Sistemas Digitales

Tomás Agustín Hernández

1. Introducción a los sistemas de representación

Magnitud

Llamamos magnitud al tamaño de algo, dicho en una medida específica. Es representada a través de un sistema que cumple 3 conceptos fundamentales:

- Finito: Debe haber una cantidad finita de elementos.
- Composicional: El conjunto de elementos atómicos deben ser fáciles de implementar y componer.
- Posicional: La posición de cada dígito determina en qué proporción modifica su valor a la magnitud total del número.

Algunos de los sistemas de representación más utilizados son: binario, octal, decimal y hexadecimal.

Bases

Una base nos indica la cantidad de símbolos que podemos utilizar para poder representar determinada magnitud.

Base	**Símbolos disponibles**
2 (binario)	0, 1
8 (octal)	0, 1, 2, 3, 4, 5, 6, 7
10 (decimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
16 (hexadecimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Tabla 1: Bases más utilizadas

La tabla anterior representa los símbolos disponibles para las bases 2, 8, 10 y 16.

Consideremos por un momento que estamos en binario; ¿sería correcto que 1 + 1 = 2? ¡No! Porque 2 no es un símbolo válido en base 2.

Para indicar la base en la que está escrito un número, se coloca la base entre paréntesis en la esquina inferior derecha.

$1024_{(10)}$: 1024 representado en base 10 (decimal)

Digítos/Bits

Sea $n \in \mathbb{Z}$, cuando decimos que tenemos n bits es lo mismo que decir que tenemos n dígitos.

- 0001: Representa el número 1 en binario, en 4 bits/dígitos.
- 0010: Representa el número 2 en binario, en 4 bits/dígitos.

Teorema de división

Es una manera de poder realizar un cambio de base de un número decimal a otra base. La representación en la otra base es el resto visto desde abajo hacia arriba.

$$a = k * d + r \ con \ 0 \leq r < |d|$$

donde:

- k = cociente
- d = divisor.
- r = resto de la división de a por d.

Pasaje del número $128_{(10)}$ a $128_{(2)}$ en 8 bits

$$128 = 64 * 2 + 0$$

$$64 = 32 * 2 + 0$$

$$32 = 16 * 2 + 0$$

$$16 = 8 * 2 + 0$$

$$8 = 4 * 2 + 0$$
$$4 = 2 * 2 + 0$$
$$2 = 1 * 2 + 0$$
$$1 = 0 * 2 + 1$$

Luego, $128_{(2)} = 1000\ 0000$

Bit más significativo / menos significativo

El bit más significativo en un número es el que se encuentra a la izquierda, mientras que el menos significativo es el que se encuentra a la derecha.

$$\mathbf{1}000000\mathbf{0}_{(2)}$$

Tipos numéricos

Representemos números naturales y enteros a partir de la representación en base 2 (binario)

Sin signo: Representa únicamente números positivos. No se pueden utilizar los símbolos de resta (**-**) ni tampoco coma (**,**)

$$1_{(10)} = 01_{(2)}$$
$$128_{(10)} = 10000000_{(2)}$$

Signo + Magnitud: Nos permite representar números negativos en binario.
El bit más significativo indica el signo

- 0: número positivo
- 1: número negativo.

$$18_{(10)} = \mathbf{0}0010010_{(2)}$$
$$-18_{(10)} = \mathbf{1}0010010_{(2)}$$

Representar números en S+M suele traer problemas porque el 0 puede representarse de dos maneras

$$+0_{(10)} = \mathbf{0}0000000_{(2)}$$
$$-0_{(10)} = \mathbf{1}0000000_{(2)}$$

Para solucionar este problema, las CPU utilizan la notación Complemento a 2 (C_2)

Exceso m: Sea $m \in \mathbb{Z}$, decimos que un número n está con exceso m unidades cuando $m > 0$

$$n_0 = n + m$$
$$n = 1 \wedge m = 10 \longrightarrow n_0 = -9$$

Nota: n_0 indica el valor original de n antes de ser excedido m unidades.

Complemento a 2: Los positivos se representan igual.
El bit más significativo indica el signo, facilitando saber si el número es positivo o negativo. Cosas a tener en cuenta

- **Rango**: -2^{n-1} *hasta* $2^{n-1} - 1$
- **Cantidad** de representaciones del cero: Una sola
- **Negación**: Invierto el número en representación binaria positiva y le sumo uno.
  - $-2_{(2)} = inv(010) + 1$
  - $-2_{(2)} = 101 + 1$
  - $-2_{(2)} = 110$
- **Extender número a más bits**: Se rellena a la izquierda con el valor del bit del signo.
- **Regla de Desbordamiento**: Si se suman dos números con el mismo signo, solo se produce desbordamiento cuando el resultado tiene signo opuesto.

Overflow / Desbordamiento

Hablamos de overflow/desbordamiento cuando

- El número a representar en una base dada, excede la cantidad de bits que tenemos disponibles.
- Si estamos en notación C_2 al sumar dos números cambia el signo.

Acarreo / Carry

Ocurre cuando realizamos una suma de números binarios y el resultado tiene más bits que los números originales que estamos sumando

Suma entre números binarios

Se hace exactamente igual que una suma común y corriente.
Es importante prestar atención a la cantidad de dígitos que nos piden para representarlo, y en caso de estar en C_2 que el signo no cambie.

Hagamos sumas en C_2 (sin límite de bits)

$$\begin{array}{r} 0010 = 2 \\ +1001 = -7 \\ \hline 1011 = -5 \end{array} \quad \begin{array}{r} 0101 = 5 \\ +1110 = -2 \\ \hline \color{blue}{1}0011 = 3 \end{array} \quad \begin{array}{r} 1011 = -5 \\ +1110 = -2 \\ \hline \color{blue}{1}1001 = -7 \end{array} \quad \begin{array}{r} 0111 = 7 \\ +0111 = 7 \\ \hline \color{red}{1}110 = \text{Overflow} \end{array} \quad \begin{array}{r} 1010 = -6 \\ +1100 = -4 \\ \hline \color{blue}{1}\color{red}{0}110 = \text{Overflow} \end{array}$$

Nota: El color azul indica el carry; El rojo indica qué es lo que produce overflow (cambio de signo).

Hagamos sumas en C_2 (límite de bits: 4)

$$\begin{array}{r} 0010 = 2 \\ +1001 = -7 \\ \hline 1011 = -5 \end{array} \quad \begin{array}{r} 0101 = 5 \\ +1110 = -2 \\ \hline \color{blue}{1}0011 = Overflow \end{array} \quad \begin{array}{r} 1011 = -5 \\ +1110 = -2 \\ \hline \color{blue}{1}1001 = Overflow \end{array} \quad \begin{array}{r} 0111 = 7 \\ +0111 = 7 \\ \hline \color{red}{1}110 = \text{Overflow} \end{array} \quad \begin{array}{r} 1010 = -6 \\ +1100 = -4 \\ \hline \color{blue}{1}\color{red}{0}110 = \text{Overflow} \end{array}$$

Nota: Al tener un límite de 4 bits, en las sumas que tenemos carry terminamos teniendo overflow.

Rango de valores representables en n bits

Sean $n, m \in \mathbb{Z}$ decimos que el rango de representación en base n y m bits acepta el rango de valores de: $[-n^m, n^m - 1]$
¿Es posible representar el 1024 en binario y 4 bits? No.

- $2^4 = 16 \implies [-16, 15]$
- Pero, $1024 \notin [-16, 15]$
- Por lo tanto, 1024 no es representable en 4 bits.

Pasar número binario a decimal

1. Si tenemos el mismo número todo el tiempo podemos usar la serie geométrica

¿Qué número decimal representa el número $1111111111_{(2)}$?

$$\textstyle\sum_{i=0}^{j-1} 1 \cdot n^i = \frac{q^{n+1} - 1}{q - 1} \ Luego,$$

$$\textstyle\sum_{i=0}^{9} 1 \cdot 2^i = 2^{10} - 1 = 1023$$

2. Si no tenemos el mismo número todo el tiempo podemos multiplicar cada dígito por la base donde el exponente es la posición del bit.

$10_{(2)} = 1 * 2^1 + 0 * 2^0 = 2_{(10)}$

Extender un número de n bits a m bits

Sea $n, m \in \mathbb{Z}$ donde n es la cantidad de bits inicial y m es la cantidad a la que se quiere extender.

$$n = 3 \wedge m = 8$$

- Signo + Magnitud y exceso m: Se extiende con 0's luego del signo.
  - En 3 bits, -2 = 110
  - En 8 bits, -2 = **1**00000010
- Complemento 2 (C_2): Se extiende con el bit más significativo.
  - En 3 bits, -2 = 110
  - En 8 bits, -2 = **1**1111110

Cambios de base

Sea $n, m \in \mathbb{Z}$ dos bases distintas, para pasar de base n a base m se debe realizar el siguiente proceso

- Pasar el número a base decimal.
- Aplicar el teorema de división utilizando la base deseada.

Encontremos en base 5, el número que corresponde a $17_{(8)}$:

- $17_{(8)} = 1 * 8^1 + 7 * 8^0 = 15_{(10)}$
- Usando ahora el teorema de división
  - $15 = 3 * 5 + 0$
  - $3 = 0 * 5 + 3$
  - Luego, $30_{(5)}$
- Por lo tanto, $17_{(8)} = 30_{(5)}$

2. Desplazamientos

Utilizamos los desplazamientos para poder mover los bits. Cada casillero representa los bits.

- Desplazamiento hacia la izquierda: Se desplazan los bits del dato tantas posiciones como se indiquen a la izquierda.
  variable $\ll$ *cantidad*

Posición	v_3	v_2	v_1	v_0
a	1	0	1	0
$c = a \ll 2$	1	0	0	0

- Desplazamiento lógico hacia la derecha: Se aplica desplazando los bits del dato tantas posiciones como se indiquen a la derecha.
  variable $\gg_l$ *cantidad*

Posición	v_3	v_2	v_1	v_0
a	1	0	1	0
$c = a \gg_l 2$	0	0	1	0

- Desplazamiento aritmético hacia la derecha: Se aplica desplazando los bits del dato tantas posiciones como se indiquen a la derecha, pero copiando el valor del bit más significativo.
  variable $\gg_a$ *cantidad*

Posición	v_3	v_2	v_1	v_0
a	1	0	1	0
$c = a \gg_a 2$	1	1	1	0

3. Operaciones lógicas

- OR (+): (1, 0), (0, 1), (1, 1) = 1
- AND (∗): (1, 1) = 1
- XOR (⊕): (1, 0), (0, 1) = 1

4. Circuitos combinatorios

Negación

Sea p una variable proposicional, el opuesto de p lo escribimos como $\bar{p}$.

$$p = 1 \iff \bar{p} = 0$$

Propiedades para operaciones lógicas

Propiedad	AND	OR
Identidad	$1.A = A$	$0 + A = A$
Nulo	$0.A = 0$	$1 + A = 1$
Idempotencia	$A.A = A$	$A + A = A$
Inverso	$A.\overline{A} = 0$	$A + \overline{A} = 1$
Conmutatividad	$A.B = B.A$	$A + B = B + A$
Asociatividad	$(A.B).C = A.(B.C)$	$(A + B) + C = A + (B + C)$
Distributividad	$A + (B.C) = (A + B).(A + C)$	$A.(B + C) = A.B + A.C$
Absorción	$A.(A + B) = A$	$A + A.B = A$
De Morgan	$\overline{A.B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}.\overline{B}$

Operaciones booleanas

Se resuelven utilizando las propiedades para operaciones lógicas

$$Verifique\ si\ son\ equivalentes\ (X + \bar{Y} = \overline{(\bar{X} * Y)} * Z + X * \bar{Z} + \overline{(Y + Z)})$$

- $\overline{\bar{X} * Y} * Z + X * \bar{Z} + (\bar{Y} * \bar{Z}) \implies De\ Morgan$
- $(X + \bar{Y}) * Z + X * \bar{Z} + (\bar{Y} * \bar{Z}) \implies De\ Morgan\ \wedge\ Distributiva$
- $(X + \bar{Y}) * Z + \bar{Z} * (X + \bar{Y})$
- $(X + \bar{Y}) * (Z + \bar{Z}) \implies Inverso$
- $(X + \bar{Y}) * 1 \implies Identidad$
- $(X + \bar{Y})$

Nota: También se pueden probar equivalencias utilizando tablas de verdad

Funciones booleanas

- AND = A ∗ B
- OR = A + B
- NOT = $\bar{A}$

Tablas de verdad

Nos permiten observar todas las salidas para todas las combinaciones de entradas dada una función.
Veamos un ejemplo con una función F:

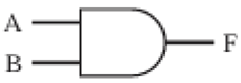
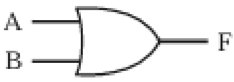
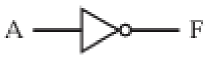
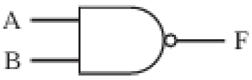
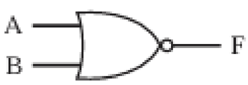
$$Sea\ F = X + \bar{Y}$$

X	**Y**	**F**
1	1	1
1	0	1
0	1	0
0	0	1

Protip: El símbolo de + indica OR porque 1 + 0 = 1 mientras que el símbolo AND indica * porque 1 * 0 = 0

Compuertas

Son modelos idealizados de dispositivos electrónicos que realizan operaciones booleanas.

Nombre	Símbolo gráfico	Función algebraica	Tabla verdad
AND		F = A • B or F = AB	A B | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		F = A + B	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT		$F = \overline{A}$ or $F = A'$	A | F 0 | 1 1 | 0
NAND		$F = (\overline{AB})$	A B | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = (\overline{A + B})$	A B | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0

Nota: XOR = $\oplus$

Compuertas Universales

Nos permiten obtener otros operadores.

- NAND = $\overline{A \wedge B}$
- NOR = $\overline{A \vee B}$
- XNOR = $\overline{A \oplus B}$ = Si son iguales es V

Compuertas en SystemVerilog

- A AND B ($A * B$) = (assign O = A & B)
- A OR B ($A + B$)= $assign\ A \mid B$
- A XOR B = $assign\ A\ \wedge\ B$
- NOT A ($\bar{a}$) = ($\sim A$)

Entradas / Salidas de un circuito

Se representan con flechas. En SystemVerilog se llaman input y output.

```
module ALU #(parameter DATA_WIDTH = 16)
        (input [DATA_WIDTH-1:0] operandoZ,
        input [DATA_WIDTH-1:0] operandoY,
        input [2:0] opcode,
        output [DATA_WIDTH-1:0] salidas,
        output overflow);
end module;
```

Nota: En las ALU no son funcionalmente iguales ni las entradas ni las salidas.

Mecanismo de Traducción fórmula a circuito

Llamaremos ϕ a una fórmula proposicional cualquiera

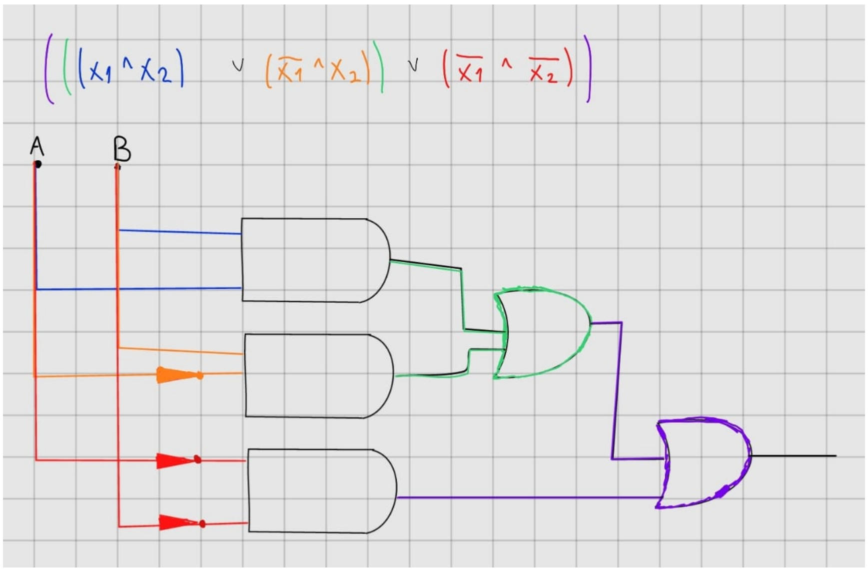
- 1. Solo consideramos de la función F, las filas verdaderas.
- 2. Cada fila verdadera tendrá su índice, y en ese índice estarán los valores de cada variable proposicional. Representamos a esa fila verdadera como t_i
- 3. Realizamos la conjunción de todas las variables de ese t_i
- 4. Realizamos la disyunción de todas las conjunciones de t_i

Un ejemplo:

$$Sea\ F = X + \bar{Y}$$

X	Y	F
1	1	1
1	0	1
0	1	0
0	0	1

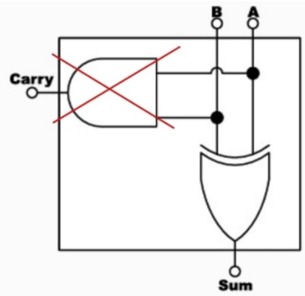
- F es solamente verdadera en la primera, segunda y tercer fila por lo tanto tenemos $t_1\ t_2\ y\ t_3$
- Por cada fila, hacemos la conjunción de los valores
  - $x_1 \wedge x_2$
  - $\bar{x_1} \wedge x_2$
  - $\bar{x_1} \wedge \bar{x_2}$
- Realizamos la disyunción de todos los t_i
  - $(x_1 \wedge x_2) \vee (\bar{x_1} \wedge x_2) \vee (\bar{x_1} \wedge \bar{x_2})$
- El resultado nos da ϕ' que es una suma de productos y nos permite traducir fácilmente a un circuito combinatorio



Carry en circuitos

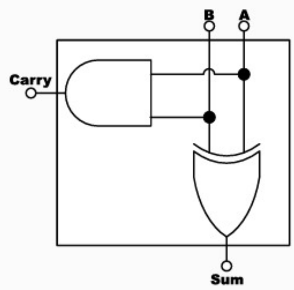
El carry debe colocarse en los circuitos en la suma porque en caso de no hacerlo, se nos pierden casos.

A	B	Sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
~~1~~	~~1~~	~~0~~	~~1~~



Si eliminamos el carry se nos pierde el caso 1 + 1, por lo tanto lo ideal sería que al hacer una suma nos quede así:

A	B	Sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Timing

En un circuito combinatorio el tiempo que tarda la salida en estabilizarse depende de la cantidad de capas de compuertas (latencia). Para enfrentar el problema usamos secuenciales.