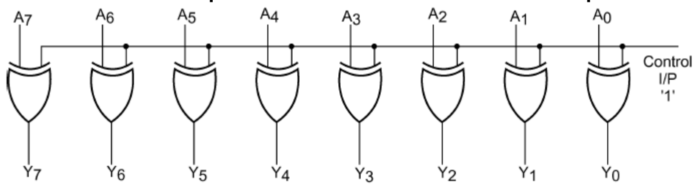
5	0101
-2	+1110
3	1 0011

subtraction

20

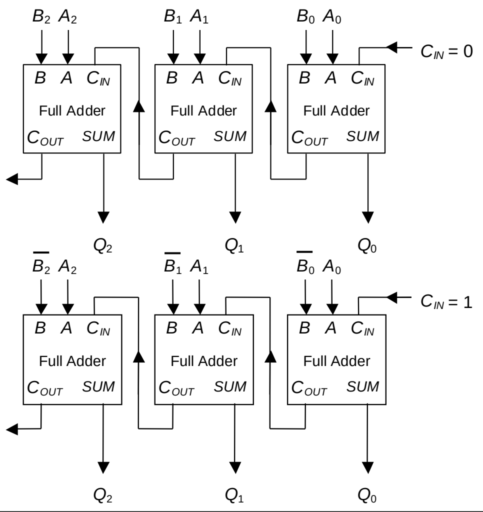
CONTROLLED INVERTER

A *controlled inverter* is needed when an adder is to be used as a subtractor. Subtraction is addition of the 2’s complement of the subtrahend to the minuend. Thus, the first step towards implementation of a subtractor is to determine the 2’s complement of the subtrahend. And for this, one needs firstly to find 1’s complement. A controlled inverter is used to find 1’s complement. A one-bit controlled inverter is a two-input EX-OR gate with one of its inputs treated as a control input.



Eight-bit controlled inverter

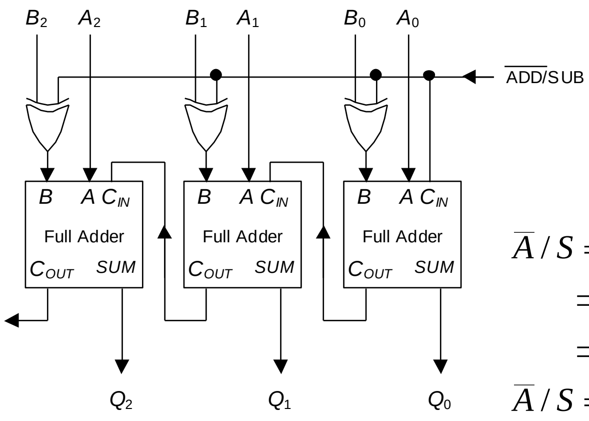
ADDITION AND SUBTRACTION



Addition

Subtraction

ADD/SUBTRACT CIRCUIT



$\overline{\text{ADD}}$/SUB	$B_{IN(n)}$	B_n
0	0	0
0	1	1
1	0	1
1	1	0

$$\bar{A}/S = 0$$
$$\Rightarrow B_{in} \rightarrow B \;\; \& \;\; C_{IN} = 0$$
$$\Rightarrow Q = A + B$$
$$\bar{A}/S = 1$$
$$\Rightarrow B_{in} \rightarrow \bar{B} \;\; \& \;\; C_{IN} = 1$$
$$\Rightarrow Q = A - B$$

XOR gates: controlled inverters

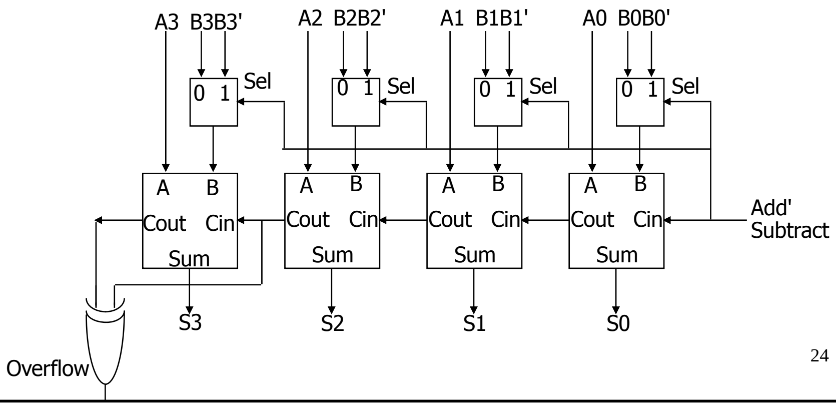
23

4-BIT ADDER/SUBTRACTER

Use an adder to do subtraction thanks to 2s complement representation

A – B = A + (– B) = A + B' + 1

Control signal selects B or 2s complement of B



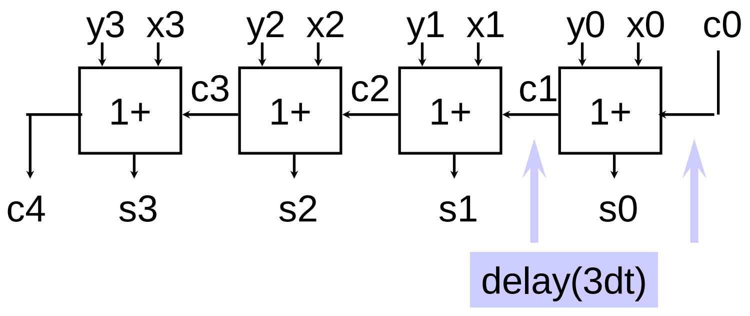
24

RIPPLE CARRY ADDER

The layout of a ripple carry adder is simple, which allows for fast design time, however, the ripple carry adder is relatively slow, since each full adder must wait for the carry bit from the previous full adder.

From C_{in} to C_{out} 2 gates should be passed through. Ergo a 32-bit adder requires 31 carry computations and the final sum calculation for a total of 31x2 + 1 = 63 gate delays.

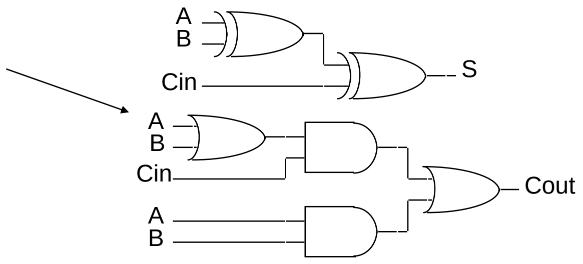
PROPAGATION DELAY OF THE RIPPLE CARRY ADDER



26

DELAY IN THE 1-BIT FULL ADDER

Standard layout
- 6 gates
- 2 XOR, 2 AND, 2 OR



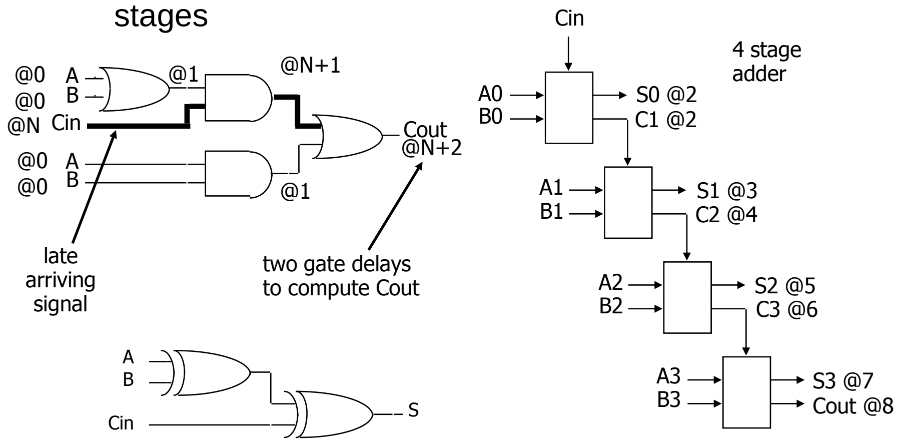
Cout = A B + Cin (A xor B) = A B + B Cin + A Cin

If A, B and Cin arrive simultaneously, the sum S will be available after a delay of 2Δt, the carry out Cout after a delay of 3Δt!

The delays with respect to the arrival of C_{in} are Δt and 2Δt respectively!

RIPPLE-CARRY ADDERS: SERIAL CARRY PROPAGATION

- Critical Delay
  - The propagation of carry from low to high order stages

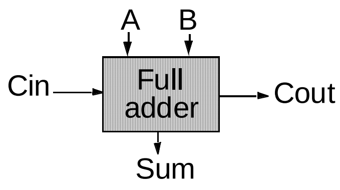


CARRY LOOK-AHEAD ADDER

Carry look-ahead adders reduce the computation time. They work creating propagate and generate signals (P and G) for each bit position, and using them the carries for each position are created.

Some multi-bit adder architectures break the adder into blocks. It is possible to vary the length of these blocks based on the propagation delay of the circuits to optimize computation time. These block based adders include the carry bypass adder which will determine P and G for each block rather than each bit, and the carry select adder which pre-generates sum and carry values for either possible carry input to the block.

FULL ADDER: CARRY



$C_o = A\,B + (A \oplus B)C_i$

vagy

$C_o = A\,B + (A + B)C_i$

$C_o = G + P\,C_i$

A	B	C_i	S	C_o	*Carry status*
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate

CARRY-LOOKAHEAD LOGIC

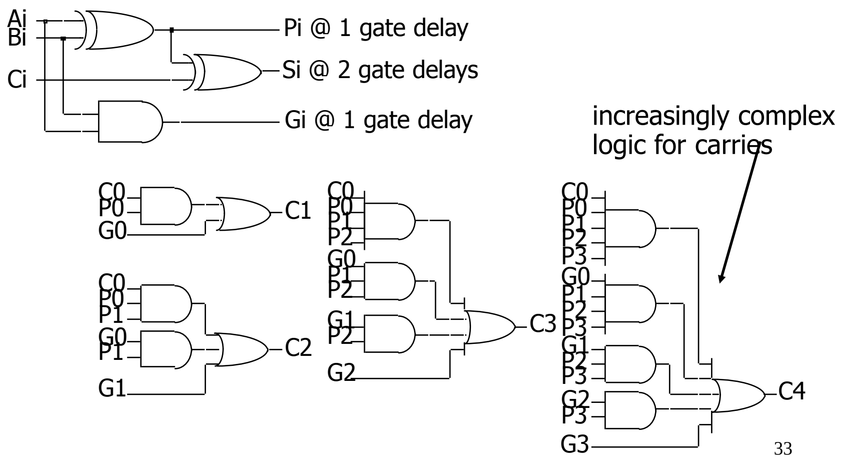
- Carry generate: Gi = Ai Bi
  - Must generate carry when A = B = 1
- Carry propagate: Pi = Ai xor Bi
  - Carry-in will equal carry-out here
- Sum and Cout can be re-expressed in terms of generate/propagate:
  - Si = Ai xor Bi xor Ci
    = Pi xor Ci
  - Ci+1= Ai Bi + Ai Ci + Bi Ci
    = Ai Bi + Ci (Ai + Bi)
    = Ai Bi + Ci (Ai xor Bi)
    = Gi + Ci Pi

CARRY-LOOKAHEAD LOGIC

- Re-express the carry logic as follows:
  - C1 = G0 + P0 C0
  - C2 = G1 + P1 C1 = G1 + P1 G0 + P1 P0 C0
  - C3 = G2 + P2 C2 = G2 + P2 G1 + P2 P1 G0 + P2 P1 P0 C0
  - C4 = G3 + P3 C3 = G3 + P3 G2 + P3 P2 G1 + P3 P2 P1 G0
    + P3 P2 P1 P0 C0
- Each of the carry equations can be implemented with two-level logic
  - All inputs are now directly derived from data inputs and not from intermediate carries
  - this allows computation of all sum outputs to proceed in parallel

CARRY LOOK AHEAD IMPLEMENTATION

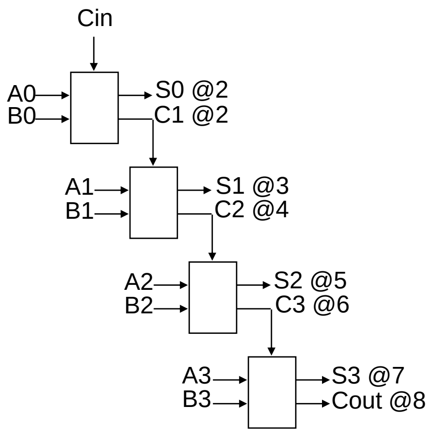
Adder with propagate and generate outputs

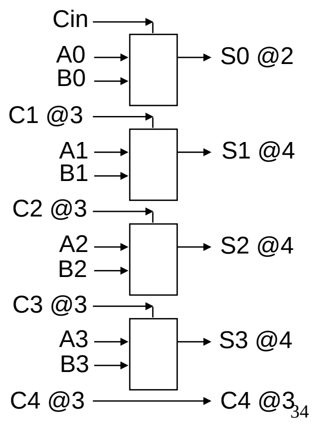


33

CARRY LOOK AHEAD IMPLEMENTATION

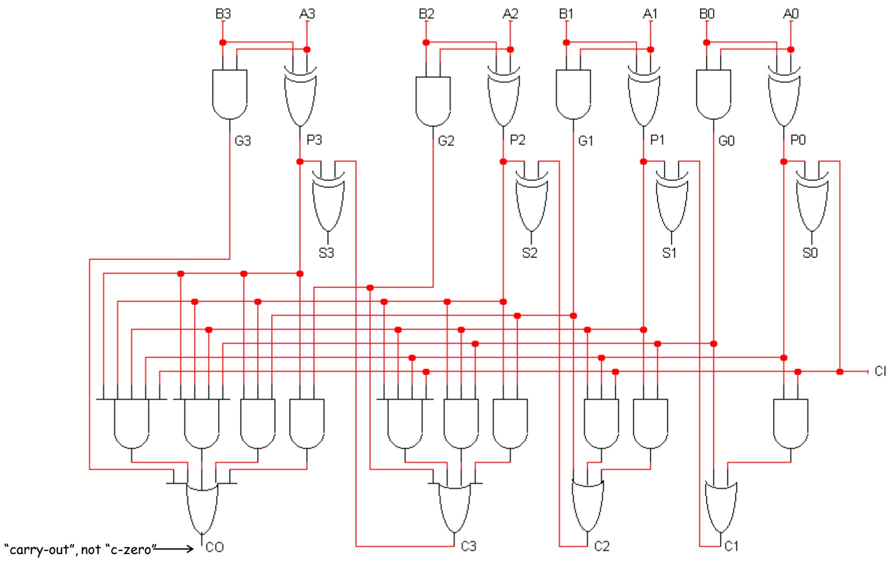
- Carry-lookahead logic generates individual carries
  - Sums computed much more quickly in parallel
  - However, cost of carry logic increases with more stages





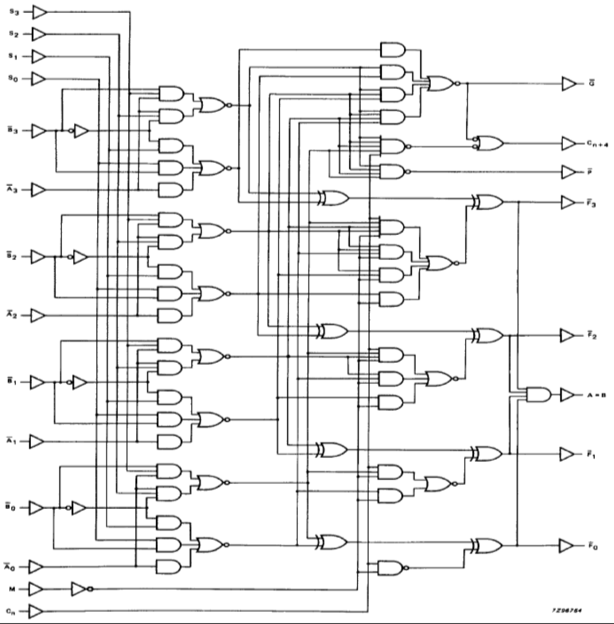
34

4-BIT CARRY LOOKAHEAD ADDER CIRCUIT



Total 26 gates, c.f. 4 standard full adders 4x6 = 24 gates

74HC/HCT181 4-BIT ALU LOGIC DIAGRAM



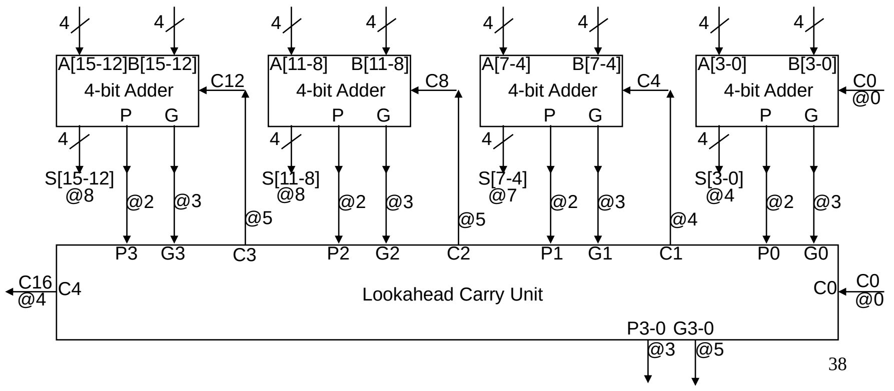
CARRY LOOKAHEAD ADDERS

- By adding more hardware, we reduced the number of levels in the circuit and sped things up.
- We can “cascade” carry lookahead adders, just like ripple carry adders. (We’d have to do carry lookahead *between* the adders too.)
- How much faster is this?
  - For a 4-bit adder, not much. There are 4 gates in the longest path of a carry lookahead adder, versus 9 gates for a ripple carry adder.
  - But if we do the cascading properly, a 16-bit carry lookahead adder could have only 8 gates in the longest path, as opposed to 33 for a ripple carry adder.
  - Newer CPUs these days use 64-bit adders. That’s 12 vs. 129 gates!
- The delay of a carry lookahead adder grows *logarithmically* with the size of the adder, while a ripple carry adder’s delay grows *linearly*.

- The thing to remember about this is the trade-off between complexity and performance. Ripple carry adders are simpler, but slower. Carry lookahead adders are faster but more complex.

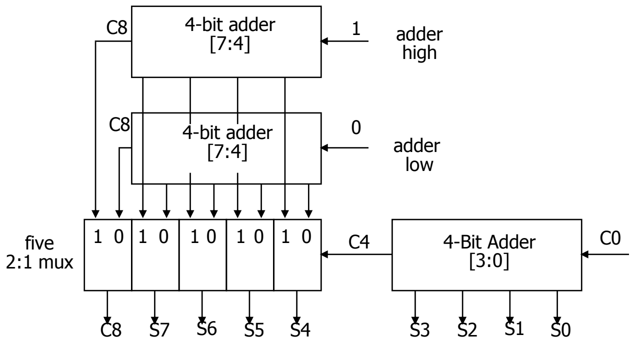
Carry-Lookahead Adder with Cascaded Carry-Lookahead Logic

- Carry-lookahead adder4 four-bit adders with internal carry lookahead
- Second level carry lookahead unit extends lookahead to 16 bits



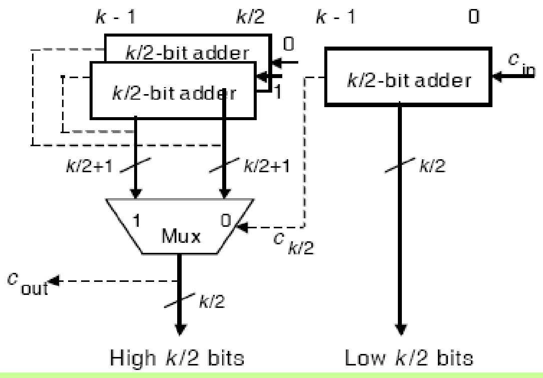
38

CARRY-SELECT ADDER



Redundant hardware to make carry calculation go faster
Compute two high-order sums in parallel while waiting for carry-in
One assuming carry-in is 0 and another assuming carry-in is 1
Select correct result once carry-in is finally comp

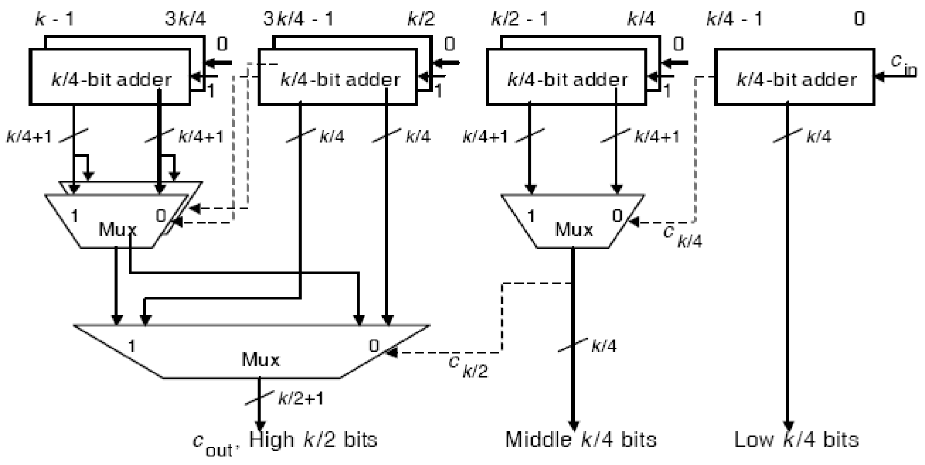
CARRY-SELECT ADDERS



Carry-select adder for *k*-bit numbers built from three *k*/2-bit adders.

$$C_{\text{select-add}}(k) = 3C_{\text{add}}(k/2) + k/2 + 1$$
$$T_{\text{select-add}}(k) = T_{\text{add}}(k/2) + 1$$

MULTILEVEL CARRY-SELECT



Two-level carry-select adder built of *k*/4-bit adders.

ARITHMETICAL OPERATIONS IN BCD

Many digital systems (processors, computers) can perform the arithmetical operations or a part of them directly on BCD numbers.

E.g. the microprocessors can perform BCD addition, several of them subtraction too. Certain special processors can perform BCD multiplication and division too.

The BCD addition is reduced to binary addition. The tetrades of the operands are added as binary numbers, and if necessary (illegal codewords or decimal carry is generated during the addition), a systematic correction is performed.

BCD ADDITION

A BCD adder is used to perform the addition of BCD numbers. A BCD digit can have any of the ten possible four-bit binary representations, that is, 0000, 0001, , 1001, the equivalent of decimal numbers 0, 1, … , 9.

When we set out to add two BCD digits and we assume that there is an input carry too, the highest binary number that we can get is the equivalent of decimal number 19 (9+9+1). This binary number is going to be $(10011)_{bin}$. On the other hand, if we do BCD addition, we would expect the answer to be $(0001\ 1001)_{BCD}$. And if we restrict the output bits to the minimum required, the answer in BCD would be $(1\ 1001)_{BCD}$.

ADDITION IN NORMAL BCD (8421) CODE

If the sum of two tetrades is not larger than 9, the result is valid, no correction is necessary.

If the sum of two tetrades is larger than 9, (decimal carry and illegal codeword or pseudotetrade is generated) the result is valid only in binary system and not in BCD. The necessary correction is to add decimal 6 or i.e. binary 0110 to the actual tetrade.

The correction should be performed beginning form the least significant tetrade and going upwards step-by-step.

ALGORITHM OF BCD (8421) ADDITION

$A_{BCD} +_{BCD} B_{BCD} = A_{BCD} +_{bin} B_{BCD}$

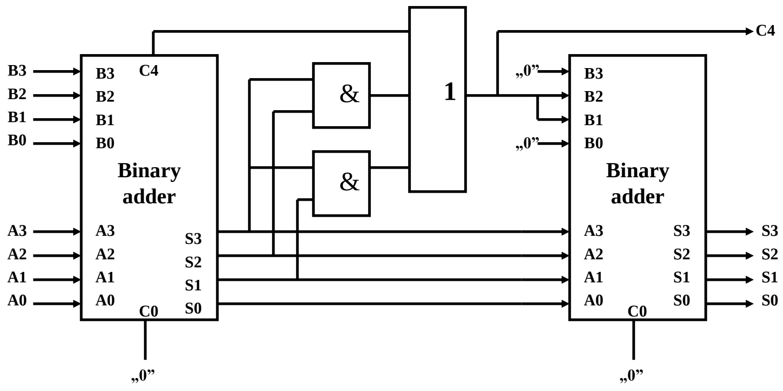
if $A_{BCD} +_{bin} B_{BCD} \leq 9$

$A_{BCD} +_{BCD} B_{BCD} = A_{BCD} +_{bin} B_{BCD} +_{bin} 6_{BCD}$

if $A_{BCD} +_{bin} B_{BCD} > 9$

45

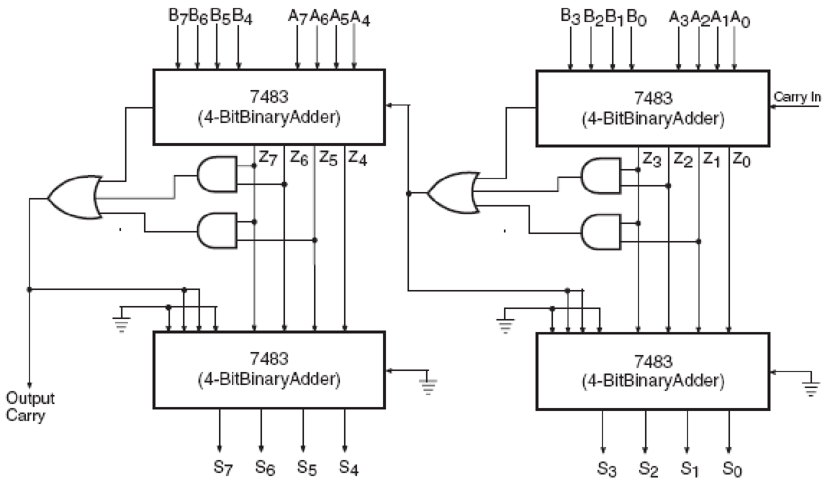
FUNCTIONAL DIAGRAM OF A BCD ADDER (1 DIGIT)



The first adder adds the two codes corresponding to the k-th decimal place, the second adds 6 if necessary.

46

APPLICATION EXAMPLE: 2-DIGIT BCD ADDER



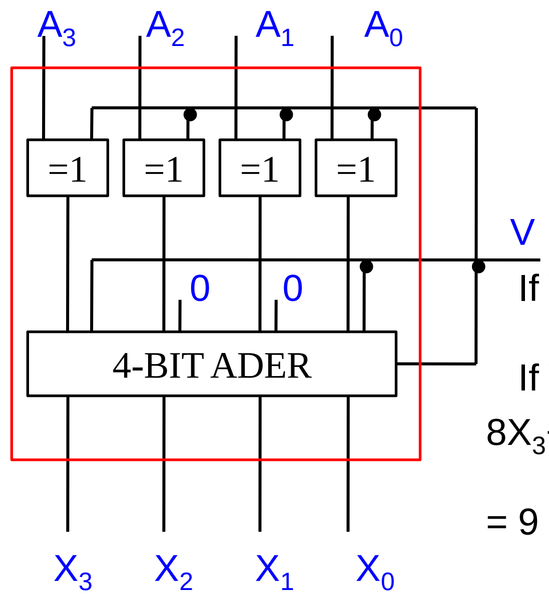
SUBTRACTION IN BCD (8421) CODE

In BCD as in binary, the subtraction is performed by complementing (the subtrahend) and addition. Generally 9's complement is used.

The circuit generating the 9's complement can be constructed from common gates or form more complex functional elements.

48

GENERATING 9'S COMPLEMENT IN BCD



If V = 0 then $X_k = A_k$

If V = 1 then

$8X_3+4X_2+2X_1+X_0 =$

$= 9 – (8A_3+4A_2+2A_1+A_0)$

MULTIPLIERS

A binary multiplier is an electronic circuit used in digital electronics, such as a computer, to multiply two binary numbers.

A variety of computer arithmetic techniques can be used to implement a digital multiplier. Most techniques involve computing a set of partial products, and then summing the partial products together. This process is similar to the method taught to primary school children for conducting long multiplication on base-10 integers, but has been modified here for application to a base-2 (binary) numeral system.

The first stage of most multipliers involves generating the partial products which is nothing but an array of AND gates. An n-bit by n-nit multiplier requires n^2 AND gates for partial product generation.
The partial products are then added to give the final results.

COMBINATIONAL MULTIPLIER

Partial Product Accumulation

				A3	A2	A1	A0
				B3	B2	B1	B0
				A3 B0	A2 B0	A1 B0	A0 B0
			A3 B1	A2 B1	A1 B1	A0 B1	
		A3 B2	A2 B2	A1 B2	A0 B2		
	A3 B3	A2 B3	A1 B3	A0 B3			
S7	S6	S5	S4	S3	S2	S1	S0

BINARY MULTIPLICATION ALGORITHM

$P = A \times B$

$$A = \sum_{i=0}^{n-1} A_i\, 2^i \quad \text{and} \quad B = \sum_{i=0}^{n-1} B_i\, 2^i$$

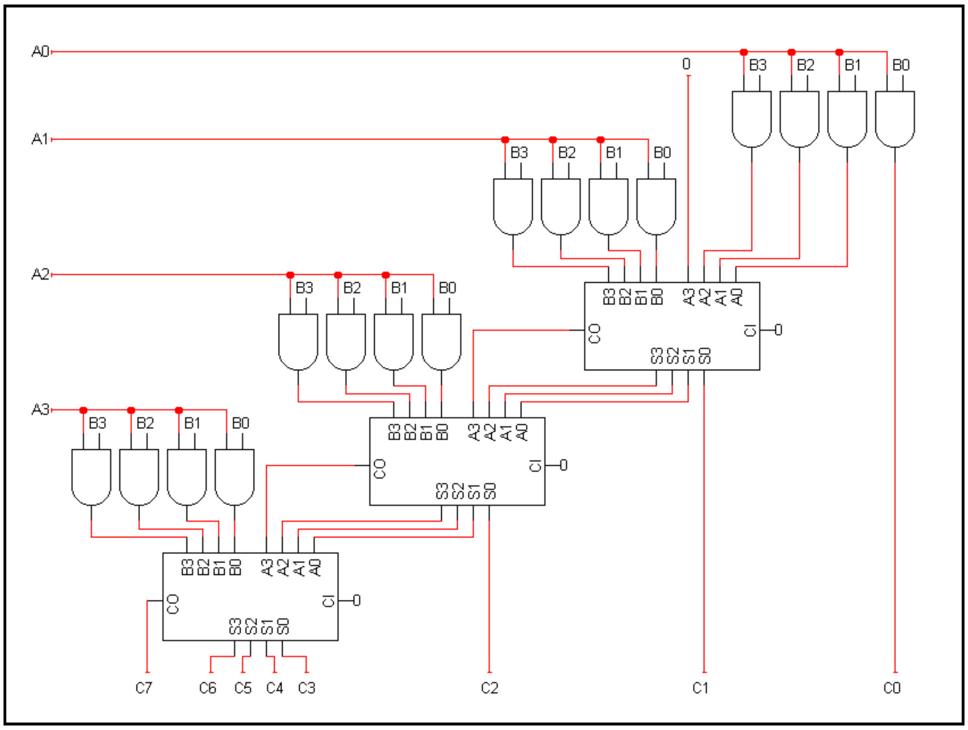
Partial products

$$P_k = B_k \sum_{k=0}^{n-1} A_i\, 2^i = 0 \text{ if } B_k = 0 \text{ and } = A \text{ if } B_k = 1$$

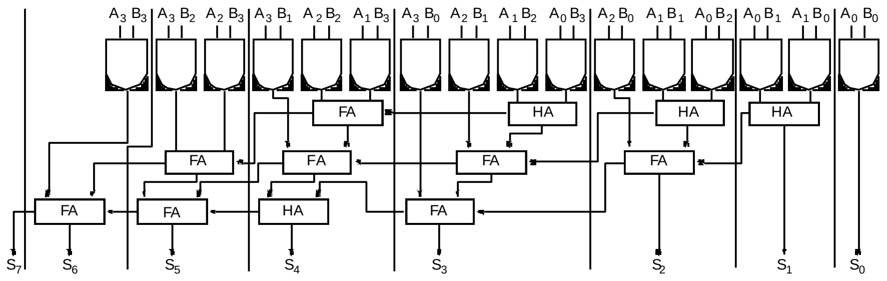
Complete product

$$P = \sum_{k=0}^{n-1} P_k\, 2^k$$

52



SUMMING UP OF PARTIAL PRODUCTS



Note use of parallel carry-outs to form higher order sums

12 Adders, if full adders, this is 6 gates each = 72 gates

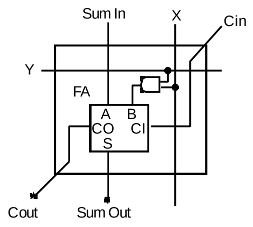
16 gates form the partial products

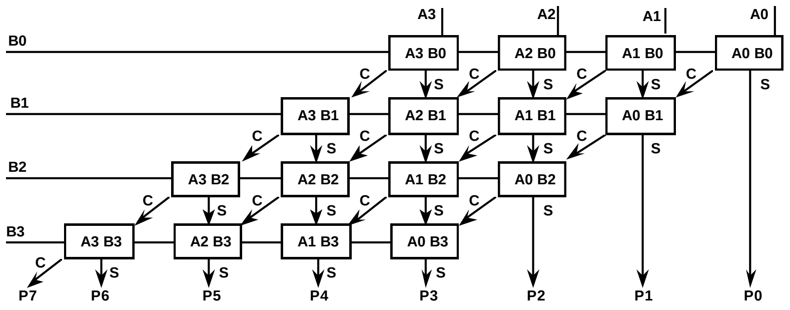
total = 88 gates!

COMBINATIONAL MULTIPLIER

Another Representation of the Circuit

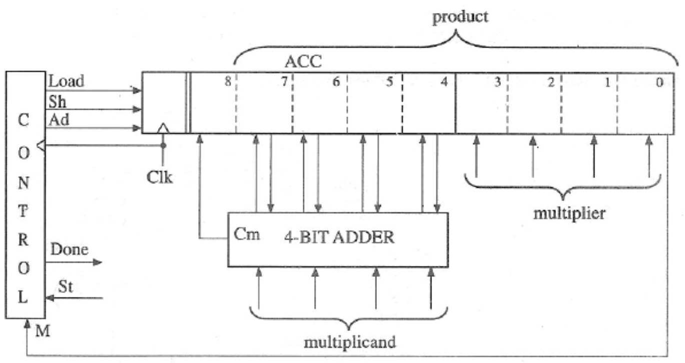
Building block: FULL ADDER + AND





4 x 4 array of building blocks

4x4 BIT SERIAL/PARALLEL MULTIPLIER



Block diagram of a 4x4 bit serial/parallel multiplier

IF multiplier bit 1 THEN add and shift

IF multipler bit 0 THEN shift

OPERATION OF THE MULTIPLIER

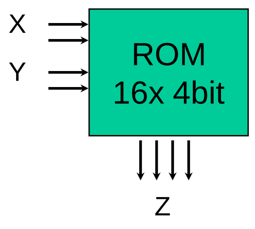
Multiplicand 1 1 0 1 Mutiplier 0 1 0 1

8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	1	0	1	
0	1	1	0	1	0	1	0	1	ADD
0	0	1	1	0	1	0	1	0	SHIFT
0	0	0	1	1	0	1	0	1	SHIFT
1	0	0	0	0	0	1	0	1	ADD
0	1	0	0	0	0	0	1	0	SHIFT
0	0	1	0	0	0	0	0	1	SHIFT

13 x 5 = 65

MULTIPLICATION USING ROM (LOOK-UP-TABLE)

	X	Y	Z
0*0	00	00	0000
0*1	00	01	0000
0*2	00	10	0000
0*3	00	11	0000
1*1	01	01	0001
1*2	01	10	0010
1*3	01	11	0011
2*0	10	00	0000
2*1	10	01	0010
2*2	10	10	0100
2*3	10	11	0110
3*0	11	00	0000
3*1	11	01	0011
3*2	11	10	0110
3*3	11	11	1001

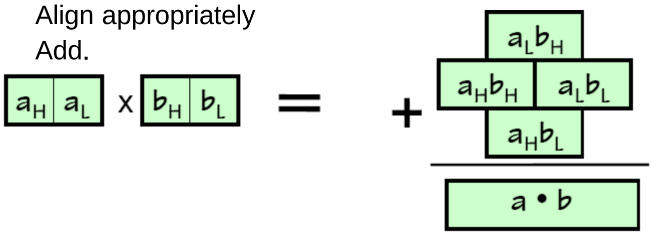


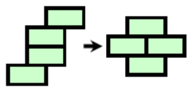
58

MAKING A 2n-BIT MULTIPLIER USING n-BIT MULTIPLIERS

2n-bit by 2n-bit multiplication:

1. Divide multiplicands into n-bit pieces
2. Form 2n-bit partial products, using n-bit by n-bit multipliers.
3. Align appropriately
4. Add.

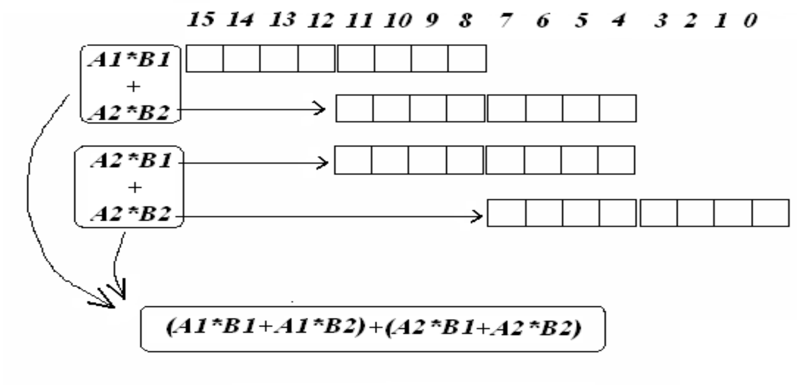




REGROUP partial products – 2 additions rather than 3!

Induction: we can use the same structuring principle to build a 4n-bit multiplier from our newly-constructed 2n-bit ones...

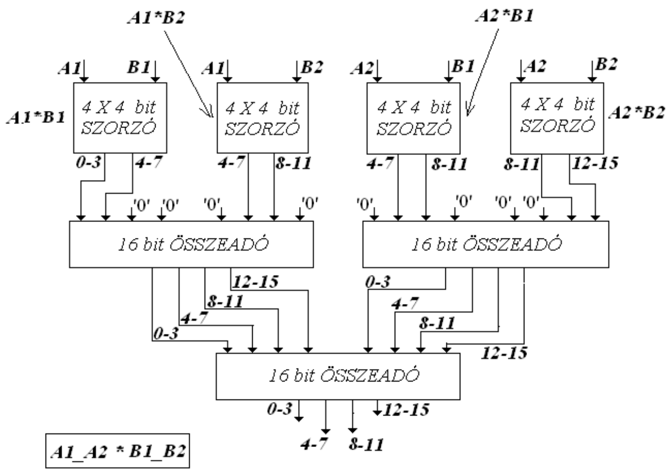
MODULAR MULTIPLIER ARCHITECTURE



8 x 8 bit multiplier built from 4 x 4 bit modules
Product MSB : 0, LSB: 15)

60

MODULAR MULTIPLIER



8 x 8 bit multiplier built from 4 x 4 bit modules
Product MSB : 0, LSB: 15)

61

MULTIPLICATION: NEGATIVE NUMBERS

The basic school method of multiplication handles the sign with a separate rule ("+ with + yields +", "+ with - yields -", etc.). Modern computers embed the sign of the number in the number itself, usually in the two’s complement representation. That forces the multiplication process to be adapted to handle two's complement numbers, and that complicates the process a bit more. Similarly, processors that use one’s complement sign-and-magnitude, IEEE-754 or other binary representations require specific adjustments to the multiplication process.

MULTIPLICATION: SPEEDING IT UP

Older multiplier architectures employed a shifter and accumulator to sum each partial product, often one partial product per cycle, trading off speed for die area.

Modern multiplier architectures use the *Baugh-Wooley algorithm*, *Wallace tree* or *Dadda* to add the partial products together in a single cycle. The performance of the *Wallace tree* implementation is sometimes improved by modified *Booth encoding* one of the two multiplicands, which reduces the number of partial products that must be summed.

FULL ADDER IMPLEMENTED IN CMOS

The simplest forms of the sum and carry function are (written in a form appropriate to CMOS implementation)

$$S = \overline{C}(A\,\overline{B} + \overline{A}\,B) + C(A\,B + \overline{A}\,\overline{B})$$

$$C_{out} = A\,B + C(A + B)$$

This is easily implemented using standard CMOS principles. The total transistor count is 34.

The disadvantage is that the circuit uses the negated values of the inputs too. So three extra inverters, i.e. 6 transisotors are needed additionally.

FULL ADDER IMPLEMENTED IN CMOS

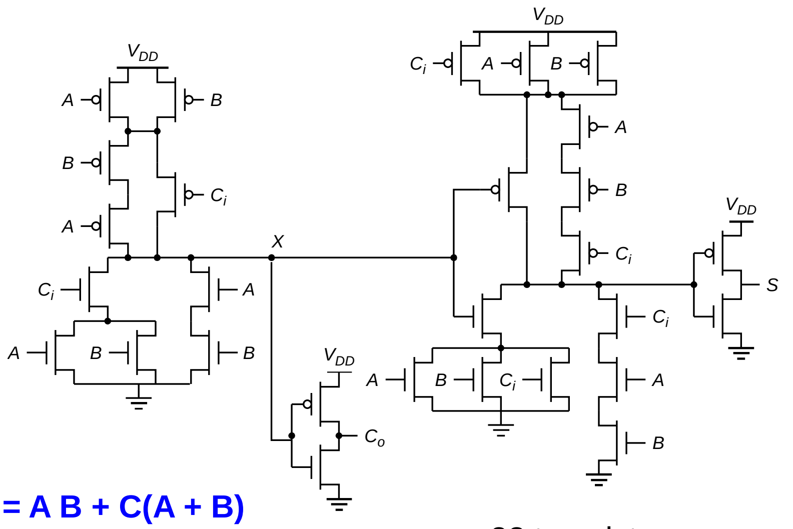
This disadvantage can be avoided, if the negated value of the generated carry C_{out} is used to calculate the sum according to

$$C_{out} = A\,B + C(A + B)$$

$$S = (A + B + C)\overline{C_{out}} + A\,B\,C$$

In this case the time delay of the sum will be larger, because three inverting operation is performed, but this is not relevant in a parallel (ripple-carry) adder, because the time necessary for a multi-bit addition is determined by the propagation time of the carry.

28 TRANSISTOR CMOS FULL ADDER



$$C_{out} = A\,B + C(A + B)$$

28 transistors

$$S = (A + B + C)\overline{C_{out}} + A\,B\,C$$

MULTIPLIERS: COMPLEXITY

Transistor count for generic multiplier circuits is based on static CMOS implementation

8-bit	3000
16-bit	9000
32-bit	21000

i.e. in the LSI range.

REVISION QUESTIONS

1. What is a half-adder? Write its truth table.

2. What is a full-adder? Draw its logic diagram with basic gates.

3. Briefly describe the concept of look-ahead carry generation with respect to its use in adder circuits. What is its significance while implementing hardware for addition of binary numbers of longer lengths?

4.Draw the logic diagram of a three-digit BCD adder and briefly describe its functional principle.

68

REVISION QUESTIONS

6. Explain the operation of the carry-select adder.

7. Explain how division and multiplication can be performed in digital systems.

8. Explain the working of the serial adder.

PROBLEMS AND EXERCISES

1. Implement a full-adder circuit using NAND gates only.

2. Implement a full-adder circuit using NOR gates only.

3. Draw the smallest possible complete circuit for a 2-bit carry-lookahead adder.

4. Design an eight-bit adder–subtractor circuit using four-bit binary adders, type number 7483, and quad two-input XOR gates, type number 7486. Assume that pin connection diagrams of these ICs are available to you.