

Sistemas Digitales

Indice

Tema 1: Introducción a los Sistemas Digitales Electrónicos.
 Tema 2: Representación Digital de la Información.
 Tema 3: Álgebra de Boole. Puertas Lógicas.
 Tema 4: Lenguaje de Descripción Hardware VHDL.
 Tema 5: Circuitos Aritméticos.
 Tema 6: Otros Circuitos Combinacionales.

Tema 1 : Introducción a los Sistemas Digitales Electrónicos

- Presentación
 - Profesor
 - Calendario
 - Sistemas Digital: TAC
 - Electrónica
 - Procesamiento de señales eléctricas
- Organización Académica
 - Programa
 - Prácticas
 - Ejercicios
 - Evaluación
 - Metodología

Profesorado

- Prof. Cándido Aramburu Mayoz.
 - Doctor Ingeniero Telecomunicación (UPNA-Universidad Politécnica de Madrid)
 - Empresa Ikusi S.A. (Sistemas de Telemedida 1989)
 - Profesor Titular UPNA (Dpto Ingeniería Electrónica y Comunicaciones 2000)
- Profesor Prácticas: Aitor Urrutia
- Profesor Euskera: Marko Galarza
- Profesor Inglés: Ignacio del Villar
- <https://www.etsit.upm.es/>
- <https://www.velatia.com/es/empresas-que-forman-velatia/ikusi/>
- <https://www.unavarra.es/eu/sites/Portada/home.html>

Contacto

- Despacho: Edificio Los Tejos 2 Planta: Despacho 2028 (Prof. Candido Aramburu)
- Mialulario → correo interno
- Clase
 - G1: A112 : Lunes (12-14) y Jueves (10-12)
 - G2: A012 : Lunes (10-12) y Miércoles (12-14)
- Tutorías
 - Lunes (14-17) y Miércoles (9-12)
 - Cita Previa

Calendario

| 5

| 6 |

Febrero

Lunes	Miércoles
30-Presentación..Repre_info:Numeros bases	1-Repre_Info:Enteros..C2:Expansion Signo
6-C2:Overflow..Tema3:Morgan	8-Tema3:Morgan..DK:examples
13-BCD..Tema3:Repaso y Ejercicios	15-Tema4_VHDL:Intro_Archit y Tema3:Dudas y Ejercicios
20-Tema5_Aritmética:Intro..Tema5:Sumador/Restador y Tema4:Paquetes:STD y Standard_Logic	27-Tema5:Semirestador..ALU y Tema4:Process()

Marzo

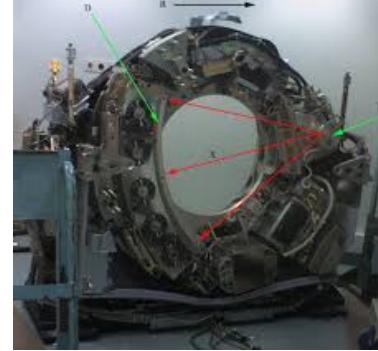
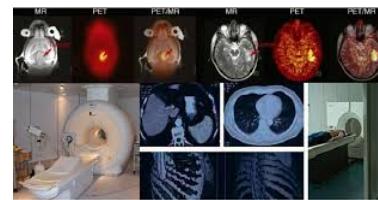
Lunes	Lunes
6	13
20	27

| 7

| 8 |

Sistemas Digitales

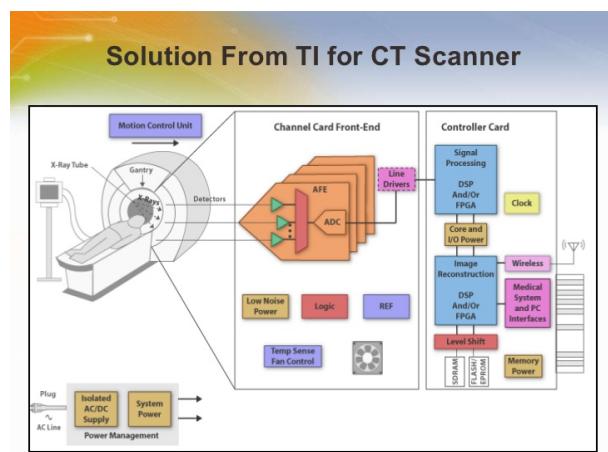
Tomografía axial computarizada



| 9

Sistema Digital

Electrónica



Entrada **Análogica** → Sensores Magnéticos.

Conversor A/D: Señal Analógica a Señales Digitales.

Circuitos **lógicos**: multiplexores, filtros, codificadores, etc ...

Procesadores lógicos: procesamiento de las señales digitales para obtener la imagen.

FPGA : Field Programming Gate Array.

DSP : Digital Signal Processing.

CPU : Centra Procesor Unit.

GPU : Graphic Procesor Unit.

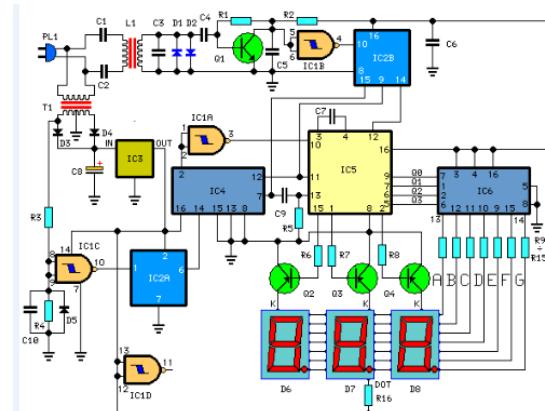
| 11

| 12 |

Equipos de Electrónica



Esquema Eléctrico



| 13

14 |

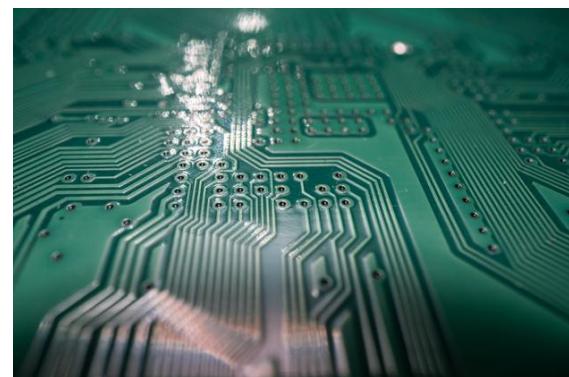
Componentes de una tarjeta de circuito impreso



Componentes:

- Discretos: resistencias, condensadores, transistores, transformadores, etc.
- Integrados ("chips", microelectrónica).
- material de semiconductor: Silicio.
- el componente básico es el transistor → un procesador puede tener cientos de millones.
- los microcircuitos hechos de transistores pueden ser tanto circuitos analógicos (un amplificador) como digitales (puertas lógicas)

Printed Circuit Board



| 15

16 |

Instrumentación



Técnico



La Electrónica en la Profesión

Digital Electronics Engineer

Ref No.	EMP439757
Location	Twickenham, London, England
Job type	Permanent
Job Status	Closed

You can not apply for this job as its status is **Closed**.

★ Save ✉ Email
Share: [f](#) [t](#) [in](#)

Introduction

Are you interested in working for a continuously growing company that really value their staff and offer a fantastic range of benefits?

Important

Digital electronics, FPGA, VHDL/Verilog, DSP, PCB

The Job

This opportunity is for a Senior Electronics Engineer to join a team that have tripled in size over the last 10 years due to continued success. The company design and manufacture sensors that use state-of-the-art acoustic resonance technology, and are distributed internationally across a number of industries including marine and defence.

¿ Profesiones relacionadas con la Electrónica?

Fases de Diseño de Circuitos Electrónicos Binarios

1. Funcional (manual): abstracción matemática
2. Automatización del proceso matemático
 - a. Herramientas de Diseño con ayuda del Computador (EDA)
 - b. Simulación del Diseño del Circuito Electrónico antes de fabricar el prototipo: Depuración
3. Fabricación del prototipo
 - a. Instrumentación
 - b. Verificación del funcionamiento en el Laboratorio
 - c. Verificación del funcionamiento en Campo
4. Comercialización
5. Producción

La Electrónica en la Carrera Universitaria

- Conocimientos de Electrónica
 - Para ... ?
- Tecnología Hardware
 - Fabricación de Prototipos
 - Diseño de Prototipos : Conceptos Teóricos y Herramientas de diseño por computador
 - Desarrollo de Sistemas: Equipos, Plataformas
 - Comercialización
 - Usuario: Equipos, Plataformas

| 21

| 22 |

Representación Científica y Prefijos de las Unidades

Table 1. Prefijos

Prefijos	Tera	Giga	Mega	Kilo	mili	micro	nano	pico
Base 10 → magnitudes:m.gr,Hz, ..	10^{12}	10^9	10^6	10^3	10^{-3}	10^{-6}	10^{-9}	10^{-12}
Base 2 → magnitudes: Byte	2^{12}	2^9	2^6	2^3	2^{-3}	2^{-6}	2^{-9}	2^{-12}

- Ejemplo: representar la magnitud=1000000000Hz debidamente
 - Notación científica → 10^9 Hz
 - Debidamente: Notación científica con prefijos f=1GHz → T=1/f= 10^{-9} seg= 1ns

Señales: Conversión Analógica Digital

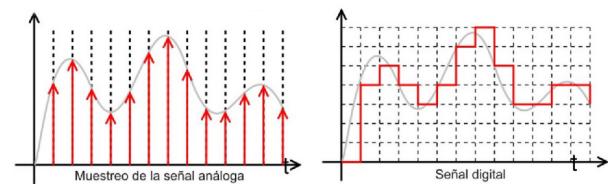
| 23

| 24 |

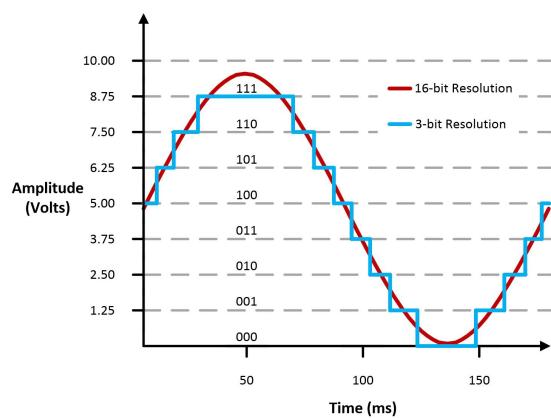
Analogica vs Digital

- Señal Continua
 - Amplitud: \approx valores posibles en el rango
 - Tiempo: \approx valores posibles en el rango
- Señal Discreta ó Digital
 - Amplitud: finitos valores posibles en el rango
 - Tiempo: finitos valores posibles en el rango

Señales : Muestreo y Cuantificación



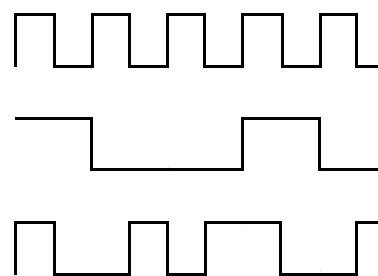
Codificación



Calcular para las resoluciones de 3 bit y 16 bits cual es el mínimo incremento de señal codificable o error de cuantificación: con 3 bits el número de niveles es $2^3=8$ niveles y el mínimo relativo es $2^{-3}=1/8$; con 16 bits el número de niveles es 2^{16} y el mínimo relativo es $2^{-16}=1/65536$.

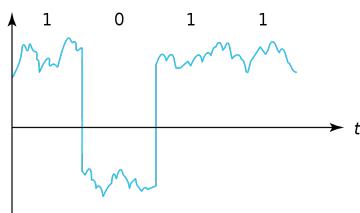
Representación de los números en código binario : [\[Tema2_Replinf\]](#)

Señales Binarias : Abstractas



Eje ordenada: valores abstractos (0/1, High/Low, ON/OFF, etc ...).
Cronograma: Representación temporal de las señales digitales binarias.
Esa representación típica de los libros de texto, pizarra de clase, etc ... es ideal ya que físicamente siempre habrá distorsión.

Señales Binarias : Físicas



Eje ordenada: magnitudes físicas (mV ó mA).

La señal física está distorsionada por causas como pej; línea larga de transmisión (efectos capacitivos e inductivos).

Un ejemplo típico de distorsión son los tiempos de subida y bajada, que no son nulos sino del orden de unos nanosegundos.

La distancia considerable entre los dos niveles (binario) a la entrada del receptor hace fácil la discriminación entre el '0' y el '1'.

Digitalización de las Señales

Ventajas

- Calidad: Fácil de recuperar a pesar de la distorsión
- Almacenamiento: Fiabilidad, Diversidad Formatos
- Compatibilidad: Diversidad de Equipos (PC, móvil, coche, etc)
- Procesamiento: Sencillo, Flexible
- Coste: Barato (componentes)

Abstracción

- Niveles: el 0 y el 1
- Lógica binaria
 - Matemáticas: Algebra de Boole

Organización Académica

Programa de la Asignatura

- [Ficha Web Upna](http://www.unavarra.es/ficha-asignaturaDOA/?languageld=100000&codPlan=246&codAsig=246110&año=2022) [http://www.unavarra.es/ficha-asignaturaDOA/?languageld=100000&codPlan=246&codAsig=246110&año=2022]
- Programa en 3 partes
 - i. **Circuitos Combinacionales**
 - ii. **Circuitos Secuenciales**
 - iii. Otros: Números, Lógica Programable (VHDL), Teoría Tecnología
- Bibliografía

Prácticas

- Tipo de prácticas:
 - Diseño manual
 - Simulación con la herramienta software Quartus de Intel.
 - Captura gráfica de Esquemas Electrónicos
 - Descripción del Circuito mediante el Lenguaje VHDL. Fabricación del Circuito en tecnología FPGA

Ejercicios

- Tipo de problemas: Libro Verde → Ejercicios tipo examen → Sin calculadora y sin libros

El libro verde se adquiere en el edificio de rectorado, en la sección de comunicación, que se encuentra en planta baja del edificio. El horario: 8 a 14:30. Precio 8.5\$.

* Capítulo 1: 1.1, 1.2, 1.4, 1.5, 1.6, 1.8, 1.9
 * Capítulo 2: 2.1
 * Capítulo 3: 3.2 3.3 -> 2º parcial
 * Capítulo 4: 4.2, 4.4, 4.6
 * Capítulo 5: 5.2, 5.3, 5.4
 * Capítulo 6: 6.1, 6.2 -> 2º parcial
 * Capítulo 7: 7.2, 7.3 y 7.4 -> 2º parcial
 * Capítulo 8: 8.1, 8.3 y 8.5 -> 2º parcial

- Los ejercicios del tema 2 (Representación de la Información) no están en el libro verde
 - Míaulario → Recursos → Ejercicios

Evaluación

- Sistema de Evaluación:
 - 75% teoría y 25% prácticas
 - Evaluación continua Teoría: dos parciales (30% 1º parcial y 45% 2º parcial). Nota mínima en el 2º parcial: 5. El Primer parcial se realizará el sábado 25 de Marzo a las 9:00, el segundo parcial el 24 de Mayo a las 8:00 y la recuperación el 12 de Junio a las 8:00
 - Recuperación Teoría: Entra todo. Nota mínima: 5.
 - Evaluación Prácticas: Un único examen el sábado XX de Mayo, no recuperable.

Metodología

- Trabajo en clase: principalmente Ejercicios con su teoría asociada
- Trabajo en casa
 - Teoría desarrollada en los apuntes PDF en mi aulario
 - Prácticas
 - En casa: Ejercicios de diseño manual
 - En casa: Utilización de Quartus y Memorias
 - Tutorías
 - Resolución de dudas

Tema 2 : Representación Digital de la Información

Indice

- Información: números, caracteres, imagen, sonido, etc ..
- Números
 - Sistemas posicionales: base 10 (decimales), base 2 (binaria)
 - Naturales: bases 10,2,8,16 . Conversión entre bases
 - Enteros: Signo Magnitud, Complemento a la base-1, Complemento a la base
 - Operaciones aritméticas: Suma,Resta
 - Reales: coma fija y coma flotante
- Caracteres
 - Alfanuméricos y Signos de Puntuación
 - ASCII standard y extendido
 - Unicode: UTF-8

Representación de los Números

Representación de los Números Decimales

- Decimal
 - 10 dígitos : 0,1,2,3,4,5,6,7,8,9
 - Pesos con base 10 : 10^n donde n es la posición del dígito dentro del número
- Ejemplo: número 5421

Table 2. Número 5421

Representación:	los símbolos 5421			
Posiciones:	3	2	1	0
Pesos:	$10^3 \rightarrow 1000$	$10^2 \rightarrow 100$	$10^1 \rightarrow 10$	$10^0 \rightarrow 1$
Dígitos:	5	4	5	1
Valores : ponderación	$5 \cdot 1000 = 5000$ =cinco mil	$4 \cdot 100 = 400$ cuatrocien	$5 \cdot 10 = 50$ cuenta	$1 \cdot 1 = 1$ uno
Valor:	$5 \cdot 1000 + 4 \cdot 100 + 5 \cdot 10 + 1 = 5421$			

Representación de los Valores Enteros en Código Binario

Representación de los Valores Enteros en Código Binario

- ¿Número? ¿Valor? ¿Código? ¿Representación?
 - 2 dígitos : 0,1
 - Pesos con base 2 : 2^n donde n es la posición del dígito dentro del número:-1024-512-256-128-64-32-16-8-4-2-1...
- Ejemplo: número 0b1011

Table 3. Número 0b110011

Representación:	los símbolos 1011			
Posiciones:	3	2	1	0
Pesos:	$2^3 \rightarrow 8$	$2^2 \rightarrow 4$	$2^1 \rightarrow 2$	$2^0 \rightarrow 1$
Dígitos:	1	0	1	1
Valores : ponderación	$1 \cdot 8 = 8$ ocho	$0 \cdot 4 = 0$ ceros	$1 \cdot 2 = 2$ dos	$1 \cdot 1 = 1$ uno
Valor:	ocho+ceros+dos+uno= once			

- ¿Cómo se representa en binario el valor 123.125? b1111011.001
- ¿Cómo se calcula el valor del número binario b1111011.001?
- Parte Entera: divisiones sucesivas por la base 2
- Parte Fracción: multiplicaciones sucesivas por la base 2

Representación de los Valores Enteros en Código Octal

- Dígitos: 0,1,2,3,4,5,6,7
- Posiciones y Pesos
- ¿Cómo se representa en octal el valor 123.125? 0o173.1
- ¿Cómo se calcula el valor del número octal 0o173.1?
- Parte Entera: divisiones sucesivas por la base 8
- Parte Fracción: multiplicaciones sucesivas por la base 8

Representación de los Números en Hexadecimal

- Dígitos: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F _ el valor de A es 10, B→11, C→12, D→13, E→14, F→15
- Posiciones y Pesos
- ¿Cómo se representa en hexadecimal el valor 123.125? 0x7B.2
- ¿Cómo se calcula el valor del número octal 0x7B.2?
- Parte Entera: divisiones sucesivas por la base 16
- Parte Fracción: multiplicaciones sucesivas por la base 16

Calculadora de Python

Python Interpreter Shell [\[https://www.programiz.com/python-programming/online-compiler/\]](https://www.programiz.com/python-programming/online-compiler/)

```
bin(123)
oct(123)
hex(123)
int(123)
int(0b1111011)
int(0o173)
int(0x7B)
```

Conversiones entre el sistema binario y sistemas con base potencia de 2

- Conversión Binaria-Hexadecimal
 - base 16=2⁴
 - grupos de 4 bits empezando por la dcha
 - b1111011 → 111 - 1011 → 0x7B
- Conversión Hexadecimal-Binaria
 - grupos de 4 bits
- Conversión Binaria-Octal
 - base 8=2³
 - grupos de 3 bits empezando por la dcha
 - b1111011 → 1 - 111 - 011 → 0o173
- Conversión Octal-Binaria
 - grupos de 3 bits

Suma binaria

- Suma $10011011 + 00011011 = 10110110$

Llevadas -->	1 1 1 1
	1 0 0 1 1 0 1 1 <--sumando
	+ 0 0 0 1 1 0 1 1 <--sumando
Valor suma	1 3 2 1 3 2 *****
Resultado -->	1 0 1 1 0 1 1 0 <--suma

- Cuando la suma en una posición específica tiene un valor es mayor o igual a la base hay que restar n veces la base y el valor n será la llevada a sumar en la posición siguiente.

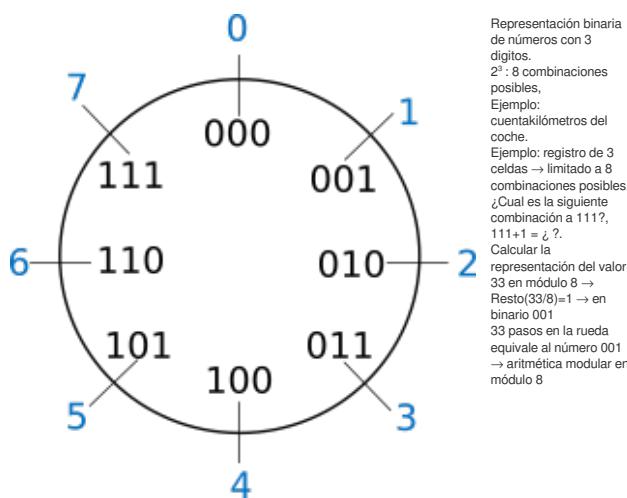
Resta binaria

- Resta $10110110 - 10011011 = 00011011$

Sumar crédito al minuendo	2 2 2 2
	1 0 1 1 0 1 1 0 <--minuendo
- 1 0 0 1 1 0 1 1 <--sustraendo	
Sumar llevada al sustraendo	1 1 1 1
Resta	0 0 0 1 1 0 1 1

- Cuando en una posición específica el minuendo es menor que el sustraendo se suma la base al minuendo antes de realizar la resta y se suma la llevada al sustraendo de la posición siguiente.

Aritmética Modular: la rueda



Operaciones aritméticas: Octal y Hexadecimal

- Base Octal

- $0675+0304 = 01201$
- $0632-0374 = 0236$

- Base hexadecimal

- $0xD1B+0xAF = 0x181A$
- $0xE53-0xBA = 0x2A9$

Representación de Números con Valores Enteros

- Signo-Magnitud
- Complemento a la base menos 1
- Complemento a la base

Representación en Signo-Magnitud

- Signo → un dígito
- Base 10:
 - valores positivos: el signo el dígito 0 en la posición MSD (More Significant Digit) y resto de dígitos representa el módulo
 - valores negativos: el signo el dígito 9 (base-1) en la posición MSD (More Significant Digit) y resto de dígitos representa el módulo
 - Ejemplo +123 → 0123 y -123 → 9123

Representación en Signo-Magnitud

- Signo → un bit (Binary digit)
- Base 2:
 - valores positivos: el signo el bit 0 en la posición MSB (More Significant Bit) y resto de bits representa el módulo
 - valores negativos: el signo el bit 1 (base-1) en la posición MSB (More Significant Bit) y resto de bits representa el módulo
 - Ejemplo +123 → 0b01111011 y -123 → 0b11111011
 - Dibujar la tabla y la rueda con todos los valores con sus representaciones.
 - ¿Cuántas representaciones son posibles? ¿Es simétrico el rango de valores representados? ¿Cuántas representaciones tiene el cero?
 - Extender el número de bits del número sin cambiar su valor

Representación en complemento a la base menos 1. C9

- Base 10: Complemento a 9 → C9
- Signo → un dígito
- Valores positivos: igual que los valores positivos en código Signo-Magnitud
- Valores negativos: Hay que restar el código del valor en positivo del minuendo 99999999 (base-1)
 - Ejemplo +123 → 0123 y -123 → 9999-0123 = 9876
- El C9 de un número positivo es el código de su valor en negativo
- El C9 de un número negativo es el código de su valor en positivo

Representación en complemento a la base menos 1. C1

- Base 2 : base-1=1 → Complemento a 1 → C1
- Signo → un dígito
- Valores positivos: igual que los valores positivos en código Signo-Magnitud
- Valores negativos: Hay que restar el código del valor en positivo del minuendo 11111111 (base-1)
 - Ejemplo '+123' → 0b01111011 y -123 → 11111111-01111011 = 10000100
 - El código del valor negativo se puede calcular invirtiendo los bits del código del valor positivo
- El C1 de un número positivo es el código C1 de su valor en negativo y del de un número negativo es el código C1 de su valor en positivo
 - Dibujar la tabla y la rueda con todos los valores con sus representaciones.
 - ¿Cuántas representaciones son posibles? Es simétrico el rango de valores representado? ¿Cuántas representaciones tiene el cero?
 - Extender el número de bits del número sin cambiar su valor

Representación en complemento a la base 10 : C10

- Signo → un dígito
- Base 10: Complemento a 10 → C10
- Valores positivos: igual que los valores positivos en código Signo-Magnitud
- Valores negativos: Hay que restar el código del valor en positivo del minuendo 00000000 (base)
 - Ejemplo '+123' → 0123 y -123 → 0000-0123 = 9877
- El C10 de un número positivo es el código de su valor en negativo
- El C10 de un número negativo es el código de su valor en positivo

Representación en complemento a la base 2 : C2

- Signo → un dígito
- Base 2: Complemento a 2 → C2
- Valores positivos: igual que los valores positivos en código Signo-Magnitud
- Valores negativos: Hay que restar el código del valor en positivo del minuendo 00000000 (base)
 - Ejemplo +123 → 0b01111011 y -123 → 00000000-01111011 = 0b100000101
 - El código del valor negativo se puede calcular invirtiendo los bits del código del valor positivo y después sumarle 1
 - Equivale a calcular el C1 y sumarle 1
 - El código del valor negativo se puede calcular a partir del código del valor positivo
 - empezando por la dcha repetir los bits hasta el primer uno e invertir el resto de bits

Representación en complemento a la base 2 : C2

- El C2 de un número positivo es el código C2 de su valor en negativo
- El C2 de un número negativo es el código C2 de su valor en positivo
 - Dibujar la tabla y la rueda con todos los valores con sus representaciones.
 - ¿Cuántas representaciones son posibles? Es simétrico el rango de valores representado? ¿Cuántas representaciones tiene el cero?
 - Extender el número de bits del número sin cambiar su valor → Extensión del bit de SIGNO

Extensión del signo en C2

Table 4. Razonamiento de la extensión de signo de un número negativo: números de 3 bits

Valor	C2 sin extensión	C2 con extensión
+33	0100001	00100001
-33	0000000 -0100001 ----- 1011111	0000000 -00100001 ----- 11011111

Se observa que en el C2 con extensión, al hacer la resta y extender con un 0 más el minuendo y el sustraendo, provoca la extensión con un bit más en la resta de valor 1 en el dígito más significante. Según añado ceros al minuendo y sustraendo, aparecen unos en la resta sin alterar su valor.

Operaciones aritméticas en C2

- Suma

- Se realiza como se ha visto para números naturales.
- Si hay llevada en el MSBit, no se tiene en cuenta, se elimina.
- A=0b11011011. Suma A+A

Llevadas -> 1 1 1 1 1

$$\begin{array}{r} 11011011 \\ + 11011011 \\ \hline 2132132 \end{array}$$

Valor suma 2132132

Resultado -> 10110110<--(Valor -74)

- Resta

- La resta de números con signo se puede realizar de dos formas: A-B ó A-B = A+(-B)
- A = 0b00110110 y B = 0b10011011
- Si hay llevada en el MSBit, no se tiene en cuenta, se elimina.

Crédito 2 2 2 2 2 2

$$\begin{array}{r} 10110110 \\ - 11011011 \\ \hline 11011011 \end{array}$$

Llevada 111111

Resta 11011011 (Valor -101)

Operaciones aritméticas C2: Overflow o Desbordamiento

- A = 0b00110110 y B = 0b10011011 → Calcular A-B
- Con 8 bits el máximo valor es 01111111 de valor $2^7 - 1 = 128 - 1 = 127$
- La resta A-(B)=A+(-B)=54+103=157>127 → Overflow o Desbordamiento

Crédito 2 2 2 2 2 2

$$\begin{array}{r} 00110110 \\ - 11011011 \\ \hline 10011011 \end{array}$$

Llevada 111111

Resta 10011011 (Valor -101)

- El valor -101 en lugar de la resta correcta +157 es debido a que el resultado está fuera de rango →
- Observamos que hemos hecho la SUMA de dos números POSITIVOS y el resultado ha sido NEGATIVO

Operaciones aritméticas C2: Overflow

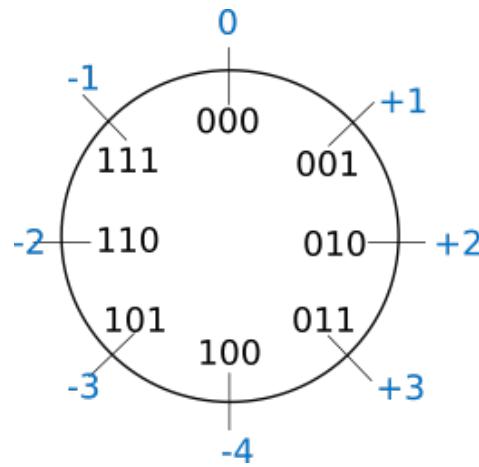


Al realizar la suma de dos valores con el mismo signo si el resultado es de signo contrario hay overflow

Operaciones aritméticas C2: Overflow

- Overflow: la operación requiere operandos con mayor número de bits manteniendo el valor para que el resultado sea correcto.
- Si dos operandos a sumar tienen diferente signo nunca hay overflow
- Si dos operandos a sumar tienen el mismo signo y resultado tiene signo contrario : **Error de Overflow**.
- Ejemplo:
 - Operando de 1 byte : $01111111 + 01111111 = 11111110$ → sumandos positivos y resultado negativo
 - Solución: **Extensión del signo** : Operandos 9 bits → $00111111 + 00111111 = 01111110$
 - la repetición del bit más significativo no altera el valor de la representación
 - el bit más significativo es 0 si es positivo y 1 si es negativo. Por lo tanto, 01010 equivale a 01010 0001010 ó 0...0001010. Por lo tanto, 1010 equivale a 11010 ó 111010 ó 1...111010

C2: Representación gráfica del Overflow



Si a partir de la posición 010 nos movemos dos posiciones en sentido horario llegamos a la posición 100.
Si a 010 le sumamos el valor 2 nos da como resultado 100
Por lo tanto
 $010 + 010 = 100$, es decir,
 $2+2=4 \rightarrow \text{overflow}$ ya que el +4 necesita 4 bits y estamos trabajando con 3 bits únicamente.

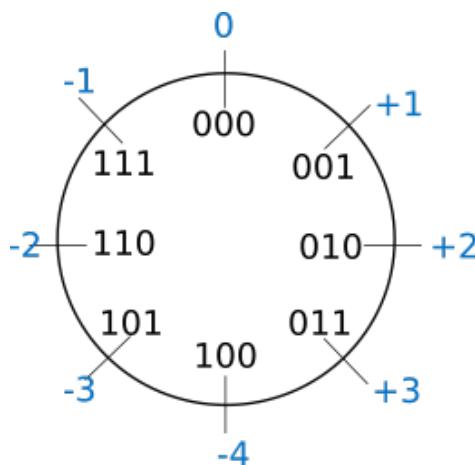
Asimetría del rango en C2: -4 con 3 bits

- Con números de 3 bits los formatos S-M y C1 son simétricos con valores en el rango (+3,-3), en cambio el formato C2 tiene el rango (+3,-4)
- En C2 el valor +4 se representa como 0b0100 y necesita por lo tanto 4 bits, no se puede representar con 3 bits, y el valor -4 se representa con el C2(0100), es decir, 1100 también con 4 bits. El 1100 se puede comprender ya que tiene el signo extendido con la repetición de 1 de bit más significativo, por lo que la representación 100 es la representación del -4

Complemento a 2 : Ejemplos

- 0b101010101 está en C2 → ¿Cuál es su valor?
 - como es negativo no es un sistema posicional
 - tenemos que calcular el valor negativo a través del valor positivo
 - La representación del valor positivo es el C2 del valor negativo
 - $C2(0b101010101) = 0b010101011$ cuyo valor es $2^7 + 2^6 + 2^3 + 2^1 + 2^0 = 128 + 32 + 8 + 2 + 1 = 171$
 - El valor de 0b101010101 es -171
- Si la representación de -123 es 0b100000101 ¿cuál es la de '+123' ?
 - $C2(0b100000101) = 0b011111011$ representa el valor '+123'

Aritmética Modular de valores representados en Complemento a 2



Representación de números binarios de 3 bits en C2
 Operaciones de suma y resta modular → método gráfico
 A partir de la posición 001 si nos movemos en sentido horario (SUMA modular) 2 posiciones obtenemos la posición 011, es decir, $1+2=3$
 A partir de la posición 110 si nos movemos en sentido horario (SUMA modular) 9 posiciones obtenemos la posición 111, es decir, $-2+9=1$
 A partir de la posición 110 si nos movemos en sentido antihorario (RESTA modular) 4 posiciones obtenemos la posición 010, es decir, $-2-4=-2$
 Los errores de **overflow** se resuelven aumentando el número de bits de la

representación, pero siempre existirá un rango que si lo traspasamos dará overflow.

Comparación S-M, C1 y C2

Table 5. Números de 3 bits

Valor	S-M	C1	C2
+3	011	011	011
+2	010	010	010
+1	001	001	001
0	000	000	000
	100	011	---
-1	101	110	111
-2	110	101	110
-3	111	100	101
-4	-	-	100

Número en complemento a 2 y base hexadecimal



Un número binario se puede representar en hexadecimal y hacer la interpretación en complemento a 2. Hay que tener cuidado con las extensiones del signo

- Calcular el valor del número 0xAAA si dicho número tiene formato en complemento a 2
 - si lo convertimos a binario el número empieza por 1, luego es negativo
 - para saber su valor calculo su complementario C2 y tendrá la representación del positivo
- 0x000-0xAAA = 0x556 → $5 \cdot 16^6 + 5 \cdot 16^5 + 5 \cdot 16^4 = 5 \cdot 256 + 5 \cdot 16 + 5 = 1280 + 80 + 5 = +213 \rightarrow 0xAAA$ tiene de valor -213

Número en complemento a 2 y base hexadecimal

- Realizar la suma de los números en formato complemento a 2: 0x80+0x80
 - sumar sin extender el signo de los operandos ¿Hay overflow?



Extender el número 0x80. ¿Por qué hay que tener cuidado ?

- sumar extendiendo un dígito el signo de los operandos 0x80

Extensión del signo en C2: problema de la BASE

Table 6. Extensión del Signo del N° 0x80 en C2 en binario, hexadecimal y octal

NºBits	Binario	Hexadecimal	Octal
8	10000000	1000_0000 → 0x80	110_000_000 → 0o600
9	110000000	1111_1000_0000 → 0xF80	110_000_000 → 0o600
10	1110000000	1111_1000_0000 → 0xF80	111_110_000_000 → 0o7600
11	11110000000	1111_1000_0000 → 0xF80	111_110_000_000 → 0x7600
12	111110000000	1111_1000_0000 → 0xF80	111_110_000_000 → 0x77600
13	1111110000000	1111_1111_1000_0000 → 0xFF80	111_111_110_000_000 → 0x77600

Suma y Resta aritmética en C1

- Ejemplos con datos de 4 bits → Rango (-7,+7).
- ¿Qué ocurre si **sumamos** dos números sin que haya overflow.
- Primer caso: dos operandos positivos 0011+0011=0110 → correcto.
- Segundo caso: dos operandos negativos donde todas ellas tienen acarreo en el MSB
--- 1111+1111=1110 → 0+0=-1; 1100+1100=1000 → -3-3=-7.; 1100+1011=0111 → -3-3=+7.
- el valor del resultado siempre da una **unidad menor** y siempre hay acarreo en el MSB.
- Solución: la suma en C1 es la suma de los sumandos **más el acarreo MSB**.
- 1111+1111=1110+1 → 0 ; 1100+1100=1000+1 → -6 ; 1100+1011=0111+1=1000 → -7.
- ¿Qué ocurre si **restamos** dos números?
- El resultado es correcto si no hay acarreo MSB.
- Si hay acarreo la resta da una **unidad mayor**, por lo tanto el resultado es la resta **menos uno**.
- El resultado es la resta **menos el acarreo MSB**.
- Caso de overflow: el resultado es incorrecto por limitación del tamaño de los datos

| 73 |

Suma aritmética en C1

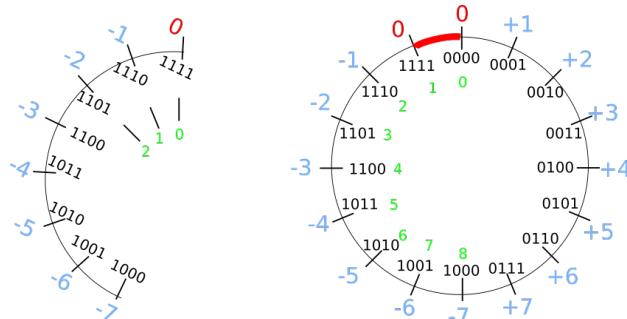


Figure 1. Razonamiento

Si nos fijamos en los números negativos su valor está desplazado una unidad de la rueda hacia la izda: el valor cero está en la marca 1 de la rueda, el valor -1 está en la marca 2 de la rueda, etc. El valor representado por el segmento perimetral de la circunferencia tiene una unidad inferior: el valor -1 tiene un segmento de 2 marcas del 0 hacia la izda.

El segmento o distancia del cero (1111) es UNO, el del -1 es DOS, el del -2 es TRES.....

| 74 |

Suma aritmética en C1

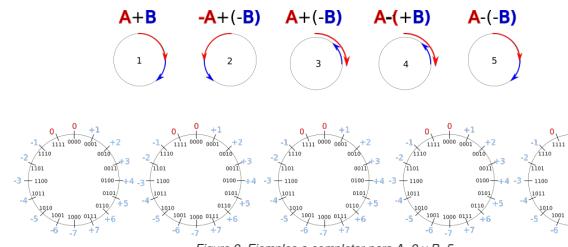


Figure 2. Ejemplos a completar para A=2 y B=5

| 75 |

Suma y Resta aritmética en C1

- Ej_1: suma: no hay conflicto $\rightarrow 2+5=7 \rightarrow 0010+0101=0111$
- Ej_2: suma: si hay conflicto : el segmento B por ser negativo tiene un segmento superior en una unidad, por lo que a la suma hay que **sumarle 1** $\rightarrow -2+(-5)=-7 \rightarrow 1101+1010=0111$ y acarreo_MSB $\rightarrow 0111+1=1000 \rightarrow -7$
- Ej_3: suma
 - si la flecha B cruza el cero (1111), el segmento B, por ser negativa tiene un segmento mayor en 1, compensa el desplazamiento entre ceros y la suma es correcta. $\rightarrow 2+(-5)=-3 \rightarrow 0010+1010=1100 \rightarrow -3$
 - si la flecha B no cruza el cero (1111), a la suma hay que **sumarle 1**
- Ej_4: resta
 - si la flecha B cruza el cero (1111), B por ser positivo no compensa el desplazamiento entre ceros. A la resta hay que **restarle 1** $\rightarrow 2-(+5)=-3 \rightarrow 0010-0101=1101$ y acarreo_MSB $\rightarrow 1101-1=1100 \rightarrow -3$
- Ej_5: resta: no cruza el cero (1111), B por ser negativo tiene un segmento mayor en 1. A la resta hay que **restarle 1** $\rightarrow 2-(-5)=+7 \rightarrow 0010-1010=1000$ y acarreo_MSB $\rightarrow 1000-1=0111 \rightarrow +7$

Números Reales Binarios

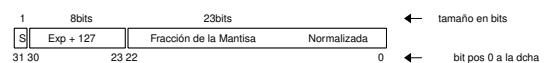
- Coma Fija
 - 123.125 $\rightarrow b1111011.001$
- Coma flotante
 - Notación científica: potencias en la base del sistema (decimal, binario, etc)
 - En decimal $\rightarrow 1.23125 \cdot 10^2$
 - En binario $\rightarrow 1.111011001 \cdot 2^6$
 - el factor que no es potencia se denomina mantisa
 - Se dice que el número real en binario y en notación científica está normalizado si la parte entera de la mantisa vale 1.

Números Reales Binarios: Norma IEEE-754

- Campos del formato en notación científica : Signo, parte entera de la mantisa, parte fracción de la mantisa, base , exponente (módulo y signo)
- ¿Es necesario representar los **seis** campos del formato de la notación científica?
 - Si esta normalizado la parte entera de la mantisa siempre vale 1
 - La base del factor potencia siempre vale 2
 - Por lo tanto la parte entera y la base no son necesario representarlas. Son implícitas a la representación.
 - Hay una forma de no tener que representar el signo del exponente del factor potencia
 - es sumarle una cantidad para que al representarlo en EXCESO siempre sea positivo
- Resumiendo, sólo es necesario representar: el signo del número , la fracción de la mantisa y el exponente en exceso. Por lo tanto el formato IEEE-754 tiene 3 campos.

Formato IEEE-754 simple

- representación de 32 bits: 1 bit para el signo / 8 bits para el exponente en exceso a 127 / 23 bits para la fracción
- bit de signo: 0 si es positivo y 1 si es negativo



Formato IEEE-754 doble

- representación de 64 bits: 1 bit para el signo / 11 bits para el exponente en exceso a 1023 / 52 bits para la fracción
- bit de signo: 0 si es positivo y 1 si es negativo



El N° -5.5/1024 en los dos Formatos IEEE-754

- Signo negativo
- Conversión binaria del módulo
 - módulo: $5.5/1024 = 5.5 \cdot 2^{-10} = 101.1 \cdot 2^{-10}$
 - Normalización de la mantisa $\rightarrow 1.011 \cdot 2^8$
- Formato Simple de 32 bits
 - Signo negativo: bit 1
 - Exponente en exceso $127 - 8 + 127 = 119 = 01110111$
 - Fracción de la mantisa=011
 - Solución:
 - $1_01110111_01100000000000000000000000000000$
 - $0b1011101110110000000000000000000000000000$
 - $0xBBB00000$
 - [calculadora ieee](http://weitz.de/ieee/) [<http://weitz.de/ieee/>]
- Formato Doble de 64 bits
 - Signo negativo: bit 1
 - Exponente en exceso $1023 - 8 + 1023 = 1015 = 0111110111$
 - Fracción de la mantisa=011
 - Solución:
 - $1_0111110111_0110...0$
 - $0b10111110110110...0$
 - $0xBF76000000000000$

| 81

| 82 |

Representación de los Caracteres

Representación de los Caracteres

- Tipos de Caracteres:
 - Alfanuméricos: a,b,...,z,0,1,...,9,A,B...,Z
 - Signos de Puntuación: !#\$%&/()=
 - de Control: Salto de Línea (\n), Find de Fichero (EOF), Fin de String (\00, ...)
- Formatos
 - ASCII: standard y extendido
 - Unicode: UTF-8

ASCII Standard

2	3	4	5	6	7	30	40	50	60	70	80	90	100	110	120
0:	0	0	P	`	p	0:	(2	<	F	P	Z	d	n	x
1:	!	1	A	Q	a	q)	3	=	6	Q	[e	o	y
2:	"	2	B	R	b	r	*	4	>	H	R	\	f	p	z
3:	#	3	C	S	c	s	!	+5	?I	S	J	g	q	{	
4:	\$	4	D	T	d	t	"	,	6	@	J	T	^h	r	
5:	%	5	E	U	e	u	#	-7	A	K	U	_i	s	}	
6:	&	6	F	V	f	v	\$.	8	B	L	V	`j	t	~
7:	'	7	G	W	g	w	%	/9	C	M	W	a	k	u	DEL
8:	(8	H	X	h	x	&	0	:	D	N	X	b	l	v
9:)	9	I	Y	i	y	1	;	E	O	Y	c	m	w	
A:	*	:	J	Z	j	z	0	;	K	[k	{			
B:	+	;	K	[k	{	C:	<	L	\	l	}			
D:	-	=	M]	m	}	E:	>	N	^	n	~			
F:	/	?	O	_	o	DEL									

American Standard Code for Information Interchange

Alfabeto anglosajón

7 bits → $2^7=128$ caracteres : 0x00 hasta 0x1F son 32 caracteres de control y el resto alfanuméricos

En hexadecimal rango [0x00-0x7F]

En decimal rango [0-127]

Upna : 0x55706E61

año 2023: 0x61—6F2032303233

ASCII Extendido

- Para poder representar caracteres de otras culturas Europeas es necesario expandir el standard con 1 bit más

- ASCII 8 bits → $2^8 = 256$ caracteres

- [Python Interpreter Shell](https://www.programiz.com/python-programming/online-compiler/) [https://www.programiz.com/python-programming/online-compiler/]

```
ord('A')
hex(ord('A'))
hex(ord('\n'))
chr(65)
chr(0x41)
[hex(ord(c)) for c in "Hola"]
[chr(c) for c in [0x48, 0x6f, 0x6c, 0x61, 0x20, 0x4d, 0x75, 0x6e, 0x64, 0x6f]]
[hex(ord(c)) for c in "\ñ"]
[hex(ord(c)) for c in "\n \t"]
```

La ñ tiene el código ASCII 0xF1

UTF-8

- [Character Set, HTML Converter, etc ...](https://www.charset.org/utf-8) [https://www.charset.org/utf-8]
- Unicode Transformation Format (UTF)
- UTF-8: Esta orientado a la transmisión de palabras de 1 byte
- Los caracteres pueden tener entre 1 y 4 bytes → 2^{21} code points ≈ 2 millones;
- The dominant encoding on the World Wide Web and on most Unix-like operating systems
- En linux comando `localectl status` : informa sobre el sistema del teclado
- ñ:
 - hex code 0xC3B1
 - unicode point U+00F1 → los primeros 256 caracteres equivalen al ascii extendido

Unicode Points

- [html css js online](https://html-css-js.com/) [https://html-css-js.com/]: �f1;
- U+2228: ∨
- U+22bc: ⚩
- U+22bd: ⚪
- U+22a6: ↨
- U+1f60b: 🌉
- U+00f1: ñ
- OrduU+00F1a: Orduña
- U+2190: ←
- U+2192: →

Tema 2: Ejercicios

- Mialario/Recursos/Ejercicios
- Fundamentos de sistemas digitales Thomas Floyd

**Tema 3 : Algebra de Comutación ó Boole.
Funciones Lógicas.**

| 89

90 |

Matemática Lógica Binaria

- Valores Lógicos Binarios : "0", "1"
 - representa dos estados: los estados de una señal binaria (High/Low), los estados de una bombilla (encendido/apagado), de un conmutador (on/off), de una condición (verdadero/falso), etc, cualquier situación que se pueda modelar mediante dos estados.
- Variables lógicas: ...u, x1, x2, y, v1, u2, ...
 - Una variable independiente que puede tomar los valores "0" y "1"
- Función lógica: z1, z2, z3, F, ...
 - Una función lógica expresa una relación lógica o/y aritmética o/y comparativa o/y etc entre las variables independientes a través de unos operadores matemáticos.
- Operadores
 - Operadores aritméticos: suma, resta, multiplicación, ...
 - Operadores lógicos: or (suma), and (producto), negación, or exclusiva, etc...
 - Operadores comparadores: > , >, ==, etc

Tablas de la Verdad de los operadores NOT, OR, AND, XOR

Table 7. NOT

x	z = \bar{x}
0	1
1	0

Table 8.
OR

x	y	z = $x + y$
0	0	0
0	1	1
1	0	1
1	1	1

Table 9.
AND

x	y	z = $x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

Table 10.
XOR

x	y	z = $x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

| 91

92 |

Tablas de la Verdad de los operadores NOR, NAND, XNOR

Table 11.
NOR

x	y	$z =$
		$x + y$
0	0	1
0	1	0
1	0	0
1	1	0

Table 12.
NAND

x	y	$z =$
		$x \cdot y$
0	0	1
0	1	0
1	0	0
1	1	0

Table 13.
XNOR

x	y	$z =$
		$x \oplus y$
0	0	1
0	1	0
1	0	0
1	1	1

Puertas Lógicas

NOT



OR



AND



XOR



NOR



NAND



XNOR

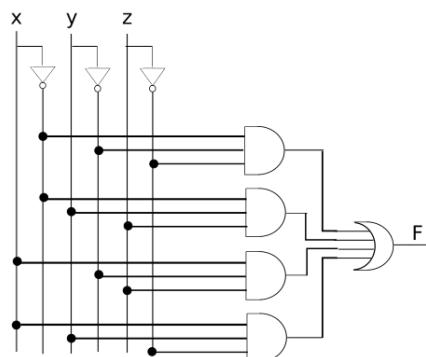
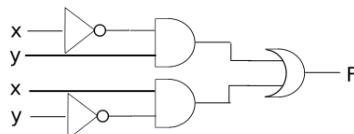


Circuitos Digitales : Expresiones Lógicas

$$F(x,y) = \bar{x}\bar{y} + x\bar{y}$$

$$F(x,y,z) = \bar{x}\bar{y}\bar{z} + xy\bar{z} + \bar{x}yz + xy\bar{z}$$

Circuito digital en 3 niveles: not-and-or.

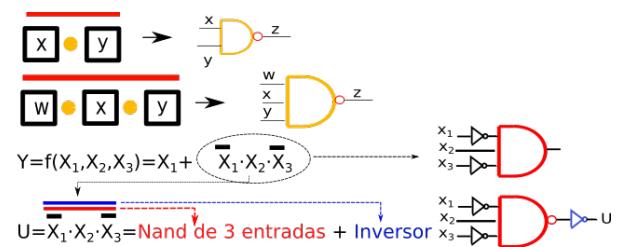


Transparencias PDF: Miaulario/Recursos/Apuntes

- Postulados del Álgebra de Boole
- Teoremas del Álgebra de Boole (Leyes de Morgan)
- $(a'b) + (c'd) = a'b + c'd$
- Generación de funciones con puertas lógicas: Ejemplo 1 a)yb)
- Simplificación de funciones mediante Teoremas: Al final
- Formas canónicas: Sum of Products (SOP) y Product of Sums (POS)
 - minitérminos y maxitérminos
 - Ejemplos básicos
- Diagramas de Karnaugh (DK)
 - Agrupar celdas adyacentes en potencias de 2^n
 - Ejemplos básicos
- Relación SOP-POS
 - ejemplo1: $a+ab$
 - ejemplo2: general 3 variables x,y,z
- Simplificación de funciones mediante Teoremas
 - Extender los términos como minitérminos
 - Dibujar DK y agrupar celdas equivale a sacar factor común

Link: álgebra de conmutación funciones.pdf

- Algebra de Boole. Funciones Lógicas [\[PDF/03_algebra_de_commutacion_funciones_logicas.pdf\]](#)



$$= \overline{x}_1 + \overline{x}_1 \cdot \overline{x}_2 \cdot \overline{x}_3 =$$

$$= \overline{\bar{X}_1} \cdot \overline{\bar{X}_1} \cdot \overline{X_2} \cdot \overline{X_3} =$$

10

98 |

Formas Canónicas de una Función: Síntesis por minitérminos y maxitérminos

- Hay dos formas canónicas (standard) de expresar una función
 - suma de productos (SOP) de variables
 - producto de sumas (POS) de variables

Lógica Positiva/Negativa: Relación y/o con */+

- Lógica positiva → ¿Cuando vale **1** una función, una expresión, una variable, etc ?
 - Lógica negativa → ¿Cuando vale **0** una función, una expresión, una variable, etc ?

Table 14
OR

x	y	$z =$
		$x +$
		y
0	0	0
0	1	1
1	0	1
1	1	1

Z= 0 si "X" e "Y" valen 0 $\rightarrow z=(x+y)$
Z= 1 si "X" o "Y" valen 1 $\rightarrow z=(x+y)$

Table 15
AND

x	y	$z =$
0	0	0
0	1	0
1	0	0
1	1	1

Z = 0 si "X" o "Y" valen 0 → (z=x + y)
Z = 1 si "X" e "Y" valen 1 → (z=x + y)

| 99

100 |

Forma Canónica SOP: Suma de Minitérminos

Table 16. Tabla de la Verdad de la Función $F(x_1, x_2, x_3)$

x1	x2	x3	F	minitérminos
0	0	0	0	$m_0 : \bar{x}_1\bar{x}_2x_3$
0	0	1	0	$m_1 : \bar{x}_1\bar{x}_2\bar{x}_3$
0	1	0	0	$m_2 : \bar{x}_1x_2\bar{x}_3$
0	1	1	1	$m_3 : \bar{x}_1x_2x_3$
1	0	0	1	$m_4 : x_1\bar{x}_2\bar{x}_3$
1	0	1	1	$m_5 : x_1\bar{x}_2x_3$
1	1	0	0	$m_6 : x_1x_2\bar{x}_3$
1	1	1	0	$m_7 : x_1x_2x_3$

Lenguaje natural $\rightarrow F$ vale 1 (lógica positiva) si $m_3 \circ m_4 \circ m_5$ vale 1 \rightarrow suma

Lenguaje natural $\rightarrow m_3$ vale 1 (lógica positiva) si \bar{x}_1 y x_2 y x_3 valen 1 \rightarrow multiplicación

Lenguaje lógico $\rightarrow F = SOP = m_3+m_4+m_5$.

$F(x_1, x_2, x_3) = \bar{x}_1x_2x_3 + x_1\bar{x}_2x_3 + x_1x_2\bar{x}_3$.

Cada minitérmino se sintetiza mediante una puerta AND.

La síntesis de la función F tendría un nivel de pueras AND de 3 entradas y un nivel con una puerta OR con tantas entradas como minitérminos hacen 1 a la función.

Forma Canónica POS: Producto de Maxiterminos

Table 17. Tabla de la Verdad de la Función $F(x_1, x_2, x_3)$

x1	x2	x3	F	maxitérminos
0	0	0	0	$M_0 : x_1+x_2+x_3$
0	0	1	0	$M_1 : x_1+x_2+\bar{x}_3$
0	1	0	0	$M_2 : x_1+\bar{x}_2+x_3$
0	1	1	1	$M_3 : x_1+\bar{x}_2+\bar{x}_3$
1	0	0	1	$M_4 : \bar{x}_1+x_2+x_3$
1	0	1	1	$M_5 : \bar{x}_1+x_2+\bar{x}_3$
1	1	0	0	$M_6 : \bar{x}_1+\bar{x}_2+x_3$
1	1	1	0	$M_7 : \bar{x}_1+\bar{x}_2+\bar{x}_3$

Lenguaje natural $\rightarrow F$ vale 0 (lógica negativa) si $M_0 \circ M_1 \circ M_2 \circ M_6 \circ M_7$ vale 0 \rightarrow multiplicación

Lenguaje natural $\rightarrow M_1$ vale 0 (lógica negativa) si x_1 y x_2 y \bar{x}_3 valen 0 \rightarrow suma

Lenguaje lógico $\rightarrow F = POS = M_0M_1M_2M_6M_7$.

$F(x_1, x_2, x_3) = (x_1+x_2+x_3)(x_1+x_2+\bar{x}_3)(x_1+\bar{x}_2+x_3)(\bar{x}_1+\bar{x}_2+x_3)(\bar{x}_1+\bar{x}_2+\bar{x}_3)$.

Cada maxitérmino se sintetiza mediante una puerta OR.

La síntesis función F tendría un nivel de pueras OR de 3 entradas y un nivel con una puerta AND con tantas entradas como maxitérminos hacen 1 a la función.

Relación entre la forma canónica SOP y POS

- Ejemplo $F = F(x_1, x_2, x_3) = m_3+m_4+m_5$
- $F = m_0+m_1+m_2+m_6+m_7$
- $F = m_0+m_1+m_2+m_6+m_7 =$
- $F = \bar{m}_0 \cdot \bar{m}_1 \cdot \bar{m}_2 \cdot \bar{m}_6 \cdot \bar{m}_7$
- $F = M_0 \cdot M_1 \cdot M_2 \cdot M_6 \cdot M_7 = F$

Simplificación de las funciones mediante los Diagramas de Karnaugh (DK)

- El Diagrama de Karnaugh es una representación gráfica multidimensional (2D, 3D, etc) mediante celdas de los minitérminos y maxitérminos de la tabla de la verdad unidimensional 1D
- Ejemplo $F(x_1, x_2, x_3) = \bar{x}_1x_2x_3 + x_1\bar{x}_2\bar{x}_3 + x_1\bar{x}_2x_3$.
 - los minitérminos y maxitérminos siguen la secuencia unidimensional 000-001-010-011-100-101-110-111
 - Los reorganizamos en una matriz de celdas **adyacentes**, donde dos celdas adyacentes tienen todas las variables comunes **excepto una**
- **Simplificación:**
 - Agrupar celdas adyacentes en grupos de un número de celdas potencia de dos $\rightarrow 2^n : 2, 4, 8, \text{etc} \dots$
 - Cuanto mayor sea el número de celdas agrupadas mayor será el número de variables y términos simplificados.

Diagrama de Karnaugh de la función F(x1,x2,x3)

		x_1x_2			
		00	01	11	10
x_3	0	0	0	0	1
	1	0	1	0	1

celdas adyacentes: todas las variables comunes menos una.

		x_1x_2			
		00	01	11	10
x_3	0	m_0/M_0	m_2/M_2	m_6/M_6	m_4/M_4
	1	m_1/M_1	m_3/M_3	m_7/M_7	m_5/M_5

miniter/maxiter adyacentes: todas las variables comunes menos una

Simplificación de la Función mediante DK

		x_1x_2			
		00	01	11	10
x_3	0	0	0	0	1
	1	0	1	0	1

celdas adyacentes: todas las variables comunes menos una.

		x_1x_2			
		00	01	11	10
x_3	0	m_0/M_0	m_2/M_2	m_6/M_6	m_4/M_4
	1	m_1/M_1	m_3/M_3	m_7/M_7	m_5/M_5

miniter/maxiter adyacentes: todas las variables comunes menos una

Si sumamos los minitermos de la ^{4^a} columna $Y = f(x_1, x_2, x_3) = m_4 + m_5 = x_1\bar{x}_2x_3 + x_1\bar{x}_2x_3 = x_1\bar{x}_2(\bar{x}_3 + x_3) = x_1\bar{x}_2$, se ha simplificado el número de variables de 3 a dos. La función simplificada es $Y = \bar{x}_1x_2x_3 + x_1\bar{x}_2$. Sumar dos minitermos adyacentes equivale a agrupar dos celdas adyacentes y reducir una variable.

- Son adyacentes las celdas de la misma columna o de la misma fila con todas las variables comunes menos una. Por eso la tercera columna ha de ser 11

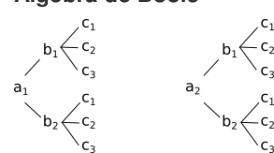
- Observar que cada celda equivale a un minitermino y un maxitermino de la Tabla de la verdad

- Por tanto, el diagrama DK representa las formas canónicas SOP y POS.

Ejercicios básicos matemáticos en el dominio del Algebra de Boole

- $x * 0 = 0; x * 1 = x; x * \bar{x} = 0; x + \bar{x} = 1$
- $x = x * 1 = x * (y + \bar{y}); x = x + 0 = x + y * \bar{y}$
- $x \cdot (x + u + v + ...) = x; x + (x \cdot u \cdot v \cdot ...) = x$
- Transformar una suma de productos de variables lógicas en producto de sumas de variables lógicas
 - $F = y\bar{z} + x\bar{y} + x\bar{y}z$
 - cambio de nomenclatura para facilitar la explicación $F = a_1a_2 + b_1b_2 + c_1c_2c_3$

Ejercicios básicos matemáticos en el dominio del Algebra de Boole



$$F = a_1a_2 + b_1b_2 + c_1c_2c_3 = (a_1 + b_1 + c_1)(a_1 + b_1 + c_2)(a_1 + b_1 + c_3) \cdot (a_1 + b_2 + c_1)(a_1 + b_2 + c_2)(a_1 + b_2 + c_3) \cdot (a_2 + b_1 + c_1)(a_2 + b_1 + c_2)(a_2 + b_1 + c_3) \cdot (a_2 + b_2 + c_1)(a_2 + b_2 + c_2)(a_2 + b_2 + c_3)$$

$$F = y\bar{z} + x\bar{y} + x\bar{y}z = (y + x + x'y)(y + x + \bar{y})(y + x + z) \cdot (y + \bar{y} + x)(y + \bar{y} + \bar{y})(y + \bar{y} + z) \cdot (\bar{z} + x + x)(\bar{z} + x + \bar{y})(\bar{z} + x + z) \cdot (\bar{z} + \bar{y} + x)(\bar{z} + \bar{y} + \bar{y})(\bar{z} + \bar{y} + z)$$

Ejercicios básicos matemáticos en el dominio del Algebra de Boole

Simplificación

$$F = (y + x)(1)(y + x + z)(1)(1)(\bar{z} + x)(\bar{z} + x + \bar{y})(1)(\bar{z} + \bar{y} + x)(\bar{z} + \bar{y})(1) = (y + x)(y + x + z)(\bar{z} + x)(\bar{z} + x + \bar{y})(\bar{z} + \bar{y} + x)(\bar{z} + \bar{y})$$

POS → Expansión para que tenga cada término las 3 variables

$$F = (y + x + z\bar{z})(y + x + z)(\bar{z} + x + y\bar{y})(\bar{z} + \bar{y} + x)(\bar{z} + \bar{y} + x\bar{x})$$

Aplico la propiedad Distributiva a cada término

$$F = (y + x + z)y + (y + x + z)z + (\bar{z} + x + y\bar{y})\bar{z} + (\bar{z} + \bar{y} + x)\bar{z} + (\bar{z} + \bar{y} + x\bar{x})\bar{z}$$

Ejercicios básicos matemáticos en el dominio del Algebra de Boole

F en la 1ª forma canónica

$$\bullet F = y\bar{z} + x\bar{y} + x\bar{y}\bar{z} = y\bar{z} \cdot (x + \bar{x}) + x\bar{y} \cdot (z + \bar{z}) + x\bar{y}\bar{z} = y\bar{z}x + y\bar{z} \cdot \bar{x} + x\bar{y}z + x\bar{y}\bar{z} + x\bar{y}z = xy\bar{z} + \bar{x}y\bar{z} + x\bar{y}\bar{z} = m_6 + m_2 + m_5 + m_4$$

Nominación Teoremas

- commutativa $a \cdot b =$
- idempotencia $a + a =$
- identidad $a \cdot 1 =$
- complementario $a + \bar{a} =$
- absorción $a + ab =$
- distributiva $ab + cd =$
- $a+1$
- a^*0

Simplificación de funciones mediante axiomas y teoremas del Algebra de Boole



celdas adyacentes equivale a minitéminos con factores comunes, que pueden ser agrupados y simplificados.

- Ejemplo 1: $Y=f(x_1,x_2,x_3)=\bar{x}_1\bar{x}_2x_3 + \bar{x}_1\bar{x}_2x_3 + x_1\bar{x}_2x_3 + x_1\bar{x}_2x_3 + x_1x_2x_3 + x_1x_2x_3$
- Dibujar la TV y el DK de la función Y
- Simplificar la función Y mediante el agrupamiento de celdas en el DK
- Partiendo del agrupamiento DK razonar la simplificación de la función Y mediante los **axiomas y teoremas del álgebra de Boole**.

Simplificación de funciones mediante el Diagrama de Karnaugh

- Agrupar celdas adyacentes en grupos de un número de celdas 2^n : 2, 4, 8, etc ...
- Cuanto mayor sea el número de celdas agrupadas mayor será el número de variables y términos simplificados.
- $y = f(x_1, x_2, x_3, x_4) = \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1\bar{x}_2x_3 + \dots$
- $y = f(x_1, x_2, x_3, x_4) = \sum(m_0 + m_1 + m_3 + m_4 + m_5 + m_7 + m_9 + m_{11} + m_{13} + m_{14} + m_{15})$
- $y = f(x_1, x_2, x_3, x_4) = \sum(0, 1, 3, 4, 5, 7, 9, 11, 13, 14, 15)$
- Simplificar la función "y" tanto simplificando la forma SOP como simplificando la forma POS y dibujar el resultado de la síntesis.

Ejercicios matemáticos en dos dominios Gráfico/Algebra de Boole

- $F = f(x_1, x_2) = x_1 = x_1 + x_2\bar{x}_2$
 - obtener la forma canónica SOP y POS mediante TV y DK
 - obtener la forma canónica SOP analíticamente: propiedad identidad
 - obtener la forma canónica POS analíticamente: propiedad distributiva
 - convertir la forma canónica POS a SOP mediante la equivalencia entre minitérminos y máxitérminos
 - convertir la forma canónica POS a SOP analíticamente
- $F = f(x_1, x_2, x_3) = x_1\bar{x}_2x_3 + x_1x_2\bar{x}_3 + x_2x_3$
 - obtener analíticamente la forma canónica SOP
 - primero expandir por la propiedad distributiva
 - simplificar cada factor
 - simplificar factores repetidos
 - extender cada factor para que tenga las 3 variables
 - aplicar la propiedad distributiva

Cuaderno de Ejercicios: Capítulo 1

- 1.1, 1.2, 1.4, 1.5, 1.6, 1.8, 1.9
- Metodología: antes de comenzar a resolver el ejercicio hay que describir el método a seguir para resolver el ejercicio.

Planteamiento de los Ejercicios Capítulo 1

- Ejercicio 1.1
 - Resolverlo primero por DK
 - asociar DK con álgebra de Boole
 - SOP,POS,factor común,ordenar,simplificar
- Ejercicio 1.2
 - Análisis,TV(combinaciones repeticiones)
 - variable indiferente → valor X
 - variable nula → TV y DK reducidas
- Ejercicio 1.3
 - lenguaje natural → lenguaje lógico
 - F=SOP
 - lógica positiva (o/y → */+ → F=1 si ...)
 - lógica negativa (o/y → */+ → F=0 si ...)
 - deducir máxitérminos y miniterminos
 - F=X si ...
 - Función: valor no definido: X
 - DK : definición libre para simplificar: 0 ó 1

Planteamiento de los Ejercicios Capítulo 1

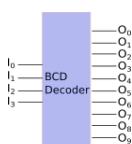
- Ejercicio 1.4
 - Resolverlo por DK
 - Formato ajedrez → Factor Común → XOR
- Ejercicio 1.5
 - lenguaje natural → lenguaje lógico
 - lógica positiva - lógica negativa
 - o/y → */+
 - Condiciones → miniterminos o maxiterminos

Planteamiento de los Ejercicios Capítulo 1

- Ejercicio 1.6
 - lenguaje natural → lenguaje lógico
 - lógica positiva - lógica negativa
 - o/y → */+
 - Condiciones → miniterminos o maxiterminos
- Ejercicio 1.8
 - Escenificación → Diferentes Casos
 - Entro al pasillo por la izda y salgo por la dcha
 - Entro al pasillo por la izda y salgo por la izda
- Ejercicio 1.9
 - Señal binaria: Relación de aspecto
 - Periodo: Duración nivel alto respecto nivel bajo
 - Módulos o subcircuitos:
 - Anidamiento de funciones → subfunciones

Binary Coded Decimal (BCD)

- El código binario BCD codifica, cada dígito decimal de un número, de forma directa con 4 bits para cada dígito decimal.
- Ejemplos
 - 23 → 0010 0011
 - 87045 → 1000 0111 0000 0100 0101
- Diseñar un circuito digital simplificado que decodifique el código binario BCD en uno de los diez dígitos: 0,1,...,9



- códigos y dígitos

Código	Dígito Decimal
0000	0
0001	1
0010	2
0011	3
0100	4

- códigos y dígitos

Código	Dígito Decimal
0101	5
0110	6
0111	7
1000	8
1001	9

- códigos y dígitos

Código	Dígito Decimal
1010	X
1011	X
1100	X
1101	X
1101	X

Tema 4: Lenguaje de Descripción Hardware VHDL

Very high speed integrated circuits Hardware Description Language (VHDL)

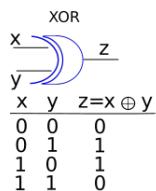
- HDL: Hardware Description Languages

- NO son lenguajes de programación sino de **descripción de Hardware**. Es una lenguaje que está pensado para describir circuitos de la misma forma que otras formas de describir un circuito digital: mediante un esquema eléctrico, mediante una tabla de la verdad, mediante diagramas de secuencias de estados, etc ...
- También sirve para describir las formas de onda cuadradas de las señales binarias de entrada de un circuito digital
- ... y por supuesto también tiene sentencias y estructuras de programación que no describen circuitos digitales, por ejemplo imprimir en la pantalla una frase como "Hello World".

Descripción del Hardware de un circuito digital.

```
-- Descripción VHDL Primavera 2023
-- Circuito light_bit.vhd:
-- Puerta lógica XOR extendida
entity of light_bit is
port (
    x,y : in bit;
    z   : out bit
);
end entity;

architecture rtl of light_bit is
signal s,t,u,v : bit;
begin
    s <= not x;
    t <= not y;
    u <= x and t;
    v <= y and s;
    z <= u or v;
end rtl;
```



$$z = x \oplus y = x \bar{y} + \bar{x} y$$

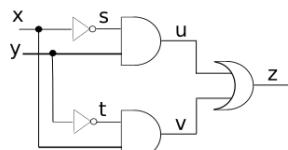
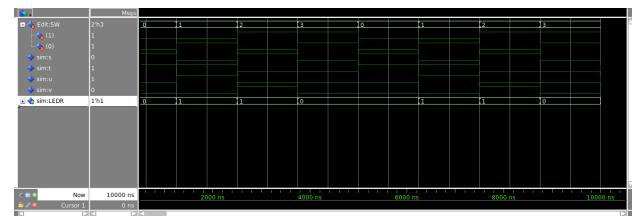


Figure 3. Circuito light_bit.vhd

Cronogramas



Señales VHDL : tipos

- Hay dos tipos de señales en el lenguaje vhdl:
 - PORT : x,y,z
 - son señales de acceso al circuito: su modo puede ser de entrada (IN) ó de salida (OUT)
 - una señal de entrada tipo IN no puede conectarse a la salida de una puerta lógica
 - una señal de salida tipo OUT no puede conectarse a la entrada de una puerta lógica
 - SIGNAL : s,t,u,v
 - son señales internas al circuito y son bidireccionales: pueden conectarse tanto a la entrada como a la salida de una puerta lógica

Señales VHDL : tipos de datos

- hay diferentes tipos de datos para las señales
- tipo de dato bit único : BIT : admite únicamente dos valores: el '0' y el '1'
- en VHDL los valores de los bits hay que entrecomillarlos para diferenciarlos de los datos de tipo INTEGER
- tipo de dato secuencia de bits: "010001010101" → doble entrecomillado si el dato se representa con más de un bit.

Señales VHDL : Buses

- Fisicamente un Bus es un conjunto de pistas metálicas que sirven para transportar señales conectando dos unidades
- Por ejemplo el "bus de direcciones" de 32 hilos ó pistas de la memoria RAM sirve para seleccionar una dirección de 32 bits de la memoria. La dirección **01100110011001100110011001100110** se transporta desde la CPU hasta la memoria RAM a través de un bus de 32 pistas. Al bus de direcciones de memoria (address bus) se le podría llamar A y a cada hilo del bus $A_{31}, A_{30}, \dots, A_1, A_0$.
- Desde el punto de vista lógico un bus es un vector o un array de dimensión "n", por ejemplo n=32.
- El tipo de datos de los buses A y B de 32 bits se podrían declarar como:
 - signal A,B : bit_vector(31 downto 0); donde el bit MSB(más a la izquierda) sería el hilo A_{31} y el bit LSB(más a la derecha) el bit A_0 y lo mismo con el bus B
 - signal A,B : bit_vector(0 to 31); donde el bit MSB(más a la izquierda) sería el hilo A_{00} el bit LSB(más a la derecha) el bit A_{31} y lo mismo con el bus B

Sentencias VHDL : Asignación Concurrente

- CAS : Concurrent Assignment Sentence
- La sentencia CAS se representa mediante el símbolo \Leftarrow
- El valor resultante de evaluar la expresión a la derecha del símbolo \Leftarrow se asigna a la señal a la izquierda del símbolo \Leftarrow

```
s <= not x;
t <= not y;
u <= x and t;
v <= y and s;
z <= u or v;
```

Sentencias Concurrentes

Concepto de concurrencia: ¿CUANDO se ejecuta una sentencia concurrente? cuando hay un **evento** en una de las **señales sensibles** de la sentencia. En el caso de la sentencia CAS la señales sensibles son las señales a la derecha del símbolo `<=`. Ejemplo:

```
s <= not x;
t <= not y;
u <= x and t;
v <= y and s;
z <= u or v;
```

Ejemplo: Ver cronograma

- 0- "x" = "y" = 0 \Rightarrow s + t = 1 \Rightarrow u = v = 0 \Rightarrow z = 0
- 1- Se produce un **EVENTO** (0→1) en la señal puerto "x"
- 2- "x" es una señal sensible en la línea 1 del código y en la linea 3 del código
- 3- Se ejecutan las líneas 1 y 3 del código
- 4- Ejecución de la linea 1: "s" (1→0)
- 5- Ejecución de la linea 3: "u" (0→1)
- 6- Hay un evento en "s": se ejecuta la linea 4 : "v" no cambia \rightarrow no evento
- 7- Hay un evento en "u": se ejecuta la linea 5 : "z" cambia (0→1)
- 8- La señal z no es una señal sensible en ninguna de las sentencias `<=` : FIN
- 9- FIN de la actualización de todas las señales hasta el próximo evento en "x" o/y "y"

Sentencias Concurrentes

Las sentencias concurrentes NO se ejecutan secuencialmente, sino **simultáneamente**, de la misma forma que en el circuito "light_bit.vhd" la puerta lógica OR procesa sus dos entradas al mismo tiempo que las señales NOT y AND del mismo circuito.

En los 4 ejemplos siguientes la actualización de los valores de todas las señales, ante el evento de una de ellas, da el MISMO resultado, ya que la ejecución no es secuencial, sino que se ejecutan UNICAMENTE las sentencias concurrentes cuyas señales sensibles varían; y las sentencias que se ejecutan lo hacen SIMULTANÉAMENTE.

```
s <= not x;
t <= not y;
u <= x and t;
v <= y and s;
z <= u or v;
```

```
z <= u or v;
v <= y and s;
u <= x and t;
t <= not y;
s <= not x;
```

```
u <= x and t;
v <= y and s;
z <= u or v;
s <= not x;
t <= not y;
```

```
u <= x and t;
t <= not y;
s <= not x;
z <= u or v;
v <= y and s;
```

Entidad

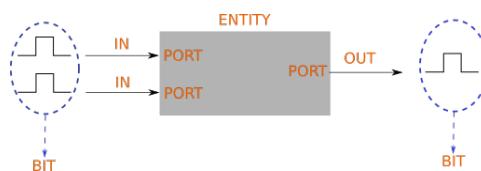
La **entidad** describe el comportamiento del circuito digital visto desde fuera, es decir, describe únicamente los accesos de entrada y salida del circuito. Los accesos de entrada y salida se realizan a través de señales digitales binarias denominadas **puertos**.

La entidad se define con el keyword ENTITY

La entidad que hay nombrarla con un nombre. Este nombre condiciona el nombre del fichero donde se almacena, que ha de tener el mismo nombre con y la extensión **.vhdl**

Las señales tipo PORT pueden ser de entrada (IN) ó salida (OUT) ó salida_y_entrada (BUFFER).

Además del modo de la señal (IN-OUT-BUFFER) es necesario declarar el tipo de los datos (BIT)



Las señales cuyos datos son de tipo BIT admiten los valores '0' y '1'

Sintaxis

```
entity of light_bit is
  port (
    x,y : in bit;
    z   : out bit
  );
end entity;
```

Arquitectura

REGISTER TRANSFER LEVEL
RTL

X+Y)*Z
(X OR Y) AND W
NAND Y
XOR

BEHAVIORAL

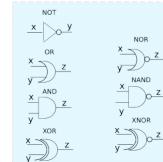
Si x e y valen 0 entonces la salida vale 0... si las entradas valen 1 la salida vale 0.....

x	y	z=x ⊕ y
0	0	0
0	1	1
1	0	1
1	1	0

STRUCTURAL



SINTESIS



3 Tipos de arquitecturas:

RTL: Expresiones lógicas

Behavioral o comportamental: funcionalidad

Structural: conectar subcircuitos

La descripción más sencilla es la **behavioral** ... pero también es la que exige un mayor esfuerzo al sintetizador.

Arquitectura

Sintaxis

```
architecture rtl of light_bit is
  signal s,t,u,v : bit;
begin
  s <= not x;
  t <= not y;
  u <= x and t;
  v <= y and s;
  z <= u or v;
end rtl;
```

La arquitectura del circuito se declara con el keyword architecture

La arquitectura del circuito hay que nombrarla con cualquier nombre: rtl, fun, etc... y relacionarla con una entidad

Las señales internas hay que declararlas con el keyword signal y definir el tipo de datos: pej bit

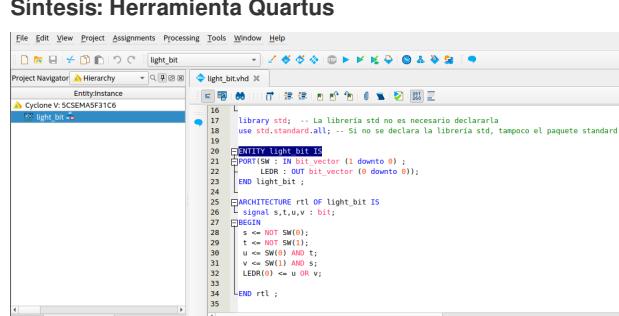
La relación entre las señales (puertos e internas) se define mediante "sentencias vhdl" entre los keywords begin y end

Hojas de Referencia

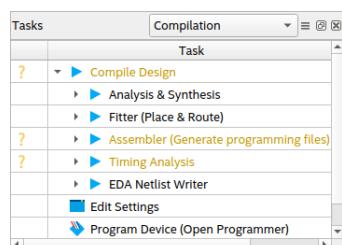
[Hoja de referencia simple](#) [/PDF/VHDL_Cheat_Sheet.pdf]

[Hoja de referencia completa](#) [/PDF/VHDL_QRC_01.pdf]

Síntesis: Herramienta Quartus

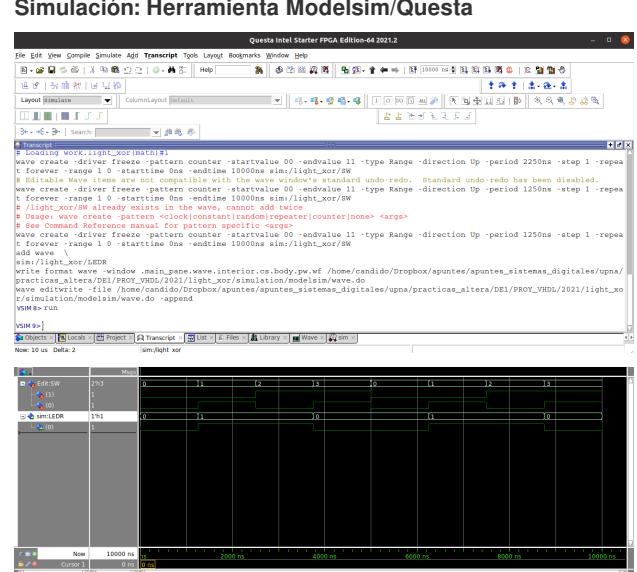


Entrada: código VHDL "light_bit.vhd"



Salida: simulación y síntesis : bitstream "quartus_light.sof"

Simulación: Herramienta Modelsim/Questa



Verificar el correcto funcionamiento del circuito antes de su Fabricación

Fabricación: FPGA Cyclone V de Intel

- Tarjeta de prototipado de Terasic DE1 SoC
- La tarjeta contiene la **FPGA Cyclone V** y sus periféricos
- El diseño "light_bit" se implementa en el chip FPGA (**Field Programmable Gate Array**)

Librerías y Paquetes

- Las definiciones de las sentencias, tipos de señales, tipos de datos, etc se encuentran definidas en las librerías.
- Por ejemplo el tipo de dato bit y bit_vector se encuentran definidos en la librería std y en el paquete standard
- Las librerías y los paquetes hay que declararlos al principio, antes de las entidades y de las arquitecturas

La librería std NO es obligado declararla

```
library std;
use std.standard.all;
entity of light_bit is
  port (
    x,y : in bit;
    z : out bit
  );
end entity;
architecture rtl of light_bit is
  signal s,t,u,v : bit;
begin
  s <= not x;
  t <= not y;
  u <= x and t;
  v <= y and s;
  z <= u or v;
end rtl;
```

Primer Diseño

Descripción del circuito minimalista $z=x$

```
entity of light_bit is
  port (
    x : in bit;
    z : out bit
  );
end entity;
architecture minima of light_bit is
begin
  z <= x;
end minima;
```

- El objetivo de este código es ser lo suficientemente simple para no dificultar su comprensión y centrarse en poner a punto la herramienta de desarrollo **Intel Quartus Prime Lite** desde cualquier computadora utilizando los recursos remotos de la UPNA.

Primer Diseño

Para poder ser fabricado en el laboratorio Remoto, es necesario que los puertos tenga los mismos nombres que se utilizan en el servidor del laboratorio Remoto. En este ejemplo la entrada serán los switches SW y la salida los leds rojos LEDR.

```
entity of light_bit is
  port (
    SW : in bit_vector (0 downto 0);
    LEDR : out bit_vector (0 downto 0)
  );
end entity;
architecture minima of light_bit is
begin
  LEDR(0) <= SW(0);
end minima;
```

Ejercicios Prácticos de Diseño de Circuitos

- Tutorial de VHDL mediante recursos Remotos: Quartus Prime Lite, Questa Intel, Laboratorio de dispositivos FPGA [vhdl_lab_remoto.html]
- La única forma de aprender un lenguaje de descripción de HW o de programación es practicando.
- La Upna brinda la posibilidad de utilizar los recursos EDA de diseño automático de circuitos integrados de forma remota, bien desde dentro del Campus Universitario o desde fuera de él, sin la necesidad de realizar ningún tipo de instalación en el portátil personal ni de acceder a ningún laboratorio.



En el escritorio virtual de la Upna al utilizar el programa Quartus no utilizar la carpeta de Descargas como ubicación del proyecto de diseño. Si se utiliza la carpeta Descargas es necesario utilizar un SUBDIRECTORIO como por ejemplo "Descargas\ssdd"

Ejercicios Prácticos de Diseño de Circuitos

1. light_bit: inicio
2. light_de1soc: señales std_logic y fabricación del diseño
3. light_signal: ver el esquema del circuito sintetizado
4. light_de1soc: simulación Questa
5. light_csa: sentencia concurrente, Tabla de la Verdad
6. light_process: sentencias secuenciales
7. light_if: sentencia secuencial
8. light_with: sentencia concurrente
9. light_case: sentencia secuencial
10. light_sum: librería ieee paquete numeric_std

Librerías y Paquetes

- Librerías: contienen paquetes que definen tipos de señales, tipos de datos, operadores, etc
 - Librería std
 - Paquete standard → definición del tipo BIT, BIT_VECTOR, etc
 - Paquete texto
 - no es necesario declararla
 - declaración
- ```
library std;
use std.standard.all;
```
- El paquete está descrito en el propio lenguaje VHDL → C:/intelFPGA\_lite/21.1/questa\_tse/vhdl\_src/std/standard.vhd

## Señales std\_logic

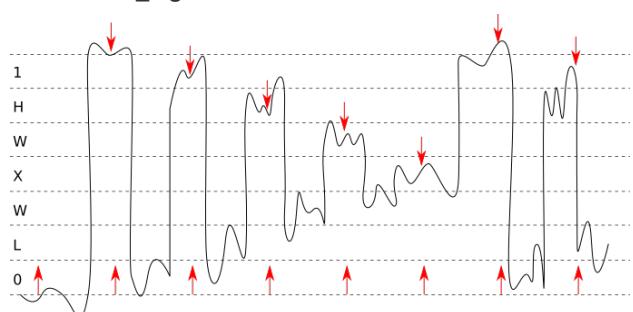


Figure 4. Señales binarias distorsionadas

- 8 Umbrales para definir el valor de la señal binaria
- 6 tipos medibles: 1,H,W,X,L,0

## Señales std\_logic

1 : forzosamente un 1  
 H : débilmente un 1  
 W : débilmente desconocido  
 X : forzosamente desconocido  
 W  
 L : débilmente un 0  
 0 : forzosamente un 0

Figure 5. Valores Medibles

8 umbrales

Valores No medibles:

- ; don't care → puede tomar cualquiera de los valores definidos "mesurables" y no afecta al funcionamiento del circuito.
- U** : indefinido → representa el estado de un circuito secuencial al encenderse y sin estado de reset de reset. Puede ser cualquiera de los 6 valores definidos pero al no tener estado inicial, no puede determinarse su estado.
- Z**: alta impedancia : salida en circuito abierto

## Librería IEEE: Paquete std\_logic\_1164

- tipos de señales: std\_logic y std\_logic\_vector

```

library ieee;
use ieee.std_logic_1164.all;

ENTITY light_de1soc IS
PORT(SW : IN std_logic_vector (1 downto 0) ;
 LEDR : OUT std_logic);
END light_de1soc ;

```

- El paquete std\_logic\_1164 está localizado en: C:/intelFPGA\_lite/21.1/questa\_fse/vhdl\_src/ieee/stdlogic.vhdl

## Simulación y Fabricación

- Simulación
  - Simulador Questa
  - Simulador Waveform
- Fabricación: acceso a través de Miaulario : Pestaña **DE1-SoC->>Remoto**

## Tema 5: Circuitos Aritméticos

## Índice

- Tema4 Circuitos Aritméticos [PDF/04\_circuitos\_aritmeticos.pdf]: PDF
- Operaciones Aritméticas: Suma, Resta, Complemento C1-C2, Multiplicación
- Circuitos sumadores
- Circuitos restadores
- Circuitos sumador/restador
- Circuito Multiplicador
- Unidad Aritmetico-Lógica (ALU)

## Operaciones Aritméticas

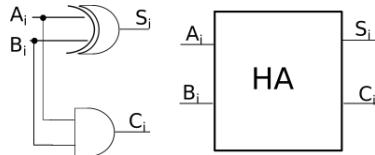
- Suma y Resta : binario, hexadecimal y números enteros C1-C2
- Multiplicación

|                                                                                                                                                                                                                                                                                 |                                                                                                                                        |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| $  \begin{array}{r}  1 \ 0 \ 1 \ 1 \ 0 \ 1 \\  \times 1 \ 0 \ 1 \ 1 \ 0 \ 1 \\  \hline  1 \ 0 \ 1 \ 1 \ 0 \ 1 \\  0 \ 0 \ 0 \ 0 \ 0 \ 0 \\  1 \ 0 \ 1 \ 1 \ 0 \ 1 \\  1 \ 0 \ 1 \ 1 \ 0 \ 1 \\  0 \ 0 \ 0 \ 0 \ 0 \ 0 \\  1 \ 0 \ 1 \ 1 \ 0 \ 1  \end{array}  $ <p>Llevadas</p> | $  \begin{array}{r}  - 1 \ 1 \ 1 \ 2 \ 1 \ 1 \ 1 \\  - \\  \hline  1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1  \end{array}  \rightarrow 2025  $ |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|

El resultado de la multiplicación tiene un tamaño de bits suma del número de bits de los factores.

## Semi-Sumador (Half Adder)

- Función: Suma de dos bits → resultado suma y carry (llevada,acarreo)
- Suma aritmética →  $Suma = A_i + B_i$
- Resultado: 2 funciones :  $S_i$  y el acarreo de la posición  $i$  a la posición  $i+1 \rightarrow C_i$
- Diseño: dibujar diagrama de bloques del circuito
  - Dos funciones y dos variables → TV, DK → expresión lógica positiva
- $S_i = \bar{A}_i \cdot B_i + A_i \cdot \bar{B}_i = A_i \oplus B_i$
- $C_i = A_i \cdot B_i$



## Sumador Completo (Full Adder)

- Función: Suma de dos bits y acarreo anterior → resultado suma y acarreo posterior
- Suma aritmética →  $Suma = A_i + B_i + C_{i-1}$
- Resultado: 2 funciones :  $S_i$  y el acarreo →  $C_i$
- Diseño: dibujar diagrama de bloques del circuito
  - Dos funciones y tres variables → TV, DK → expresión lógica positiva
- $S_i = \bar{A}_i \cdot \bar{B}_i \cdot C_{i-1} + \bar{A}_i \cdot B_i \cdot \bar{C}_{i-1} + A_i \cdot \bar{B}_i \cdot \bar{C}_{i-1} + A_i \cdot B_i \cdot C_{i-1} = \bar{A}_i \cdot (\bar{B}_i \cdot C_{i-1} + B_i \cdot \bar{C}_{i-1}) + A_i \cdot (\bar{B}_i \cdot \bar{C}_{i-1}) = \bar{A}_i \cdot (\bar{B}_i \oplus C_{i-1}) + A_i \cdot (\bar{B}_i \oplus \bar{C}_{i-1}) = A_i \oplus B_i \oplus C_{i-1}$
- $C_i = A_i \cdot B_i + A_i \cdot C_{i-1} + B_i \cdot C_{i-1}$

## Sumador Completo

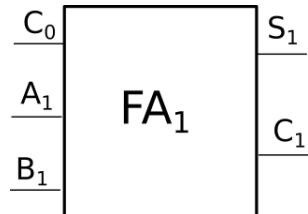


Figure 6. Full Adder

## Sumador Completo usando semi-sumadores



Concepto nuevo: Diseñar un circuito no como combinación de puertas lógicas sino como combinación de otros circuitos o bloques



Concepto nuevo: introducir en la tabla de la verdad variables-funcion nuevas como Sa,Ca,Sb,Cb para obtener las funciones C1 y S1 en función de ellas.

## Sumador Completo usando semi-sumadores

Partitionamiento: utilizar señales intermedias  
 $A_1 + B_1$  i puede ser sustituido por un Half Adder  
 Razonar la suma de 3 bits con sumas parciales de 2 bits

|                                                                                                                                                        |  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| $  \begin{array}{r}  1 & & 1 \\  1 & & + 1 \\  + 1 & -> & --- \\  --- & & C_a S_a \\  & & + 1 \\  & & C_b \\  & & --- \\  & & C_{1S} b  \end{array}  $ |  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------|--|

$S_{1S} = S_b$   
 $\& C1 ?$

$$Suma = (A_1 + B_1) + C_0 = (S_a, C_a) + C_0 = C_a, (S_a + C_0) = C_a, (S_b, C_b) = (S_1, C_1)$$

$$HA\_a \rightarrow (S\_a, C\_a)$$

$$HA\_b \rightarrow (S\_b, C\_b)$$

$\& C1 ?$

## Sumador Completo usando semisumadores

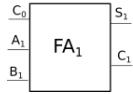
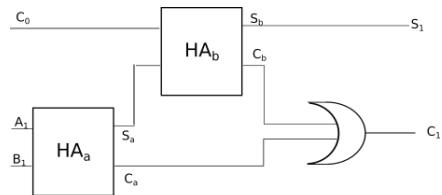
Tabla de la verdad con 3 variables y 6 funciones  
 $A_1, B_1, C_0, S_a, S_b, C_b, C_1, S_1$   
 $S_1 = S_b$   
 $C1 = \bar{C}_a \cdot C_b + C_a \cdot \bar{C}_b = C_a \oplus C_b$

| $C_1$ | $C_b$ | 0 | 1 |
|-------|-------|---|---|
| $C_a$ | 0     | 0 | 1 |
| 1     | 1     | X |   |

$$C1 = C_a + C_b$$

Table 18.  
 Full Adder

### Sumador Completo usando semisumadores



### Sumador Paralelo Acarreo Serie

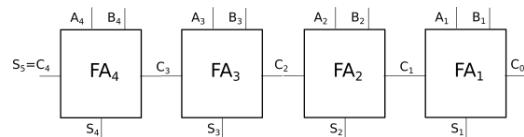


Figure 7. Full Adders en Cadena

### Sumador Paralelo Acarreo Paralelo

- El acarreo serie ralentiza la obtención de la suma → ¿solución? → Carry Look Ahead
- Full Adder: Tabla de la verdad :
  - segmentar la tabla de la verdad
  - fijarse en la relación entre los dos acarreos , el anterior y el posterior, cada dos filas, ya que son el problema.
  - 3 casos: no hay acarreo, se propaga el acarreo, se genera el acarreo → lo expresamos lógicamente mediante las funciones **G** y **P**
- Particionamiento: Variables ó Funciones Intermedias → generación y propagación
  - 2 funciones-variables intermedias para codificar 3 casos → **G** y **P** se pueden calcular en paralelo en función de los sumandos ya que NO DEPENDEN de los acarreos → Los acarreos de salida se pueden calcular en función de **G** y **P**, que no dependen de los acarreos de entrada
- Diseñar el circuito
  - Más rápido pero más complejo

### Sumador Paralelo Acarreo Paralelo

Table 19. Funciones de Generación y Propagación del Acarreo

| A <sub>i</sub> | B <sub>i</sub> | C <sub>i-1</sub> | C <sub>_i</sub> | Descripción                | G | P |
|----------------|----------------|------------------|-----------------|----------------------------|---|---|
| 0              | 0              | 0                | 0               | Ni se Genera Ni se Propaga | 0 | 0 |
| 0              | 0              | 1                | 0               |                            | 0 | 0 |
| 0              | 1              | 0                | 0               | Se Propaga                 | 0 | 1 |
| 0              | 1              | 1                | 1               |                            | 0 | 1 |
| 1              | 0              | 0                | 0               |                            | 0 | 1 |
| 1              | 0              | 1                | 1               |                            | 0 | 1 |
| 1              | 1              | 0                | 1               | Se Genera                  | 1 | 0 |
| 1              | 1              | 1                | 1               |                            | 1 | 0 |

## Sumador Paralelo Acarreo Paralelo

$$C_i = \overline{A}_i B_i C_{i-1} + A_i \overline{B}_i C_{i-1} + A_i B_i \overline{C}_{i-1} + A_i B_i C_{i-1} = (A_i \oplus B_i) \cdot C_{i-1} + A_i B_i = P_i \cdot C_{i-1} + G_i$$

| $P_i$     | $A_i B_i$   | $G_i$     | $A_i B_i$   |
|-----------|-------------|-----------|-------------|
| $C_{i-1}$ | 00 01 11 10 | $C_{i-1}$ | 00 01 11 10 |
| 0         | 0 0 1 0 1   | 0         | 0 0 1 0     |
| 1         | 0 1 0 1     | 1         | 0 0 1 0     |

Conclusión:  $P_i$  y  $G_i$  permiten expresar el acarreo de forma más clara.

$$C_1 = G_1 + P_1 \cdot C_0$$

$$C_2 = G_2 + P_2 \cdot C_1 = G_2 + P_2 G_1 + P_2 P_1 C_0$$

$$C_3 = G_3 + P_3 \cdot C_2 = \dots = G_3 + P_3 P_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_0$$

## Sumador Serie

Sumador serie:

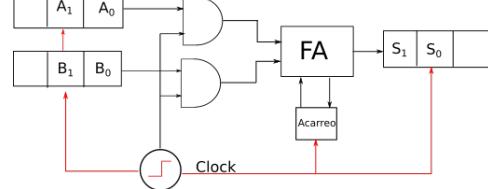
-Un Full Adder con 3 entradas (sumando y acarreo anterior) y dos salidas (suma y acarreo)

-Necesita de 3 "registros de desplazamiento" para memorizar los operandos y la suma. Cuando el reloj genera un disparo positivo, el contenido de las celdas del registro se desplazan una posición hacia la derecha.

-Celda de memoria para recordar el acarreo previo. La escritura de la memoria es síncrona con el reloj.

-Puerta de bloqueo: mientras el reloj está a cero los sumandos valen cero

-Dispositivo de sincronismo: el reloj



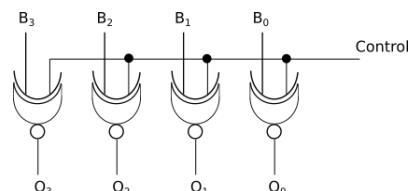
## Sumador/Restador de 4 bits

- Operando: números con signo en formato C1 ó C2
- Diagrama de bloques del circuito:
  - Señal de control para sumar o restar → la resta equivale a sumar cambiando de signo al sustraendo
  - Resta: cambiar el signo para sólo sumar
    - bloque complementador para generar el complementario
    - bloque sumador completo de 4 bits
- Diseño C1 :
  - C1 : complementario
  - suma: la última llevada (MSB) se suma. Ver [Suma-Resta C1](#)
- Diseño C2 :
  - C2 : complementario más 1
  - suma: la última llevada no se tiene en cuenta

## Complementador

| Control | Entrada | Salida |
|---------|---------|--------|
| 1       | 0       | 0      |
|         | 1       | 1      |
| 0       | 0       | 1      |
|         | 1       | 0      |

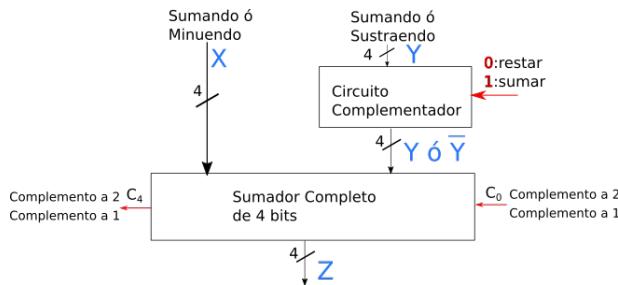
$$\text{Salida}_i = \overline{\text{Control}} \oplus \text{Entrada}_i$$



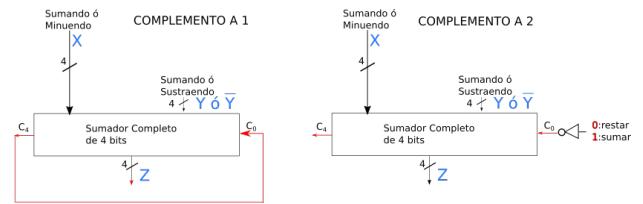
$$\text{Control} = 1 \quad O_i = B_i$$

$$\text{Control} = 0 \quad O_i = \overline{B}_i$$

### Sumador/Restador de 4 bits



### Sumador/Restador de 4 bits



### Semi-Restador

- Mismos conceptos de diseño que para el Semi-Sumador
  - TV → lógica positiva o negativa → Obtener Expresión lógica
- La llevada (L) en lugar de acarreo recibe el nombre de Borrow

### Restador Completo

- Mismos conceptos de diseño que para el Sumador Completo
  - TV → lógica positiva o negativa → Obtener Expresión lógica

## Acumulador

- Almacenar resultados parciales y reutilizarlos como operandos

## Multiplicador

- Concepto de diseño:
  - Estructura matricial: el bus A en "m" Filas y el bus B en "n" columnas
  - En cada punto de la matriz  $(i,j)$  se dispone de un sumador Full-Adder para realizar las sumas de dos en dos verticalmente y teniendo en cuenta el acarreo de la columna anterior, al igual que se realiza manualmente.
  - Tener en cuenta que el número de bits del resultado es  $m+n$ .

## Unidad Aritmético Lógica (ALU)

- Puede realizar diferentes operaciones: aritméticas , lógicas, etc
- El circuito tiene como entrada señales de control para seleccionar la operación a realizar
- ALUs comercial 714181 → Interpretación de las Hoja de Características
  - la entrada  $M$ : selecciona la categoría de las operaciones: Aritmética o Lógica.
  - la entrada  $C_n$  : selecciona la subcategoría de las operaciones dentro de la categoría Aritmética:
  - En total hay  $3 \times 2^4$  operaciones = 48 operaciones
  - las entradas  $S_3, S_2, S_1, S_0$  seleccionan el tipo de operación dentro de las 2 categorías.
  - Los datos de entrada A y B y el de salida F son de 4 bits
  - Los datos de entrada y salida siguen la lógica negativa : son activas las señales de valor 0.
  - Lógica negativa → el dato 1000 representa al valor 7.

## Tema 6: Otros Circuitos Combinacionales

## Tema 6: Índice

- Tema5 Otros Circuitos Combinacionales [PDF/05\_otros\_circuitos\_combinacionales.pdf]: PDF
- 1. Circuitos Combinacionales.
- 2. Multiplexor
- 3. Codificador (Encoder)
- 4. Decodificador / Demux
- 5. Convertidores de Código
- 6. Generador-Comprobador de Paridad
- 7. Comparador Binario

## Circuitos Combinacionales

- Concepto: Circuitos cuya salida un instante es resultado de la entrada en ese mismo instante sin tener cuenta la historia pasada
- No tienen MEMORIA

## Multiplexor

1. Concepto
2. Símbolo
3. Diseño
4. Esquema Eléctrico
5. Hoja de características comercial
6. Extensión
7. Aplicaciones: Generador de Funciones

## Concepto - Símbolo

El circuito multiplexor de N entradas de datos y 1 salida de datos tiene la función de seleccionar una de las entradas y conectarla a la salida. Para ello se sirve de M entradas de control donde  $2^M=N$

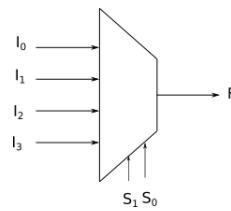
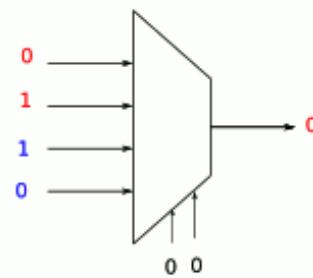


Figure 8. Multiplexor 4x1



## Diseño

lenguaje natural: función del multiplexor  $\rightarrow F=I_0 \text{ si } S_1=0 \text{ y } S_0=0 \text{ ó } F=I_1 \text{ si } S_1=0 \text{ y } S_0=1 \text{ ó } \dots$

$$F = I_0 \text{ si } S_1=0 \text{ y } S_0=0 \text{ ó } F = I_1 \text{ si } S_1=0 \text{ y } S_0=1 \text{ ó } F = I_2 \text{ si } S_1=1 \text{ y } S_0=0 \text{ ó } F = I_3 \text{ si } S_1=1 \text{ y } S_0=1$$

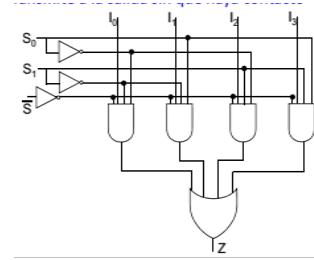
Table 20. TV

| I <sub>3</sub> | I <sub>2</sub> | I <sub>1</sub> | I <sub>0</sub> | S <sub>2</sub> | S <sub>1</sub> | F              |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| X              | X              | X              | X              | 0              | 0              | I <sub>0</sub> |
| X              | X              | X              | X              | 0              | 1              | I <sub>1</sub> |
| X              | X              | X              | X              | 1              | 0              | I <sub>2</sub> |
| X              | X              | X              | X              | 1              | 1              | I <sub>3</sub> |

## Esquema Eléctrico

- Señal Strobe : señal de validación, habilitación. Es necesario que este activa para que el circuito procese los datos de entrada y la salida sea válida.

- Se implementa mediante la función ( $Z = \text{Strobe} \cdot F$ ) donde es necesario activar Strobe para que la salida Z tenga el valor de F, en caso contrario Z=0. Si la señal strobe se activa con lógica negativa  $\rightarrow Z = \overline{\text{Strobe}} \cdot F$

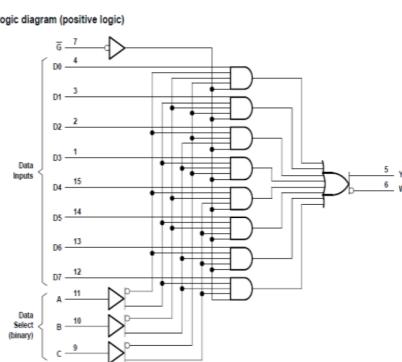


## Hoja de Características Comercial

| FUNCTION TABLE |   |        | logic diagram (positive logic) |       |
|----------------|---|--------|--------------------------------|-------|
| INPUTS         |   | SELECT | STROBE                         | Y W   |
| X              | X | X      | H                              | L H   |
| L              | L | L      | L                              | 00 00 |
| L              | L | H      | L                              | 01 01 |
| L              | H | L      | L                              | 02 02 |
| L              | H | H      | L                              | 03 03 |
| H              | L | L      | L                              | 04 04 |
| H              | L | H      | L                              | 05 05 |
| H              | H | L      | L                              | 06 06 |
| H              | H | H      | L                              | 07 07 |

D<sub>0</sub>, D<sub>1</sub> ... D<sub>7</sub> = the level of the respective D input

Dispositivo buffer: adapta la señal (I,V)  
Símbolo círculo "O": negación

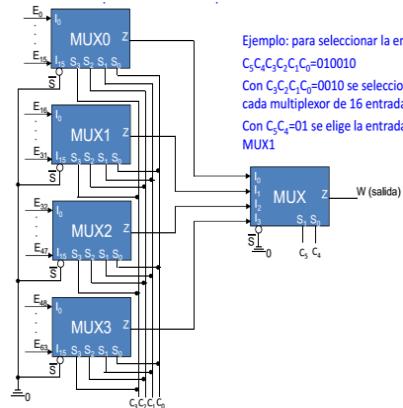


## Extensión de Multiplexores

- El número de entradas de los multiplexores es limitado. No hay multiplexores con 64 entradas. Para diseñar un multiplexor de 64 entradas es necesario utilizar multiplexores con un número de entradas inferior: 16, 8, 4, 2 entradas.

- Pej cómo diseñar un mux de 64 entradas con mux de 16 entradas de datos
  - Un mux de 64 entradas necesita 6 señales de selección o control  $\rightarrow 2^6$
  - Un mux de 64 entradas es la combinación de 4 mux de 16 entradas de datos y 4 entradas de selección. Las 4 salidas de los 4 mux se multiplexan con un segundo mux de 4 entradas.
  - Las señales de selección del 1º nivel de mux están interconectadas dando lugar a 4 entradas de control y el 2º nivel de mux tendría 2 entradas de control. Los dos niveles suman 6 entradas de control.

## Extensión de Multiplexores



## Aplicaciones: Generador de Funciones

- Las salida del multiplexor tiene el formato de una función canónica SOP donde las entradas de control son las variables independientes de la función. Las entradas de datos tomar valores "0" ó "1" o ser también variables.
- Por lo tanto un multiplexor con 3 entradas de control puede implementar una función lógica de 3 o más variables independientes.
- Ejemplos.

## Ejemplos: Generador de Funciones

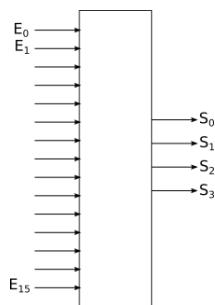
1. Generar con un MUX 8x1 la función  $F = \overline{B}CD + A\overline{B}D + \overline{A}\overline{B}C$ 
  - Para hacer la equivalencia con la función de un MUX la función F tiene que tener formato SOP
  - Variables independientes: las 3 entradas de selección del MUX 8x1 → P ej las variables A,B y C
  - La variable independiente D se implemente mediante las entradas de datos del MUX.
2. Implementar mediante un MUX Nx1 la función  $F = \overline{A} \cdot \overline{B} \cdot C \cdot D + \overline{A} \cdot B \cdot C \cdot E + A \cdot \overline{B} \cdot \overline{C} \cdot F + A \cdot B \cdot C \cdot G$ 
  - Se observan variables comunes en todos los sumandos → SOP

## Codificador (Encoder)

1. Concepto
2. Símbolo
3. Ejemplo de Diseño: BCD sin prioridad, 4x2 con prioridad

## Concepto

- Un codificador realiza la conversión de un número con cualquier tipo de código no binario a código binario.
- Por ejemplo: un codificador de código hexadecimal a código binario se implementa mediante un circuito con un bus de datos de entrada de 16 hilos donde únicamente (codificador sin prioridad) se activa el hilo asociado a uno de los 16 dígitos hexadecimales. Para codificar en binario los 16 dígitos son necesarios 4 bits ( $2^4$  combinaciones), por lo que el bus de datos de entrada tiene un tamaño de 16 pistas y el de salida tiene un tamaño de 4 pistas.



| 185

## Codificador Hexadecimal : Diseño

- No se puede recurrir directamente a un DK ya que el número de celdas es muy elevado
- Es necesario recurrir al lenguaje natural para obtener las 4 salidas y optimizarlas mediante la matemática booleana.
- Función  $S_0 \rightarrow$  lógica positiva
  - vale 1 si: filas 1<sup>a</sup>, 3<sup>a</sup>, 5<sup>a</sup>...15<sup>a</sup>
  - fila 1<sup>a</sup>: 0000000000000010 → es el minitérmino  $m_2$  de 16 variables → optimización: ¿se puede reducir el N° de variables?
  - Si de las filas RESTO capturo todas las filas  $m_i$  donde  $E_1$  valga 1 y las sumo a la fila 1<sup>a</sup> → puedo sacar factor común  $E_1$
  - $E_1 * \text{sum}(m_i) = E_1 \text{cdot} 1$  y así con las filas 3<sup>a</sup>, 5<sup>a</sup>, etc
  - $S_0 = E_1 + E_3 + E_5 + E_7 + E_9 + E_{11} + E_{13} + E_{15}$

## Codificador Hexadecimal sin Prioridad : Diseño

Table 21. TV del Codificador Hexadecimal

| $E_{15}$                        | $E_{14}$ | $E_{13}$ | .. | $E_1$ | $E_0$ | $S_3$ | $S_2$ | $S_1$ | $S_0$ | Fila            |
|---------------------------------|----------|----------|----|-------|-------|-------|-------|-------|-------|-----------------|
| 0                               | 0        | 0        | .. | 0     | 1     | 0     | 0     | 0     | 0     | 0 <sup>a</sup>  |
| 0                               | 0        | 0        | .. | 1     | 0     | 0     | 0     | 0     | 1     | 1 <sup>a</sup>  |
| ..                              | ..       | ..       | .. | ..    | ..    | ..    | ..    | ..    | ..    | ..              |
| 0                               | 0        | 1        | .. | 0     | 0     | 1     | 1     | 0     | 1     | 13 <sup>a</sup> |
| 0                               | 1        | 0        | .. | 0     | 0     | 1     | 1     | 1     | 0     | 14 <sup>a</sup> |
| 1                               | 0        | 0        | .. | 0     | 0     | 1     | 1     | 1     | 1     | 15 <sup>a</sup> |
| RESTO de COMBINACIONES POSIBLES |          |          |    |       |       | X     | X     | X     | X     |                 |

- El resto de combinaciones "no van a ocurrir" por lo que salida queda "X" en DK ó "don't care" en VHDL, es decir, no importa que la salida valga cero ó uno para dichas entradas y las tenemos que definir 0 ó 1 en función de su optimización.

| 186 |

## Codificador Hexadecimal : Diseño

- Aplicamos el mismo método a las 4 salidas
  - $S_0 = E_1 + E_3 + E_5 + E_7 + E_9 + E_{11} + E_{13} + E_{15}$
  - $S_1 = E_2 + E_3 + E_6 + E_7 + E_{10} + E_{11}$
  - $S_2 = E_4 + E_5 + E_6 + E_7 + E_{12} + E_{13} + E_{14} + E_{15}$
  - $S_3 = E_8 + E_9 + E_{10} + E_{11} + E_{12} + E_{13} + E_{14} + E_{15}$
- ¿ Hubiésemos llegado al mismo resultado mediante el lenguaje natural?
  - La salida  $S_k$  igual a 1 si la entrada  $E_i = 1$  ó la entrada  $E_j = 1$  ó etc ...

| 187

### Ejemplo de Diseño: BCD sin prioridad

- Mismo procedimiento que el codificador Hexadecimal sin prioridad.

### Ejemplo de Diseño: Codificador 4x2 con prioridad

Se puede activar más de una entrada...pero se selecciona únicamente UNA → la de mayor prioridad  
Ejemplo 1 : Codificador 4x2 con prioridad  $E_3 > E_2 > E_1 > E_0$

Table 22. Tabla de la verdad

| $E_3$ | $E_2$ | $E_1$ | $E_0$ | $S_1$ | $S_0$ |
|-------|-------|-------|-------|-------|-------|
| 1     | X     | X     | X     | 1     | 1     |
| 0     | 1     | X     | X     | 1     | 0     |
| 0     | 0     | 1     | X     | 0     | 1     |
| 0     | 0     | 0     | 1     | 0     | 0     |
| 0     | 0     | 0     | 0     | 0     | 0     |

- lógica positiva

$$\circ S_0 = E_3 + \overline{E}_3\overline{E}_2E_1 = \text{optimizar} = E_3 + E_3\overline{E}_2E_1 + \overline{E}_3\overline{E}_2E_1 = E_3 + \overline{E}_2E_1$$

$$\circ S_1 = E_3 + \overline{E}_3E_2 = \text{optimizar} = E_3 + E_3E_2 + \overline{E}_3E_2 = E_3 + E_2$$

### Decodificador / Demux

- Concepto
- Diseño: Salidas mutuamente excluyentes
- Ejemplo de Diseño: Decodificador BCD
- Excitador de un Display de 7 segmentos
- Aplicaciones:
  - Generador de Funciones
  - Conversor Serie Paralelo
  - Generador de Funciones: Decodificador y Multiplexor

### Concepto

- El Decodificador realiza la función inversa del codificador, es decir, convierte la entrada de datos binaria en una salida de datos en código no binario.

## Diseño: Salidas mutuamente excluyentes

Salidas Mutuamente excluyentes: únicamente se activa una de las salidas

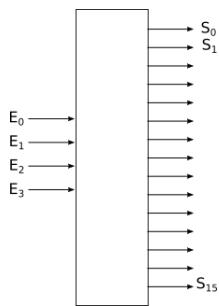


Figure 9. Decodificador Hexadecimal

## Diseño: Salidas mutuamente excluyentes

- Lenguaje Natural
  - La salida  $S_0$  vale 1 si la entrada  $E_3E_2E_1E_0$  vale 0000 → minitérmico  $m_0$
  - $S_0 = \overline{E}_3\overline{E}_2\overline{E}_1\overline{E}_0$
- La salida  $S_1$  vale 1 si la entrada  $E_3E_2E_1E_0$  vale 0001 → minitérmico  $m_1$
- $S_1 = \overline{E}_3\overline{E}_2\overline{E}_1E_0$
- La salida  $S_{15}$  vale 1 si la entrada  $E_3E_2E_1E_0$  vale 1111 → minitérmico  $m_{15}$
- $S_{15} = E_3E_2E_1E_0$

## Ejemplo de Diseño: Decodificador BCD

- Decodificador BCD/DEC sn74ls42 [<http://bit.ly/1ohdG23>]
- símbolo línea con lógica negativa



## Extensión de los Decodificadores

- 3-LINE TO 8-LINE DECODERS/DEMUTLIPLEXERS [<https://www.ti.com/lit/ds/symlink/cd54act138.pdf>]: Diseñar un Decodificador de 32 Salidas mediante decodificadores CD54ACT138 3x8
  - Hacen falta 4 decodificadores CD54ACT138 cuyas características son:
  - 2 tipos de entradas: de selección y de datos
  - La función **enable** (habilitación) se obtiene mediante el operador & (and) de las 3 señales:  $\overline{G}2A$ ,  $\overline{G}2B$  y  $G1$  correspondientes a los pines 4,5 y 6 del circuito integrado CD54ACT138.
  - Las 8 salidas de datos operan en lógica **negativa**: se activan con un 0

## Extensión de los Decodificadores mediante CD54ACT138.

- El decodificador resultante de 32 salidas tendría una entrada binaria de 5 bits  $\rightarrow 2^5$ ,  $A_4, A_3, A_2, A_1, A_0$  y el decodificador CD54ACT138 3x8 solo tiene 3 entradas de datos, por lo que es necesario utilizar las señales "Enable".
- Las entradas  $A_2, A_1, A_0$  se asignan al bus de datos del decodificador 3x8
  - Una combinación  $A_2, A_1, A_0$  seleccionará la misma salida de los 4 decodificadores 3x8 : tendremos seleccionadas 4 salidas del decodificador 5x32, por lo que es necesaria una segunda selección para seleccionar únicamente uno de los 4 decodificadores 3x8.
- Para realizar la segunda selección las entradas  $A_4, A_3$  se asignan al bus de control Enable de los 4 decodificadores 3x8

## Extensión de los Decodificadores mediante CD54ACT138.

Table 23. Tabla de la Verdad

| Decodificador Seleccionado | $A_4$ | $A_3$ | $\bar{G}_{2B}$ | $\bar{G}_{2A}$ | $G_1$ |
|----------------------------|-------|-------|----------------|----------------|-------|
| Primer                     | 0     | 0     |                |                |       |
| Segundo                    | 0     | 1     |                |                |       |
| Tercero                    | 1     | 0     |                |                |       |
| Cuarto                     | 1     | 1     |                |                |       |

- Hay que conectar las 3 señales de control con las 2 entradas de datos  $A_4, A_3$

## Diodo L.E.D.

- Light-Emitting Diode : diodo LED

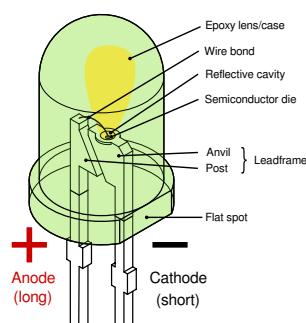
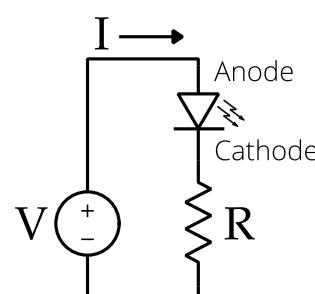


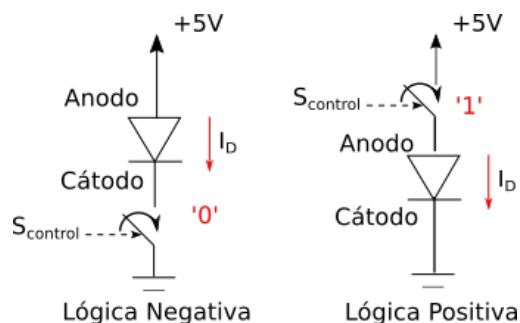
Figure 10. Estructura

By Inductiveload - Own work by uploader, drawn in Solid Edge and Inkscape., Public Domain, <https://commons.wikimedia.org/w/index.php?curid=6431789>

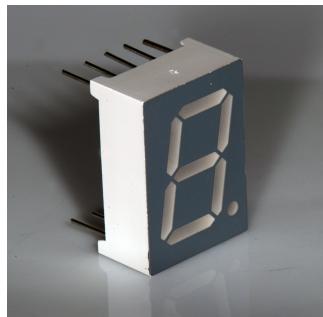
## LED: Circuito Excitador



<a href="https://creativecommons.org/licenses/by-sa/2.5" title="Creative Commons Attribution-Share Alike 3.5">CC BY-SA 2.5,</a>, <a href="https://commons.wikimedia.org/w/index.php?curid=2550282">Link</a>



## Display 7 segmentos



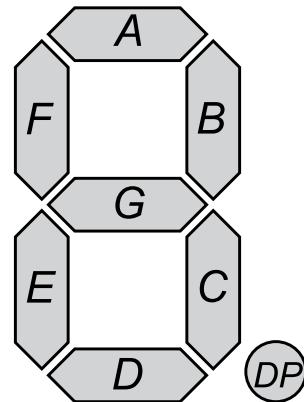
<a href="https://creativecommons.org/licenses/by-sa/3.0" title="Creative Commons Attribution-Share Alike 3.0">CC BY-SA 3.0</a>, <a href="https://commons.wikimedia.org/w/index.php?curid=2550282">Link</a>

Cada segmento del Display es un LED



<a href="https://creativecommons.org/licenses/by-sa/4.0" title="Creative Commons Attribution-Share Alike 4.0">CC BY-SA 4.0</a>, <a href="https://commons.wikimedia.org/w/index.php?curid=105121223">Link</a>

## Excitador de un Display de 7 segmentos



<a href="https://commons.wikimedia.org/w/index.php?curid=582271">Link</a>

Los 7 LEDs del Display se nombran con 7 letras según su posición.

Los 7 LEDs del Display tienen el Anodo ó el Cátodo común.

Si el Display es de Anodo Común, se puede controlar el Cátodo de cada LED libremente con un **0** : lógica negativa.  
 -Iluminar el 0 : activar los segmentos a-b-c-d-e-f → 0000001.  
 -Iluminar el 5: activar los segmentos a-c-d-f-g → 0100100.

Si el Display es de Cátodo Común, se puede controlar el Anodo de cada LED libremente con un **1** : lógica positiva.  
 -Iluminar el 0 : activar los segmentos a-b-c-d-e-f → 1111110.  
 -Iluminar el 5: activar los segmentos a-c-d-f-g → 1011011.

## Decodificador Display 7 Segmentos

- Diseñar un Decodificador de Hexadecimal a Display\_7\_segmentos de Anodo Común en código VHDL. Verificar su funcionamiento mediante la tarjeta DE1-Soc
  - Utilizar como puerto de entrada las señales de los switches SW(3 downto 0)
  - Utilizar como puerto de salida las señales del Display de 7 segmentos HEX0(6 downto 0)
  - Cada dígito hexadecimal de entrada estará codificado en binario con 4 bits y la salida será un bus de 7 líneas donde cada línea irá a uno de los LEDs

## Aplicaciones: Generador de Funciones

- El Decodificador de salidas mutuamente excluyentes activa una salida para combinación de la entrada por lo que cada salida responde a un minítermo.
- Las entradas del Decodificador representan las variables independientes.
- Se puede generar una función mediante la primera forma canónica SOP realizando la suma lógica, exclusivamente de las salidas correspondientes a los miníterminos que hacen uno a la función, mediante una puerta OR cuya salida representa a la Función.

## Aplicaciones: Conversor Serie Paralelo

- Aplicación como **Demultiplexor**: 1 entrada ENABLE que se conecta a una de las salidas seleccionada mediante las entradas de selección.
  - en realidad la entrada enable no se conecta físicamente sino que **inhibe** (todas las salidas desactivadas) o **habilita** (salida seleccionada activa) el dispositivo.
- Mediante un Decodificador con entrada ENABLE como bus de datos serie se puede enviar cada bit de dicha entrada a una salida diferente del decodificador si hacemos un barrido de las combinaciones de entrada mediante la secuencia binaria de un contador.
  - El contador
    - El tamaño del contador binaria ha de ser igual al número de entradas binarias del decodificador.
    - El contador ve incrementada su cuenta a cada disparo positivo del reloj de entrada.
    - El valor de la cuenta es accesible en el bus paralelo de salida del contador, donde cada línea del bus representa cada bit de la cuenta.

## Aplicaciones: Generador de Funciones: Decodificador y Multiplexor

- Se puede generar una función mediante un Decodificador y/o un Multiplexor.

## Convertidores de Código

1. BCD

## Generador-Comprobador de Paridad

1. Concepto de Paridad
2. Detección de Errores
3. Hoja de Datos Comercial

## Comparador Binario

1. Concepto
2. Diseño comparador de 4 bits
3. Extensión con Entradas de Expansión
4. Extensión sin Entradas de Expansión
5. Hoja de Datos Comercial

## Fin del Primer Parcial

| 209

210 |

## listas

- hola
  - caracola
  - caracola

## matematicas

$Y = f(x_1, x_2, x_3) = x_1 + \bar{x}_1 x_2 x_3 =$  transformar suma en producto con morgan  
 $= x_1 + \bar{x}_1 x_2 x_3 =$  aplico morgan =  $\bar{x}_1 \bar{x}_1 x_2 x_3$

x

$$\frac{\bar{x}_1 x_2 x_3}{\bar{x}_1 \bar{x}_2 \bar{x}_3}$$

| 211

212 |

## Text Formatting

### text with id and role

some text  
some text

```
printf("Hello world %d\n", x)
```

## step 1



Finally, a pro tip...



First, watch out for...



Second, don't forget...

## step 2

The greatest glory in living lies not in never falling, but in rising every time we fall.  
The greatest glory in living lies not in never falling, but in rising every time we fall.

## step 3

blue

white

red

**overline**

Z = X

Z = X

Z = X

**tablas**