17.03.2011 ©

```
LIBRARY ieee;

USE ieee.std_logic_1164.all;                <≠>

USE work.fulladd_package.all;                        1-bit FA

ENTITY adder4 IS              4-bit FA
PORT ( Cin : IN STD_LOGIC;
         x : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         Y :   "      "      "      "      ;
         S : OUT     "      "      "      ;
      Cout : OUT STD_LOGIC);
 END adder4;
```

```
ARCHITECTURE Structure of adder4 is
SIGNAL C : STD_LOGIC_VECTOR(1 TO 3);
BEGIN
Stage0: fulladd PORTMAP ( Cin, X(0), Y(0), S(0), C(1));
Stage 1: fulladd PORTMAP( C(1), X(1), Y(1), S(1), C(2));
Stage 2: fulladd PORTMAP (C(2), X(2) Y(2), S(2), C(3));
Stage 3 : fulladd PORTMAP ( C(3), X(3), Y(3), S(3), Cout);
END Structure;
```

Adder

$X + Y = S$

STRUCTURAL WAY:

$x(i) \oplus y(i) \oplus c_i = s_i$

It is possible to do the same thing with

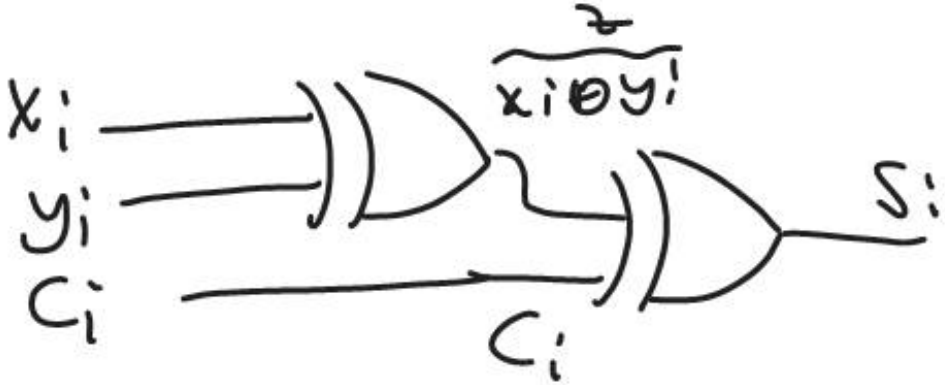
$(x(i)$ AND $y(i))$ OR $(x(i)$ AND $c(i))$ OR $(y(i)$ AND $c(i)) = c_{i+1}$

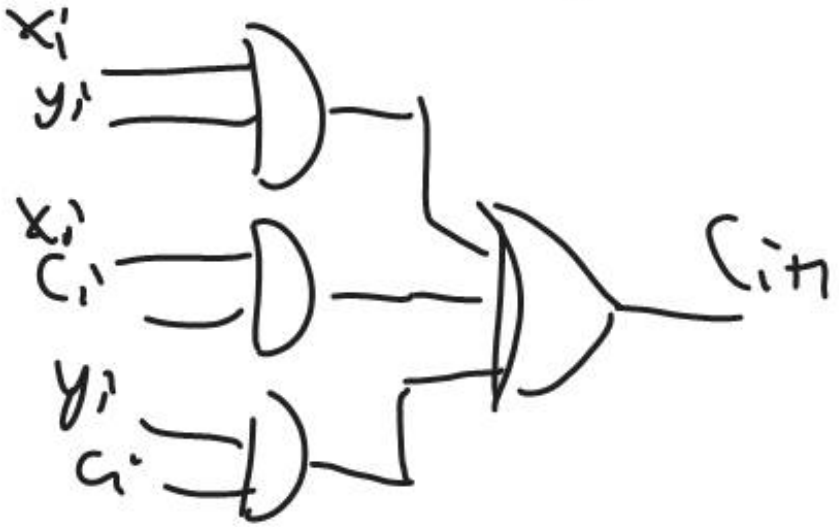
ARITHMETIC STATEMENTS

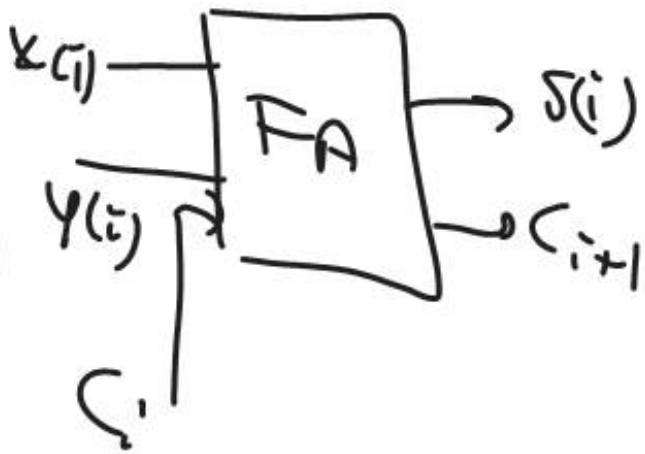
Behavioral

$z \oplus c_i$

$(x_i \oplus y_i) \oplus c_i = x_i \oplus y_i \oplus c_i$







It is possible to DESIGN ENTITY with ARITHMETIC STATEMENTS

Arithmetic → signed numbers
Arithmetic → unsigned numbers

⇒ ieee.std_logic_signed.ALL;

Then we can use arithmetic statements

USE ieee.std_logic_arith.ALL;

this way we can do arithmetic with
- signed
- unsigned

} numbers.

ARITHMETIC ASSIGNMENT STATEMENTS

ex:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;
ENTITY Adder16 IS
 PORT(X,Y: IN STD_LOGIC_VECTOR(15 DOWNTO 0);
      S : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
END Adder16;
ARCHITECTURE Behavior OF Adder16 IS
 BEGIN
  S <= X+Y;
END BEHAVIOR;
```

(no carry out, no overflow
Carry in

to consider carries and overflow:

```
:
USE ieee.std_logic_signed.all;
ENTITY Adder16 IS
PORT( X, Y : IN STD_LOGIC_VECTOR(15 DOWNTO 0);   -- 16-bit
      Cin : IN STD_LOGIC;
      S : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
      Cout, Overflow : OUT STD_LOGIC);
END Adder16;

ARCHITECTURE Behavior OF Adder16 IS
SIGNAL SUM : STD_LOGIC_VECTOR (16 DOWNTO 0);   -- 17-bit
```

```
BEGIN
SUM <= ('0' & X) + Y + Cin ; (?)
S <= Sum(15 DOWNTO 0);
Cout <= Sum(16);
OVERFLOW <= Sum(16) XOR X(15) XOR Y(15) XOR
Sum(15);
END BEHAVIOR;
```

16 bits

17 bit

$C_n \oplus C_{n-1}$

Recall:

-3

$+\ -6$

-9

$n = 4$

0011 → 1100 → 1101 (−3)

0110 → 1001 → 1010 (−6)

1101 X 4-bit

+ 1010 Y 4-bit 5-bit

1 | 01111 = Sum (4 Down to 0)

$C_4\ C_3\ C_2\ C_1$

$C_4 \oplus C_3$ = OVERFLOW

5-bit

10111 −9

01000

1

01001

= 9

MULTIPLICATION

UNSIGNED INTEGERS

$X = 1001_2$

$X * 2 \Rightarrow$ shift 1 bit left and add zero

10010_2 / 18

$2^4 \quad 2$

1011.011

$2^3 \; 2^1 \; 2^0 \; 1$

$2^{-1} . 2^{-2} \; 2^{-3}$

11

$0 + \frac{1}{4} + \frac{1}{8} = 0.25 + 0.125 = 0.375$

Divide by 2:

Shift 1 bit right and add 0 from left

$X \div 2 = 0100* = 0100_4$ 11.375

Signed Numbers

$n = 4$

ex $0101 = x$
$+5$

$x * 2 = 01010_{10}$ ✓
$8\ 2$

ex $y = 1100_{-4}$

$y * 2 = 11000_{-8}$

Divide by 2

ex $x = 0110_b$

$0011 = x \div 2$
3

$y = 1100$ -4 $\quad 0100 \to 0101$ 5

$y \div 2 = 1110$ $\quad 0011 \to 0100$ -4

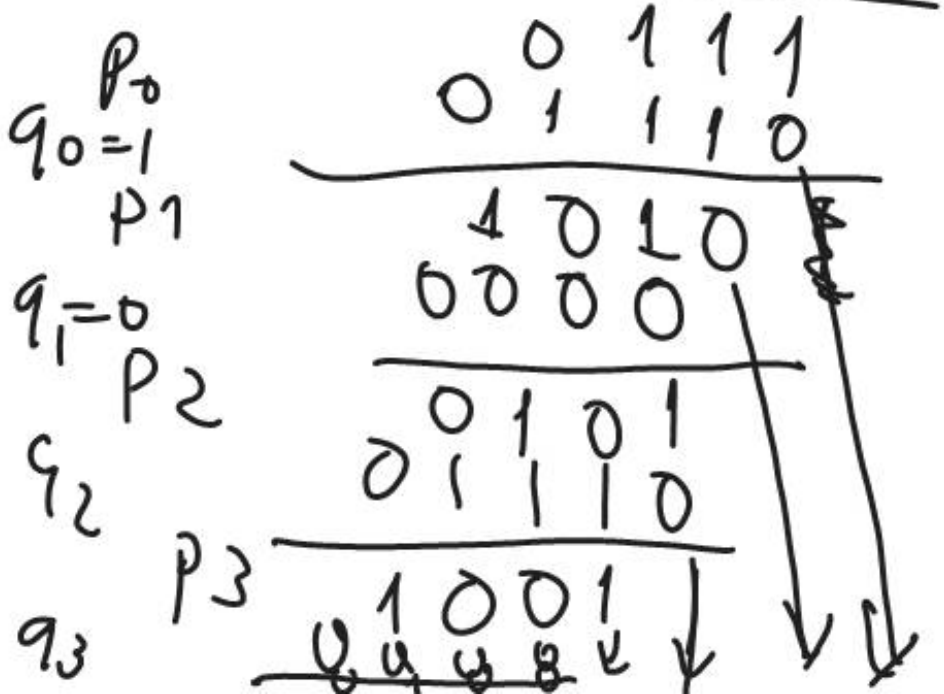
$0001 \to 0010$ -2

Multiply by 2^k is done by shifting left by k times.

eg 4 bit numbers

M ⇒ 0 1 1 1 = multiplicand $m_3 m_2 m_1 m_0$

Q ⇒ 0 1 0 1 = multiplier $q_3 q_2 q_1 q_0$



Product

0 1 0 0 1 1 0 1

✗