Universidad
Carlos III de Madrid

Descripción de circuitos digitales

CIRCUITOS COMBINACIONALES ARITMÉTICOS

Índice

- Representación de números con y sin signo
- Operaciones aritméticas
- Circuitos aritméticos
- Descripción de circuitos aritméticos en VHDL

Representación de números con signo

- Los números con signo tienen dos partes: signo y magnitud
- Es necesario **codificar el signo** del número para poder representarlo utilizando sólo 0s y 1s:
  - Normalmente se codifica el signo con un 0 para números positivos y un 1 para números negativos
- Según diferentes formas de **codificar la magnitud** se obtienen 3 diferentes sistemas de representación:
  - Signo y magnitud
  - Complemento a 1
  - Complemento a 2

Representación Signo y Magnitud

- Los números en el sistema de Signo y Magnitud se codifican de la siguiente forma:
  - El MSB es el signo (0 si es positivo o 1 si es negativo)
  - El resto de los bits son la magnitud del número a representar, codificada en binario natural

$$25_{10} = 11001_{BIN}$$

- Ejemplos:

$$+25_{10} = 011001_{SM}$$

$$-25_{10} = 111001_{SM}$$

Complemento a 1

- Los números en el sistema de Complemento a 1 se codifican:
  - Si el número es positivo:
    - El MSB es un 0 (signo)
    - El resto de los bits son la magnitud en binario natural
  - Si el número es negativo:
    - El MSB es un 1 (signo)
    - El resto de los bits son el complemento (a 1) de la magnitude
  - Cambiar de signo es cambiar ceros y unos
- Ejemplos: $+25_{10} = 011001_{Ca1}$

  $-25_{10} = 100110_{Ca1}$

Complemento a 2

- Los números en el sistema de Complemento a 2 se codifican:
  - Si el número es **positivo**:
    - El MSB es un 0 (signo)
    - El resto de los bits son la magnitud en binario natural
  - Si el número es **negativo**:
    - El MSB es un 1 (signo)
    - El resto de los bits son el complemento a 2 de la magnitud.
      - El complemento a dos de un número es su complemento + 1

$$Ca2(A) = \overline{A} + 1$$

      - Equivalentemente, se puede definir como (siendo n el número de bits):

$$Ca2(A) = 2^n - A$$

$$(2^n - A = 2^n - 1 + 1 - A = 11\ldots11 - A + 1 = \overline{A} + 1)$$

- Ejemplos: $+25_{10} = 011001_{Ca2}$

  $-25_{10} = 100111_{Ca2}$

Complemento a 2

- No hay que confundir los conceptos de “operación de complementar a 2” y “representación en complemento a 2” de un número:
  - Operación de complementar a 2 de un número A:

$$Ca2(A) = 2^n - A = \overline{A} + 1$$

  - Representación en complemento a 2 de un número A:
    - Hay que distinguir si es positivo o negativo, sólo en el caso de que sea negativo dicha representación se obtiene aplicando la operación de complementar a 2.
- La representación en complemento a 2 es la más utilizada en los sistemas digitales para números con signo.

Extensión del número de bits

- Un mismo número se puede representar con diferente número de bits:

$$+25_{10} = 011001_{SM} = 0000011001_{SM} = 0000000000011001_{SM}$$

- Para extender el signo:
  - En SM:
    - Añadir ceros justo después del signo
  - En Ca1 y Ca2:
    - Si es positivo añadir ceros a la izquierda
    - Si es negativo añadir unos a la izquierda

$$+25_{10} = 011001_{Ca2} = 0000011001_{Ca2} = 0000000000011001_{Ca2}$$

$$-25_{10} = 100111_{Ca2} = 1111100111_{Ca2} = 1111111111100111_{Ca2}$$

Complemento a 2

Codificación			SM	Ca1	Ca2
0	0	0	0	0	0
0	0	1	1	1	1
0	1	0	2	2	2
0	1	1	3	3	3
1	0	0	-0	-3	-4
1	0	1	-1	-2	-3
1	1	0	-2	-1	-2
1	1	1	-3	-0	-1

- En SM y en Ca1 hay dos codificaciones distintas que representan el 0
- En Ca2 la representación del 0 es única

Complemento a 2

- Otra propiedad del Ca2 es que el complemento a 2 del complemento a 2 de un número es el mismo número (la operación inversa del Ca2 es el propio complemento a 2):

$$Ca2(Ca2(A_{Ca2})) = A_{Ca2}$$

Dem: $Ca2(Ca2(A_{Ca2})) = 2^n - (Ca2(A_{Ca2})) = 2^n - (2^n - A_{Ca2}) = A_{Ca2}$

- De lo que se deduce esta otra propiedad:

$$-A_{Ca2} = Ca2\,(A_{Ca2})$$

Dem: Si A_{Ca2} es positivo, entonces por la propia definición de representación en Ca2

Si A_{Ca2} es negativo, entonces $-A_{Ca2}$ se obtiene haciendo la operación inversa del Ca2, pero como $Ca2(Ca2(A_{Ca2})) = A_{Ca2}$, se tiene que $-A_{Ca2} = Ca2\,(A_{Ca2})$

- Ejemplo:

```
Partiendo de un número positivo:    00001001  (+9)
Realizando la operación de Ca2:     11110111  (-9)
Realizando la operación de Ca2:     00001001  (+9)
```

Complemento a 2

- Otra forma de calcular el complemento a 2 de un número:
  - Empezando por el LSB, dejar iguales los bits hasta encontrar el primer 1 e invertir el resto:
  - Ca2(11100100) =00011100

    Comprobación: Ca2(11100100) = 00011011 + 1 = 00011100
- Otra forma de convertir de Ca2 a decimal (Ca2 es un código ponderado):
  - Considerar el peso del signo como negativo
  - Ejemplos:
    - $1110_2 = 1*(-2^3) + 1*2^2 + 1*2^1 + 0*2^0 = -8 + 4 + 2 = -2_{10}$
    - $0110_2 = 0*(-2^3) + 1*2^2 + 1*2^1 + 0*2^0 = 4 + 2 = 6_{10}$

Suma binaria

- Las sumas de números naturales en binario se realizan igual que en decimal:

```
   1001   ( 9)
+  1101   (13)
  10110   (22)
```

- Utilizando la representación de los números en Ca2, el método es válido también para números con signo, si se siguen las siguientes reglas.
  - Operandos con el mismo número de bits
  - Se descarta el acarreo final
  - Si los dos operandos tienen el mismo signo, y el resultado de la operación tiene signo diferente el resultado no es válido. Se dice que hay desbordamiento (**"overflow"**):
    - Esto sucede porque haría falta un bit adicional para poder representar el resultado.

Suma binaria en Ca2: Ejemplos

- Dos números positivos: (+9) + (+4)

$$\begin{array}{rr} 0\ 1001_{Ca2} & (+9) \\ +\ 0\ 0100_{Ca2} & (+4) \\ \hline 0\ 1101_{Ca2} & (+13) \end{array}$$

- Número positivo grande y número negativo pequeño: (+9) + (-4)

$$\begin{array}{rr} 0\ 1001_{Ca2} & (+9) \\ +\ 1\ 1100_{Ca2} & (-4) \\ \hline \cancel{1}\ 0\ 0101_{Ca2} & (+5) \end{array}$$

- Números iguales de signo contrario: (+9) + (-9)

$$\begin{array}{rr} 0\ 1001_{Ca2} & (+9) \\ +\ 1\ 0111_{Ca2} & (-9) \\ \hline \cancel{1}\ 0\ 0000_{Ca2} & (\ 0) \end{array}$$

- Dos números negativos: (-9) + (-4)

$$\begin{array}{rr} 1\ 0111_{Ca2} & (\ -9) \\ +\ 1\ 1100_{Ca2} & (\ -4) \\ \hline \cancel{1}\ 1\ 0011_{Ca2} & (-13) \end{array}$$

- Número positivo pequeño y número negativo grande: (-9) + (+4)

$$\begin{array}{rr} 1\ 0111_{Ca2} & (-9) \\ +\ 0\ 0100_{Ca2} & (+4) \\ \hline 1\ 1011_{Ca2} & (-5) \end{array}$$

- Dos números grandes (overflow): (+9) + (+9)

$$\begin{array}{rr} 0\ 1001_{Ca2} & (+9) \\ +\ 0\ 1001_{Ca2} & (+9) \\ \hline 1\ 0010_{Ca2} & (-14) \end{array}$$

Resta binaria

- Para realizar restas binarias se utiliza la propiedad vista anteriormente del complemento a 2:

$$-A_{Ca2} = Ca2\,(A_{Ca2})$$

Y por tanto:

$$A_{Ca2} - B_{Ca2} = A_{Ca2} + Ca2\,(B_{Ca2})$$

O equivalentemente:

$$A_{Ca2} - B_{Ca2} = A_{Ca2} + \overline{B_{Ca2}} + 1$$

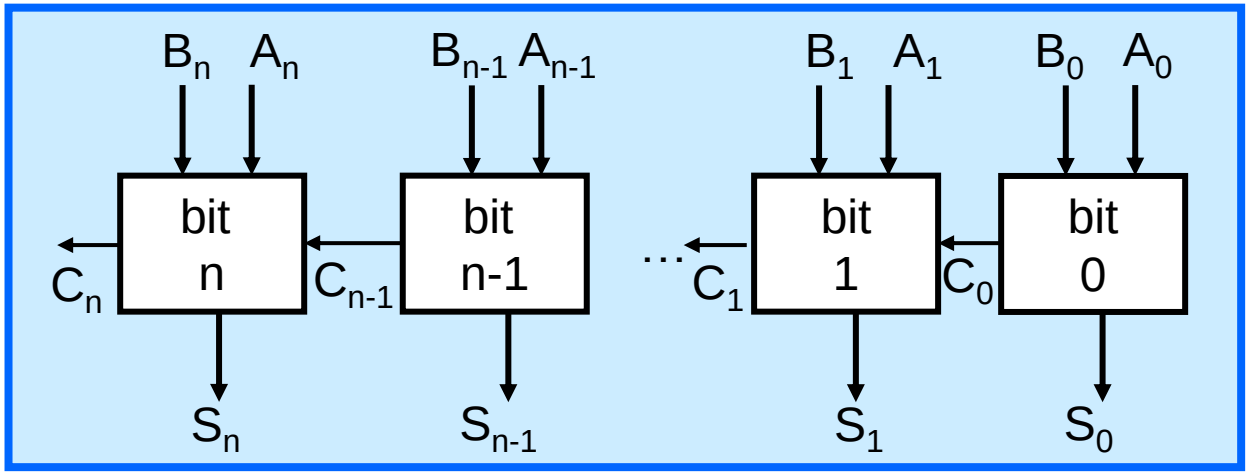
- En los sistemas digitales se representan los números con signo en Ca2, y no se utilizan restadores (no hacen falta), sólo sumadores

Sumador con propagación de acarreo serie

Suma decimal y binaria

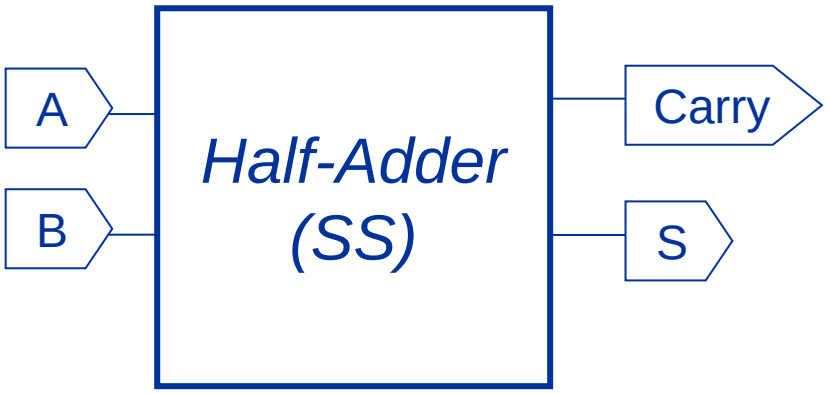
1 1		
86d	→	1010110b
25d	→	0011001b
111d	→	1101111b

- Operandos: **n bits**
- Resultado: **n+1 bits**

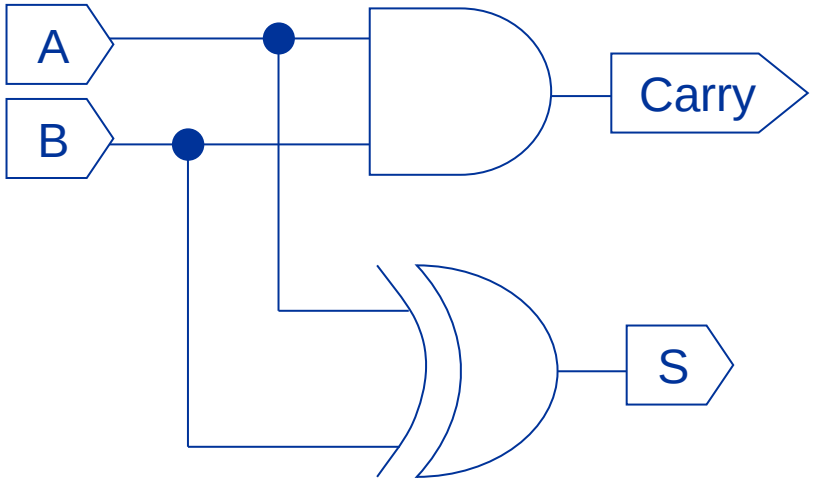


Sumador con propagación de acarreo serie

- Semisumador

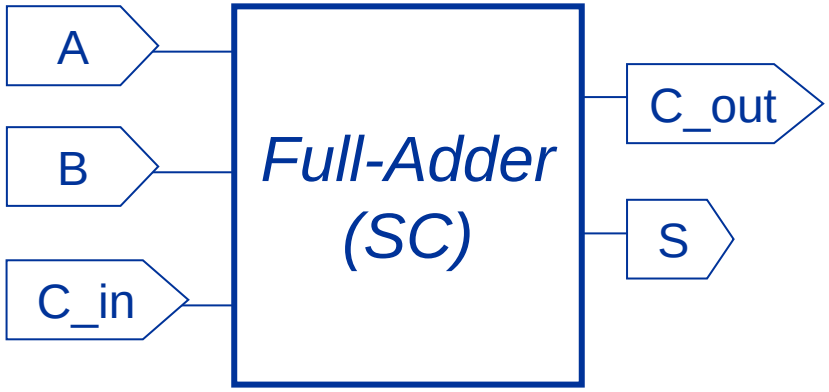


A	B	S	Carry
0	0	**0**	**0**
0	1	**1**	**0**
1	0	**1**	**0**
1	1	**0**	**1**

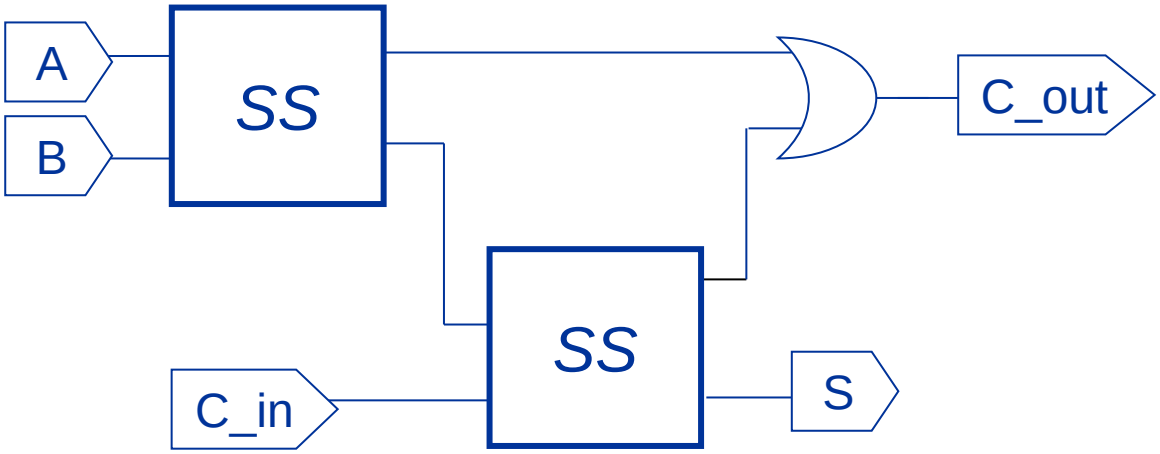


Sumador con propagación de acarreo serie

- Sumador completo



A	B	C_in	S	C_out
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**



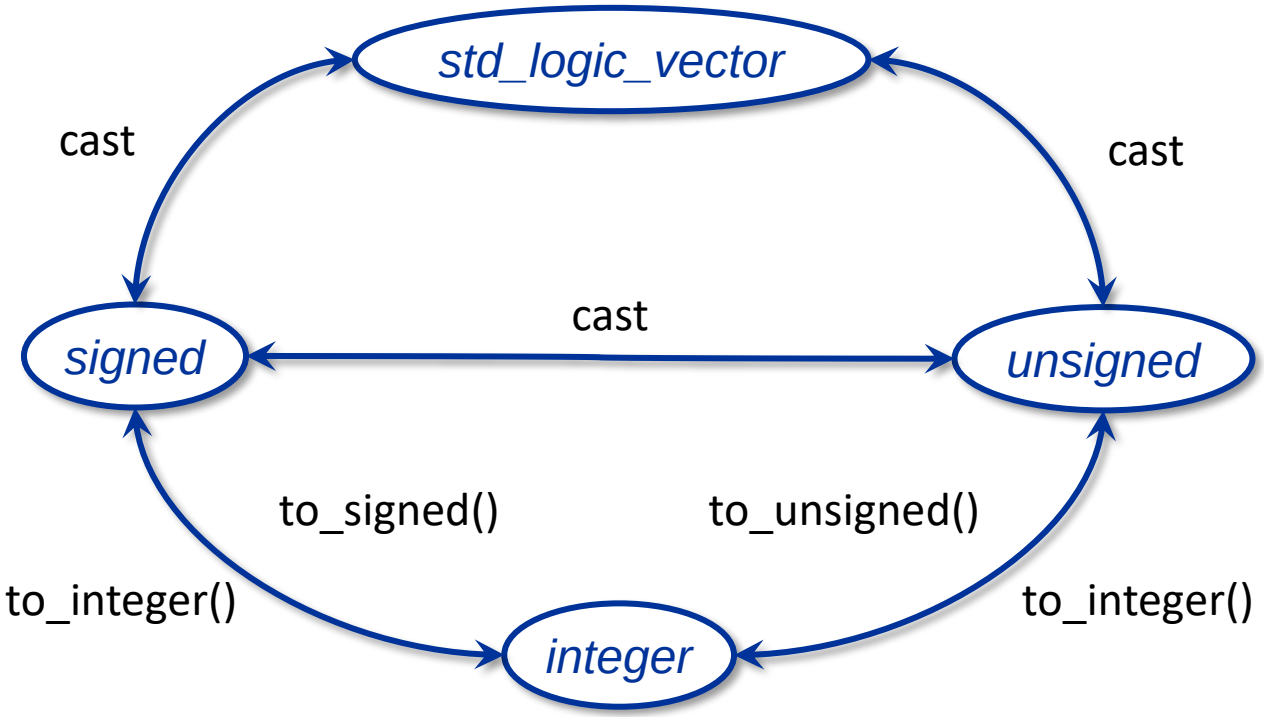
Aritmética VHDL: IEEE.NUMERIC_STD

- Tipos:
  - Vector de bits: no numérico
    - `STD_LOGIC_VECTOR(7 DOWNTO 0)`
  - Binario natural: números positivos
    - `NATURAL RANGE 0 to 255 (no necesario, usar INTEGER)`
    - `UNSIGNED (7 downto 0)`
  - `Ca2: números positivos y negativos`
    - `INTEGER RANGE -128 to 127`
    - `SIGNED (7 downto 0)`
- Operadores: +,-,*,/,REM,MOD
  - `INTEGER y NATURAL se pueden mezclar`
  - `SIGNED <= SIGNED operador SIGNED/INTEGER/NATURAL`
  - `UNSIGNED <= UNSIGNED operador UNSIGNED/INTEGER/NATURAL`
- Operadores: <,>,<=,>=,=, /=
  - `BOOLEAN <= SIGNED operador SIGNED/INTEGER/NATURAL`
  - `BOOLEAN <= UNSIGNED operador UNSIGNED/INTEGER/NATURAL`

Aritmética VHDL: IEEE.NUMERIC_STD

- Funciones: devuelven mismo tipo que el operando, salvo STD_MATCH que devuelve BOOLEAN
  - `SHIFT_LEFT(), SHIFT_RIGHT()`: desplazamientos
  - `ROTATE_LEFT(), ROTATE_RIGHT()`: rotaciones
  - `RESIZE()`: cambia el tamaño, teniendo en cuenta el signo
  - `STD_MATCH()`: comprueba la igualdad teniendo en cuenta dont_care ('-')
- Funciones de conversión (cast):
  - `STD_LOGIC_VECTOR()` `SIGNED,UNSIGNED => STD_LOGIC_VECTOR`
  - `SIGNED()` `UNSIGNED,STD_LOGIC_VECTOR => SIGNED`
  - `UNSIGNED()` `SIGNED,STD_LOGIC_VECTOR => UNSIGNED`
- Funciones de conversion:
  - `TO_INTEGER()` `SIGNED,UNSIGNED => INTEGER/NATURAL`
  - `TO_SIGNED()` `INTEGER/NATURAL => SIGNED`
  - `TO_UNSIGNED()` `INTEGER/NATURAL => UNSIGNED`

Aritmética VHDL IEEE.NUMERIC_STD



- *std_logic_vector* no representa números, no se pueden convertir a enteros ni ser usados como argumentos de operaciones aritméticas (suma, mult, …)
- Hacer cast entre *std_logic_vector*, *signed* y *unsigned* no cambia el contenido en bits del vector: usar con cuidado.

Aritmética VHDL: ejemplos

- Calificadores de tipo

```
signal a_un, b_un, c_un: unsigned(7 downto 0);
signal a_sg,b_sg,c_sg,d_sg: signed(7 downto 0);
. . .
a_un <= "1111"; -- a_sg vale 15
b_un <= a_un + "0001";   -- No válido: ambiguo
c_un <= a_un + unsigned("0001");
d_un <= a_un + 1;
...
a_sg <= "1111"; -- a_sg vale -1
c_sg <= a_sg + signed("0001");
d_sg <= a_sg + 1;
```

Aritmética VHDL: ejemplos

- Sumador con generador de acarreo

```
signal a,b,s: unsigned(7 downto 0);
signal cy: std_logic;
. . .
process(a,b)
  variable result: unsigned(8 downto 0);
begin
  result := resize(a,9)+resize(b,9); -- Suma con 9 bits
  cy <= result(8); -- Acarreo es el noveno bit
  s <= result(7 downto 0);
end process;
```

Aritmética VHDL: ieee.std_logic_arith

std_logic_arith

Tipos: signed, unsigned
Operadores:
+,-,*,/,rem,mod
Funciones:
SHR, SHL, EXT, SEXT
Funciones de conversión:
SIGNED, UNSIGNED, STD_LOGIC_VECTOR
CONV_UNSIGNED, CONV_SIGNED
CONV_INTEGER
CONV_STD_LOGIC_VECTOR

Creados por Synopsys, no estándar

std_logic_signed
std_logic_unsigned

std_logic_vector se usa como signed o unsigned

Operadores:
+,-,*,<,>,<=,>=,=,/=
Func. de conversión (std_logic_vector):
CONV_INTEGER

Aritmética VHDL

Usos de bibliotecas típicos

```
-- Declaración mínima para usar STD_LOGIC y STD_LOGIC_VECTOR
library IEEE;
use IEEE.std_logic_1164.all;
```

```
-- Operaciones aritméticas con tipos SIGNED y UNSIGNED
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
```

```
-- Operaciones aritméticas con tipos SIGNED y UNSIGNED
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
```

```
-- Operaciones aritméticas que usan STD_LOGIC_VECTOR como
-- enteros sin signo
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
```

Aritmética VHDL: recomendaciones

- Acotar los rangos facilita la depuración
  - INTEGER RANGE 0 to 255;
- Trabajar en enteros es más sencillo… pero los puertos deben ser vectores
- Usar STD_LOGIC_VECTOR para agrupaciones de bits no numéricas y SIGNED/UNSIGNED para números
- Controlar el tamaño de los datos:
  - La señal destino debe ser lo suficientemente grande para contener el resultado
  - El truncamiento de bits se hace conservando los bits de menos peso