Design of remarkable circuits: adder, comparator, complementer and subtractor

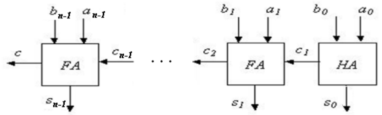
Prof. Daniele Gorla

Parallel Adder at *n* bits

Request: design a binary adder that performs the arithmetical sum of two n bits strings $A = a_{n-1}...a_0$ and $B = b_{n-1}...b_0$, seen as natural numbers.

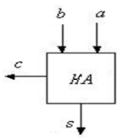
Idea: compute the sum as we are used to

- Sum the less signifying bits a_0 and b_0;
- This generates the less signifying bit of the result s_0 and a possible carry c_1;
- Now sum a_1, b_1 and c_1; this generates s_1 and c_2;
- ...and so on until the most signifying bits;
- If the last sum yields a carry c, then there is an overflow.

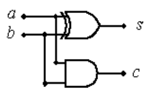


The elementary cell HA

The sum of a_0 and b_0 (here, simply denoted as a and b) does not have to consider any preceding carry (it is the first sum of the sequence); however, it can generate a carry c:



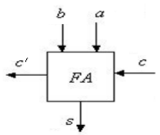
b	a	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



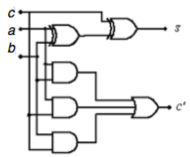
$$s = a \oplus b$$
$$c = ab$$

The elementary cell FA

If we denote with c the carry coming from the sum of a_{i-1} and b_{i-1}, and with c' the carry from the sum of a_i and b_i, we have the following truth table for the circuit that sums a_i and b_i (here, simply called a and b):



c	b	a	s	c'
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



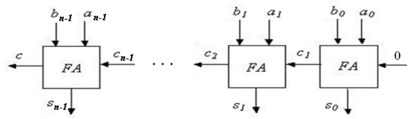
$$s = (a \oplus b)\overline{c} + \overline{(a \oplus b)}c = (a \oplus b) \oplus c$$
$$c' = ac + bc + ab$$

A Uniform Adder

Having two elementary circuits (HA and FA) makes the project more complex and costy.

To simplify, we can adopt a "uniform" version of the adder that only relies on FAs: it suffices to set the initial carry at 0 in the first elementary cell (that for the less signifying bits).

REMARK: in this way, we have a few more gates, but I have to produce one single kind of elementary cell!!



Adder for integer numbers

As we saw, for integers represented in 2-complement the sum is done exactly in the same way; hence, the circuit is the same!

The only difference is the overflow condition:

- For naturals, we just have to check the last carry bit (1 → overflow)
- For integers, we have an overflow if
  - Operands have the same sign that is different from the result's sign
  - We obtain the "forbidden" sequence 10…0

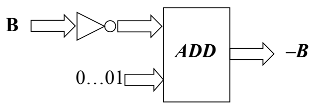
Hence, the BE associated to the overflow for integers is

$$a_{n-1}b_{n-1}\overline{s}_{n-1} + \overline{a}_{n-1}\overline{b}_{n-1}s_{n-1} + s_{n-1}\overline{s}_{n-2}\ldots\overline{s}_0$$

Opposite and Subtraction

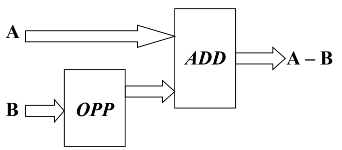
Opposite

Recall that the opposite of **B** = $b_{n-1}\ldots b_0$ is $\overline{b}_{n-1}\ldots\overline{b}_0 + 1$, we have that the circuit for calculating the opposite of a number is:



Subtraction

To compute **A – B**, we can do **A** + (–**B**) and so the circuit for the difference is:



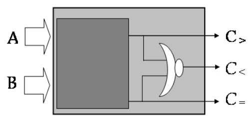
Arithmetical Comparator

Problem: given two binary n bits strings **A** and **B** representing two natural numbers, establish whether **A > B, A = B** or **A < B**.
The circuit will be something like

where:

- $c_> = 1$ iff **A > B**
- $c_< = 1$ iff **A < B**
- $c_= = 1$ iff **A = B**

OBS.: $c_<$ = NOR($c_>$, $c_=$); hence, we shall only design circuits for computing $c_>$ e $c_=$, from which



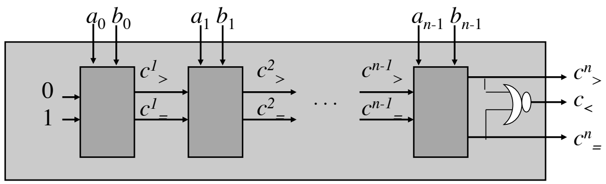
Structure of the comparator

SAPIENZA Università di Roma Dipartimento di Informatica

The idea is similar to the adder, with the circuit made up of n elementary comparator cells put in cascade.
To this aim, we use some partial results defined as follows:

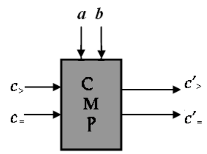
for every $i = 1, \dots, n$

- $c^i_> = 1$ iff $a_{i-1} \dots a_0 > b_{i-1} \dots b_0$
- $c^i_= = 1$ iff $a_{i-1} \dots a_0 = b_{i-1} \dots b_0$



The elementary comparing cell

SAPIENZA Università di Roma Dipartimento di Informatica



The truth table for **CMP** is

a	b	$c_>$	$c_=$	$c'_>$	$c'_=$
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	-	-
0	1	0	0	0	0
0	1	0	1	0	0
0	1	1	0	0	0
0	1	1	1	-	-
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	-	-
1	1	0	0	0	0
1	1	0	1	0	1
1	1	1	0	1	0
1	1	1	1	-	-

Synthesis of $c'_>$

SAPIENZA Università di Roma Dipartimento di Informatica

The KM is

$c_>$ $c_=$ \ a b	00	01	11	10
00	0	0	0	1
01	0	0	0	1
11	-	-	-	-
10	1	0	1	1

from which $c'_> = a\bar{b} + ac_> + \bar{b}c_> = a\bar{b} + (a + \bar{b})c_>$

Synthesis of $c'_=$

SAPIENZA Università di Roma Dipartimento di Informatica

The KM is:

$c_>$ $c_=$ \ a b	00	01	11	10
00	0	0	0	0
01	1	0	1	0
11	-	-	-	-
10	0	0	0	0

from which $c'_= = \bar{a}\bar{b}c_= + abc_= = (\bar{a}\bar{b} + ab)c_= = (\overline{a \oplus b})c_=$

Circuit Schema

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA