DIGITAL TECHNICS

Dr. Bálint Pődör

Óbuda University,
Microelectronics and Technology Institute

4. LECTURE: COMBINATIONAL LOGIC DESIGN: ARITHMETICS (THROUGH EXAMPLES)

2nd (Autumn) term 2018/2019

COMBINATIONAL LOGIC DESIGN: EXAMPLES AND CASE STUDIES

- Arithmetic circuits
  - General aspects, and elementary circuits
  - Addition/subtraction
  - BCD arithmetics
  - Multipliers
  - Division

ARITHMETIC CIRCUITS

- Excellent Examples of Combinational Logic Design
- Time vs. Space Trade-offs
  - Doing things fast may require more logic and thus more space
  - Example: carry lookahead logic
- Arithmetic and Logic Units
  - General-purpose building blocks
  - Critical components of processor datapaths
  - Used within most computer instructions

ARITHMETIC CIRCUITS: BASIC BUILDING BLOCKS

Below I 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*.

HALF-ADDER AND FULL-ADDER

Half-adder
This circuit needs 2 binary inputs and 2 binary outputs.
The input variables designate the augend and addend bits: the output variables produce the sum and carry.

Full-adder
Is a combinational circuit that forms the arithmetic sum of 3 bits.
Consists of 3 inputs and 2 outputs.
When all input bits are 0, the output is 0.
The output S equal to 1 when only one input is equal to 1 or when all 3 inputs are equal to 1.
The C output has a carry of 1 if 2 or 3 inputs are equal to 1.

CIRCUITS FOR BINARY ADDITION

- 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

ON THE IMPLEMENTATION OF THE FULL ADDER

A possible technique for implementing the 1-bit full adder is to generate the two relevant logic function directly

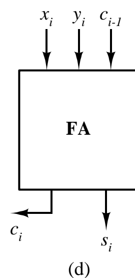
$$S_i = \overline{A_i}\,\overline{B_i}C_{i-1} + \overline{A_i}B_i\overline{C_{i-1}} + A_i\overline{B_i}\,\overline{C_{i-1}} + A_iB_iC_{i-1}$$

$$C_i = A_iB_i + A_iC_{i-1} + B_iC_{i-1}$$

Both require a two level AND-OR circuitry, therefore the time required to achieve the sum and the carry is equal to the propagation delay of two gates, or 2 t_{pd}.

An other possibility is to generat the sum using XOR gates.

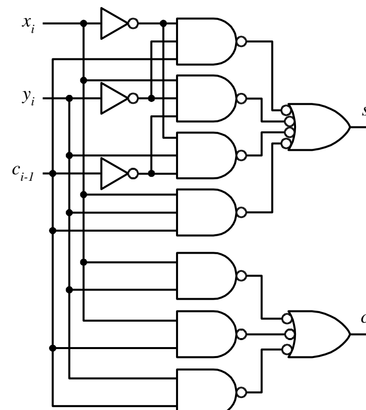
1-BIT FULL ADDER IMPLEMENTATIONS



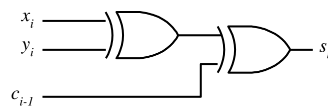
(d)

x_i	y_i	c_{i-1}	c_i	s_i
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

(e)



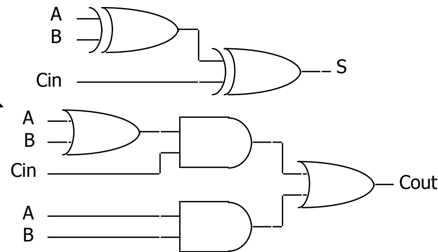
(f)



(g)

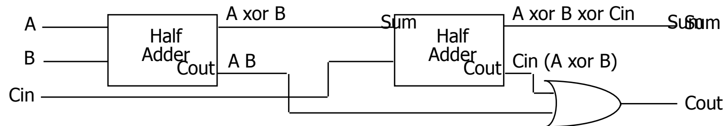
FULL ADDER IMPLEMENTATIONS

- 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 IMPLEMENTED IN CMOS

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

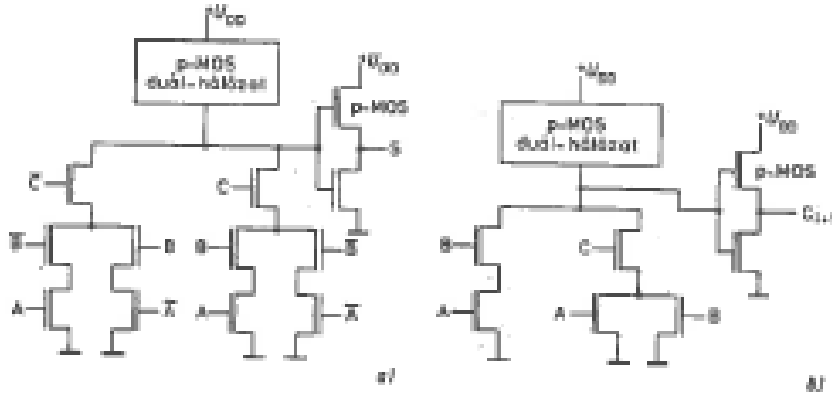
$$S = \overline{C}(A\,\overline{B} + \overline{A}\,B) + C(A\,B + \overline{A}\,\overline{B})$$

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

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 input variables too.

FULL ADDER IMPLEMENTED IN CMOS



Static CMOS adder, a. sum, b. carry circuit

11

FULL ADDER IMPLEMENTED IN CMOS

The sum and carry functions can be rearranged:

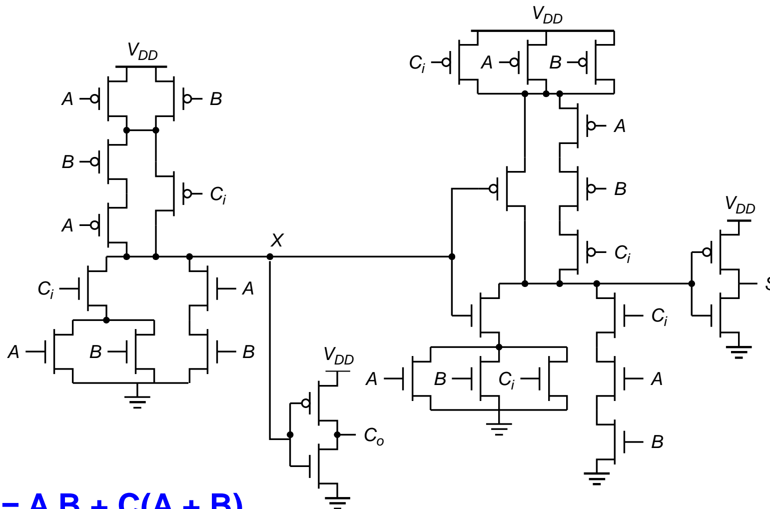
$$C_{out} = A\,B + C(A + B)$$

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

The advantage gained is that the complemented opearands are not needed.
This is also easily implemented using standard CMOS principles. The total transistor count is only 28.

The disadvantage is that the circuit has three levels.

28 TRANSISTOR CMOS FULL ADDER



C_{out} = A B + C(A + B)

28 transistors

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

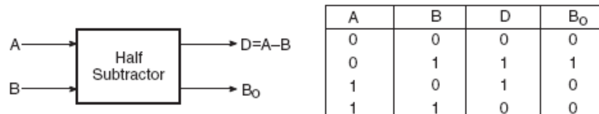
HALF- AND FULL SUBTRACTOR

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

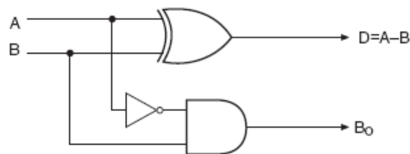
However, we will also briefly look at the counterparts of half-adder and full adder circuits in the *half-subtractor* and *full subtractor* for direct implementation of subtraction operations using logic gates.

HALF-SUBTRACTOR

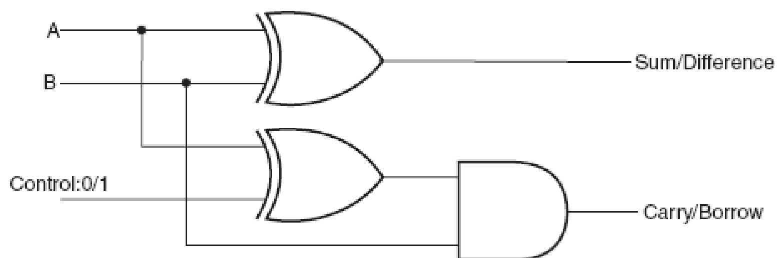
A *half-subtractor* is a combinational circuit that can be used to subtract one binary digit from another to produce a DIFFERENCE output and a BORROW output. The BORROW output here specifies whether a ‘1’ has been borrowed to perform the subtraction.



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

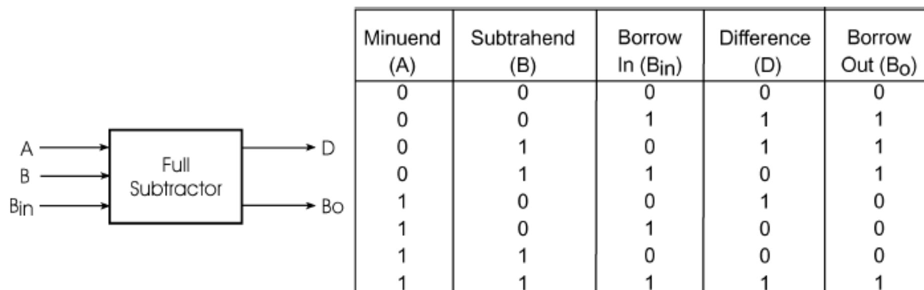


Control 0 ADD
Control 1 SUBTRACT

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.

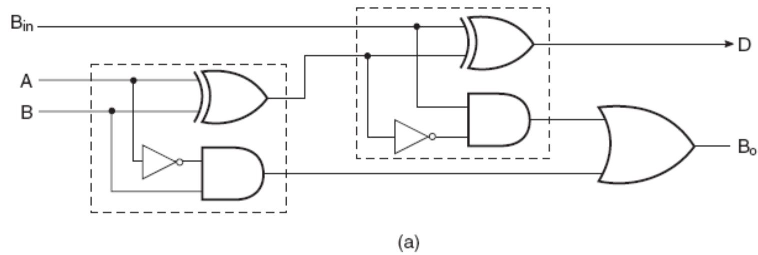
FULL SUBTRACTOR



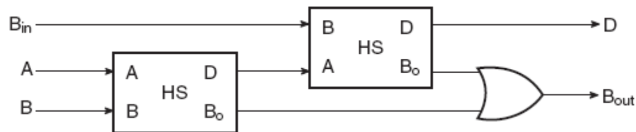
Minuend (A)	Subtrahend (B)	Borrow In (B_{in})	Difference (D)	Borrow Out (B_o)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Truth table of a full subtractor

FULL SUBTRACTOR



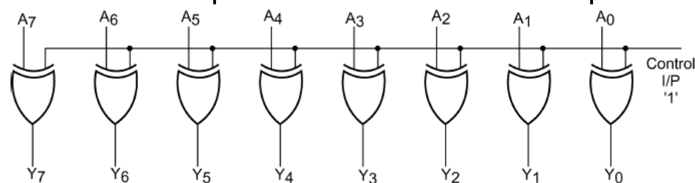
(a)



Logic implementation of a full subtractor with half-subtractors.

CONTROLLED INVERTER

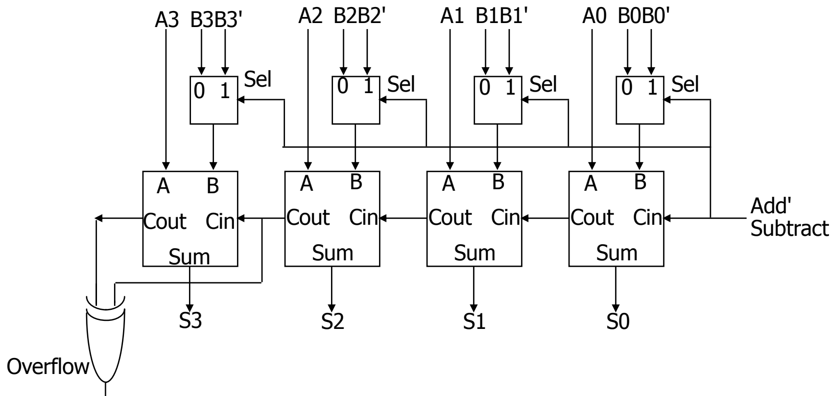
A *controlled inverter* is needed when an adder is to be used as a subtractor. Subtraction is addition of the 2's complement of the subtrahend to the minuend. Thus, the first step towards implementation of a subtractor is to determine the 2's complement of the subtrahend. And for this, one needs firstly 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.



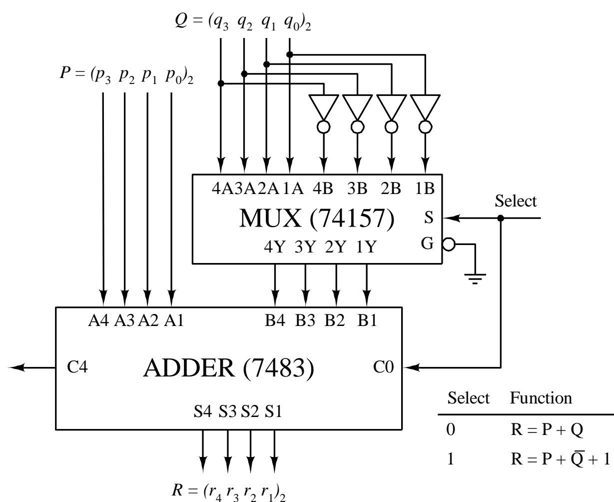
Eight-bit controlled inverter

ADDER/SUBTRACTOR

- 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



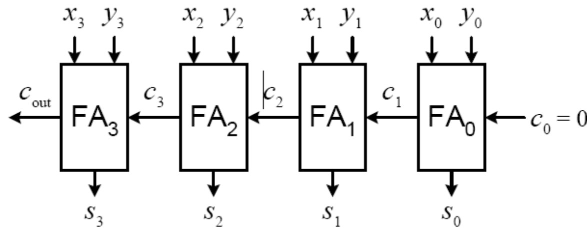
TWO'S COMPLEMENT ADDER/SUBTRACTOR



Select	Function
0	$R = P + Q$
1	$R = P + \overline{Q} + 1$

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.

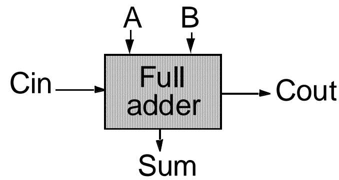
CARRY-LOOKAHEAD ADDER

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.

In the *carry-lookahead* adder, each bit slice eliminates this dependency on the previous carry-out signal and instead uses the values of the two input operands, directly to deduce the needed signals. This is possible from the following observations regarding the carry-out signal.

FULL ADDER: GENERATION AND PROPAGATION OF CARRY



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

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

or

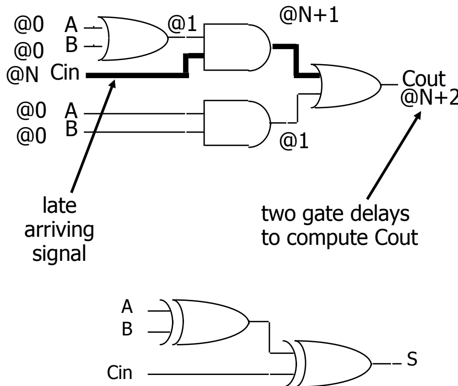
$C_o = A\,B + (A + B)C_i$

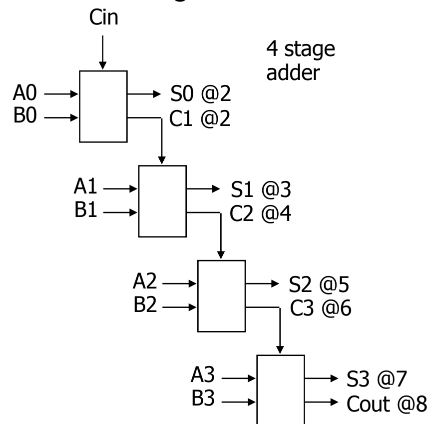
$C_o = G + P\,C_i$

Define G and P auxiliary functions

RIPPLE-CARRY ADDERS

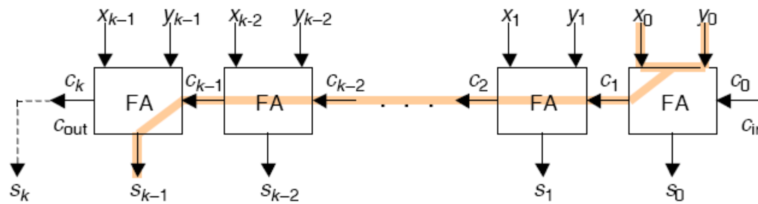
- Critical Delay
  - The propagation of carry from low to high order stages





CRITICAL PATH TROUGH A RIPPLE CARRY ADDER

$$T_{\text{ripple-add}} = T_{FA}(x,y \rightarrow c_{out}) + (k-2) \times T_{FA}(c_{in} \rightarrow c_{out}) + T_{FA}(c_{in} \rightarrow s)$$



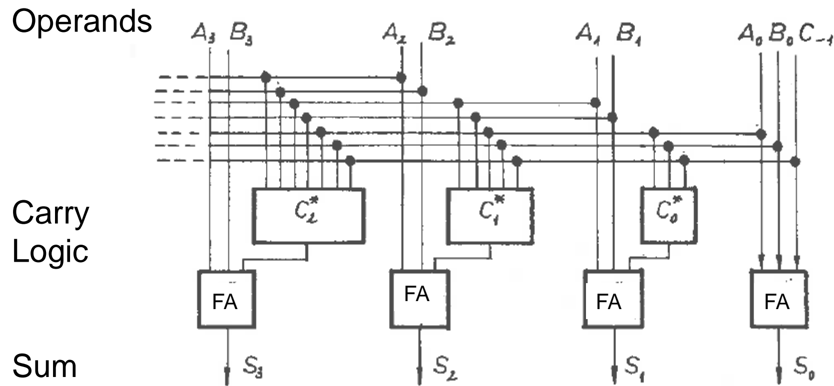
Critical path in a k-bit ripple-carry adder.

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.

FASTER ADDITION: CARRY LOOKAHEAD LOGIC



Principal layout of carry lookahead adder.

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

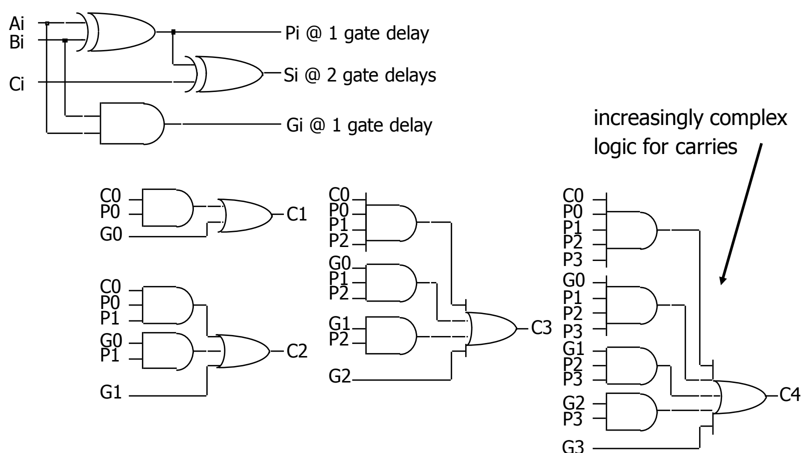
CARRY-LOOKAHEAD LOGIC

- 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

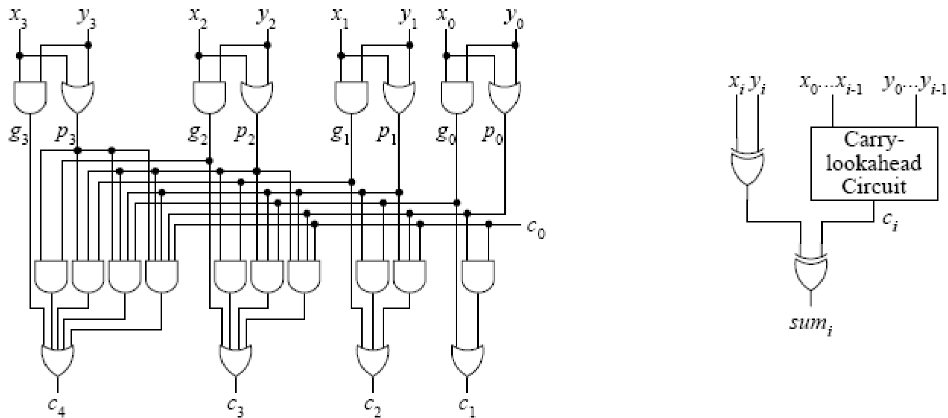
- Each of the carry equations can be implemented with two-level logic
  - All inputs are now directly derived from data inputs and not from intermediate carries
  - this allows computation of all sum outputs to proceed in parallel

CARRY-LOOKAHEAD IMPLEMENTATION

Adder with propagate and generate outputs



CARRY LOOKAHEAD CIRCUITRY

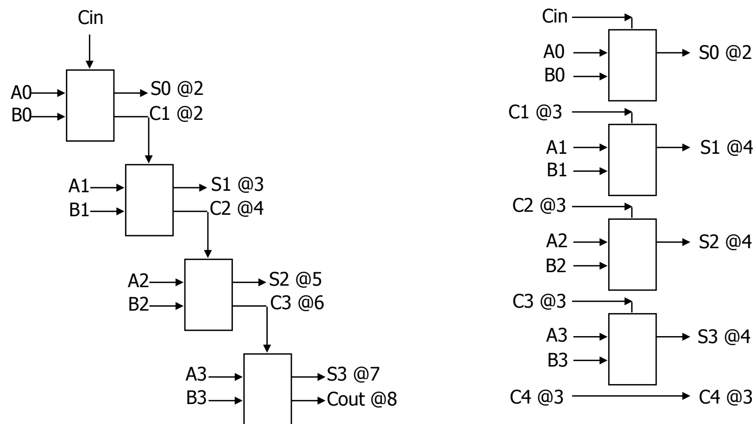


(a) Circuit for generating the carry-lookahead signals, c1 to c4;
(b) One bit slice of the carry-lookahead adder.

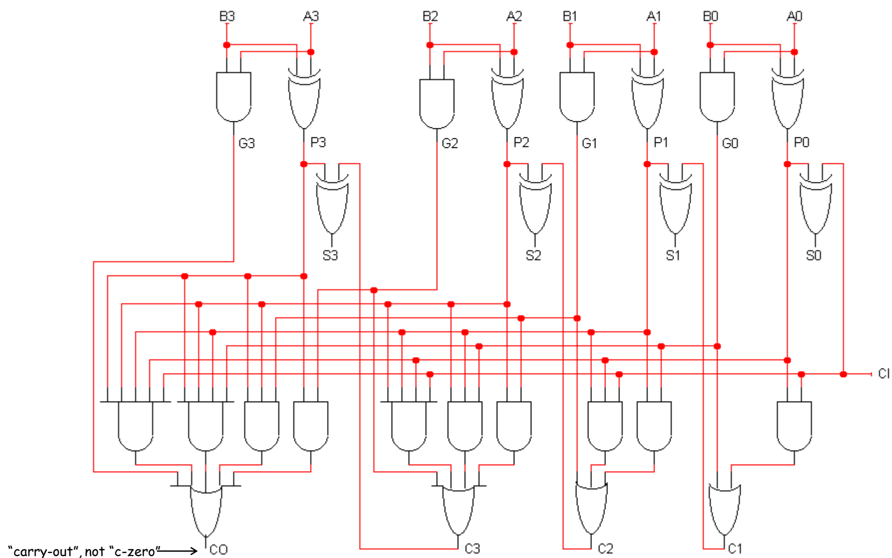
CARRY-LOOKAHEAD IMPLEMENTATION

Carry-lookahead logic generates individual carries

- Sums computed much more quickly in parallel
- However, cost of carry logic increases with more stages

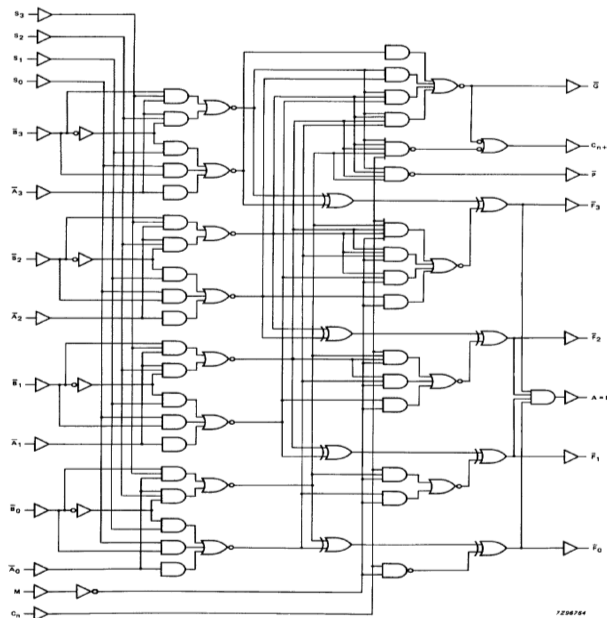


4-BIT CARRY LOOKAHEAD ADDER CIRCUIT



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

74HC/HCT181 4-BIT ALU LOGIC DIAGRAM



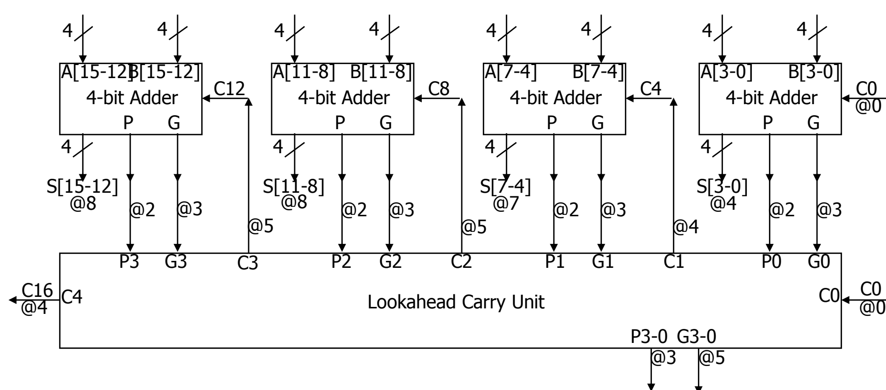
CARRY LOOKAHEAD ADDERS: FEATURES

- 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.

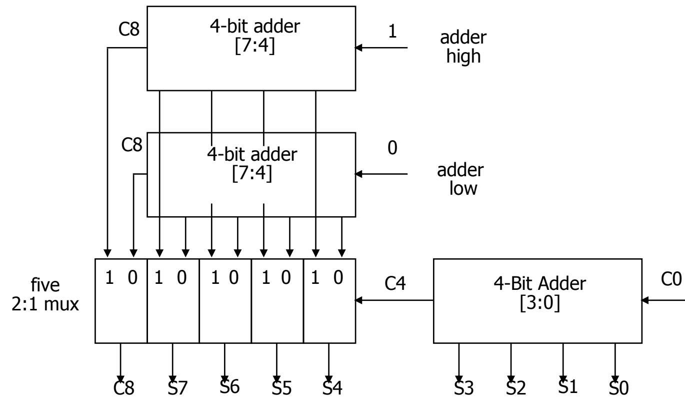
CARRYLOOKAHEAD ADDERS WITH CASCDED CARRY-LOOKAHEAD LOGIC

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

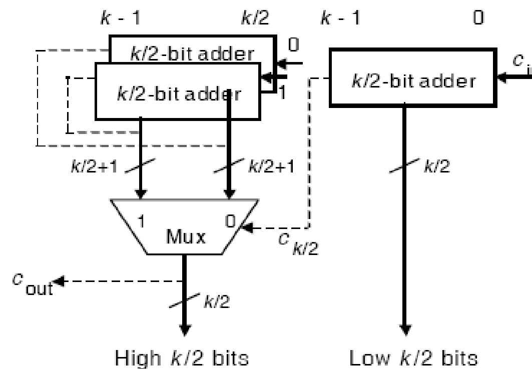


CARRY-SELECT ADDER

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 computed



CARRY-SELECT ADDERS

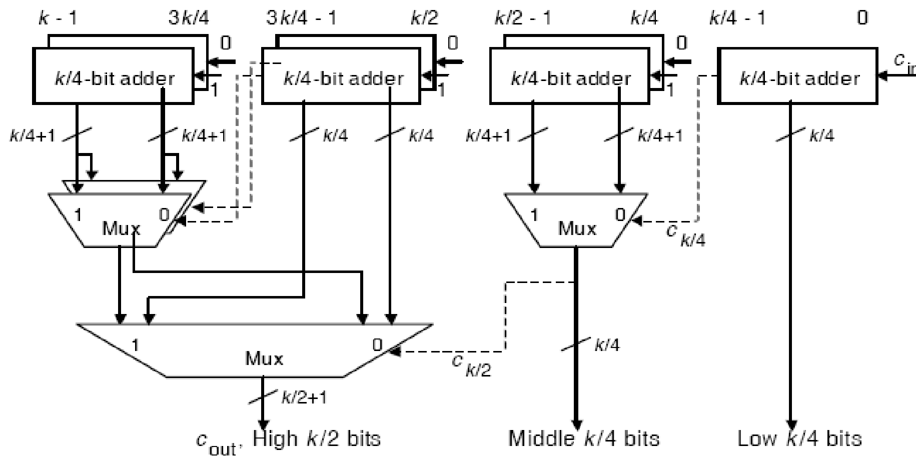


Carry-select adder for *k*-bit numbers built from three *k*/2-bit adders.

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

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

MULTILEVEL CARRY-SELECT ADDERS



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

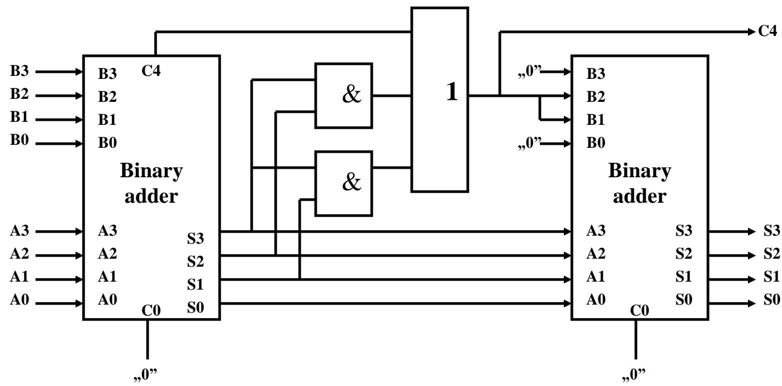
If the sum of two tetrades is not larger than 9, the result is valid, no correction is necessary.

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.

44

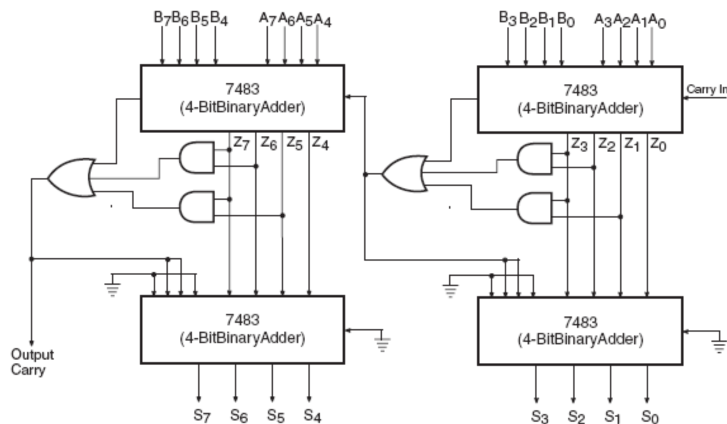
FUNCTIONAL DIAGRAM OF A BCD ADDER (1 DIGIT)



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

45

APPLICATION: 2-DIGIT BCD ADDER



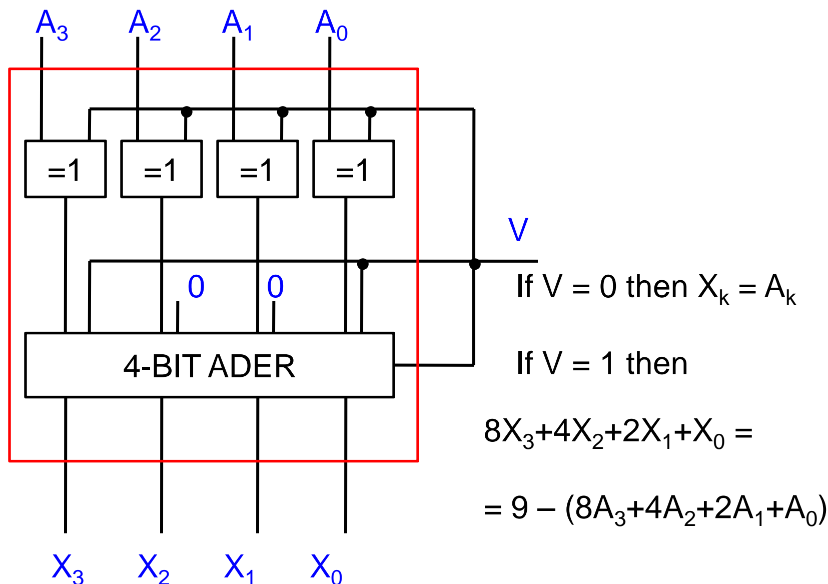
SUBTRACTION IN BCD (8421) CODE

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.

47

GENERATING 9'S COMPLEMENT IN BCD



SSI MODULAR LOGIC: 4-BIT BCD ADDER

4-bit BCD adder **74F583**

FEATURES

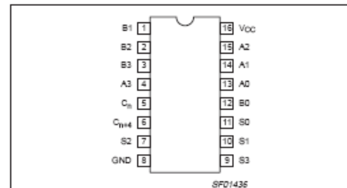
- Adds two decimal numbers
- Full internal look-ahead
- Fast ripple carry for economical expansion
- Sum output delay 19.5 ns max.
- Ripple carry delay 8.5 ns max.
- Input to ripple delay 13.0 ns max.
- Supply current 60 mA max.

DESCRIPTION

The 74F583 4-bit coded (BCD) full adder performs the addition of two decimal numbers (A0–A3, B0–B3). The look-ahead generates BCD carry terms internally, allowing the 74F583 to then do BCD addition correctly. For BCD numbers 0 through 9 at A and B inputs, the BCD sum forms at the output.

In addition of two BCD numbers totalling a number greater than 9, a valid BCD number and carry will result. For input values larger than 9, the number is converted from binary to BCD. Binary to BCD conversion occurs by grounding one set of inputs, An or Bn, and applying a 4-bit binary number to the other set of inputs. If the input is between 0 and 9, a BCD number occurs at the output. If the binary input falls between 10 and 15, a carry term is generated. Both the carry term and the sum are the BCD equivalent of the binary input. Converting binary numbers greater than 16 may be achieved by cascading 74F583s.

PIN CONFIGURATION



TYPE	TYPICAL PROPAGATION DELAY	TYPICAL SUPPLY CURRENT (TOTAL)
74F583	9.0 ns	45 mA

ORDERING INFORMATION

PACKAGE	COMMERCIAL RANGE V_{CC} = 5 V ±10% T_{amb} = 0°C to +70°C	DRAWING NUMBER
16-pin plastic DIP	N74F583N	SOT38-4
16-pin plastic SO	N74F583D	SOT109-1

49

SSI MODULAR LOGIC: 4-BIT BCD ADDER

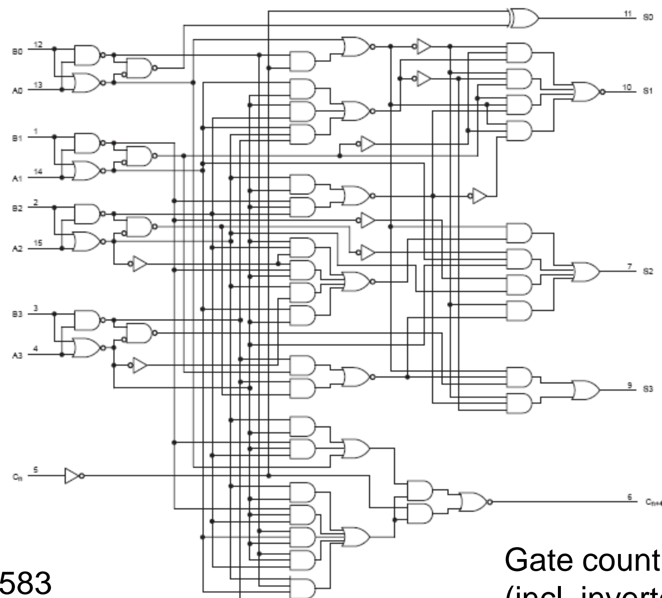
The 74F583 4-bit coded (BCD) full adder performs the addition of two decimal numbers (A0–A3, B0–B3). The look-ahead generates BCD carry terms internally, allowing the 74F583 to do BCD addition correctly.

For BCD numbers 0 through 9 at A and B inputs, the BCD sum forms at the output.

In addition of two BCD numbers totalling a number greater than 9, a valid BCD number and carry will result.

50

4-BIT BCD ADDER LOGIC DIAGRAM



74F583

Gate count 75
(incl. inverters)

51

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.

THEORY OF MULTIPLICATION

Basic Concept

multiplicand	1101	(13)
multiplier	* 1011	(11)
Partial products	1101	
	1101	
	0000	
	1101	
	10001111	(143)

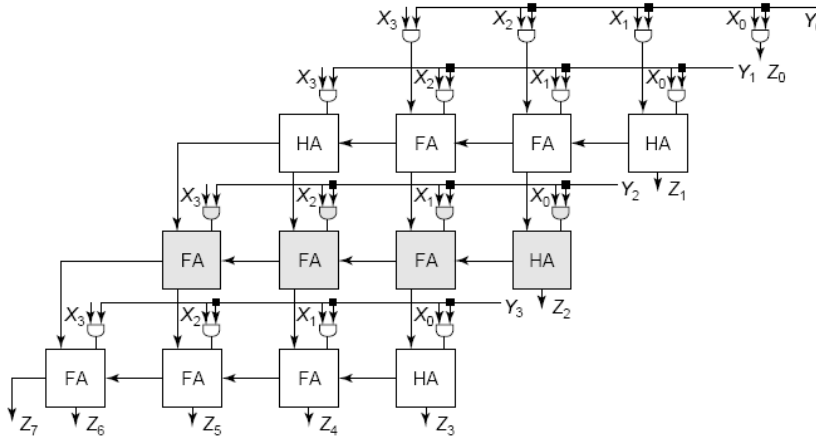
product of two 4-bit numbers
is an 8-bit number

COMBINATIONAL MULTIPLIER

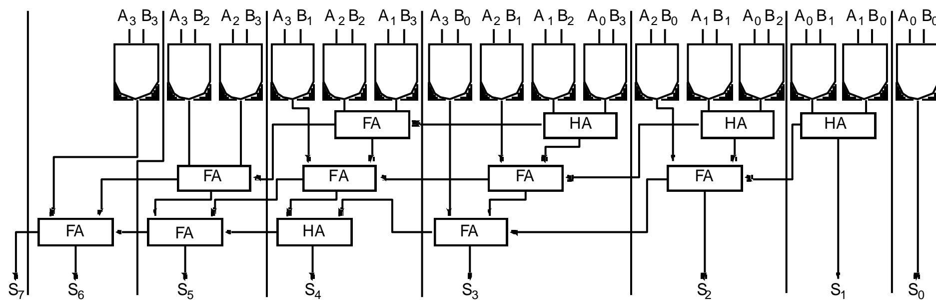
Partial Product Accumulation

				A3	A2	A1	A0
				B3	B2	B1	B0
				A2 B0	A2B0	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

THE ARRAY MULTIPLIER (4x4 BIT)



PARTIAL PRODUCT ACCUMULATION



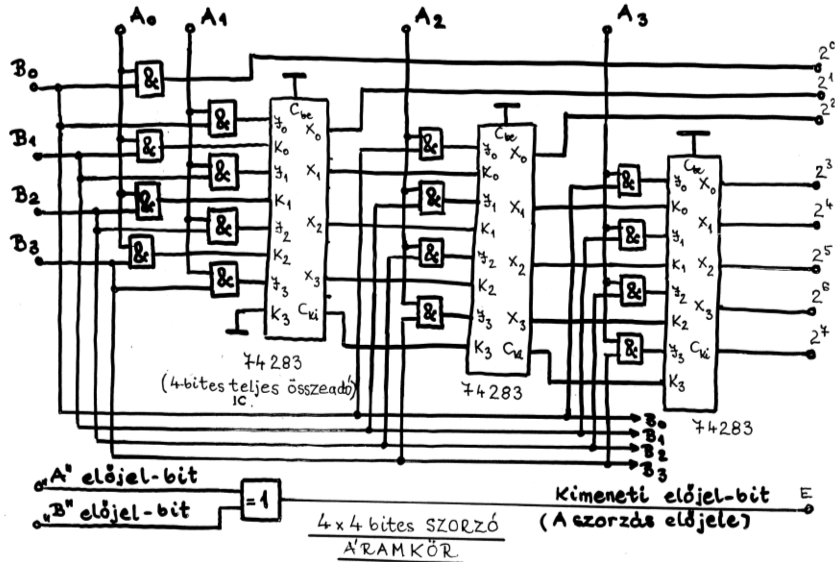
Note use of parallel carry-outs to form higher order sums

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

16 gates form the partial products

total = 88 gates!

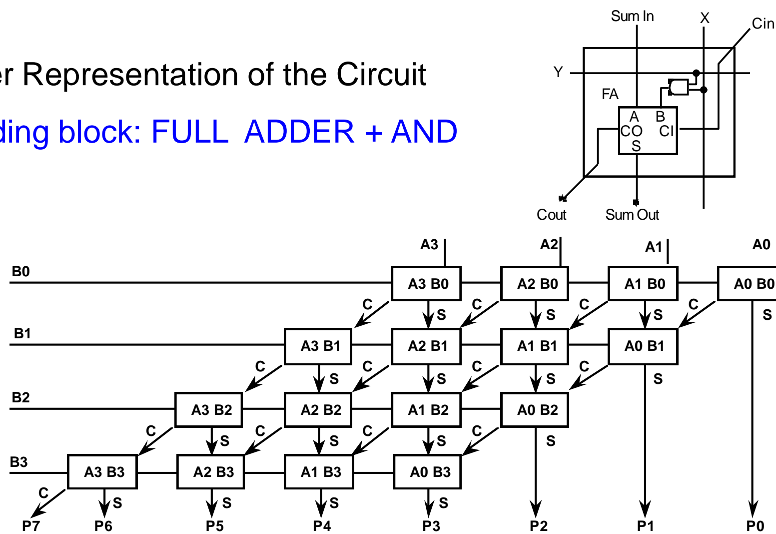
SSI REALIZATION OF 4x4 BIT MULTIPLIER



COMBIATIONAL MULTIPLIER

Another Representation of the Circuit

Building block: FULL ADDER + AND



4 x 4 array of building blocks

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

E.g. in the case of a 8-bit multiplier, it ia possible to partition the problem by splitting both the multiplier and multiplicand into two 4-bit words.

$N1 = (2^4H1 + L1)$
$N2 = (2^4H2 + L2)$

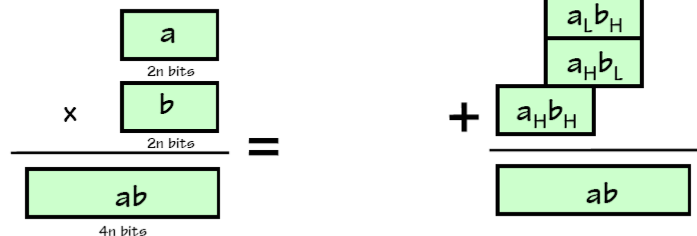
Multiplying out

$N1N2 = 2^8H1H2 + 2^4(H1L2 + H2L1) + L1L2$

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

Given n-bit multipliers:

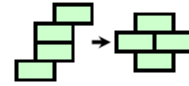
Synthesize 2n-bit multipliers:

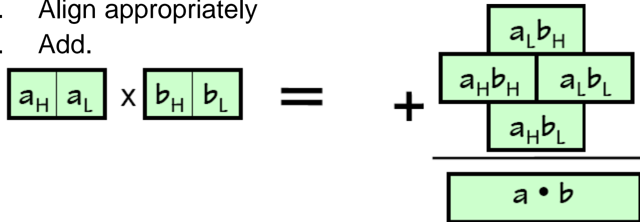


MAKING A 2n-BIT MULTIPLIER USING n-BIT MULTIPLIERS

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.

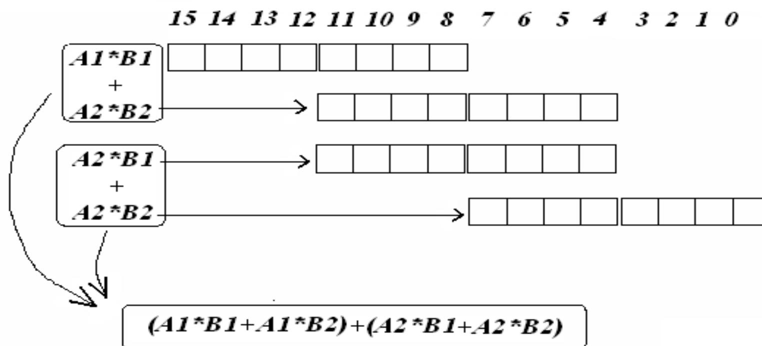




REGROUP partial products – 2 additions rather than 3!

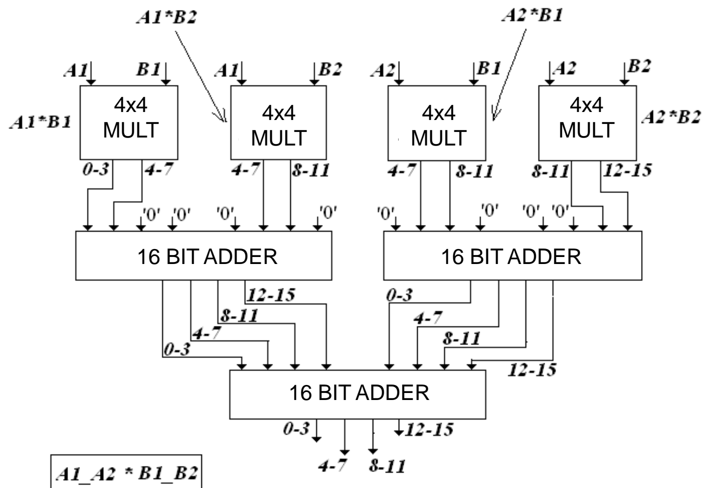
Induction: we can use the same structuring principle to build a 4n-bit multiplier from our newly-constructed 2n-bit ones...

MULTIPLIER: MODULAR STRUCTURE



8 x 8 bit multiplier built from 4 x 4 bit modules

MULTIPLIER: MODULAR STRUCTURE



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

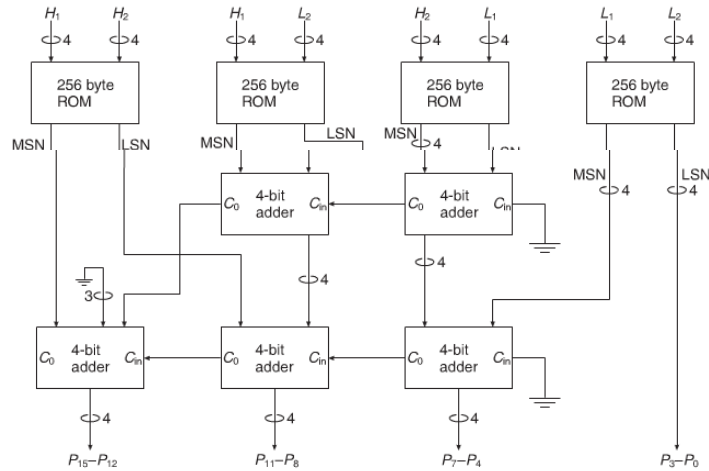
63

ROM IMPLEMENTED MULTIPLIER

Binary multiplication can be achieved by using a ROM as a "look-up table". E.g., multiplication of two 4-bit numbers requires a ROM having eight address lines, four of them X4XRX2X1 being allocated to the multiplier, and the remaining four, Y4Y3Y2Y1 to the multiplicand. Since the multiplication of two 4-bit numbers can result in a double-length product, the ROM should have eight output lines, and a room with capacity of 256 bytes is required.

For two 8-bit numbers 2^{16} = 65336 memory locations and 16 output lines for the double-length products are required. This requires a ROM of 128 kbytes. For 16-bit multiplication the required ROM capacity is formidable (16 Gbytes!).

8x8 BIT COMBINATIONAL MULTIPLIER



4x4 bit partial products are generated by four 256x8 bit ROMs

MULTIPLICATION: NEGATIVE NUMBERS

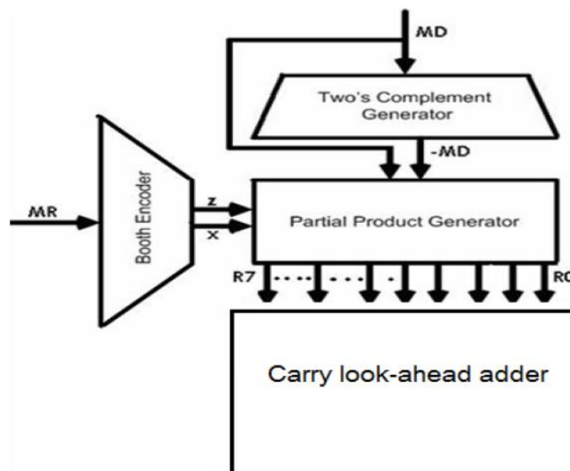
The basic school method of multiplication handles the sign with a separate rule ("+ with + yields +", "+ with - yields -", etc.). Modern computers embed the sign of the number in the number itself, usually in the two’s complement representation. That forces the multiplication process to be adapted to 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.

BOOTH ENCODING MULTIPLIER



System layout of Booth encoding 8-bit multiplier

BOOTH ENCODING MULTIPLICATION

The multiplier takes in two 8-bits operands:
the multiplier(MR) and the multiplicand (MD), then produces 16-bit multiplication result of the two as its output.

The architecture comprises four parts:

- Complement Generator,
- Booth Encoder,
- Partial Product and
- Carry Look-ahead Adder.

BOOTH ENCODING DEMO

$$X \times 00111110 = X \times (2^5 + 2^4 + 2^3 + 2^2 + 2^1) = X \times 62,$$

The number of partial product and the number of operations can be reduced to two by rewriting the equation as

$$X \times 00111110 = X \times (2^6 - 2^1) = X \times (64 - 2) = X \times 62$$

When Booth encounters the first digit of a block of ones (0 1), it follows this scheme.
When Booth encounters the end of the block (1 0), it follows a subtraction.

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. Present the layout of the two's complement adder/substractor and explain its operation.

2. Describe the operation of the carry lookahead adder.

3. Describe the operation of the carry-select adder.

4. Present the layout of a (one digit) BCD adder and explain its operation.

5. Describe the layout and operation of the combinational multiplier.

PROBLEMS AND EXERCISES

1. Implement the 2-bit adder function (i.e., 2-bit binary number *AB* plus 2-bit binary number *CD* yields 3-bit result *XYZ*) using three 8:1 multiplexers. Show your truth table and how you derived the inputs to the multiplexers.

2. Construct a circuit which multiplies a 4-bit binary number (X3 X2 X1 X0) by six. Use a 4-bit adder (functional block), and a minimum number of other gates.

3. 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. Give short description of your design and its operation.

PROBLEMS AND EXERCISES

4. Design a BCD adder circuit capable of adding BCD equivalents of two-digit decimal numbers. Indicate the IC type numbers used if the design has to be TTL logic family compatible.

5. Using representation on 8 bits perform in 2's complement the following operations:

25+30=?   25-30=?   30-25=?