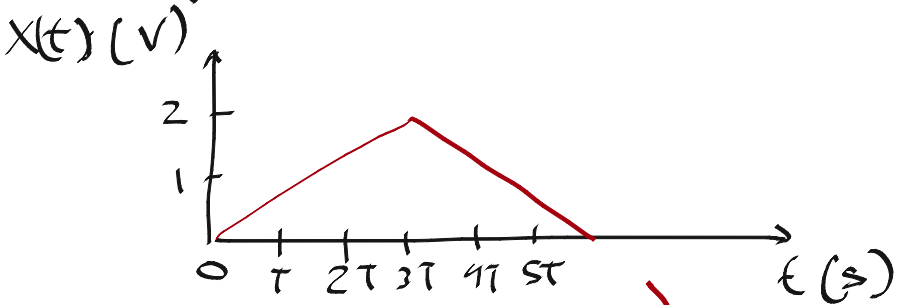
CLASE 16

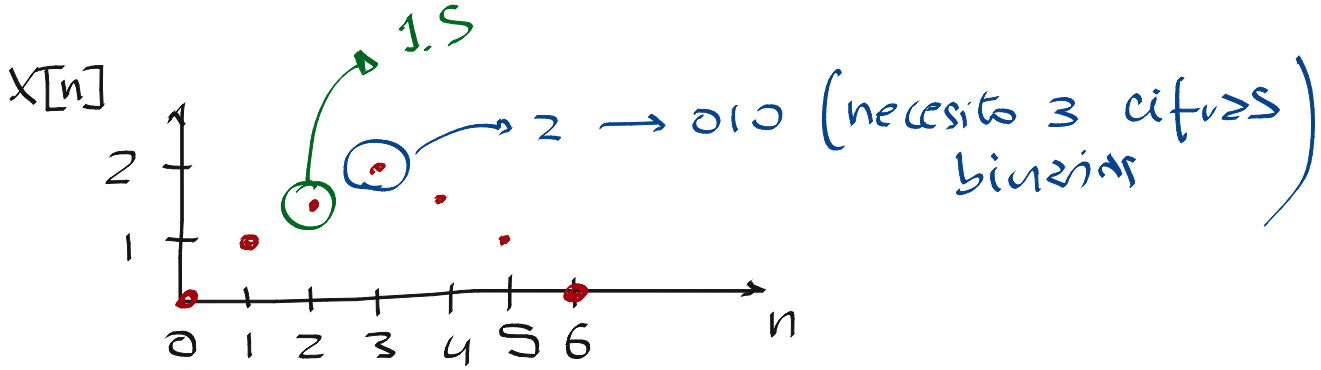
Diseño digital

Información analógica vs digital

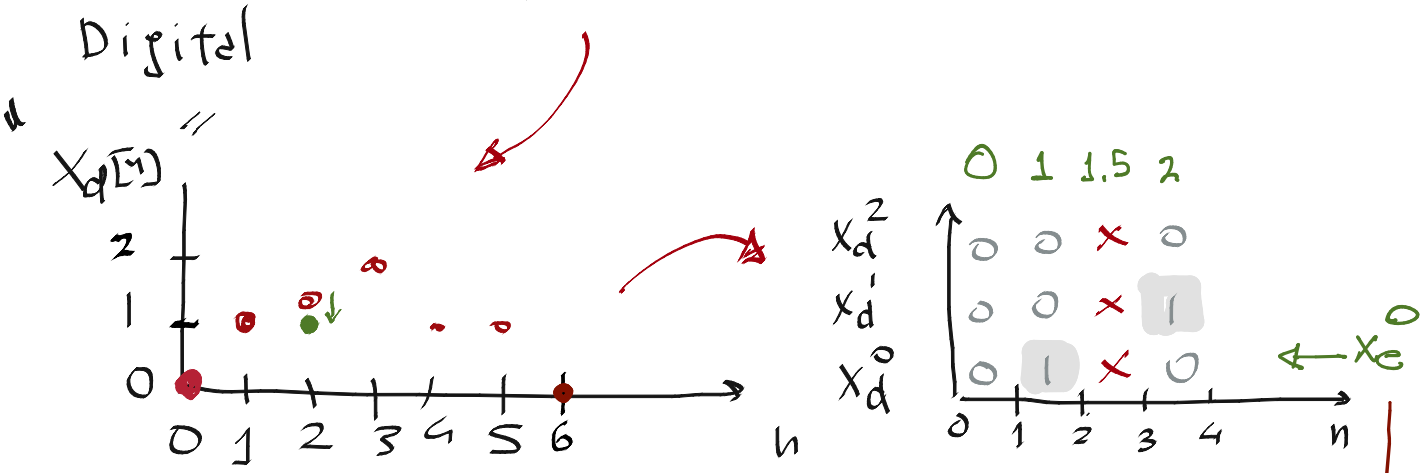
1) Analógico / continuo.



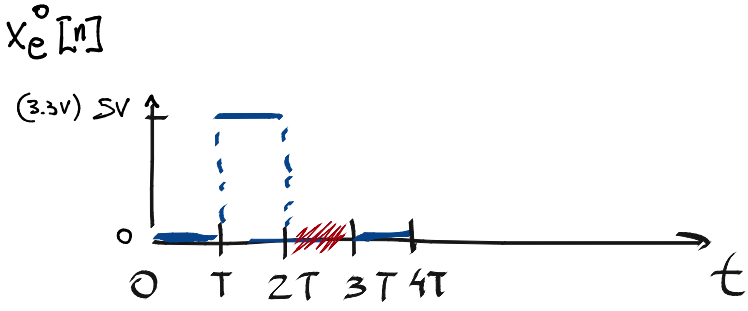
2) Discreto ($T = 1.0\,ms$)



3) Digital



4) Representación física (analógica) de lo digital



$T = 1\,\mu s \Rightarrow f_{clock} = 1\,MHz$

Compuertas lógicas

* transformación entrada-salida:
  1) funcional
  2) tabla de verdad
  3) circuito

Ejemplo: "OR Lógico"

- entradas: A, B
- salida: O

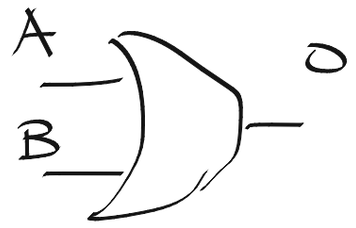
2)

A	B	O
0	0	0
0	1	1
1	0	1
1	1	1

cantidad combinaciones ← $2^2 \rightarrow 2^{\#E}$

tabla verdad

3)



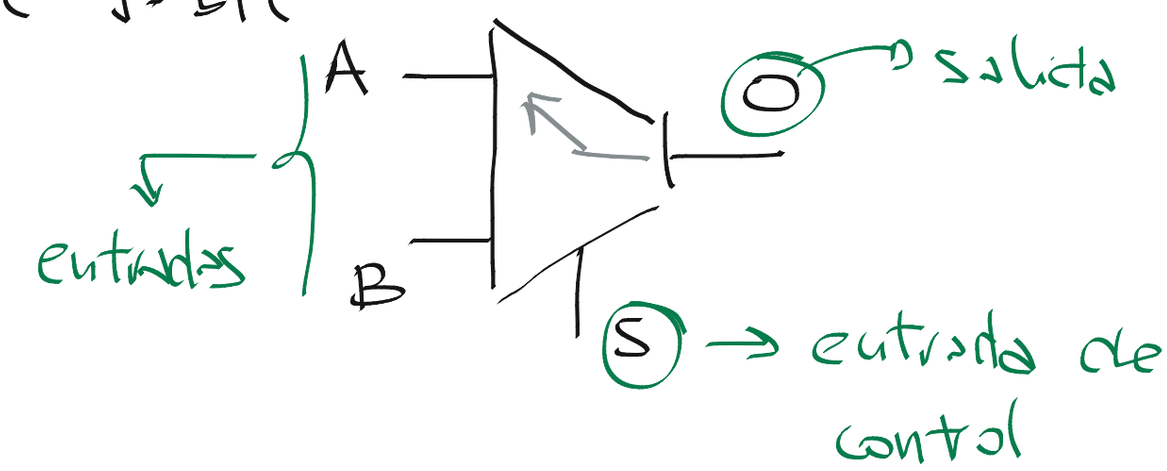
circuito (1 compuerta)

1) $A \oplus B = O$ ← Álgebra de Boole.

⊕ → OR (no suma)

funcional

Ejemplo: "multiplexer de 1-bit"



1) 2^3 combinaciones

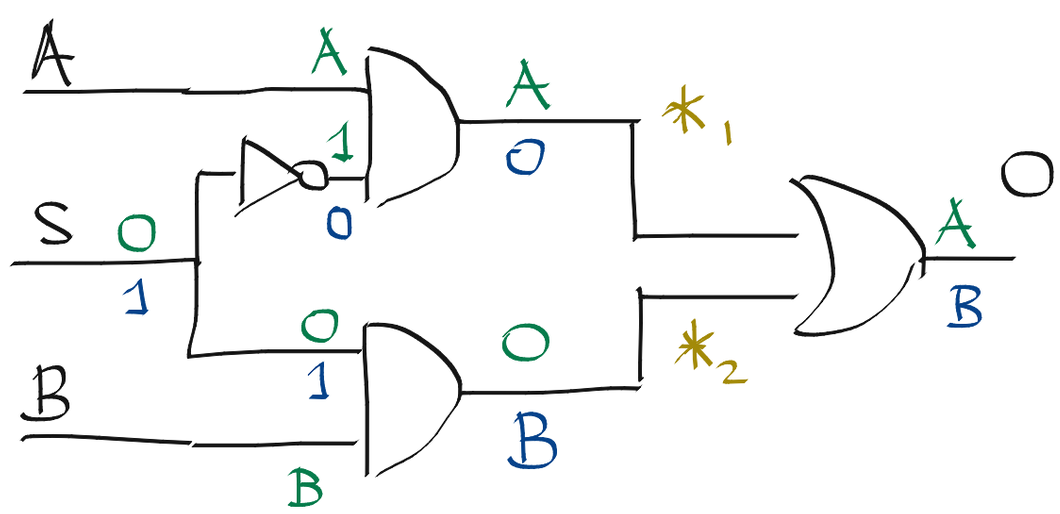
S	A	B	O
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

S=0 / S=1

defino si $S=\emptyset \Rightarrow O=A$ (deja pasar A)

$O = \begin{cases} A & \text{si } S=\emptyset \\ B & \text{si } S=1 \end{cases}$

3)



2) $O = \underbrace{\bar{S} \cdot A}_{*_1} + \underbrace{S \cdot B}_{*_2}$ ← Algebra de Boole

obs: * las tres representaciones son intercambiables

CLASE 17

Sistemas de numeración

SISTEMAS DE NUMERACIÓN

Sistemas posicionales

$$D = \underbrace{d_1 \cdot 10^1 + d_0 \cdot 10^0}_{\text{parte entera}} + \underbrace{d_{-1} \cdot 10^{-1}}_{\text{parte fraccionaria}}$$

base: $r \geq 2$

dígitos: * $d_i < r \quad \forall i$

$$D = \sum_{i=-n}^{p-1} d_i \cdot r^i$$

* p dígitos en la p.e
* n dígitos en la p.f

expresión: $d_{p-1} \ldots d_1 d_0 \bullet d_{-1} \ldots d_{-n}$

- d_{p-1}: dígito más significativo
- $\bullet$: pto. decimal
- d_{-n}: dígito menos significativo

bases comunes:

base 10 (decimal): $73_{10} = 7 \cdot 10^1 + 3 \cdot 10^0 \qquad r = 10 \quad , d_i = \{0..9\}$

$d_1 d_0$

base 2 (binario): $r = 2 \quad , d_i = \{0, 1\}$

$$1^4\,0^3\,0^2\,1^1\,1^0 = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

$\uparrow$ MSB (most significant bit) $\qquad \uparrow$ LSB (least sig. bit) $\qquad = 19_{10}$

Obs: * python: bin2dec, dec2bin

$$B = \sum_{i=-n}^{i=p-1} b_i \cdot 2^i$$

obs: * la representación es única: $5 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$

$5 = 2 \cdot 2^1 + 1 \cdot 2^0$ ✗

* La única excepción son los ceros a la izquierda: 1 → 01

ejemplo: $101.001_2 = 2^2 + 1 + 0.1 \cdot 2^{-3} = 5,125_{10}$

base 16 (hexadecimales) "abreviar números binarios largos"

$1001\,0111 = 97_{16} = \emptyset x97$ ← conversión por sustitución !!

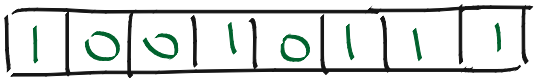
9 7 d_1

$2^7 + 2^4 + 2^2 + 2 + 1 = 151_{10}$ $\quad 9 \times 16 + 7 = 151_{10}$

Representación física

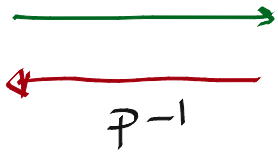
* Existe un elemento eléctrico llamado FF → almacena 1 bit
* Registro: elemento capaz de almacenar N bits
  → formado por N FF
  → unidad mínima de almacenamiento (palabra o word)
* representar con N cifras es: $\underbrace{r\ r\ r\ r}_{N}$

r^N números. $\neq (2^N)$

* byte: 8 bits 

* Conversión por sustitución

Conversión general



* expandir la fórmula: $D = \sum_{i=-n}^{p-1} d_i \cdot r^i$

$$1CE8_{16} = 1 \cdot 16^3 + 12 \cdot 16^2 + 14 \cdot 16^1 + 8 \cdot 16^0 = 7400_{10}$$

* Conversión por recurrencia:

$$D = d_{p-1} \cdot r^{p-1} + \dots + d_2 r^2 + d_1 r^1 + d_0$$

$$D = \underbrace{\left(d_{p-1} \cdot r^{p-2} + \dots + d_2 r + d_1\right)}_{q_0} r + d_0$$

$$D = \left(\underbrace{\left(d_{p-1} \cdot r^{p-2} + \dots + d_2\right)}_{q_1} r + d_1\right) r + d_0$$

* Convertir a base r: $d_0 < r$

$$D = q_0 \cdot r + d_0 \leftarrow \text{división entera}$$

→ ej: $7/3 = 2.33\ldots$ → $7 = 2.33 \cdot 3$ (coc, div)

7	3
1	2

→ $7 = 2 \cdot 3 + 1$

D	r
d_0	q_0

→

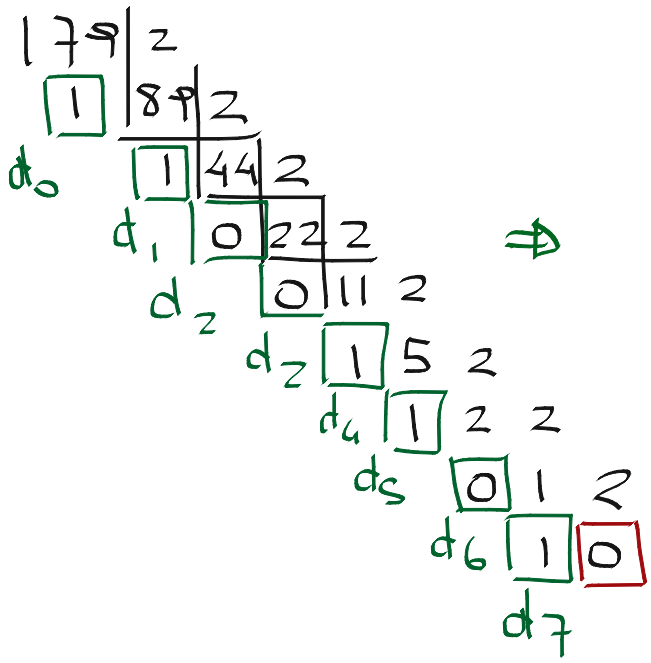
q_0	r
d_1	q_1

python: $7/3 = 2$

$7/3.0 = 7.33$

$7.0/3 = 2.33$

Ejemplo: 179_{10} a binario.



$\Rightarrow$ 1 0 1 1 | 0 0 1 1 $= B3_{16} = \emptyset x b3$

B 3

OPERACIONES

Suma

Algoritmo

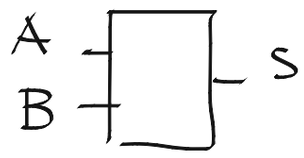
	C	0	1	0	0
+	X		1	9	0
	Y		1	4	1
	S		3	3	1

← acarreo (carry)

Obs: * sólo tengo elementos (compuertas básicos

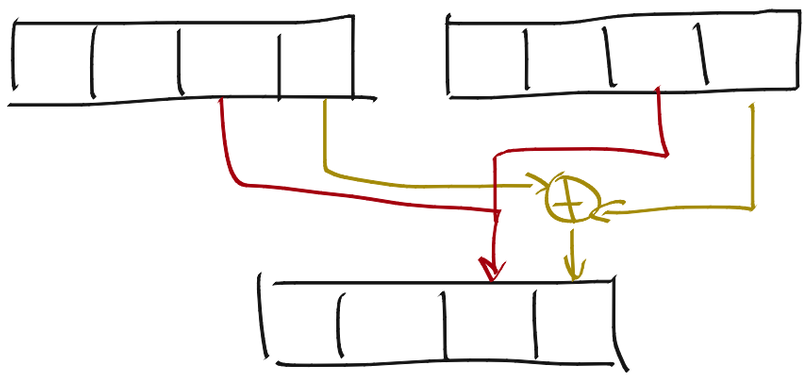
* supongo que suma dos dígitos a la vez

* necesito acarreo y suma



Half - Adder (semi-sumador)

* Suma binario

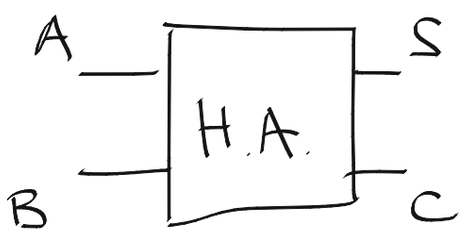


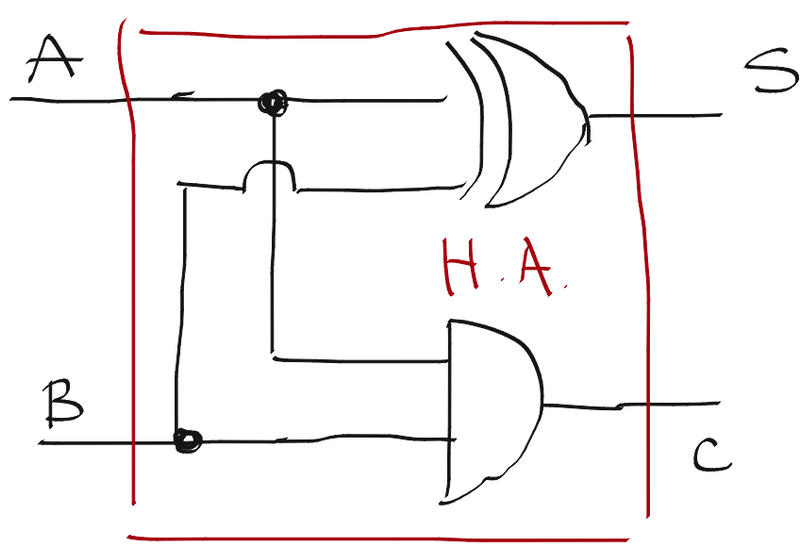
Obs: * la suma de 1 bit tiene 3 entradas y 2 salidas

* asumo que no hay acarreo

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

AND XOR





CLASE 18

Sistemas de numeración

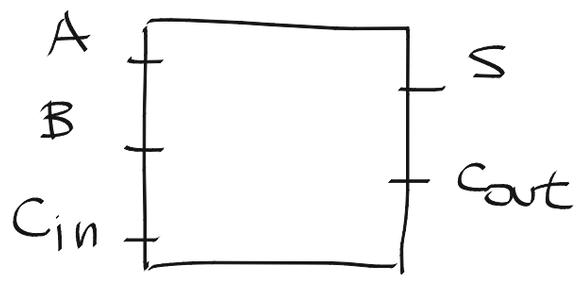
Numeros con signo y multiplicación

Suma (cont)

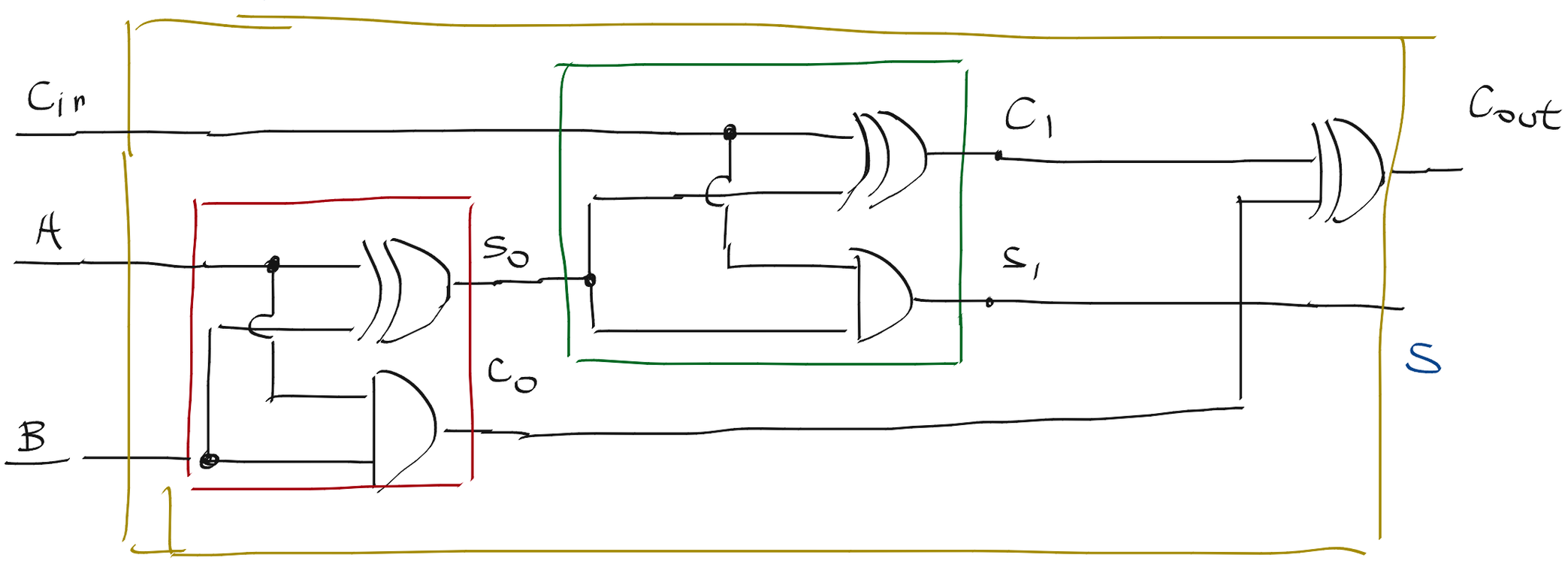
Full - Adder (sumador completo)

C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

C_0	S_0	C_1	S_1
0	0	0	0
0	1	0	1
0	1	0	1
1	0	0	0
0	0	0	1
0	1	1	0
0	1	1	0
1	0	0	1



C_0	C_1	C_{out}
0	0	0
1	0	1
0	1	1
1	1	0



dss:
* Suma completa de 1 bit: → 5 compuertas
  → 2 integrados (3 XOR, 2 AND)
* Sumar un decimal (8 full-adders): → 40 compuertas
* Puedo calcular uso de recursos:
  → espacio en chip (PLD) (40/N compuertas)
  → Consumo (2 integrados)
  → Retardo (3 compuertas) ~ 45 nS

Resta:

	B	0	1	0
−	X		2	2	9
	Y		0	4	6
	D		1	8	3

←

B	0	0	1	1	1	1	1	0	
X		1	1	1	0	0	1	0	1
Y		0	0	1	0	1	1	1	0
D		1	0	1	1	0	1	1	1

Obs: puedo hacer un comparador sólo mirando el bit de borrow

Suma hexadecimal

	C	1	1	0	0
+	X	1	9	B	9
	Y	C	7	E	6
	S	E	1	9	F

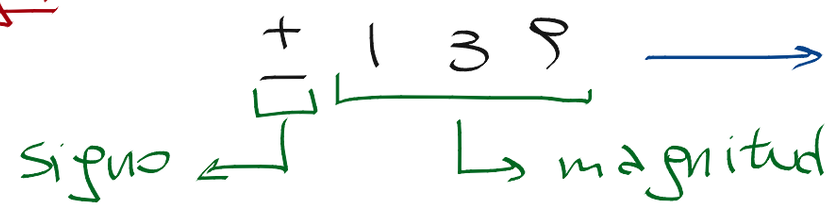
	1	1	0	0
	1	9	11	9
	12	7	14	6
	14	17	25	15
		16+1	16+9	
	E	1	9	F

Obs: * cálculo del carry:

25	16
9	1 → carry

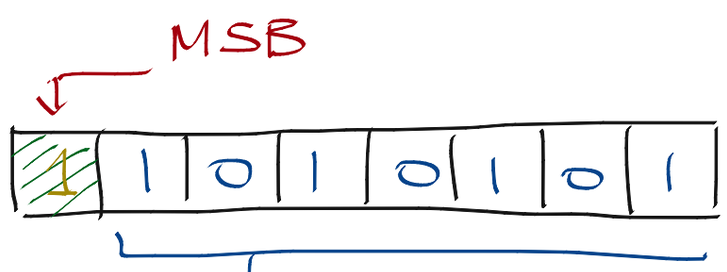
NÚMEROS CON SIGNO

Magnitud con signo



$\longrightarrow \pm 0$ (dos rep $\neq$)

Bit de signo



ejemplo $n=8$

* binario sin signo: $2^7+2^6+2^4+2^2+2^0 = 213_{10}$
* binario con signo: $2^6+2^4+2^2+2^0 = -85_{10}$
  - → MSB: _ 1 negativo
    _ 0 positivo
* "pierdo" 1 bit: → antes: $[0, 2^8-1] = [0, 255]$
  → ahora: $[-2^7+1, 2^7-1] = [-127, 127]$
* tengo 2 ceros
* sumas y restas: decisiones (complejas)
* multiplicar y dividir: son fáciles
* es bueno poder testear fácil si es negativo

Complemento

* difícil de representar
* algunas operaciones se simplifican
* complemento a la base ó reducido

Complemento a la base

$$C = r^n - D \quad , \quad n \text{ cantidad cifras}$$

* si $D \in [1, r^n - 1] \Rightarrow C \in [1, r^n - 1]$

ejemplo

$n = 4$

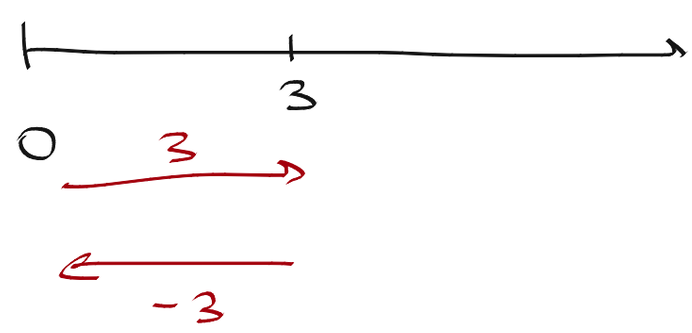
$r = 2^4 = 16$

$D \in [1,15] \Rightarrow C \in [1,15]$

* si $D = 0$, $C = r^4$ $\qquad D = 0 \rightarrow C = 16$

* r^n no es representable con n cifras: $r^n = \cancel{1}\,|0|0|0|0| \rightarrow C = 0$

* No necesito resta, la expresamos como suma del opuesto, al que ahora llamaremos complemento



Receta

$$C = \overline{\left(\left(r^n - 1\right) + D\right)} + 1 \quad , \quad D_{max} = \underbrace{m\,m\,\ldots\,m}_{n \text{ veces}}$$

D_{max}

* m es el dígito máximo. $m = \max\{dig\}$

Ejemplo: $r=10$, $n=4$, $D=53$

A) fórmula directa: $C = r^n - D = 10^4 - 53 = 10000 - 53 = 9947$

	0	1	1	1	0
r^n	1	0	0	0	0
D				5	3
C	0	9	9	4	7

B) Receta:

1) $m = r - 1 = 9 \rightarrow D_{max} = 10^4 - 1 = 1000 - 1 = \underbrace{9999}_{n=4}$ (n)

2) $\bar{C} = D_{max} - D$ complemento dígito a dígito.

3) $C = \bar{C} + 1$

	D_{max}	9	9	9	9
−	D			5	3
	$\bar{C}$	9	9	4	6
+					1
	C	9	9	4	7

c_i: complemento al dígito máximo.

obs: * Sólo sumas y restas simples
* no hay préstamo ni acarreo
* comp. al dígito. $\boxed{c_i = m - d_i}$

Complemento a 2

* es negativo sii MSB=1
* peso del MSB es -2^{n-1}
* rango representable $[-2^{n-1}, 2^{n-1}-1]$

ejemplo

$n=8$

-128

$[-2^7, 2^7-1] = [-128, 127]$

17_{10} = 0 0 0 1 | 0 0 0 1 D

1 1 1 0 1 1 1 0 $\bar{C}$

1 C2

1 1 1 0 1 1 1 1 Z

$-2^7+2^6+2^5+0\ 2^3+2^2+2^1+2^0 = -17_{10}$

Suma y resta en complemento a2

+ 3 | 0 0 1 1
+ 4 | 0 1 0 0
= 7 | 0 1 1 1

+ 6 | 0 1 1 0
+ -3 | 1 1 0 1
= 3 | X 0 0 1 1 (acarreo descartado)

+ -2 | 1 1 1 0
+ -6 | 1 0 1 0
= -8 | X 1 0 0 0 (acarreo descartado)

-2^3

-8

3 → 0 0 1 1 D

1 1 0 0 $\bar{C}$

1 C2

1 1 0 1 C

2 → 0 0 1 0 D

1 1 0 1 $\bar{C}$

1 C2

1 1 1 0 C

$n=4 \rightarrow [-2^3, 2^3-1]$

$[-8, 7]$

* El acarreo en el MSB se ignora

Desbordamiento (overflow)

1 0 0 0 0

+ -3 | 1 1 0 1
+ -6 | 1 0 1 0
= 7 | X 0 1 1 1

obs: * "representación gráfica" [ver tabla en slide #5]

CLASE 19

Códigos y arquitectura

Multiplicación

```
  1 1
X 1 3
-------
  3 3      ← 2 mul
1 1 //     ← 2 mul
-------
1 4 3      ← 1 suma
```

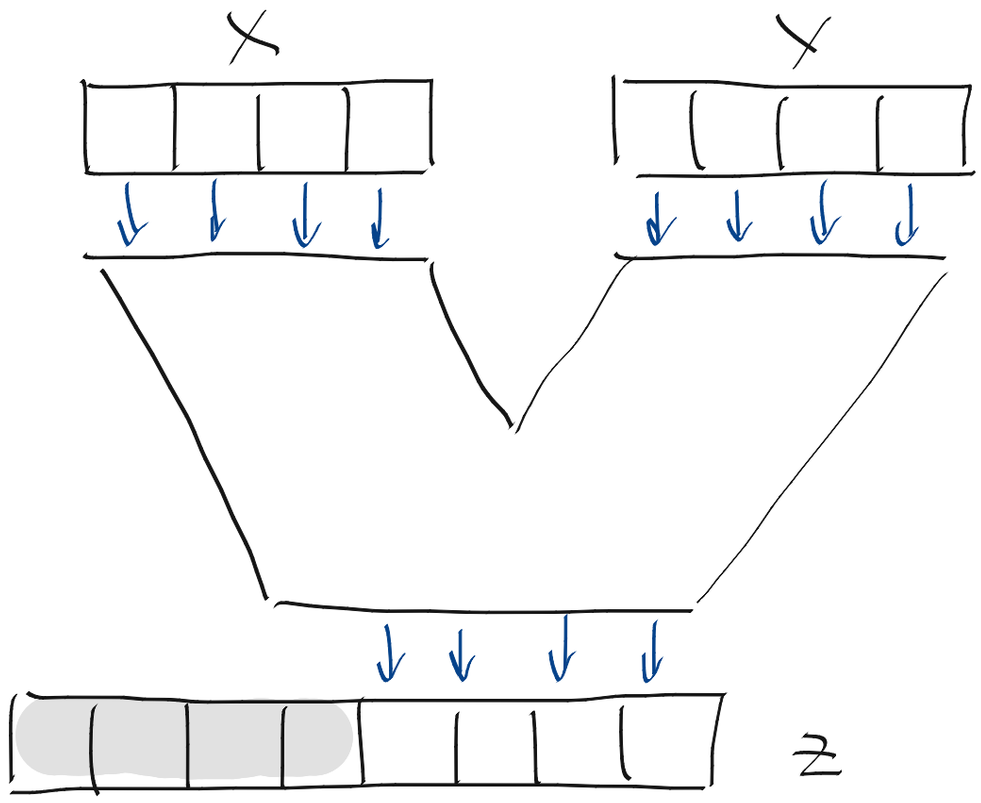
←

```
    1 0 1 1
  X 1 1 0 1
-------------
(1)(1)(1)1 0 1 1      ← 4 ands
     0 0 0 0          ← 4 ands
   1 0 1 1            ← 4 ands
 1 0 1 1              ← 4 ands
-------------
1 0 0 0 1 1 1 1
  1 2 3 2            ← 6 sumas
```

8 bits

$z^7 + 2^3 + 2^2 + 2^1 + 2^0 = 143_{10}$

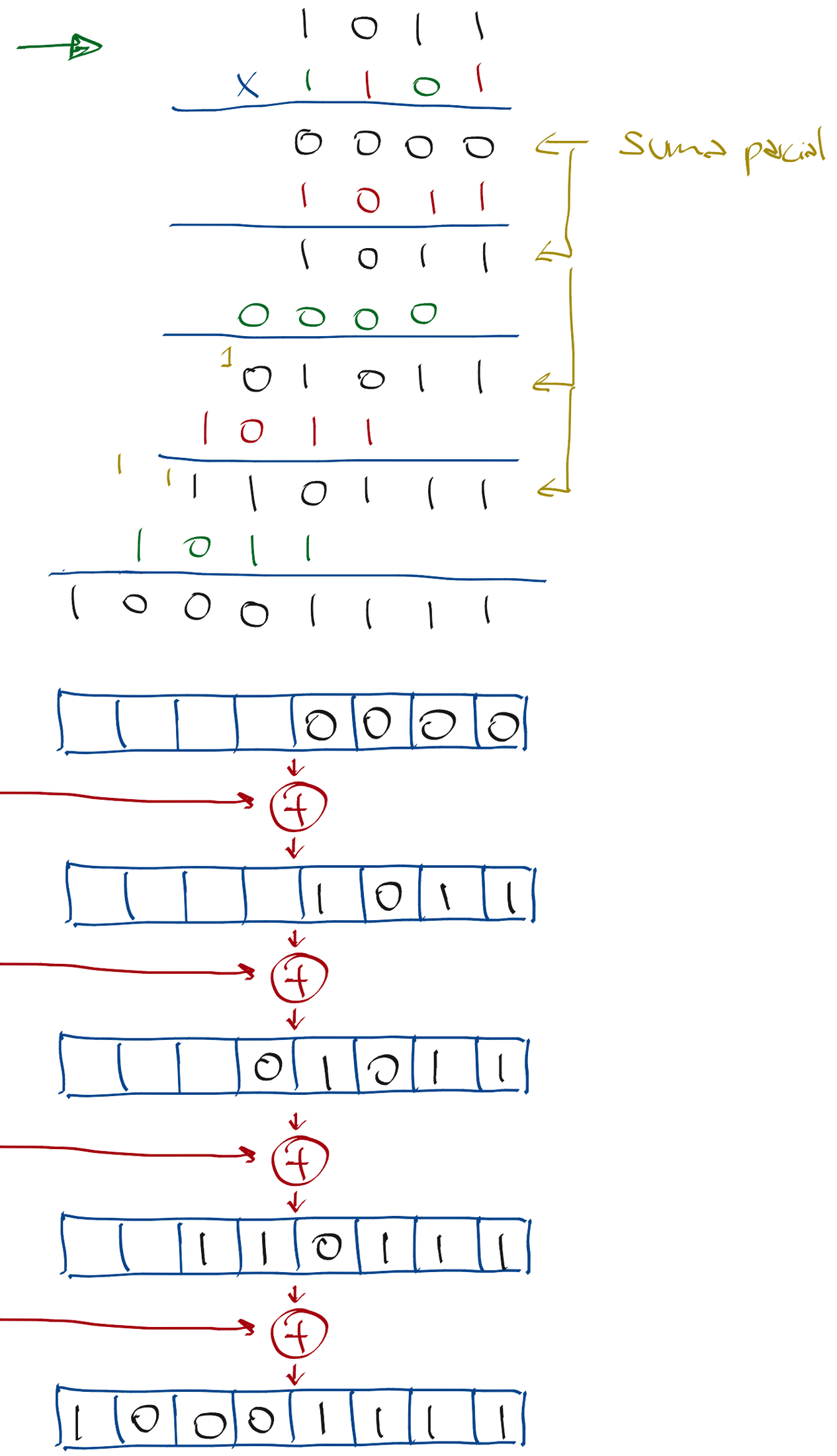
ejemplo: n=4

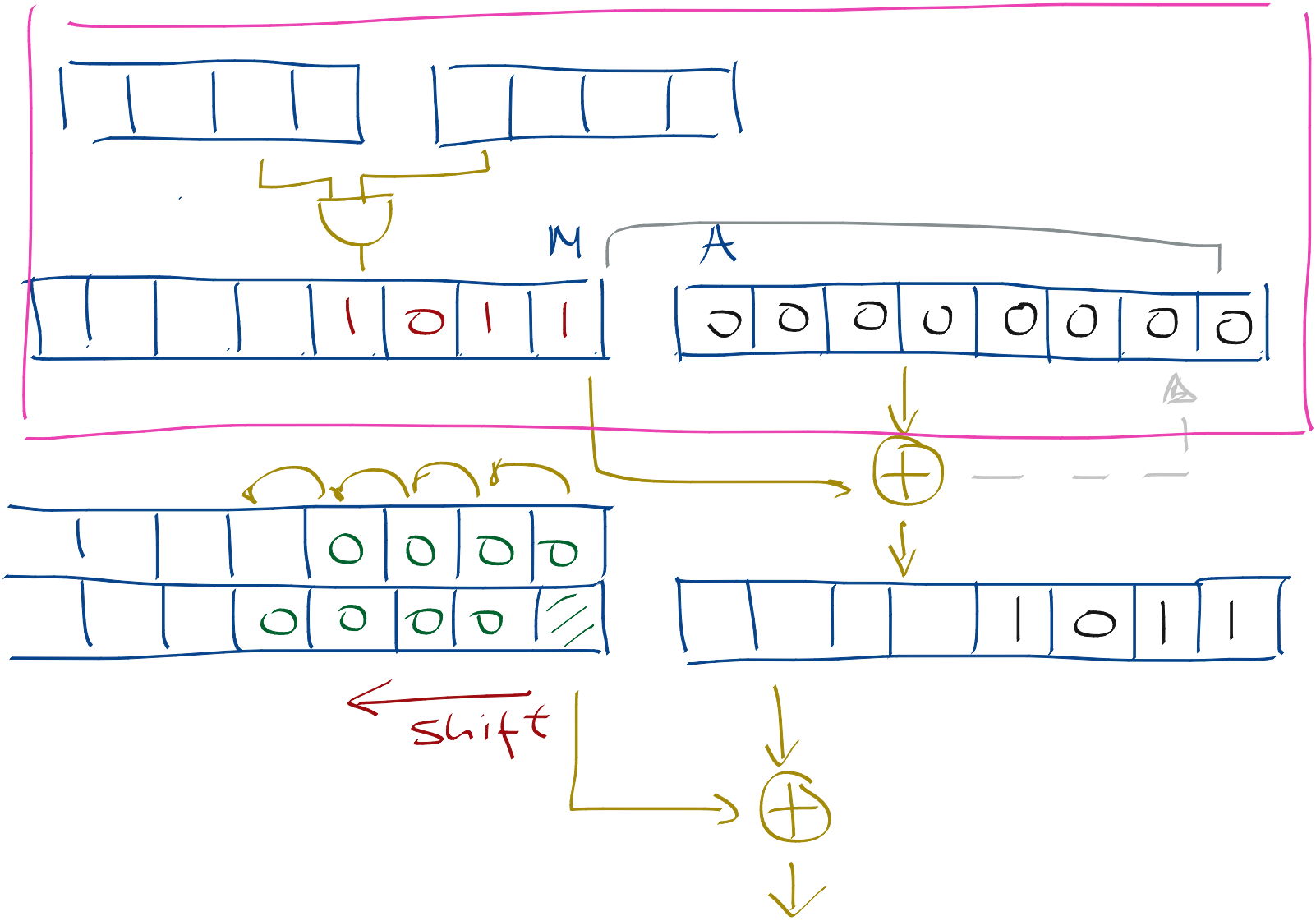


obs: * la multiplicación es mucho más complicada
* en complemento a dos es más difícil de hacer

ejemplo: "implementación"

```
  1 1
X 1 3
```




M
A
1 0 1 1
0 0 0 0 0 0 0 0

1 0 0 0 0
0 0 0 0
1 0 1 1

shift

CÓDIGOS Y ARQUITECTURA

Tipos de objetos

* datos: enteros, reales, texto (strings)
* programación: → ¿Qué son? ¿Cómo se representan?
  ↕ → ¿Cómo se opera con ellos?
* Arquitectura: → ¿Cómo se almacenan?
  → ¿Cómo se implementan las operaciones en HW?

Unidades: * bit: dígito binario, valor 1 o 0
* palabra: secuencia ordenada de bit
* largo de palabra: n=8 (byte) → n=64

Código: * asociación entre el objeto real y la palabra digital que lo representa
* no es única: → mag. y signo
  → comp. a 2
* largo N: cuantos objetos puedo representar

Que podemos representar

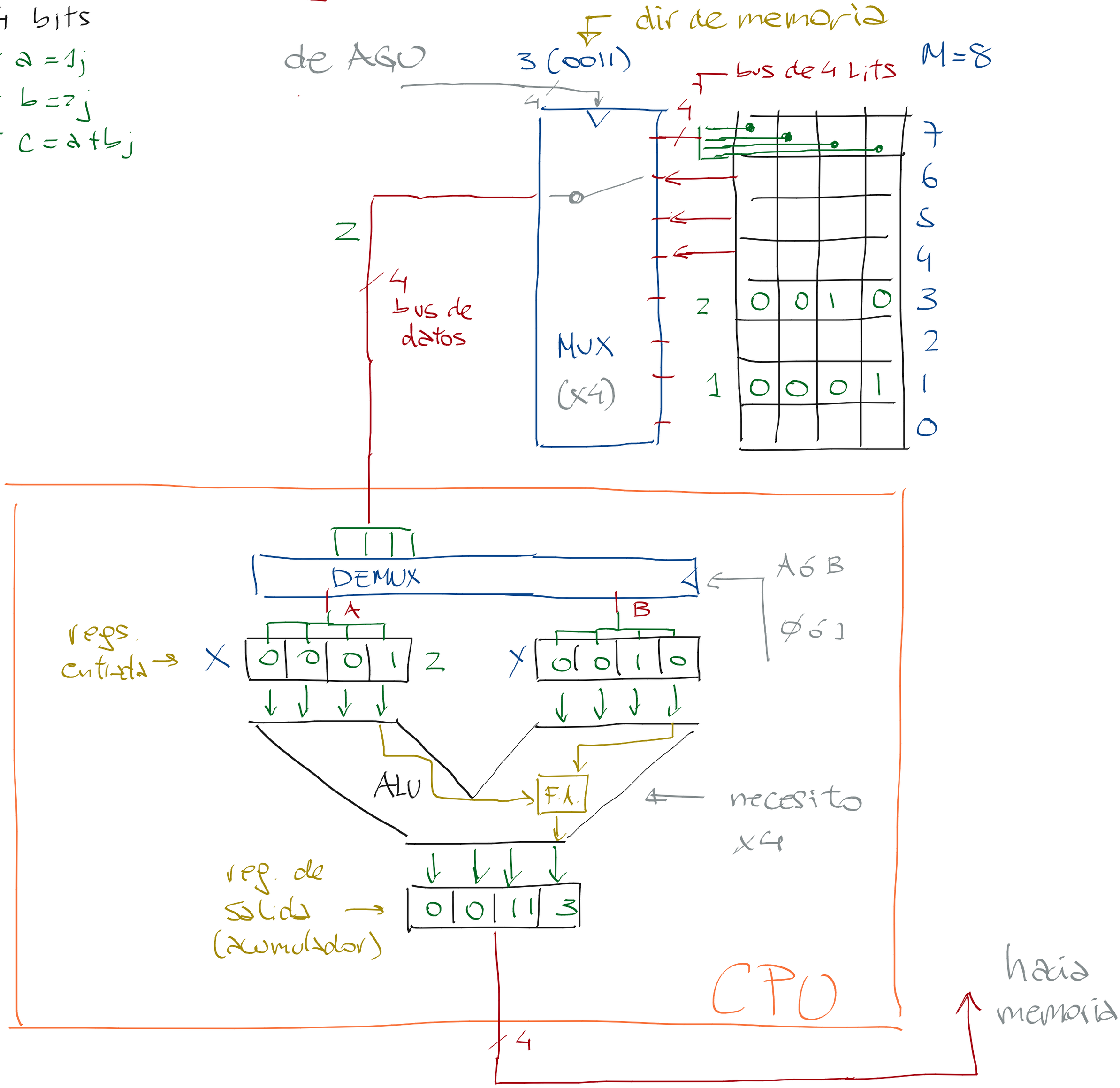
* datos: → números
  → caracteres
  → objetos compuestos: persona (edad, nombre, peso)

  natural | string | real
* funcionamiento del sistema digital
  → programa (instrucciones)
  → estados
  → direcciones de memorias

Diagrama de una arquitectura sencilla

n=4 bits

```
int a = 1;
int b = 2;
int c = a+b;
```



Representaciones para otros usos (números decimales)

* binario común es más apropiado para la representación interna y para operaciones aritméticas
* la interfaces humanas usan decimales
* algunos sistemas de alto nivel (PL) fueron "procesar en decimal"

Obs: * 10 números decimales (dígitos), N = 10, n bits cantidad $= 2^n$.

Victoria → 0
Juan → 1
Manel
Renato
Lucía.

] 1 bit.

] 2 bit.

$z^n = 10 \Rightarrow \log_{10} z = n^*$

$n^* = 3.3 \rightarrow \boxed{n_0 = 4}$

Código BCD: "Binary Coded Decimal"

* 4 bits → 1 dígito (9 números)

* en un sistema de palabra n = 8
  - → representación empaquetada: 1 nibble por dígito
  - → n = 8 — represento de 0 a 99
    — en binario "común" 0 a 255
  - → menos eficiente

* mapeo "casi" directo a decimal
* las operaciones aritméticas son más difíciles

Ponderación

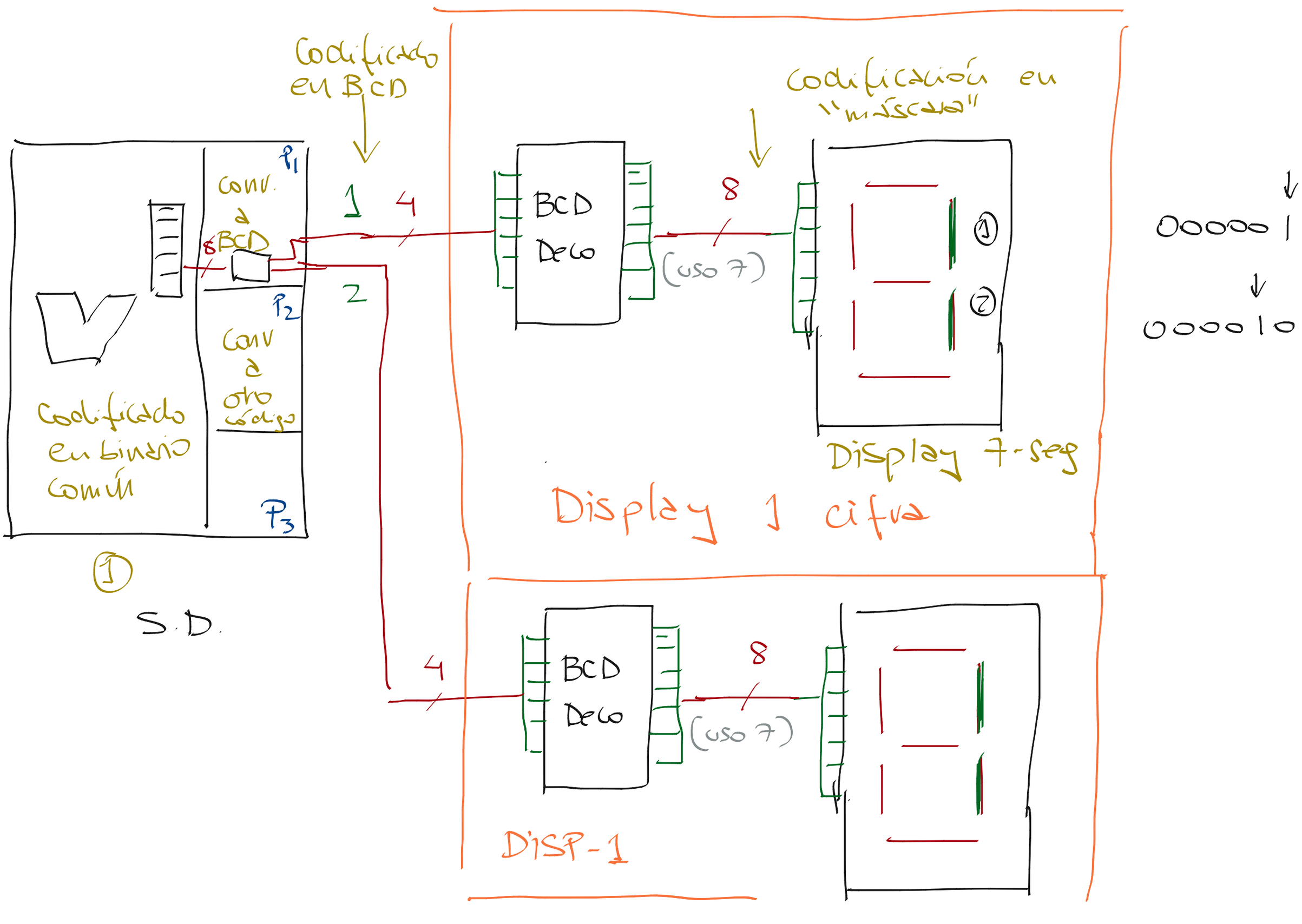
* código (8421): pesos ordenados.
* código (2421): autocomplementado (complemento a la base)

Periféricos

* dispositivo externo conectado a un sistema digital
  - → interacción con el mundo exterior: sensores, actuadores
  - → comunicación: redes, teclado, mouse, pantalla
  - → almacenamiento persistente: disco duro.

* todo lo que no es CPU + memoria
  - ALU AGU control almacenamiento temporal.

Ejemplo: "display de 2 cifras"

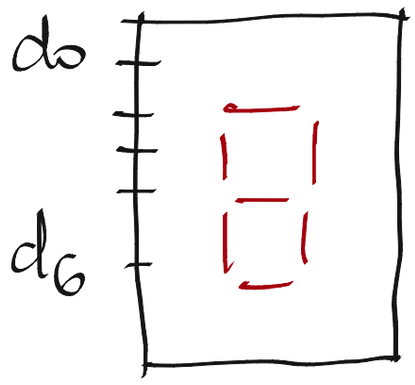


① S.D.

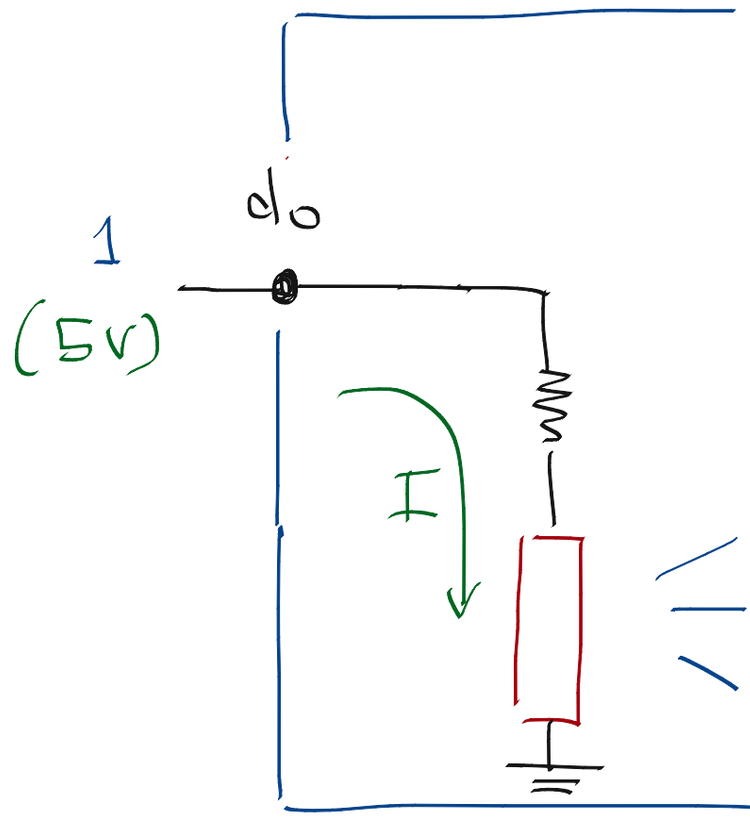
CLASE 20

Códigos y errores

Ejemplo: "display 7-segmentos"



* Periférico:
  - → salida
  - → transducción de voltaje a luz
  - → D.A o D.D.
* Cada luz (led) es un bit:
  - → 0 apagado } ⇒ activo por nivel alto
  - → 1 prendido }
* Cómo prendo un led?
  - → 1 de 7 (1 de N), máscara
  - → muy usada en el control de periféricos



D:	d_7							d_0
0	0	0	0	0	1	0	0
G				C	B	A	

* la palabra D necesita un código distinto
  - → Cómo prendo el 1?

D:	d_7							d_0
0	0	0	0	0	1	1	0
G							A

vs | | | | |
|---|---|---|---|
| 0 | 0 | 0 | 1 |

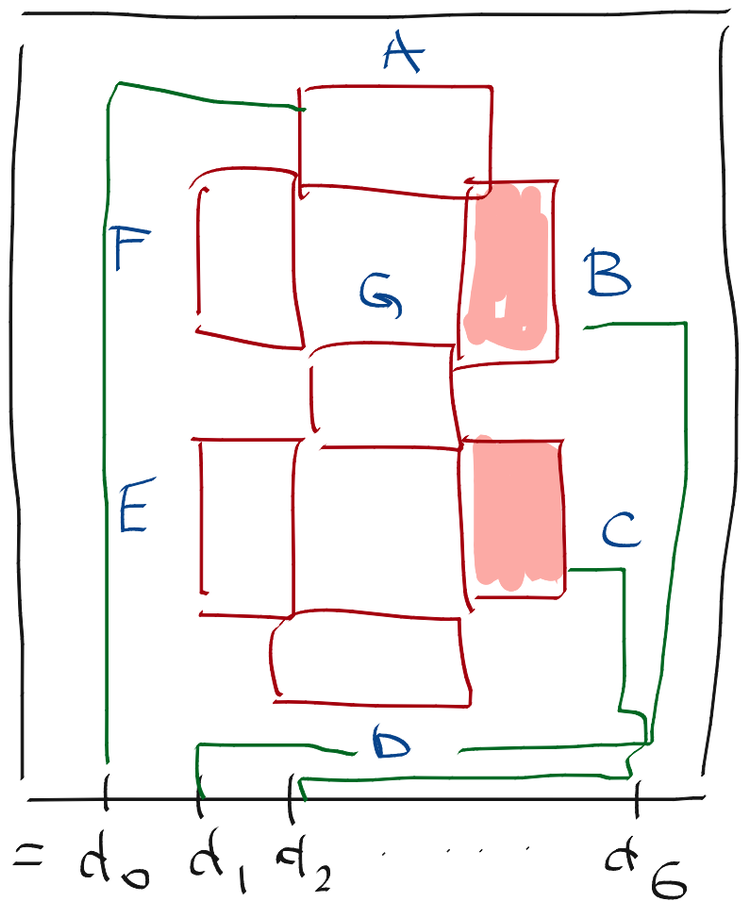
||

0	0	0	0	0	0	1	0	B

+

0	0	0	0	0	1	0	0	C

← OR lógico.



* Código (ARDUINO):

```
w_B = 0000 0010;
w_C = 0000 0100;

w = w_B | w_C;   <- OR bit a bit
```

Ejemplo: "Decodificador BCD"

	BCD				DISPLAY (D)						
decimal	b_3	b_2	b_1	b_0	d_6	d_5	d_4	d_3	d_2	d_1	d_0
0	0	0	0	0	0	1	1	1	1	1	1
1	0	0	0	1	0	0	0	0	1	1	0
2											
3											
4											
5											
6											
7											
8											
9											

* 7 funciones lógicas
* Diseñ cada una con el método anterior

Def: "Puerto"

* interfaz para conectar periféricos
* protocolo: lenguaje de comunicación con el periférico.

Ejemplo: "Puerto digital del Arduino"

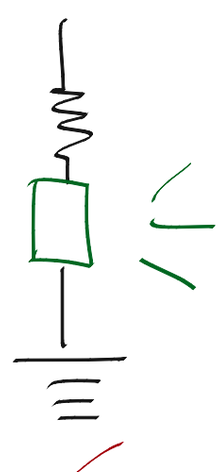
* Puerto digital multipropósito

* el código o conjunto de palabras de control es propio del periférico

LUZ ADAT X

0	0	1	0
33	32	31	30

0 1 0 0



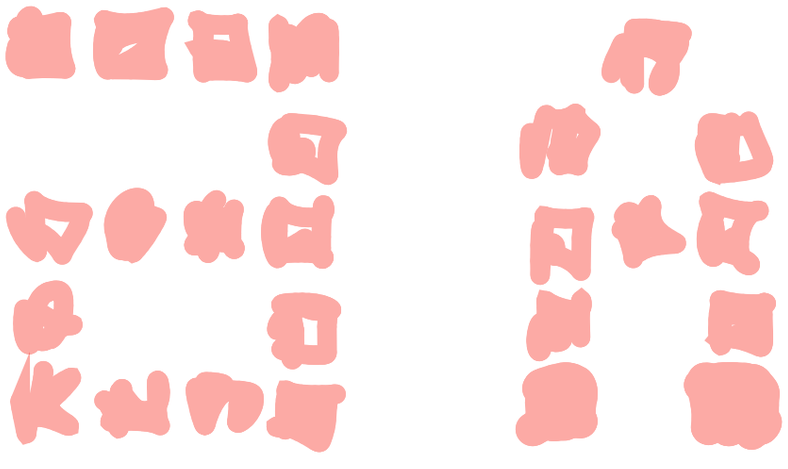
* no todas las palabras son válidas: 0 1 X 0

* Control del autito:
  - → configuran: – palabras de control: altera el comportamiento
    - configurePin(31, OUT)
    - Serial.begin(9600)
  - → Envío de datos: – digitalPinOut(31, HIGH) → 1 → 5V
    - digitaPortOut(0010)

REPRESENTACIÓN DE CARACTERES

ASCII * números y caracteres en 1 byte ($n=8$), $N=256$

* cantidad:
  - → 10 números
  - → 27 letras (inglés)
  - → 27 letras mayúsculas
  - → 54
  - → acentos
  - → puntuación: . , ; -
  - → caracteres de control \n \t



Ejemplo:

* string: h o l a
* código (decimal): 104 | 111 | 108 | 097
* código (binario): 01101000 | 01101111 | 01101100 | 01100001

byte

[ver tabla en slide #7]

Tamaño:

* m caracteres
* 1 caracter = 1 palabra del S.D.
* Largo palabra n

Ejemplo:

* Sólo letras (mayúsculas y minúsculas): $m=54$
* puedo representar $N=2^n$

$$2^n = N \geq m \Rightarrow n \geq \log_2 m = n^* = 5.75 \Rightarrow n=6$$

me sobra

$$N = 2^6 = 64$$

UTF-8:
* 1 a 4 bytes = 32 bits
* super-conjunto de ASCII: los primeros 128 son iguales al ASCII
* "todos" los idiomas:
  - caracteres + acentos
  - occidentales
* largo variable

Resumen códigos

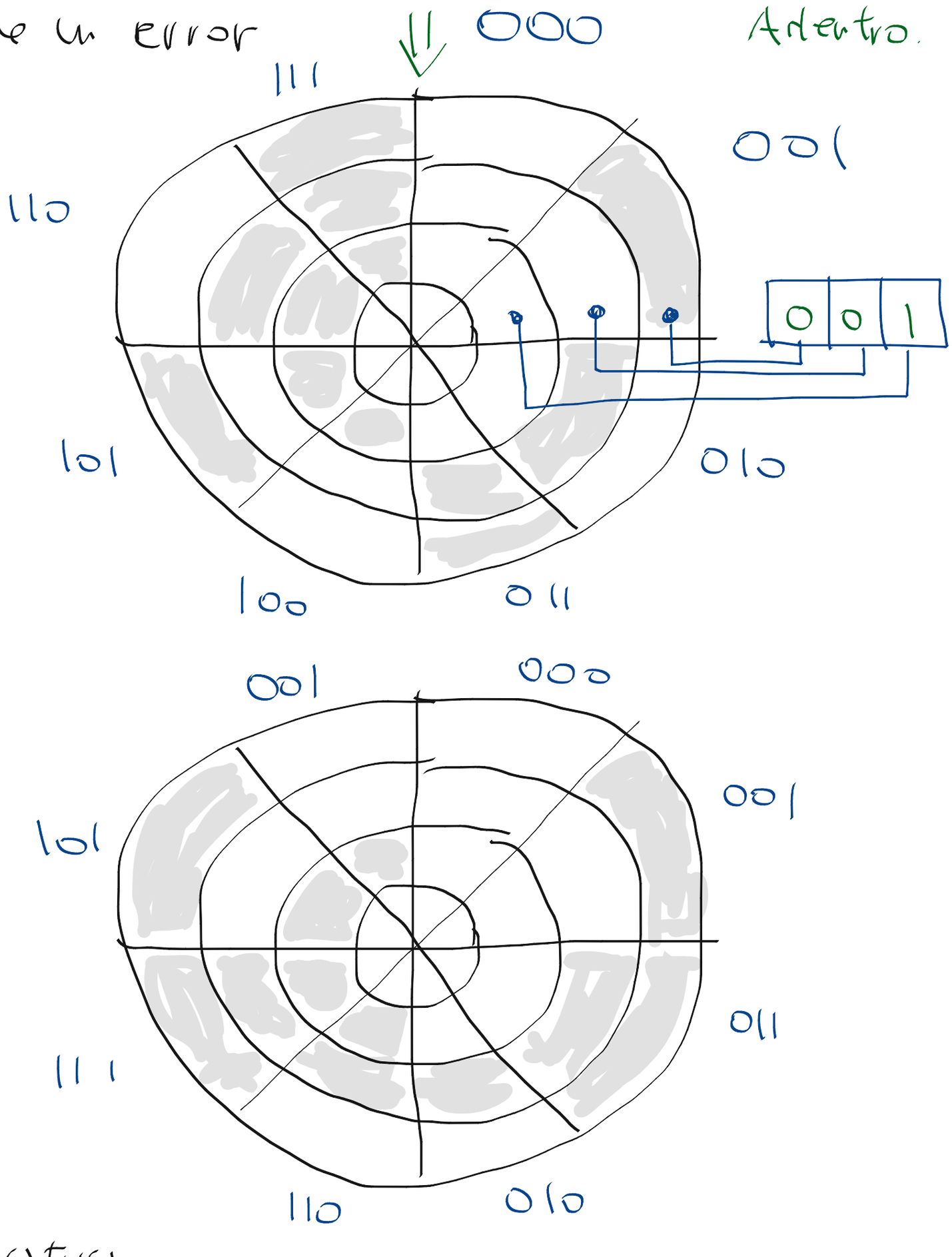
* mapeo arbitrario de objetos a bits
* diseñados para facilitar una tarea
* se mapea a circuitos diferentes

CÓDIGOS DE MANEJO DE ERRORES

Códigos de Gray

* minimizar el impacto de un error

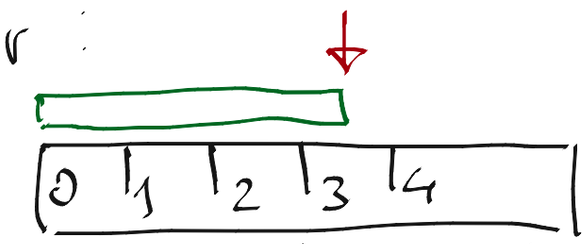
MSB
Adentro.



[mostrar dibujo hecho]

Errores:

* falla física: - temperatura
  - interferencia electromagnética
  - deterioro de la señal en la transferencia
* modelos de error:



  - → estadístico: $e \sim U[0,1]$
  - → ejemplos: - independiente por bit
    - ráfaga (raya disco)
  - → 1 error: cambio de 1 bit

distancia

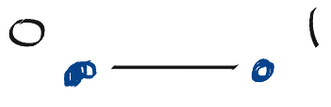
- entre palabras de un código.
- cantidad de bits que cambian

$d(000, 001) = 1$

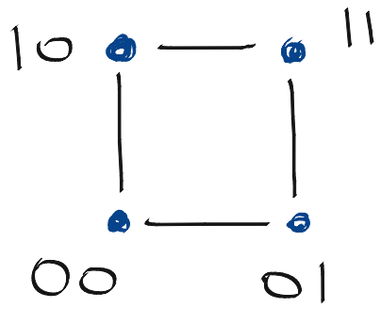
$d(101, 000) = 2$

cubos n

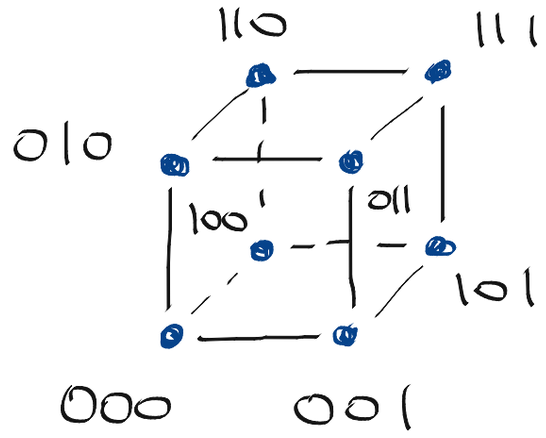
Cubo 1



Cubo 2



Cubo 3



distancia de Hamming

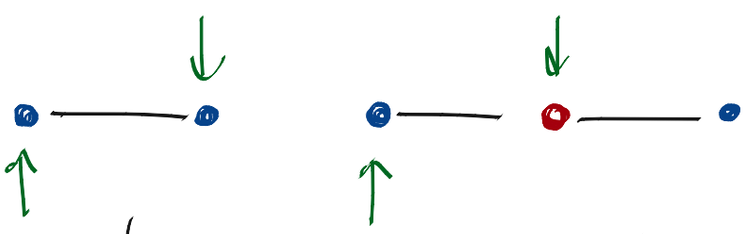
- largo (medido en aristas) del camino que une dos palabras.
- Greg: camino que visita los vértices una sola vez

CLASE 21

Códigos óptimos

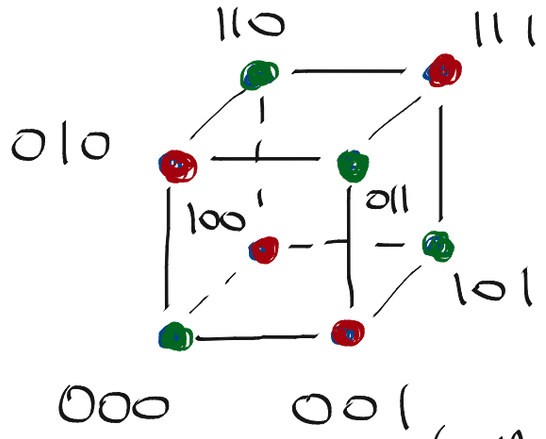
Códigos de detección

* Un código no tiene porque usar todos los bits (BCD)
* se definen palabras válidas:
  - si aparece una palabra inválida se fue hubo error
* Uso un subconjunto del total:
  - distancia sea mayor a 1
  - no haya vértices adyacentes en el cubo.



* Para detectar un error de 1 bit: $d_{min} \geq 2$

000
011
101
110
} tiene $d_{min}=2$



* Para detectar un error de 1 bit necesito el doble $(2^n . 2) = 2^{n+1}$

Bit paridad

* se agrega un bit para que el número de 1 sea par siempre.

Checksum (suma de verificación)

* cédula 4.139.701-5

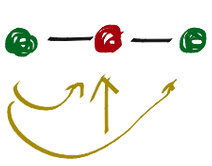
DETECCIÓN Y CORRECCIÓN DE ERRORES

* detección de largo $\underline{k}$, $d_{min} > k$

* Agregar $\underline{c}$ bits y $\underline{d}$ bits : → corregir $\underline{c}$ → detectar $\underline{d}$ } $k = c + d$

1) $d_{min} = 2$

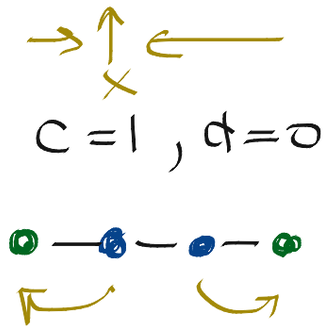
$c = 0$, $d = 1$



2) $d_{min} = 3$

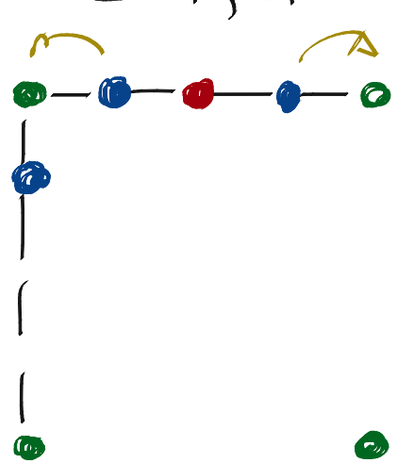
a) $c = 0$, $d = 2$

b) $c = 1$, $d = 0$



3) $d_{min} = 4$

$c = 1$, $d = 2$



$d_{min} = 2 \cdot c + d + 1$

$4 = 2 \cdot \underline{1} + \underline{1} + 1$

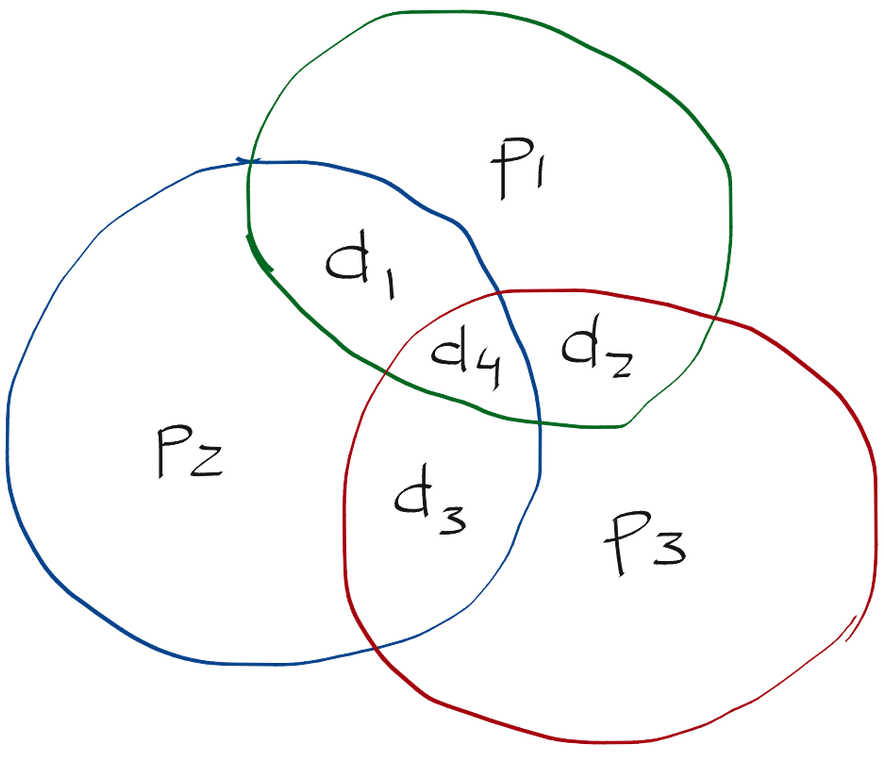
Códigos de Hamming

* 1950, R.W. Hamming

* $d_{min}=3$ → largo: 2^k-1 bits
  → información: 2^k-1-k

* Ejemplo: Hamming (7,4) → 7 en total (2^3-1)
  → 4 info (2^3-1-3)
  → 3 redundancia.

* mensaje: → info: $d_1 d_2 d_3 d_4$
  → paridad: $p_1 p_2 p_3$



* número de 1 en cada círculo sea par

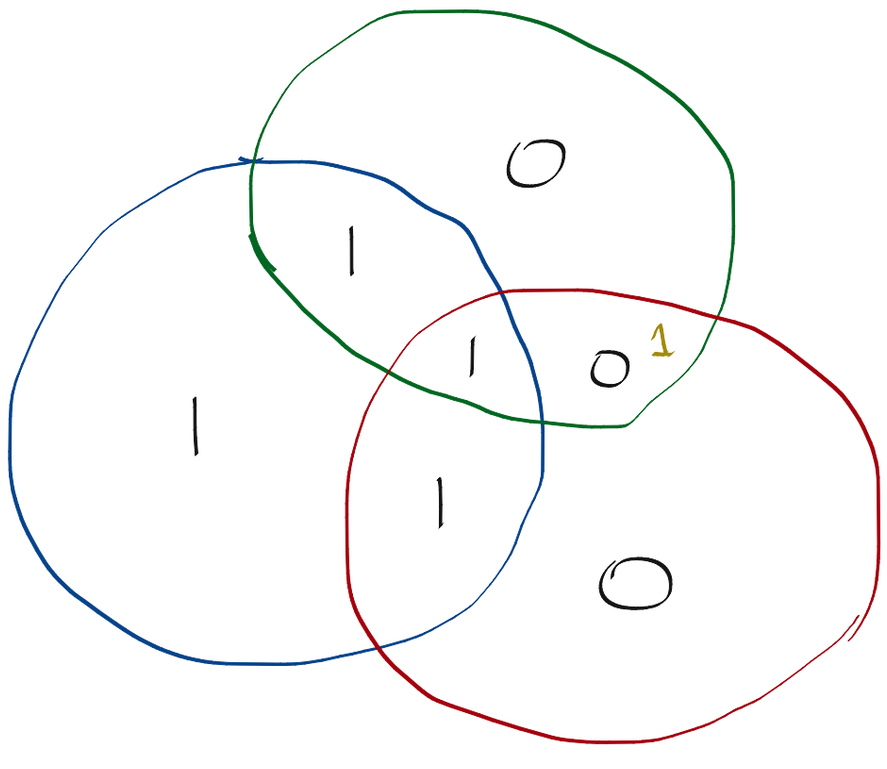
Ejemplo:

info: 1011

par: 0

mensaje: $p_1 p_2 d_1 p_3 d_2 d_3 d_4$

0 0 1 1 0 1 1



* protocolo: acuerdo entre partes (emisor y receptor)

CÓDIGOS ÓPTIMOS

Teoría de la información

C. Shannon "A mathematical theory of communication"

Objetivo: "crear un código que minimize el largo del mensaje"

Alfabeto:

* Conjunto de símbolos $D = \{d_i\}$, largo $\#D$
* Caracter dado $\underline{c}$ es V.A. con probabilidad $p_i = P(C = d_i)$
* Largo de palabra (en bits): $n \;/\; 2^n > \#D$ ← largo máx

Mensajes:

* $M = c_1 \ldots c_K$ secuencia ordenada de $\underline{K}$ caracters.
* Largo del caracter (en bits): $\ell(c_k) \leq n$
* Largo mensaje (en bits): $L(M) = \sum_{k=1}^{K} \ell(c_k)$

Probabilidad

$$P_{evento} = \frac{\#\text{casos favorables}}{\#\text{casos posibles}}$$

ej: prob. de recibir una letra $\frac{1}{256}$

ej: $P(c_{i+1} = u \mid c_i = q) = 1$

incertidumbre/variabilidad $\propto \frac{1}{\text{común del símbolo}}$

* Cuanto más raro m símbolo más bits necesito

Información:

$$I(c_k) = -\log_2(p_k) \rightarrow p_k = \frac{1}{256} \Rightarrow I(c_k) = -\log_2\left(\frac{1}{256}\right) = \boxed{8} \text{ bits}$$

$$p_k = 1 \Rightarrow I(c_k) = -\log_2(1) = \boxed{0}$$

Entropía

$$H(M) = \sum_k p_k \cdot I(c_k) = -\sum_k p_k \cdot \log_2(p_k)$$

Ejemplo: $D = \{a, b, c, d, e\}$ $\#D = 5$

* cuántos bits necesito para representar todos los símbolos

→ $2^n \geq \#D \Rightarrow 2^n \geq 5 \Rightarrow n^* = 2.3 \Rightarrow n = 3$

* cuanto puedo representar con $n=3$ bits $\Rightarrow 2^n = 8 \Rightarrow$ me sobra !!

Largo de Código

$$L(C) = \sum_{i=1}^{n} p_i \cdot \ell_i$$ ← largo promedio de un caracter en bits

* ej anterior fue asumir que los símbolos eran equiprobables y que el largo por símbolo era fijo.

* $p_i = \frac{1}{5}$, $L = \sum_{i=1}^{5} \frac{1}{5} \cdot 3 = 3 \frac{1}{5} \cdot 5 \neq 3$

Ejemplo:

	a	b	c	d	e
p	0.1	0.15	0.3	0.16	0.29 ← suma 1
c	000	001	010	011	100	101	110	111
ℓ	3	3	3	3	3

$L = 0.1 \cdot 3 + 0.15 \cdot 3 + 0.3 \cdot 3 + 0.16 \cdot 3 + 0.29 \cdot 3 = (\overbrace{0.1 + \ldots + 0.29}^{\text{suma 1}}) \cdot 3 = 3$ bits/símbolo

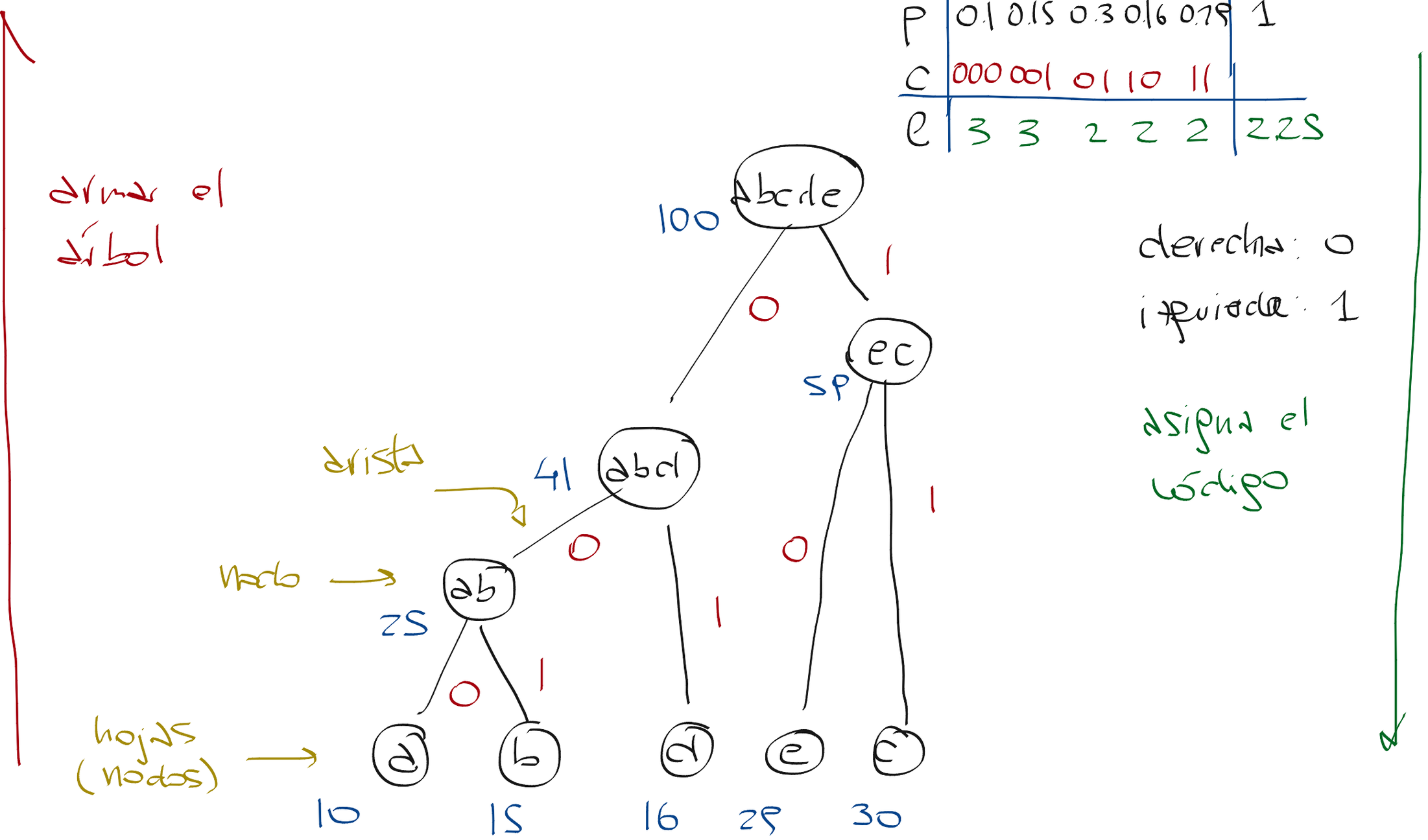
$H = -0.1 \cdot \log(0.1) - \ldots - 0.29 \cdot \log(0.29) \neq 2.205$ bits/símbolo

diferencia

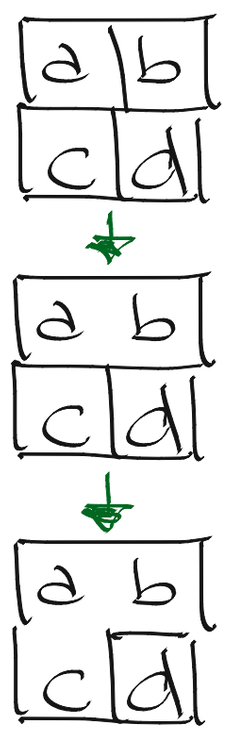
Códigos de Huffman

* árbol (grafo)

	a	b	c	d	e	T
f	10	15	30	16	29	100
p	0.1	0.15	0.3	0.16	0.29	1
c	000	001	01	10	11	
l	3	3	2	2	2	2.29



$$L = (0.1 + 0.15)\cdot 3 + (0.3 + 0.16 + 0.29)\cdot 2 = 2.29$$



obs:

* Propiedad prefijo: ninguna palabra es prefijo de otra.

0 1 0 0 0 1 1

* 1952: D. Huffman, trabajo de fin de curso

"A method for the construction of minimum-redundancy codes"

CLASE 22

Algebra de Boole

ALGEBRA DE BOOLE

Introducción:

* base del diseño de sistemas digitales binarios
* definir la estructura formal de los circuitos lógicos
* no veremos detalles formales

Álgebra:
* Estudio de los símbolos matemáticos
  → y las reglas para manipularlos
* Grupos, anillos y cuerpos

Grupo:
* Par $(G, \bullet)$: G conjunto y $\bullet$ es una operación
* $\bullet : G \times G \rightarrow G$
* Propiedades:
  → clausura / cerrada
  → asociatividad
  → ∃ neutro
  → ∃ inverso

Algebra de Boole

* Define el comportamiento de los números binarios
* Parte de un conjunto arbitrario de axiomas (hay varios)
* Axiomas diferentes: dan lugar a propiedades diferentes (geometría proyectiva)

Axiomas (Huntington, 1904)

* Consistentes e independientes

1) Equivalencia: → Existe una relación de equivalencia $=$
   → Sustitución de expresiones formadas con elementos de G.
   → $a+0 \neq a$

2) Reglas de combinación:
   a) "+": si $a,b \in G \Rightarrow a+b \in G$ → cerrada en G
   b) "·": si $a,b \in G \Rightarrow a \cdot b \in G$

3) Neutros:
   a) $\exists\ 0 \in G\ /\ \forall a \in G,\ a+0=a$
   b) $\exists\ 1 \in G\ /\ \forall a \in G,\ a \cdot 1 = a$

4) Conmutatividad: $\forall a,b \in G$ se cumple
   a) $a+b=b+a$
   b) $a \cdot b = b \cdot a$

5) Distributiva: $\forall a,b \in G$ se cumple,
   a) $a+(b \cdot c) = (a+b)\cdot(a+c)$ ← esto no para en la suma común
   b) $a \cdot (b+c) = (a \cdot b)+(a \cdot c)$

6) Complemento: $\forall a \in G,\ \exists\ \bar{a} \in G$ tal que,
   a) $a \cdot \bar{a} = 0$
   b) $a+\bar{a} = 1$ ← no es el opuesto !!

7) $\exists$ al menos dos elementos $x, y \in G$ / $x <> y$

obs: * Si bien si se parece al álgebra "común", hay diferencias.
* la cantidad de elementos no se define, sino fue demostra

Modelo Aritmética

* Instancia o ejemplo del Álgebra de Boole: $\mathbb{G} = \{0,1\}$ (Ax. 7)
* Deben coincidir con los neutros: $0 \rightarrow +$ (Ax. 3)
  $1 \rightarrow \cdot$
* La reglas y funcionamiento de las operaciones se deducen de los axiomas

Ax. 3

	a)	b)
$(a=0)$	$0+0=0$	$0 \cdot 1 = 0$
$(a=1)$	$1+0=1$	$1 \cdot 1 = 1$

Ax. 4

$0+1=1$ $\quad$ $1 \cdot 0 = 0$

+	·
$0+0=0$	$0 \cdot 0 = 0$
$0+1=1$	$0 \cdot 1 = 0$
$1+0=1$	$1 \cdot 0 = 0$
$1+1=1$	$1 \cdot 1 = 1$

*2

Ax. 5

a) $1 + (1 \cdot 0) = (1+1) \cdot (1+0)$

↓ Ax.1 $\qquad$ ↓ Ax.1

$1 + 0 = (1+1) \cdot 1$

↓ Ax. 3a $\qquad$ ↓ Ax. 3b

$1 = 1+1$ *1

b) $0 \cdot (0+1) = 0 \cdot 0 + 0 \cdot 1$

↓ $\qquad$ ↓

$0 \cdot 1 = 0 \cdot 0 + 0$

↓ $\qquad$ ↓

$0 = 0 \cdot 0$ *2

obs: * las reglas de combinación (operaciones) se presentan en tablas de verdad
* son funciones lógicas de dos variables

Propiedades

Dualidad: → postulados vienen de a pares — Nueva
→ Cambiar 0 por 1 y + por ·

Asociativa: a) $a+(b+c)=(a+b)+c$
b) $a\cdot(b\cdot c)=(a\cdot b)\cdot c$

Idempotencia: a) $a+a=a$ — Nueva
b) $a\cdot a=a$

Dem
$0+0=0$ ✓
$1+1=1$ ✓

Neutros creados: a) $a+1=1$ ←
b) $a\cdot 0=0$

obs: * usaremos esta construcción binaria (generalmente)

Complemento del complemento

$\overline{\overline{a}}=a$ * Dem:

a	$\overline{a}$	$\overline{\overline{a}}$
0	1	0
1	0	1

✓

Leyes de De Morgan

a) $\overline{a+b}=\overline{a}\cdot\overline{b}$ — Nueva
b) $\overline{a\cdot b}=\overline{a}+\overline{b}$

Modelo Lógico

Definiciones

* Lógica formal: Juicio → afirmación
* Valores lógicos: → verdadero (V) $\quad \mathbb{B} = \{V, F\}$
  → falso (F)
* Operadores lógicos: O e Y → ($\vee$ e $\wedge$)

Propiedades

* $V \vee V = V$
* $V \vee F = V$ } ⇒ OR
* $V \vee V = V$
* $F \vee F = F$

Correspondencia

* O y 1 ⇒ F y V
* + y · ⇒ $\vee$ e $\wedge$

obs:
* modelo lógico y el aritmético son isomorfos
* por eso llamaremos +, · → OR, AND

Ejemplo: "silogismos"

$$\frac{A \Rightarrow B \quad B \Rightarrow C}{A \Rightarrow C}$$ ✓

$$\frac{A \Rightarrow B \quad C \Rightarrow B}{A \Rightarrow C}$$ ✗

A	B	A ⇒ B	A ⇔ B
V	F	F	F
V	V	V	V
F	F	X	V
F	V	X	F

← tabla de verdad

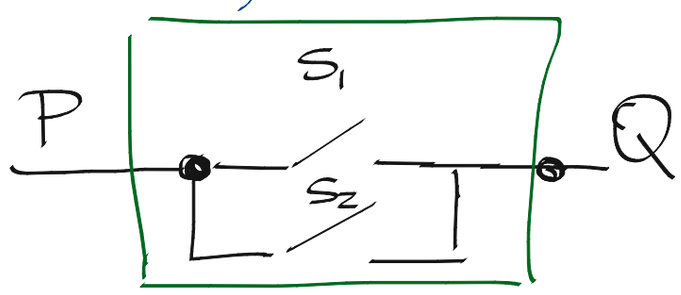
X no sé

Modelo circuital

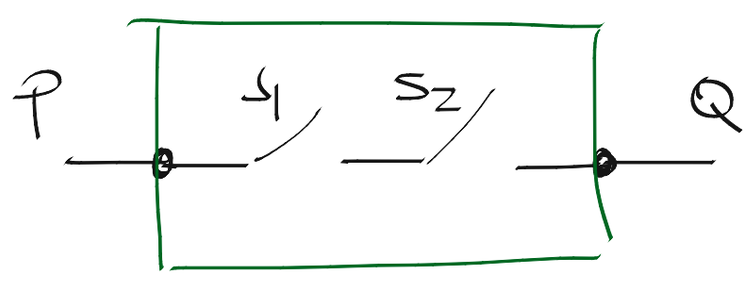
Definición:
* llaves eléctricas
* A: abierto, no pasa corriente
* C: cerrado, pasa corriente

Reglas de combinación

a) Paralelo



b) Serie



S_1	S_2	Paralelo
A	A	A
A	C	C
C	A	C
C	C	C

OR

S_1	S_2	Serie
A	A	A
A	C	A
C	A	A
C	C	C

AND

Correspondencia:
* A, C ⇒ 0, 1
* paralelo, serie ⇒ +, ·

Expresiones booleanas

Def: * Constante: elemento del conjunto $\mathbb{G}$
* Variable: puede tomar cualquier valor en $\mathbb{G}$

Expresión: (def. recursiva)

1) Constantes y/o variables
2) Complemento de una expresión
3) el AND y el OR de dos expresiones

obs: la expresiones siempre se pueden evaluar

Funciones booleanas

Def: $F: \mathbb{G}^n \longrightarrow \mathbb{G}$

$\{x_1, \dots, x_n\} \rightarrow \{0,1\}$

Representación

1) tabla de verdad

a	b	c	F	
0	0	0	0	0
0	0	1	1	1
0	1	0	0	2
0	1	1	0	3
1	0	0	1	4
1	0	1	1	5
1	1	0	0	6
1	1	1	1	7

2) notación Σ o Π

a) $f(a,b,c) = \sum(1,4,5,7)$

b) $f(a,b,c) = \Pi(0,2,3,6)$

3) $f(a,b,c) = a.c + a.\bar{b} + \bar{a}.b.c$

OR Exclusivo (XOR)

* XOR es importante: suma aritmética módulo 2

* Aritmética módulo N, $a \in [0, N-1]$, $a+1 \to 0$ si $a = N-1$

* Propiedades:
  - asociativa
  - conmutativo
  - distributiva: $a \cdot (b \oplus c) = ab \oplus ac$
  - $a \oplus 0 = a$
  - $a \oplus 1 = \bar{a}$
  - $a \oplus a = 0$
  - $a \oplus \bar{a} = 1$

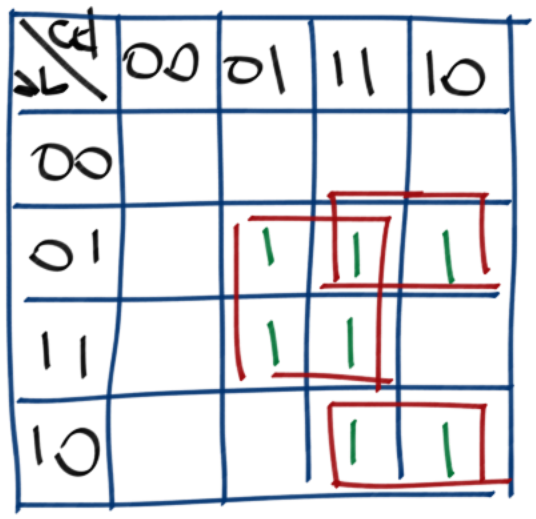
a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

Práctico 13

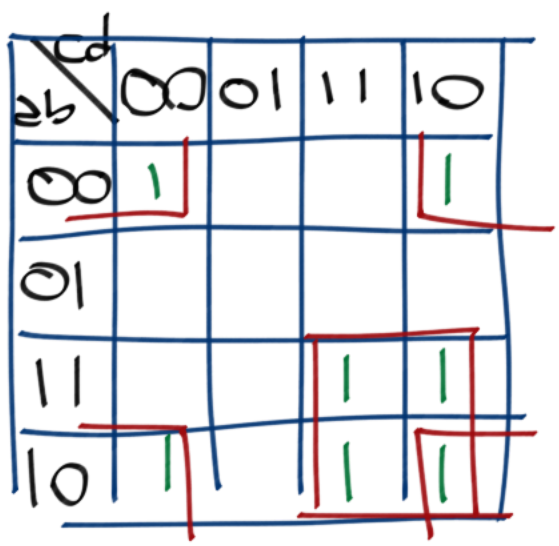
Introducción a PLD

TUTORIAL

$$dl_1 = a\bar{b}c + \bar{a}bc + bd$$



$$dl_2 = ac + \bar{b}.\bar{d}$$



$N: \{n_i \in [0,12]\}$ μ nota media modelo ← datos

① $e_i = (n_i - \mu)^2$ $\quad E = \frac{\sum_i (n_i - \mu)^2}{\#N}$ ← métrica $\quad N = \{0,0,2,3,8,8,12\}$

μ? $E(\mu)$ mín $\quad \frac{\partial E(\mu)}{\partial \mu} = \frac{2 \sum (n_i - \mu) \cdot (-1)}{\#N} = 0$ ← Algoritmo

$\sum n_i - \mu = 0$

$\sum n_i = \sum \mu \Rightarrow \sum n_i = \#N \cdot \mu$

$$\mu = \frac{\sum_i n_i}{\#N}$$ ← parámetro óptimo

$\mu^* = 4.85$

② $\mu_k = [\overbrace{0, 1, \ldots, 12}^{\Delta\mu}]$ ← parámetros de algo

$E_k = \sum_{i=0}^{\#N} \frac{(n_i - \mu_k)^2}{\#N} \quad \forall k$ ← fuerza bruta

$D = \{\mu \in [0,12]\}_{\mu \in \mathbb{R}}$

$k^* = \arg\min_k \{E_k\}$ óptimo μ_{k^*} $\quad \mu_{k^*} = 5$

$E(5) = 0.14 = 1.4 \times 10^{-1}$

$E(4.85) = 3.6 \times 10^{-4}$

$\Delta\mu = 1$

$$\mu^* = \lim_{\Delta\mu \to 0} \mu_k$$

$\Delta\mu = 0.1 \quad E(4,8) = 0.02$

$E(4.9) = 0.01$

CLASE 23

Algebra de Boole

Suma de productos

* dada la tabla de verdad de f, encontrar su expresión algebraica

$$f(x_1,x_2) = \bar{x}_1\bar{x}_2 \cdot f(0,0) + \bar{x}_1 x_2 \cdot f(0,1) + x_1\bar{x}_2 \cdot f(1,0) + x_1 x_2 \cdot f(1,1)$$

ej: XOR

$$f(x_1,x_2) = \bar{x}_1\bar{x}_2 \cdot 0 + \bar{x}_1 x_2 \cdot 1 + x_1\bar{x}_2 \cdot 1 + x_1 x_2 \cdot 0$$

$$\boxed{f(x_1,x_2) = \bar{x}_1 x_2 + x_1\bar{x}_2}$$

x_1	x_2	XOR	
0	0	0	← $f(0,0)$
0	1	1	
1	0	1	
1	1	0	

Obs: * sólo importan las combinaciones donde $f(\,,\,)=1$

Ejemplo:

$$\boxed{f(a,b,c) = \bar{a}\bar{b}c + \bar{a}b\bar{c} + a\bar{b}\bar{c}}$$

a	b	c	f	
0	0	0	0	
0	0	1	1	← $\bar{a}\bar{b}c$
0	1	0	0	
0	1	1	1	← $\bar{a}bc$
1	0	0	1	← $a\bar{b}\bar{c}$
1	0	1	0	
1	1	0	0	
1	1	1	0	

Producto de sumas

* Análogo pero con ceros.

0 1 0 → 0

$(a+\bar{b}+c)$

* $f(x) = x * x$
$= x^2$

Operadores lógicamente completos

* Conjunto de operadores capaz de representar cualquier función lógica
* El teorema de los productos canónicos prueba que el conjunto {AND, OR, NOT} es lógicamente completo
* Para probar que el conjunto {F, G, H} es lógicamente completo basta con implementar el AND, OR, NOT con esas funciones
* Puede no ser mínimo: en {AND, OR, NOT} el AND es redundante
  → De Morgan: $\overline{ab} = \overline{a} + \overline{b}$ → $\boxed{ab = \overline{\overline{a} + \overline{b}}}$

* {NAND} es lógicamente completo → $a \# b = \overline{ab}$

  → $a \# a = \overline{aa} = \overline{a}$ NOT

  → $(a \# b) \# (a \# b) = \overline{(a \# b) \cdot (a \# b)} = \overline{a \# b} = \overline{\overline{ab}} = ab$ AND

  → $(a \# a) \# (b \# b) = \overline{a} \# \overline{b} = \overline{\overline{a} \cdot \overline{b}} = \overline{\overline{a + b}} = a + b$ OR

a	a . a
0	0
1	1

Simplificación

* Tenemos un método sistemático, pero no óptimo
* Mínimo número de compuertas lógicas

Método Algebraico

1) $f \cdot \bar{f} = 0$
2) $f + \bar{f} = 1$
3) $gf + \bar{g}f = f$
4) $gf + f = f$
5) $f + \bar{f}g = f + g$

Ejemplo $f_1 = \bar{a}bc + \bar{a}b\bar{c} + ab\bar{c}$ ← 8 op (12 p) con NEG.

a) $f_1 = \bar{a}b + ab\bar{c}$ ← 4 op

b) $f_1 = \bar{a}bc + b\bar{c}$

c) $f_1 = \bar{a}bc + \bar{a}b\bar{c} + \bar{a}b\bar{c} + ab\bar{c}$

$f_1 = \bar{a}b + b\bar{c}$

Ejemplo $f_2 = \bar{a}\bar{b}c + a\bar{b}c + a\bar{b}\bar{c} + \bar{a}bc$ ← 11 op

a) $f_2 = \bar{b}\cdot c + a\bar{b}\bar{c} + \bar{a}bc$ (dist)

$f_2 = \bar{b}(c + a\bar{c}) + \bar{a}bc$ (5)

$f_2 = \bar{b}(c + a) + \bar{a}bc$ (dist)

$f_2 = bc + \bar{b}a + \bar{a}bc$ (dist)

$f_2 = c(\bar{b} + \bar{a}b) + \bar{b}a$ (3)

$f_2 = c(b + a) + \bar{b}a$ (dist)

$\boxed{f_2 = ac + bc + \bar{b}a}$ ← 5 op

b) $f_2 = \bar{a}\bar{b}c + a\bar{b}c + a\bar{b}\bar{c} + \bar{a}bc$

$\boxed{f_2 = \bar{a}c + a\bar{b}}$ ← 3 op

Mapas de Karnaugh

$$f_2 = \bar{a}\bar{b}c + a\bar{b}c + a\bar{b}\bar{c} + \bar{a}bc$$

c \ ab	00	01	11	10
0	0	0	0	1
1	1	1	0	1

$$f_2 = \bar{a}c + a\bar{b}$$

a	b	c	f_2
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

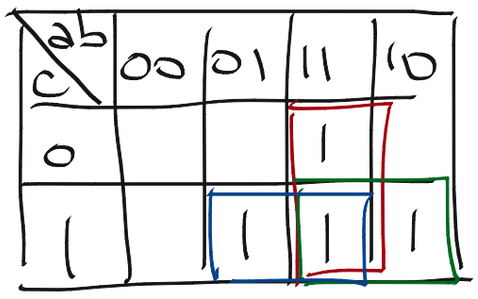
CIRCUITOS COMBINATORIOS

[ver slides 1-14]

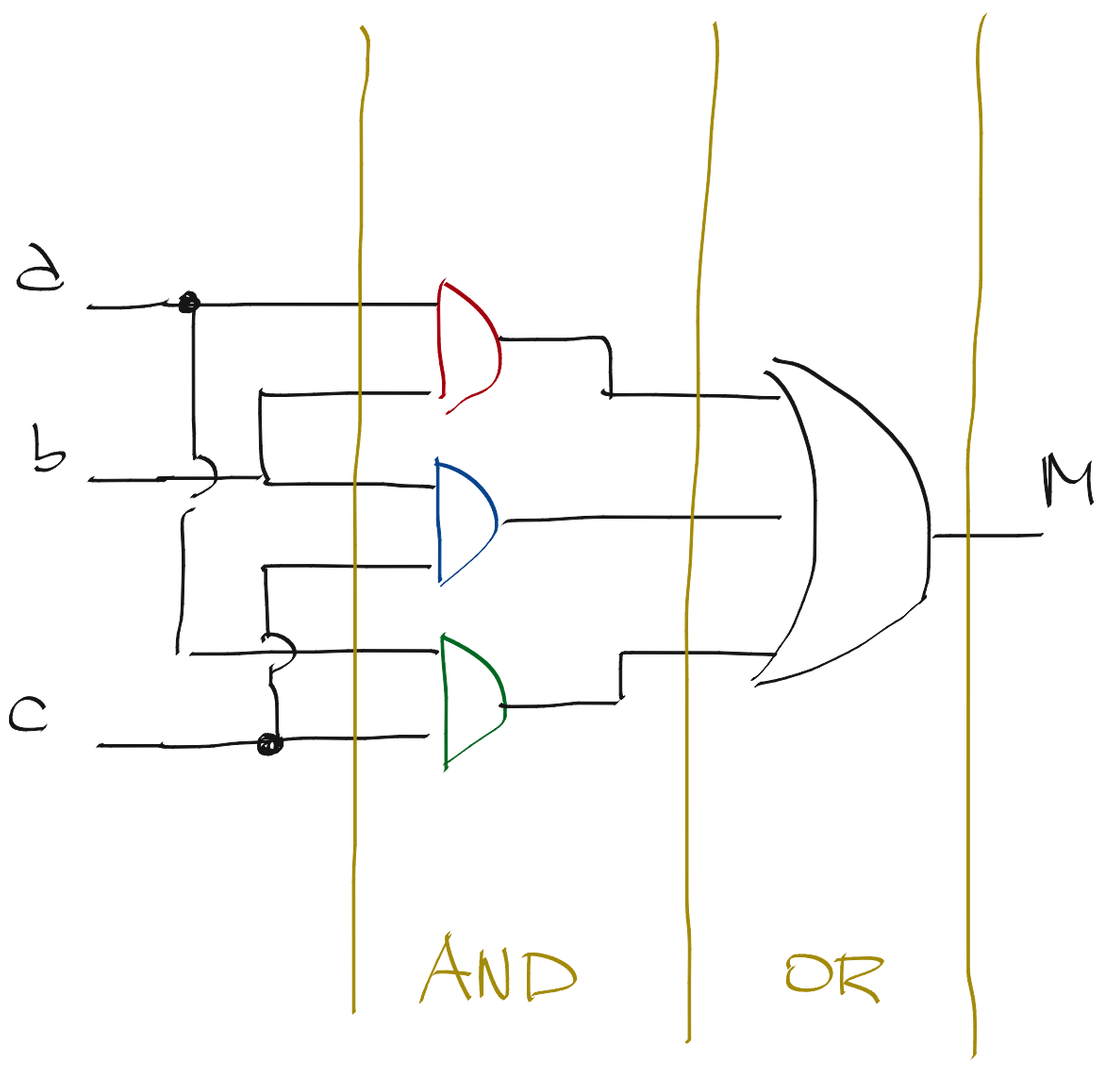
Circuito mayoría

* da 1 si hay más unos que ceros

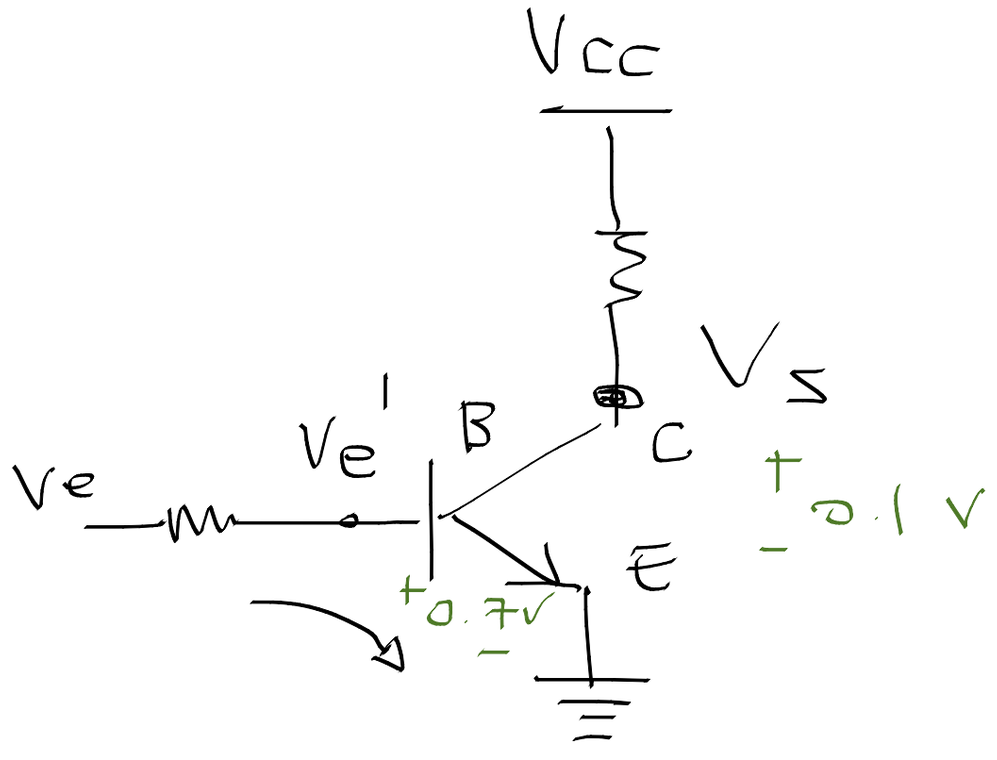
a	b	c	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



M = ab + ac + bc



QLS:
* Siempre da un circuito de dos capas (AND, OR, NOT)
* mínimo para este modelo
* no es mínimo global.



$$i = \frac{Ve - Ve'}{Rs}$$

CLASE 24

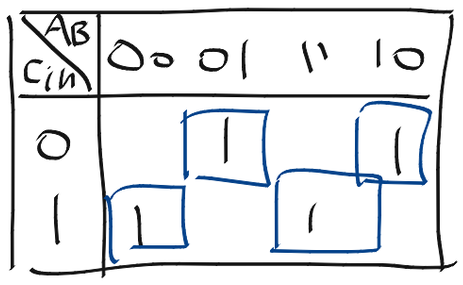
Circuitos combinatorios

Síntesis

Ejemplo: "sumador completo de 1-bit"

C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

S



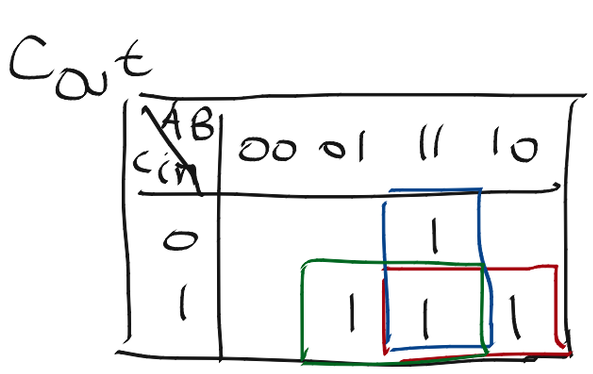
← no se simplifica!!

$$S = \bar{a}\,b\,\bar{C}_{in} + a\,\bar{b}\,\bar{C}_{in} + \bar{a}\,\bar{b}\,C_{in} + a\,b\,C_{in}$$

11 comp.

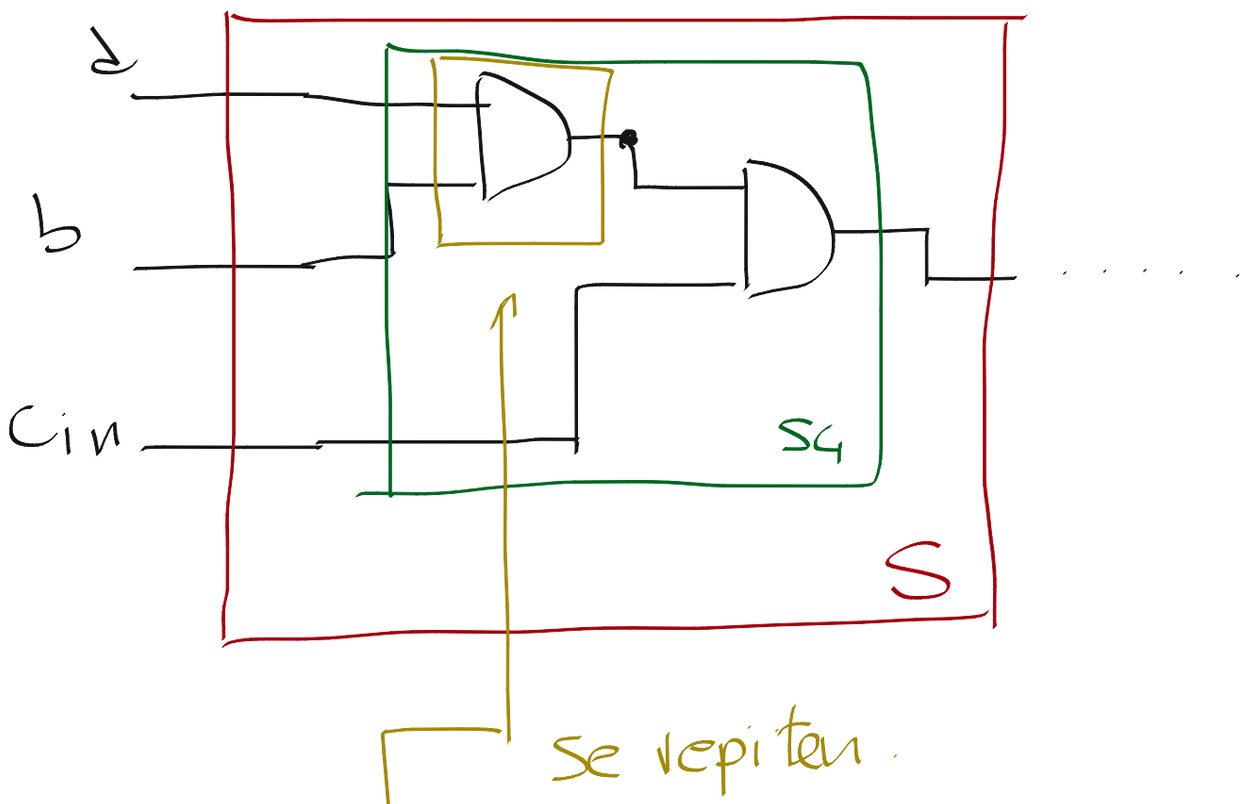
S_4

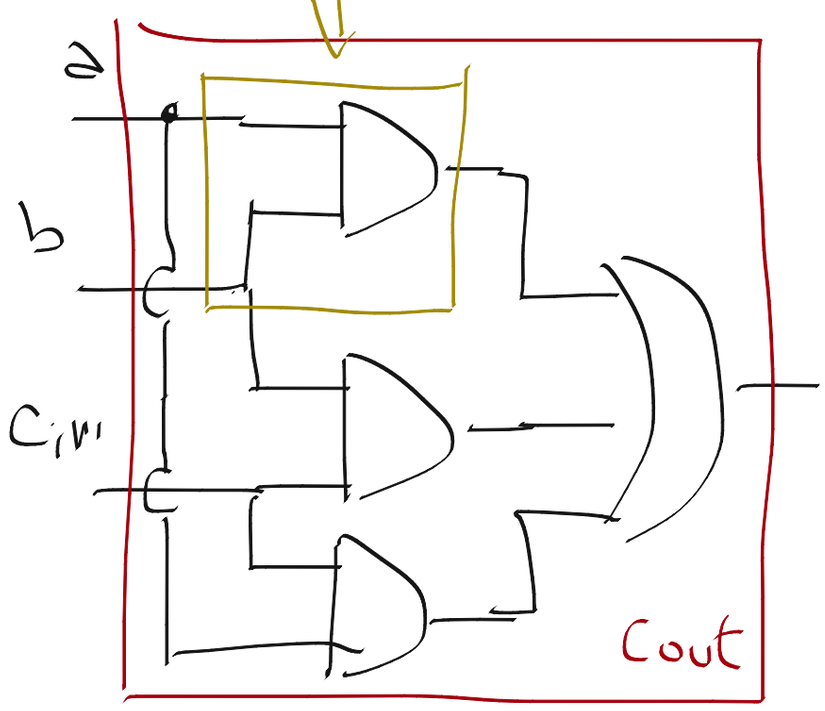
C_{out}



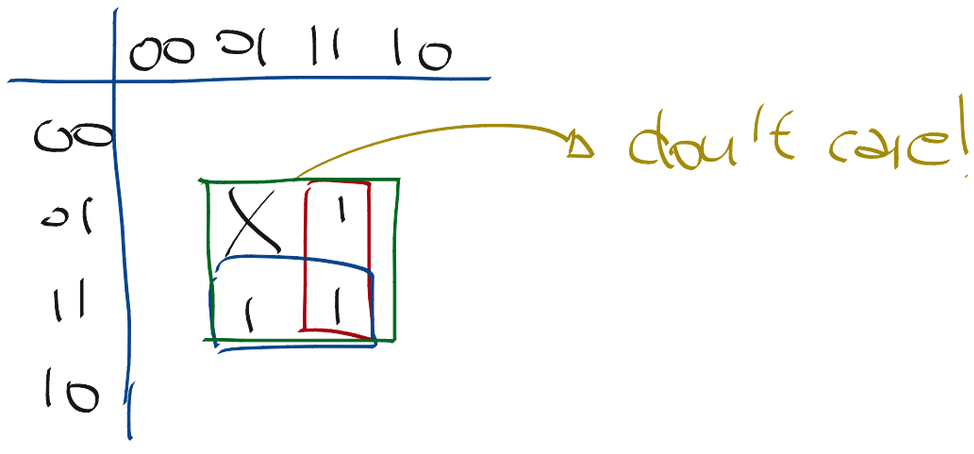
$$C_{out} = ab + bc + ac$$

5 c





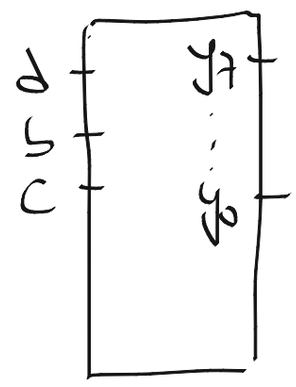
Comodines



* Le puedo dar el valor que quiera.

BLOQUES CONSTRUCTIVOS

Ejemplo: "Decodificador" $(N=3)$



a	b	c	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0						1		
0	1	1					1			
1	0	0				1				
1	0	1			1					
1	1	0		1						
1	1	1	1							

obs: * $D(a,b,c): \mathbb{G}^3 \to \mathbb{G}^8$ (8 funciones lógicas)

* 1 mapa de Karnaugh por función

y_0

ab \ c	0	1
00	1	
01		
11		
10		

$\rightarrow \quad y_0 = \bar{a}\bar{b}\bar{c}$

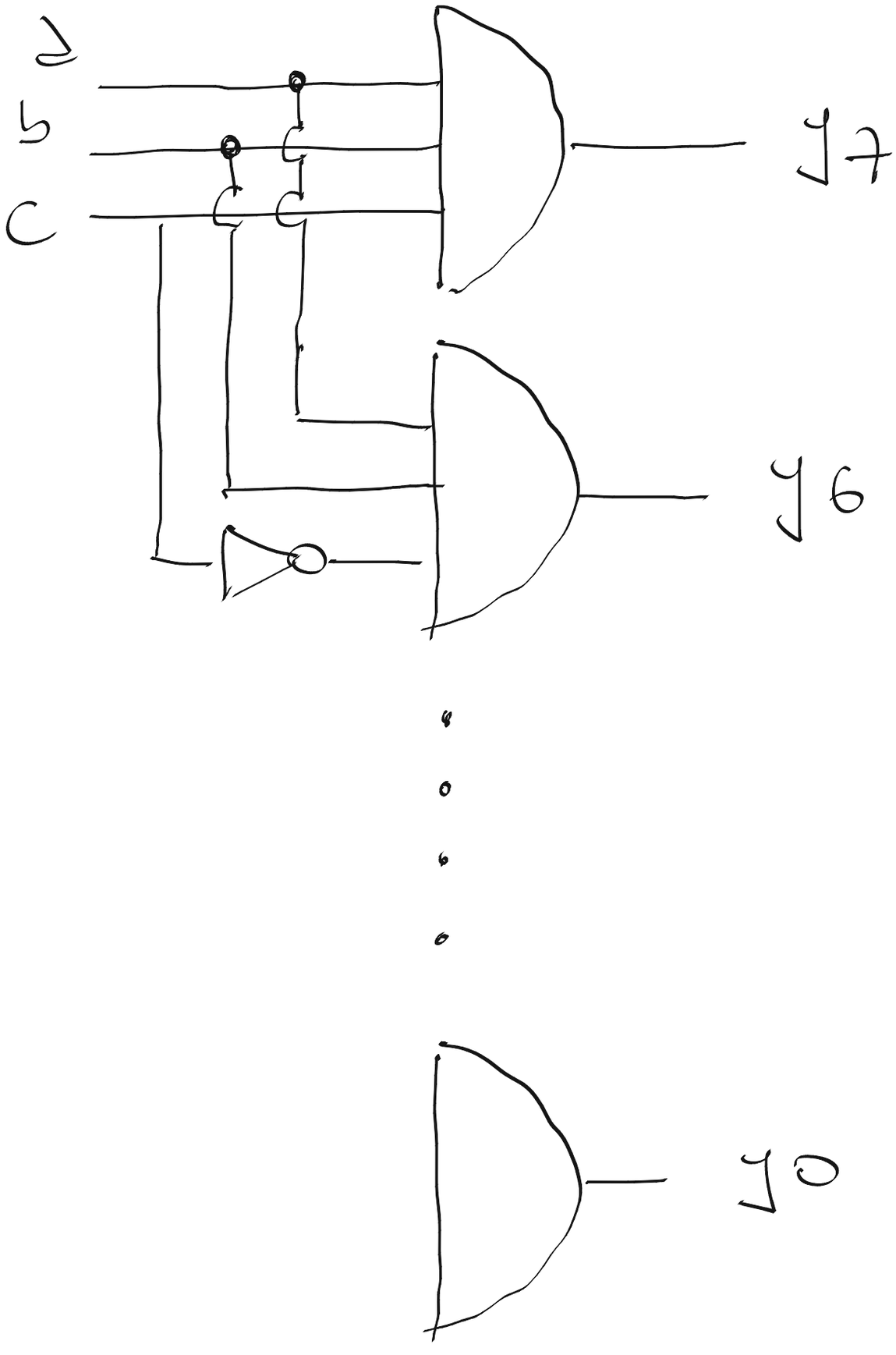
y_1

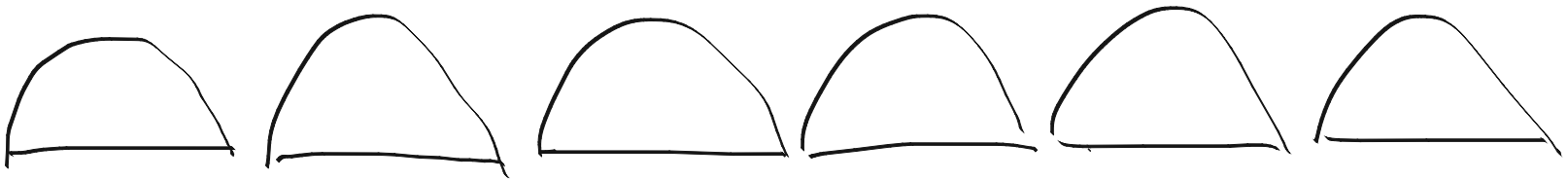
ab \ c	0	1
00		
01		
11		
10		

$\rightarrow \quad y_1 = \bar{a}\bar{b}c$

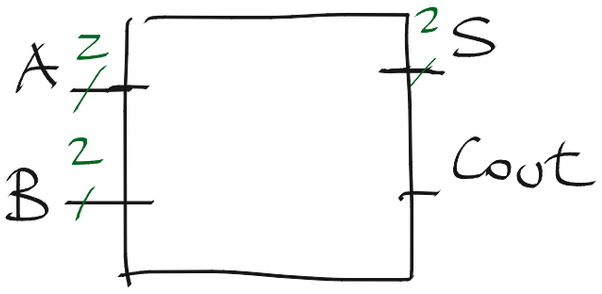
Obs:

* 1 caso





Ejemplo: Semi-Sumador 2-bits



$A = a_1 a_0 \qquad S = S_1 S_0$

$B = b_1 b_0$

a_1	a_0	b_1	b_0	C	S_1	S_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0			
0	0	1	1			
0	1	0	0			
0	1	0	1	0	1	0
0	1	1	0			
0	1	1	1	1	0	0
1	0	0	0			
1	0	0	1			
1	0	1	0			
1	0	1	1			
1	1	0	0			
1	1	0	1			
1	1	1	0			
1	1	1	1	1	1	0

máx: 3+3 = 6 →

S_1 $a_1 a_0$

$b_1 b_0$ \ A	00	01	11	10
00		1	1	
01		1	1	
11	1		1	
10	1	1		

$$S_1 = a_1 \bar{b}_1 \bar{b}_0 + a_1 \bar{a}_0 b_1 + \bar{a}_1 a_0 \bar{b}_1 b_0 + a_1 a_0 b_1 a_0 + \bar{a}_1 \bar{a}_0 b_1 + \bar{a}_1 b_1 \bar{b}_0$$

19 Comp

S_0

B \ A	00	01	11	10
00		1	1	
01	1			1
11	1			1
10		1	1	

$$S_0 = \bar{a}_0 b_0 + a_0 \bar{b}_0$$ 3 Comp

C

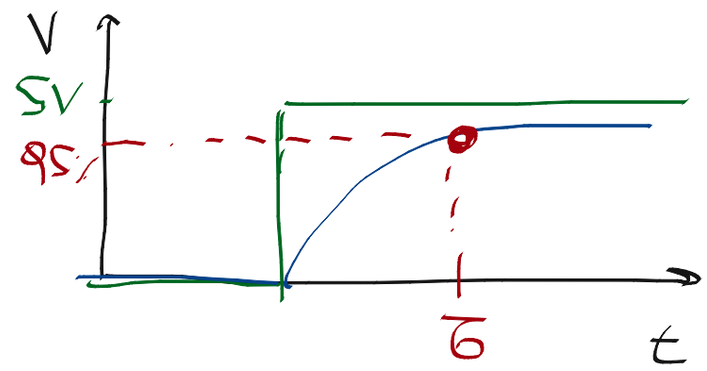
B \ A	00	01	11	10
00				
01		1		
11	1	1	1	
10		1	1	

$$C = a_1 b_1 + a_1 a_0 b_0 + a_0 b_1 b_0$$ 8 comp.

Tiempo de propagación (retardo)

retardo:

* respuesta al escalón (SyS)
* tiempo que demora el circuito en cambiar de estado
* usualmente demora unos 20-80 ns



[ver hoja datos]

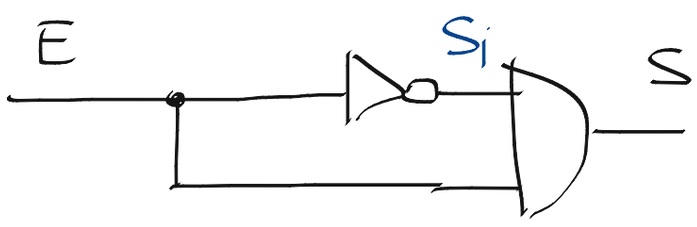
velocidad circuito:

* análogo al costo computacional
* retardo acumulado del camino más largo.

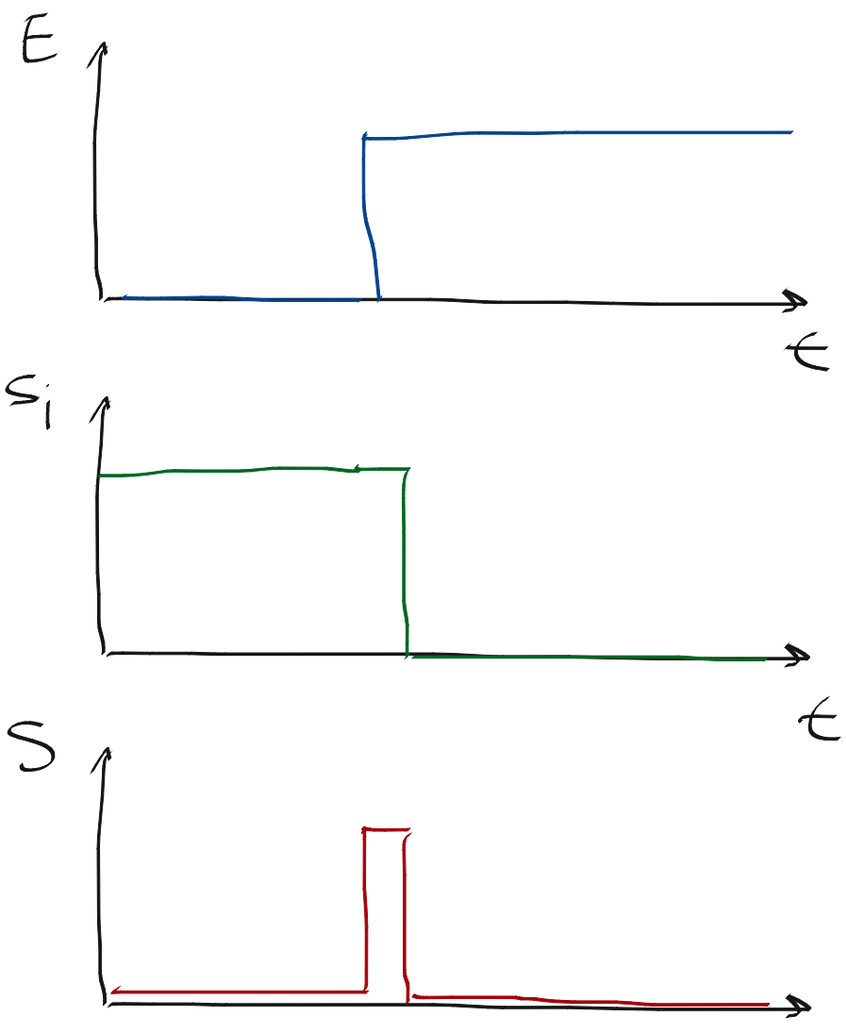
Glitches

* Cambio espúreos de estado

$$S = E \cdot \bar{E} = 0$$



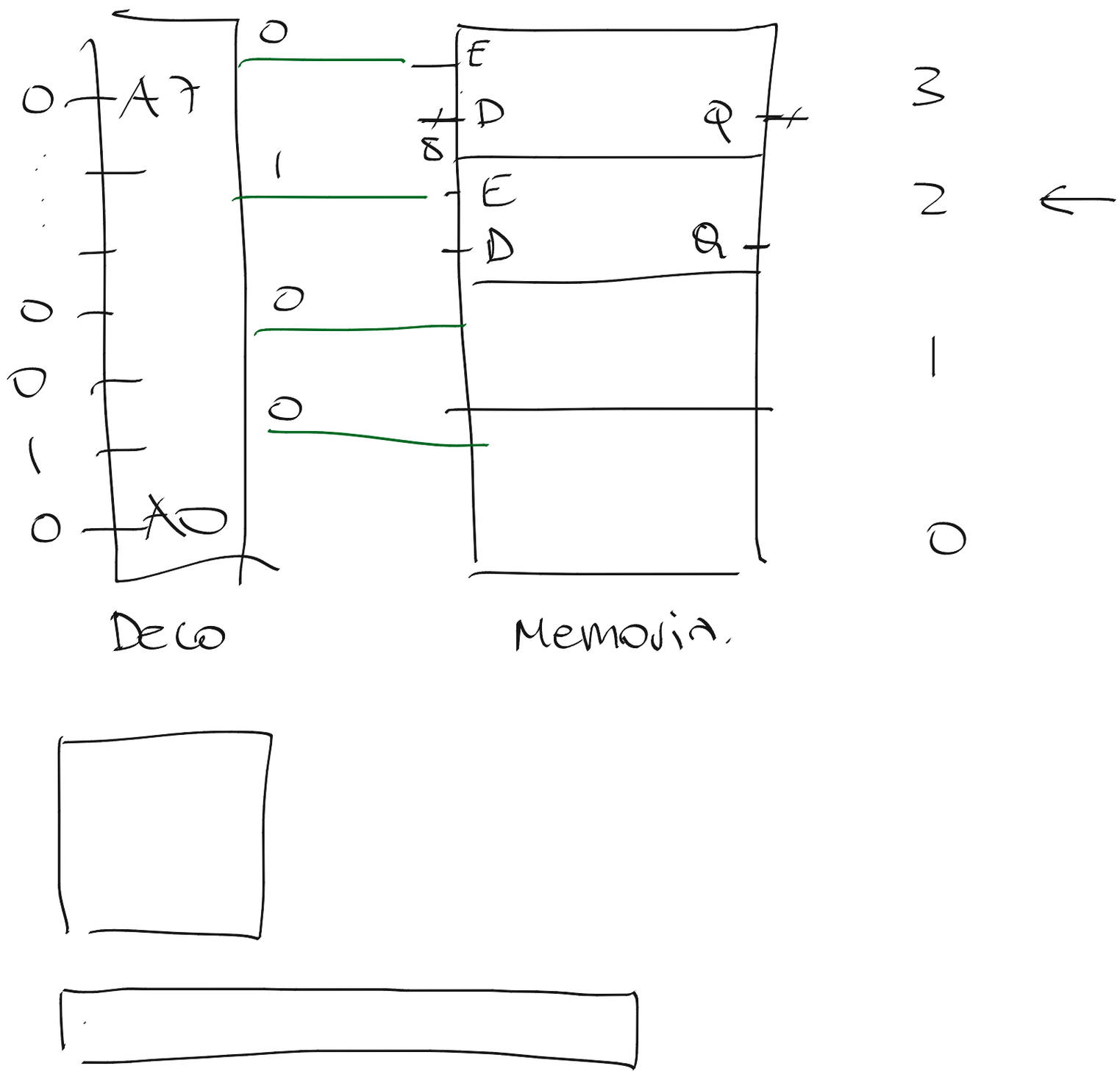
E	Si	S
0	1	0
1	0	0



OLs:

* los entornos de desarrollo permiten eliminar estos glitches.

Ejemplo: "Uso del deco para habilitar memoria.



Deco Memoria.

CLASE 25

Circuitos secuenciales

Flip-flops

Flip-flops

Circuitos secuenciales

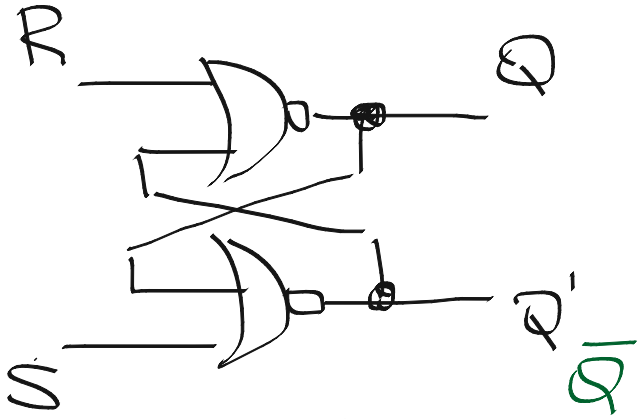
* el valor de la salida depende la entrada actual y de las anteriores
* tienen memoria o estado

Esquema:
* FF como circuito secuencial más sencillo
* Contadores
* Máquinas de estado

Flip-Flop

* elemento más básico de memoria
* recuerda un valor previo de la entrada
* hay varios tipos:
  → veremos qué hacen todos
  → veremos como lo hacen algunos

FF Asíncrono (Latch)



* dos entradas: R, S
* dos salidas: Q, $\bar{Q}$

A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

SET-RESET

1) caso R=1, S=0
   a) R=1 fuerza Q=0
   b) S=0, Q=0 ⇒ Q'=1
2) caso R=0, S=1
   a) S=1 fuerza Q'=0
   b) R=0, Q'=0 ⇒ Q=1
3) R=0, S=0 no fuerzan determinadas, dependen
   a) de Q y Q'
   b) Combinaciones válidas de Q y Q'? $Q \neq Q'$
      * si Q=Q'=1 ⇒ los dos NOR fuerzan 0. Absurdo
      * si Q=Q'=0 ⇒ los NOR dan 1. Absurdo
        como R=S=0
   c) hay que analizar de donde venimos
      * R=1, S=0 → R=S=0 ⇒ cuando R pasa de 1 a 0
        R=1, S=0 → Q=0, Q'=1
        ⇒ Q=0, Q'=1 ← quedan igual
      * si venía de R=0, S=1 ⇒ Q'=0, Q=1

4) a) $R = S = 1 \Rightarrow Q = Q' = 0$ porque ambos NOR fuerzan a 0.

Que pasa si cambiar R o S

b) $R: 1 \to 0$, $S = 1$ $\Rightarrow$ $Q: 0 \to 1$, $Q' = 0$

c) $R = 1$, $S: 1 \to 0$ $\Rightarrow$ $Q = 0$, $Q': 0 \to 1$

} opuestos

d) Que pasa si $R: 1 \to 0$ al mismo tiempo $S: 1 \to 0$

la salida queda indeterminada ($Q \neq Q'$)

no valen otras combinaciones (por cont)

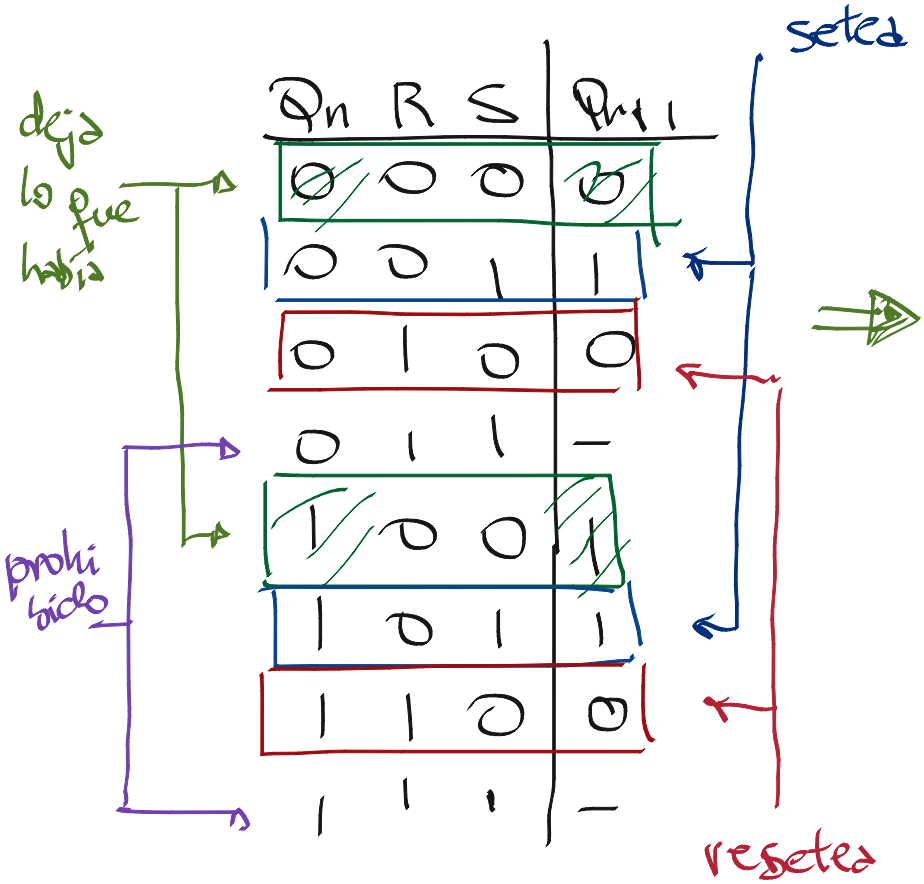
obs: → en la práctica no existe tal cosa como "al mismo tiempo", siempre uno pasa antes que la otra

→ igual queda indeterminado

→ Aparecen glitches

→ La combinación (1,1) se prohibe. (pero se puede) depende del usuario

Tabla de verdad

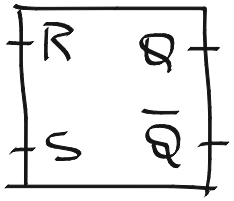


Qn	R	S	Qn+1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	–
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	–

notación

R	S	Qn+1
0	0	Qn
0	1	1
1	0	0
1	1	–

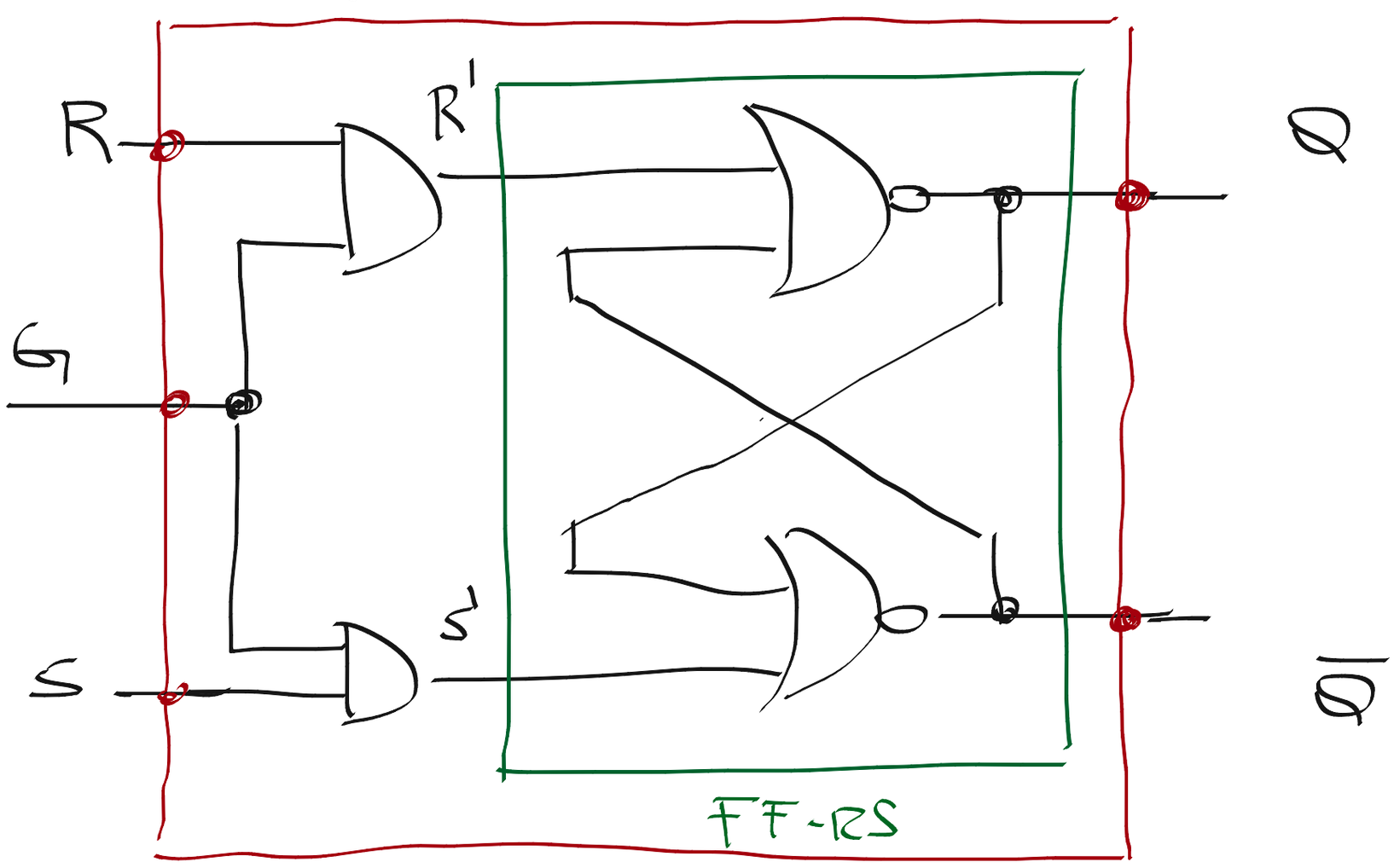
Símbolo:



* No es que necesite $\bar{Q}$, pero me sale gratis.

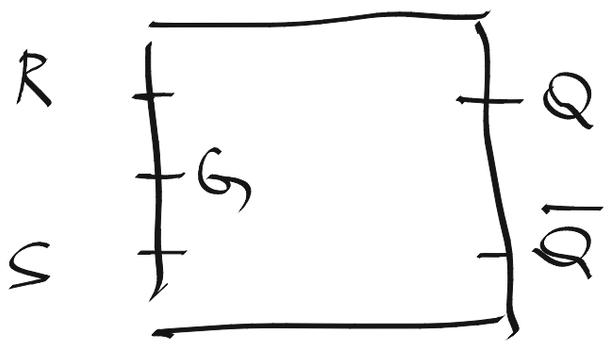
Flip-Flop R-S síncrónico (nivel)

* Los FF asinc pueden cambiar sus entradas en momentos no deseados y su salida cambiar
* Hace difícil construir circuitos complejos
* Para evitar esto se introduce una señal de sincronismo: → habilitación o reloj
  → general del circuito (orquesta)



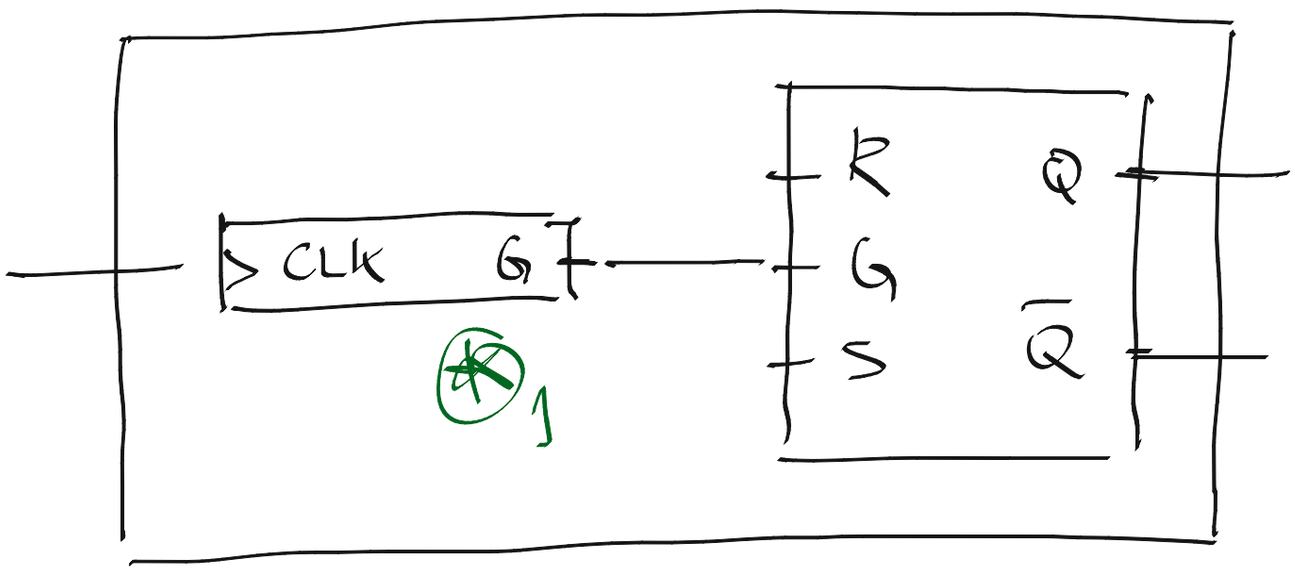
FF-RS sync

* G → entrada de control
  → G=1 las AND no hacen nada (queda el mismo circuito que antes)
  → G=0 ⇒ R'=S'=0 ⇒ el circuito no hace nada.

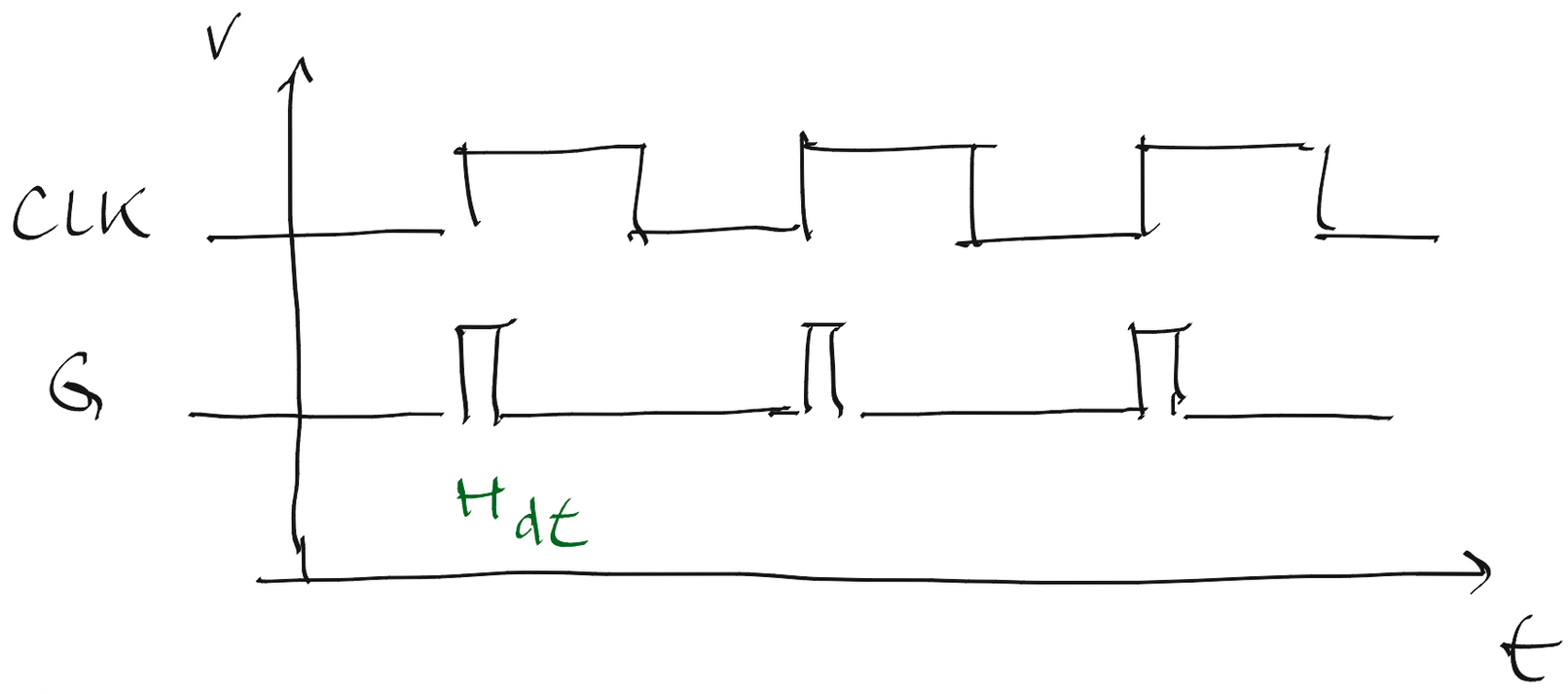


FF-RS síncrono (por flanco)

2021.
hasta 58:59



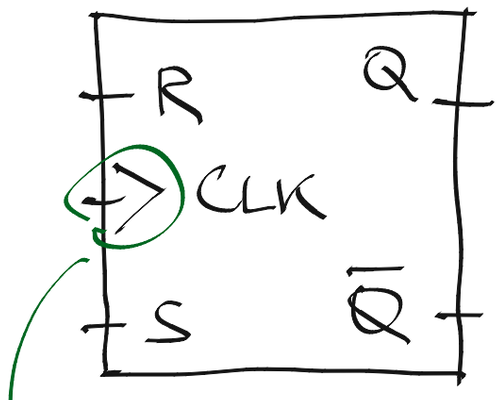
Convierto flanco a nivel (ilusión)



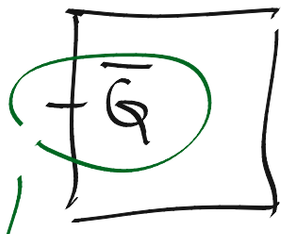
Obs: * $dt << CLK$

* es más fácil generar una onda cuadrada

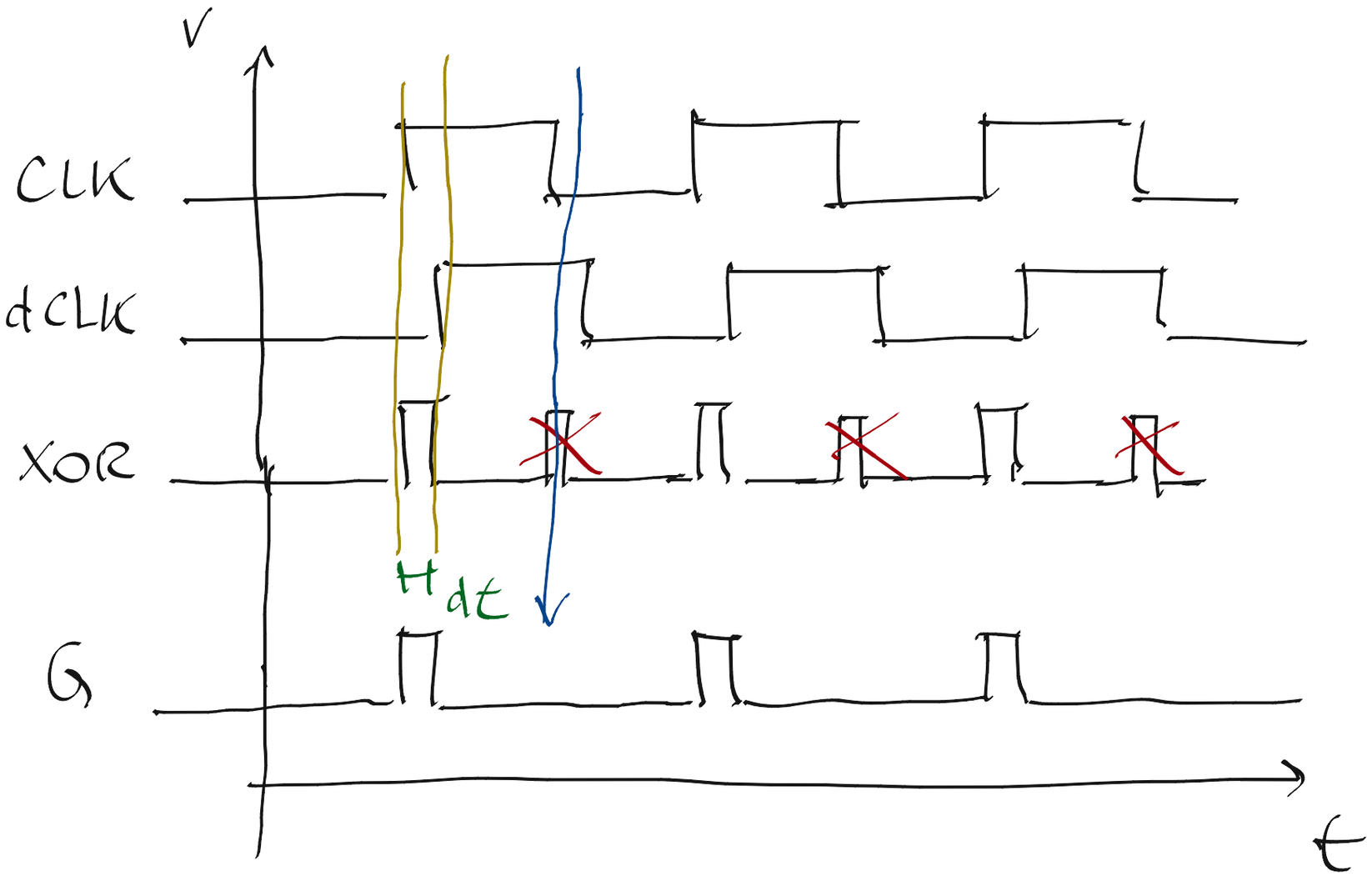
Notación

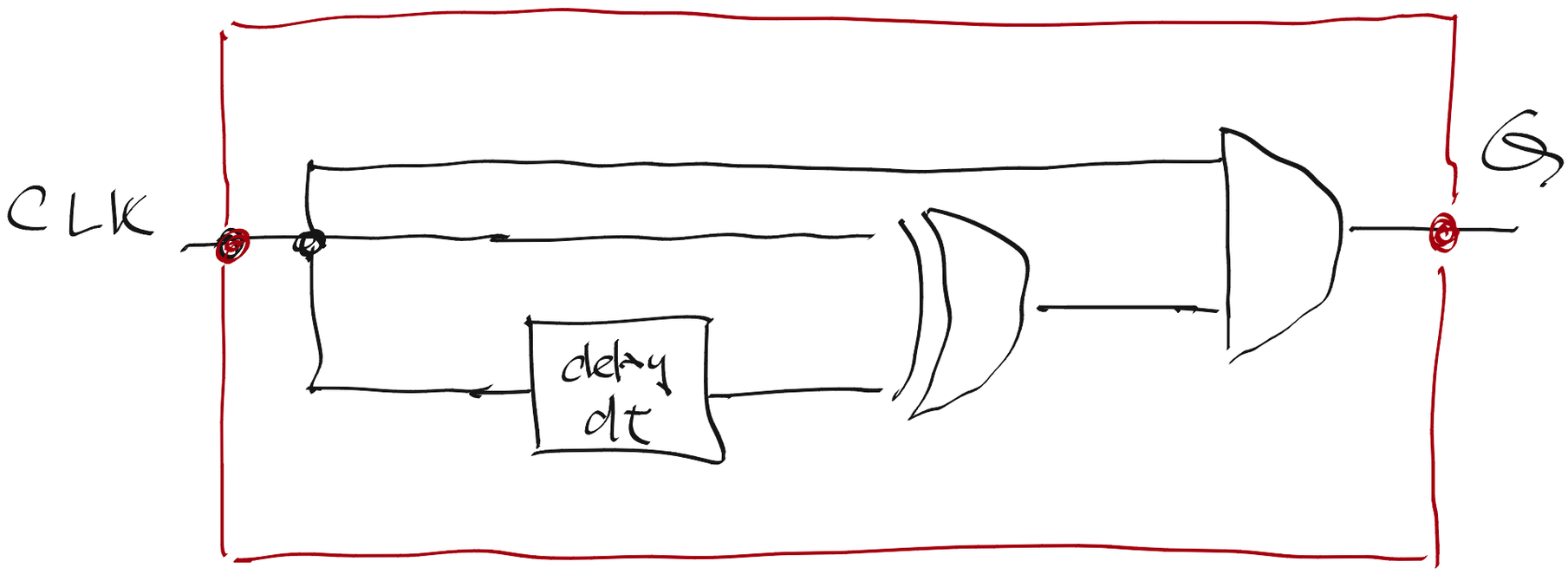


sensible al flanco.



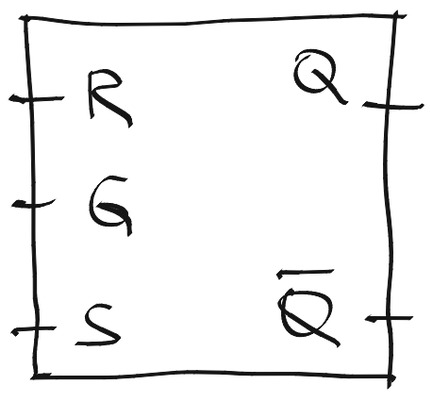
activo por nivel bajo



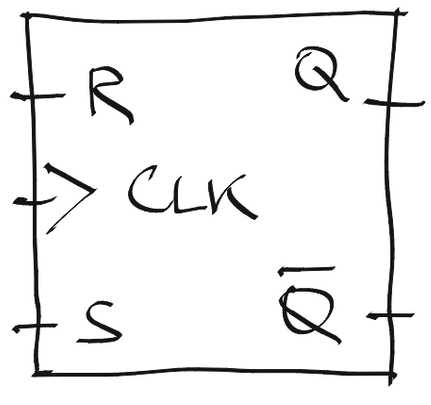


Convierte flanco a nivel
(de subida)

Resumen



FF-RS
síncrono
nivel (alto)



FF-RS
síncrono
flanco (subida)

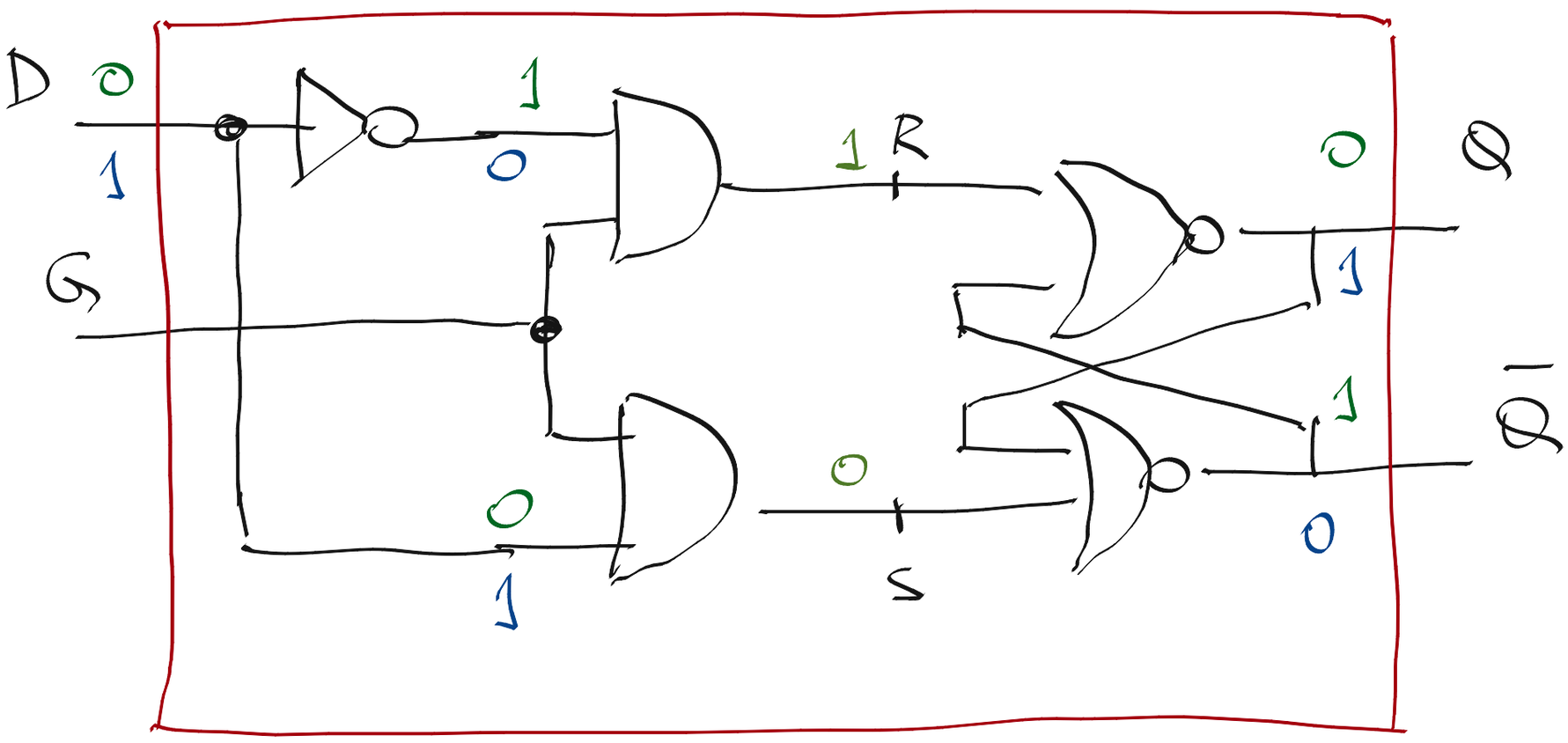
Tipos de FF

FF-D

* simplifica el comportamiento del R-S
* es el tipo más usado

G = 1

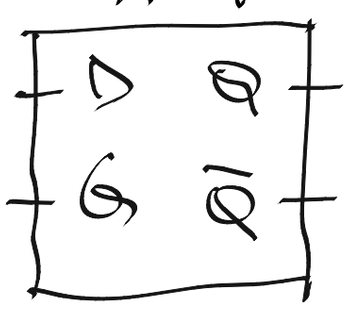
D	Q_{n+1}
0	0
1	1



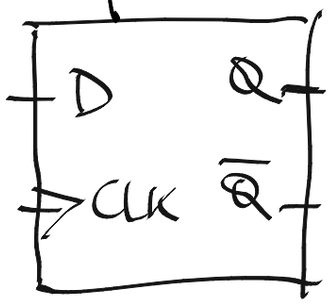
$Q_{n+1} = D_n$

guarda lo que pongo en D

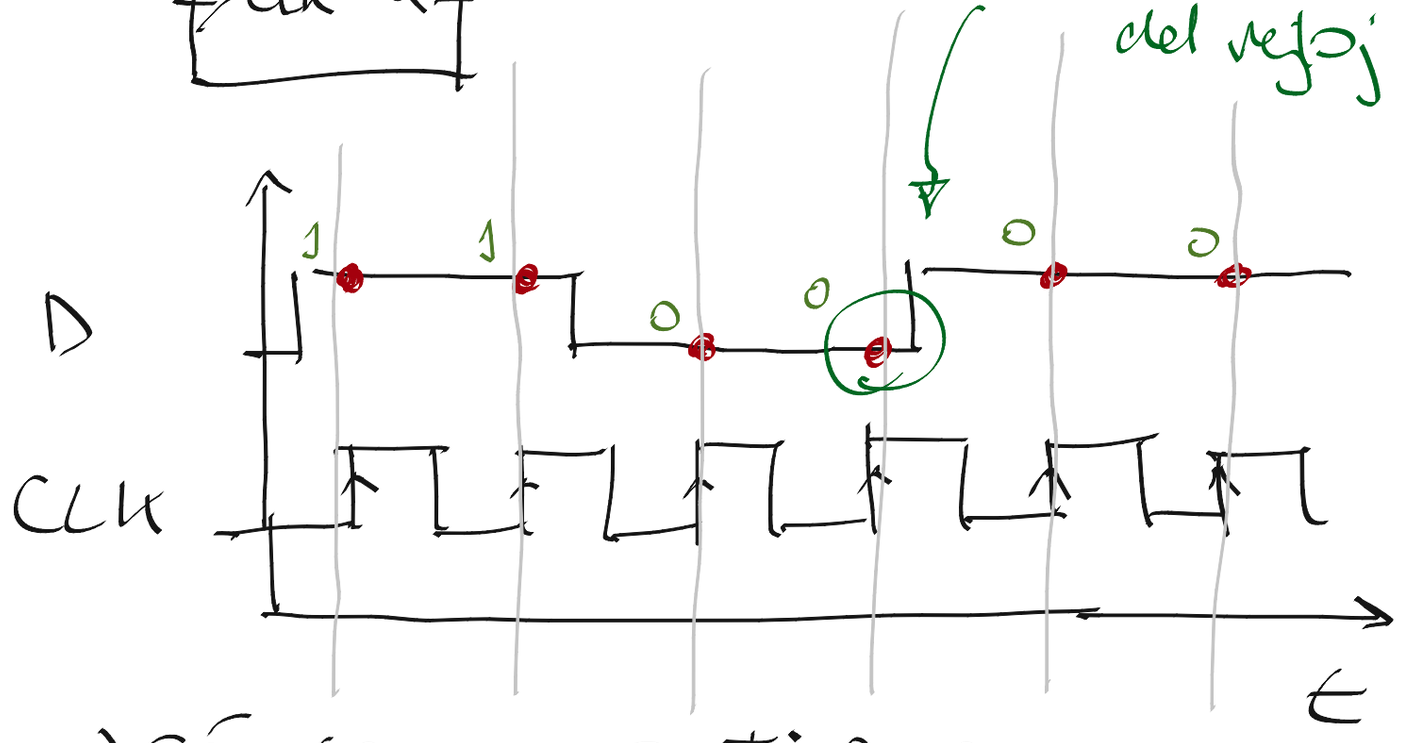
nivel



flanco



D no debe cambiar cerca del reloj



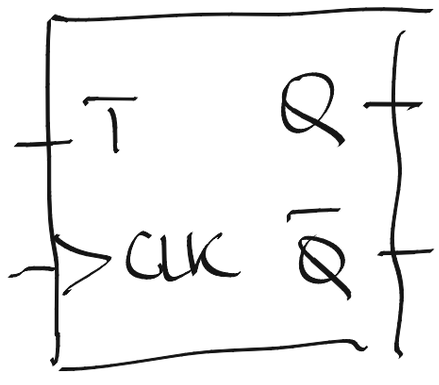
Obs: * el FF-D asíncrono no tiene sentido (sólo copia)

* el síncrono por flanco de subida sirve para implementar el muestreo.

FF-D funcionamiento dinámico [slides 46]

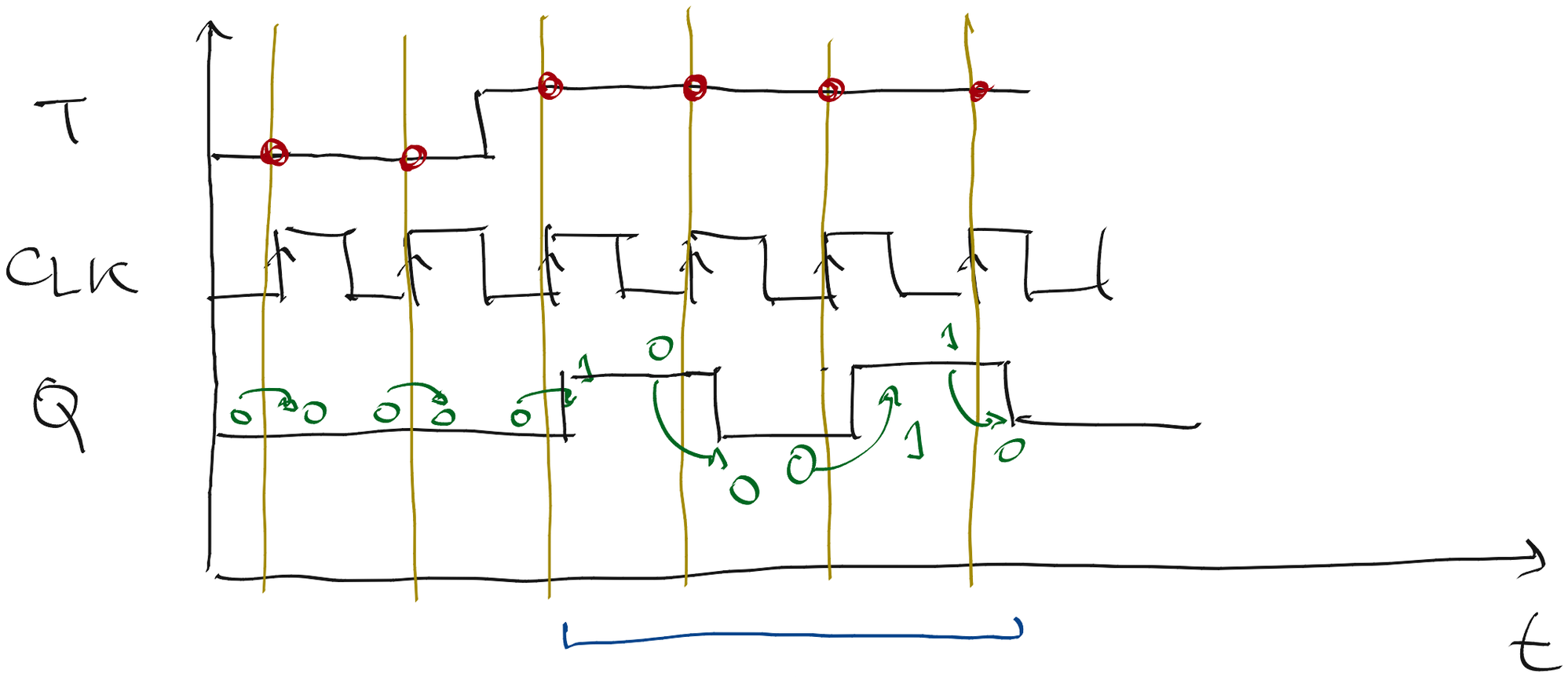
FF tipo T (toggle)

* invierte el valor anterior (T=1)



T	Q_{n+1}
0	Q_n
1	$\overline{Q_n}$

$$Q_{n+1} = \overline{T}.Q_n + T.\overline{Q_n}$$



¿Qué frecuencia tiene Q?

Usos:

* sirve para implementar un divisor de frecuencia
  - alimentar circuitos más lentos
  - bloque constructivo

FF - JK deducción

JK 1:28

*

FF - JK Resumen

1:48

* se puede usar como sinonimo universal.

CLASE 26

Circuitos secuenciales

Flip-flops y aplicaciones

Flip-Flop: aplicaciones

hasta el 5:00! rs vepso

CLASE 27

Circuitos secuenciales

Memorias RAM

CLASE 28

Circuitos secuenciales

Maquinas de estado

MÁQUINAS DE ESTADO

Introducción

* Circuitos secuenciales → tiempo
  → no vimos un método sistemático
* Máquinas de estados → especificar → diseñar ⇒ circuito secuencial
* Circuito secuencial → parte combinatoria
  → memoria: FFs
* propiedades → depende de la entrada
  → toda la secuencia pasada
  → infinitas secuencias de entrada

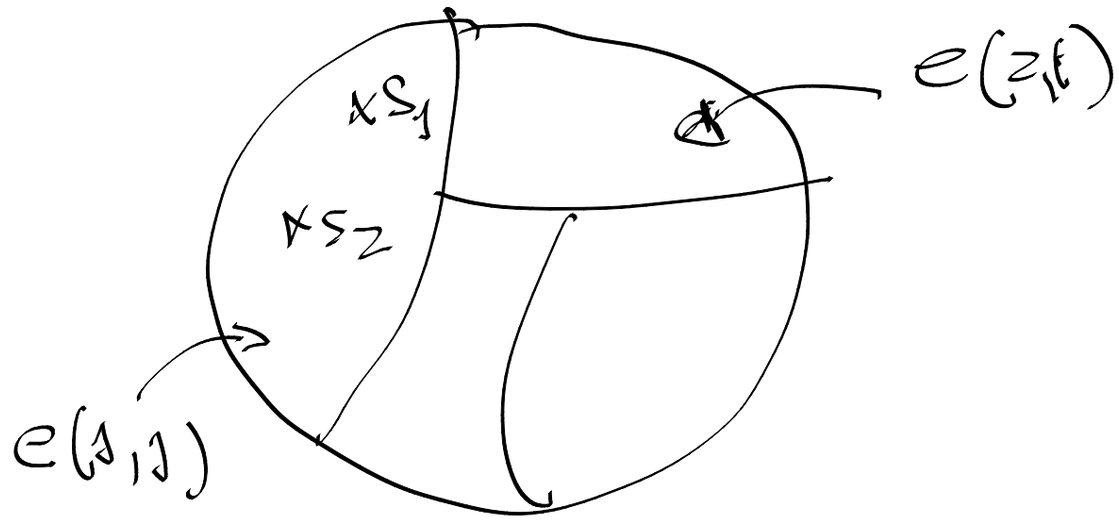
Estado

* Relación de equivalencia de secuencias:
  → coinciden a partir de un cierto punto (entrada)
  → a partir de ese punto la salida coincide

ejemplo: "ascensor"

e									
					0	2	2	2	
S_1	1	2	1	0	1	0	1	2	
S_2				2	1	0	1	2	

* Agrupamos las infinitas secuencias de entrada en clases de equivalencia (estados)



Autómatas finitos deterministicos (AFD)

* AFD: → modelo matemático de sistema con entradas y salidas discretas
  → Número finito de configuraciones o estados

* Resumir la información de entradas anteriores en estados

* determina las salidas futuras en función de las entradas y los estados

* Teoría general: → ciencias de la computación
  → computadora
  → programa
  → sistemas en gral.

Definición tupla $M = (E, \Sigma, \delta, e_0)$

* E: conjunto de estados
* Σ: alfabeto de entrada
* δ: $E \times \Sigma \rightarrow E$ función de transición de estados
* e_0: $e_0 \in E$, estado inicial

AFD con salida

* dos tipos: → Moore: salida esta asociada al estado
  → Mealy: salida esta asociada a la transición

Mealy: tupla $M = (E, \Sigma, \Delta, \delta, \lambda, e_0)$

conj. func. elem.

* E: conj. estados
* Σ: Alfabeto de entrada
* Δ: Alfabeto de salida
* δ: $E \times \Sigma \rightarrow E$ función transición estado
* λ: $E \times \Sigma \rightarrow \Delta$ función transferencia
* e: $e_0 \in E$ estado inicial

Ejemplo: $E = \{e_0, e_1, e_2\}$ $\Sigma = \{0, 1\}$ $\Delta = \{y, n\}$

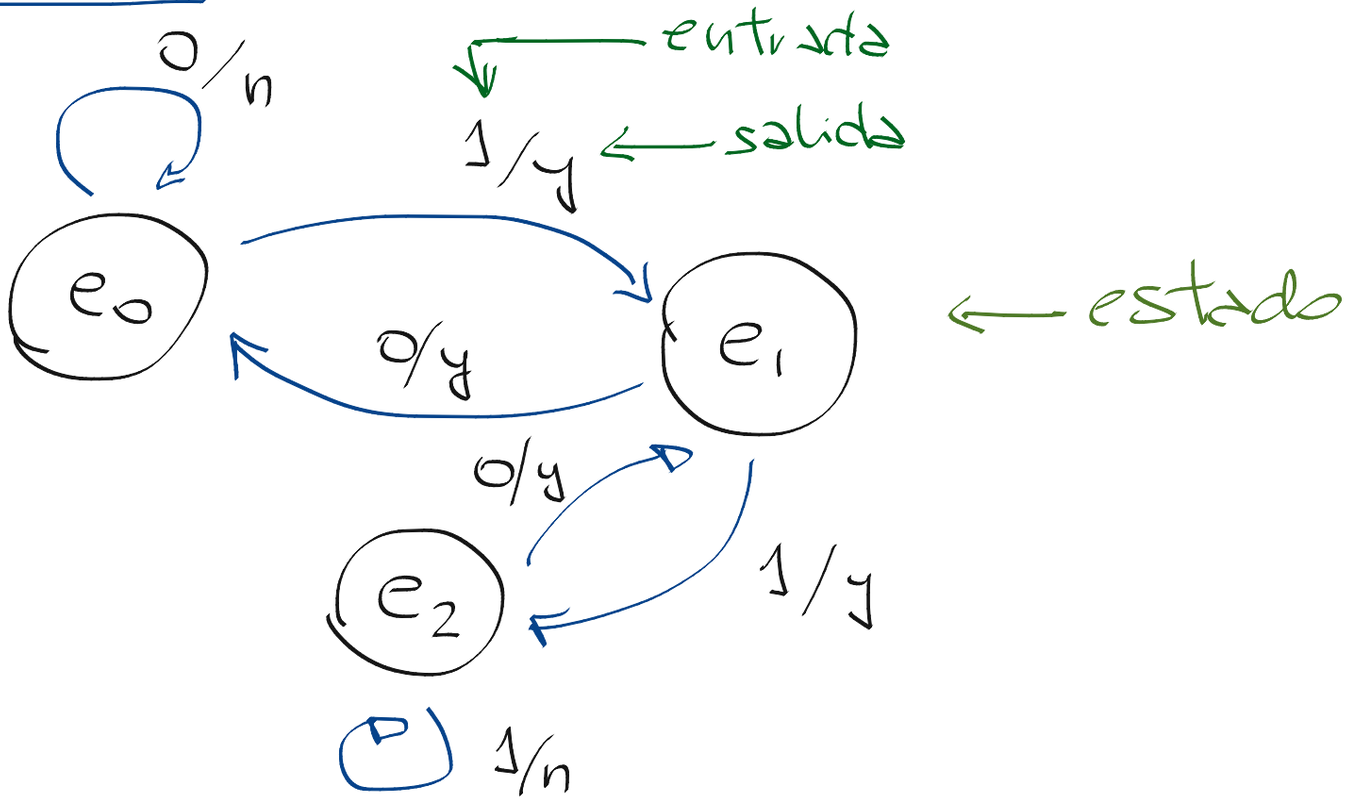
δ	Σ	
	0	1
e_0	e_0	e_1
e_1	e_0	e_2
e_2	e_1	e_2

tabla de transición

λ	Σ	
	0	1
e_0	n	y
e_1	y	y
e_2	y	n

tabla de salida

Diagrama de estados

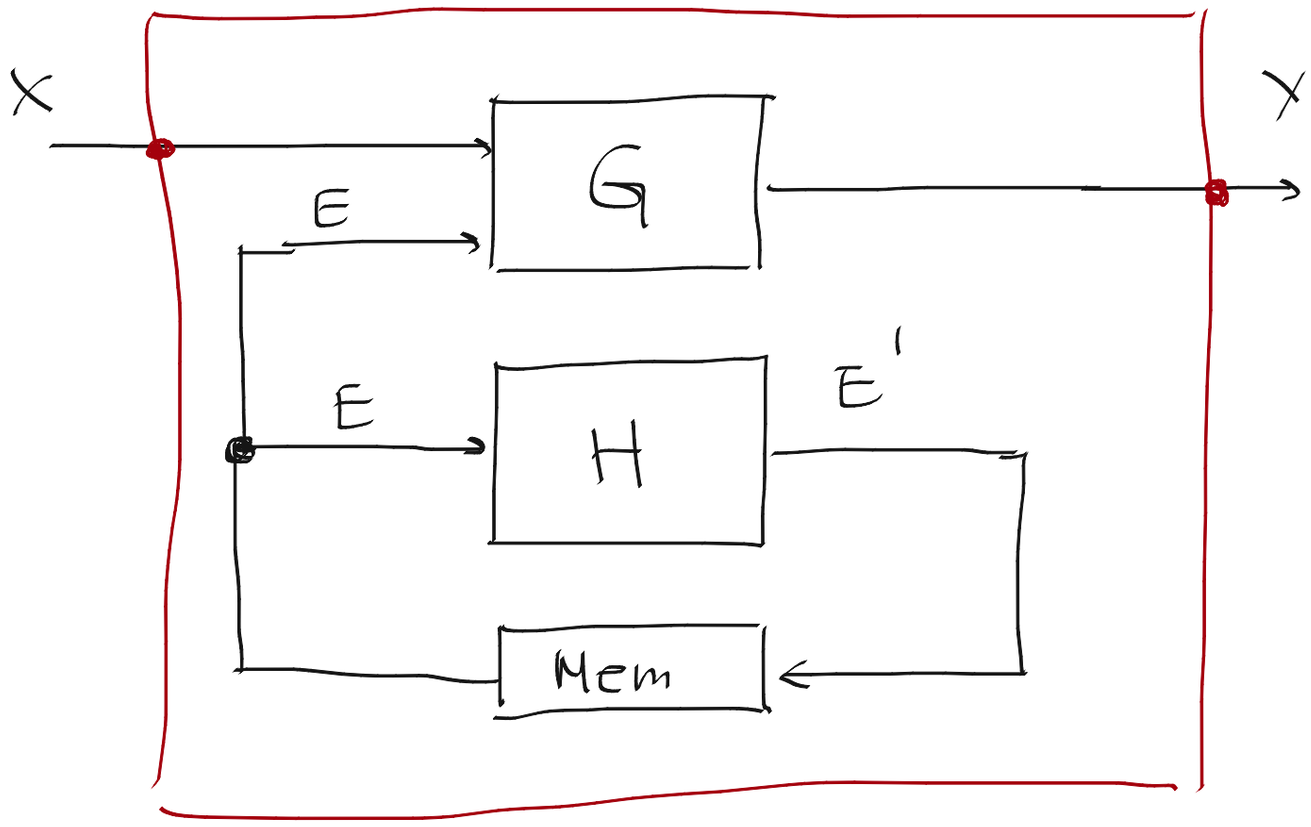


DISEÑO DE CIRCUITOS SECUENCIALES

Def:

$$\begin{cases} Y = G(X, E) & \text{transferencia (salida)} \\ E' = H(X, E) & \text{transición (estado)} \end{cases}$$

* E: estado actual
* E': estado siguiente
* X: entrada
* Y: salida



Diseño

* diseño en base a un AFD
* almacenar el estado en FF_S
* las funciones de transición H y de transferencia G se implementan en lógica combinatoria

Pasos

1) Modelado del sistema mediante un AFD
   - usualmente expresado como diagrama de estados (DE)
   - y definiendo los conj. E, Σ, Δ

2) Deducir la tabla de estados y salida (a partir del DE)

3) Determinar el número de bits y elegir la codificación de las entradas y las salidas

4) Determinar la cantidad de FFs (bits) y elegir la codificación de los estados

5) Incorporar la codificación de estados, entradas y salidas, para obtener las tablas de transiciones y salidas (TTS)

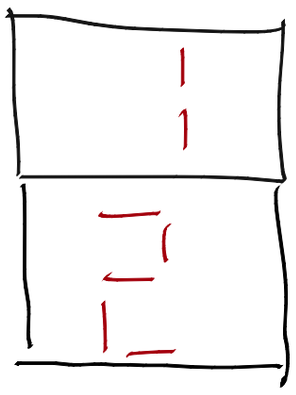
6) Seleccionar los FFs y en base a ellos y TTS crear las las tablas de verdad

7) Minimizar las funciones lógicas resultantes (karnaugh)

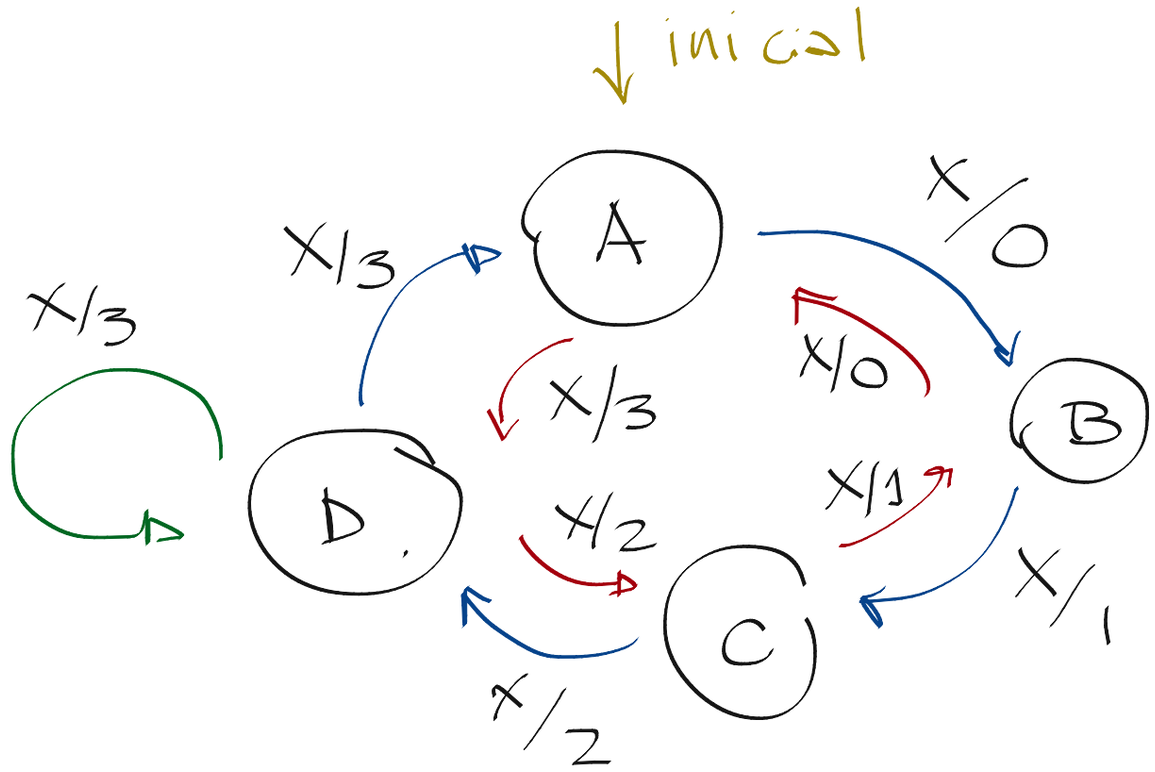
8) Dibujo el circuito en base a las Compuertas y los FFs

Ejemplo: Contador módulo 4

* Cuenta solo (no tiene entrada)
* Cuenta con cada flanco de subida del reloj
* módulo 4: cuando llega a 4 se reinicia



1) Especificar el AFD con DE (Mealy)



3 versiones:

1) circular adelante
2) lineal
3) regresivo circular

2) Tabla estados

E_n	E_{n+1}	S
A	B	0
B	C	1
C	D	2
D	A	3

3) Defino el AFD

$$M = \left(\underbrace{\{A,B,C,D\}}_{E}, \underbrace{\{\ \}}_{\Sigma}, \underbrace{\{0,1,2,3\}}_{\Delta}, \delta, \lambda, \underbrace{A}_{e_0} \right)$$

4) Codificación

* 4 estados → 2 FFs { A → 00, B → 01
* 4 salidas → 2 bits en binario común (3 → 11)

5) Tablas de transición y salida

E_n	E_{n+1}	S
A	B	0
B	C	1
C	D	2
D	A	3

TTS

cambia con la codificación

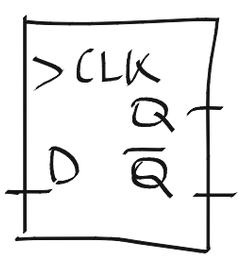
G función lógica

En		Entl		S	
a_1	a_0	b_1	b_0	s_1	s_0
0	0	0	1	0	0
0	1	1	0	0	1
1	0	1	1	1	0
1	1	0	0	1	1

H función lógica

cambia según el tipo de FF elegido

6) Elección de FF → tipo D



$Q = E_n$

$D = E_{n+1}$

En		Entl		S	
f_1	f_0	d_1	d_0	s_1	s_0
0	0	0	1	0	0
0	1	1	0	0	1
1	0	1	1	1	0
1	1	0	0	1	1

tabla de verdad.

* obs: la tabla de verdad resultante puede cambiar dependiendo de la elección del FF

7) Minimizar

* salida: $S = E_n$ → no hay que hacer nada

→ $S_1 = f_1$ $\quad$ $S_2 = f_0$

* estado: → d_1

f_1 \ f_0	0	1
0	0	1
1	1	0

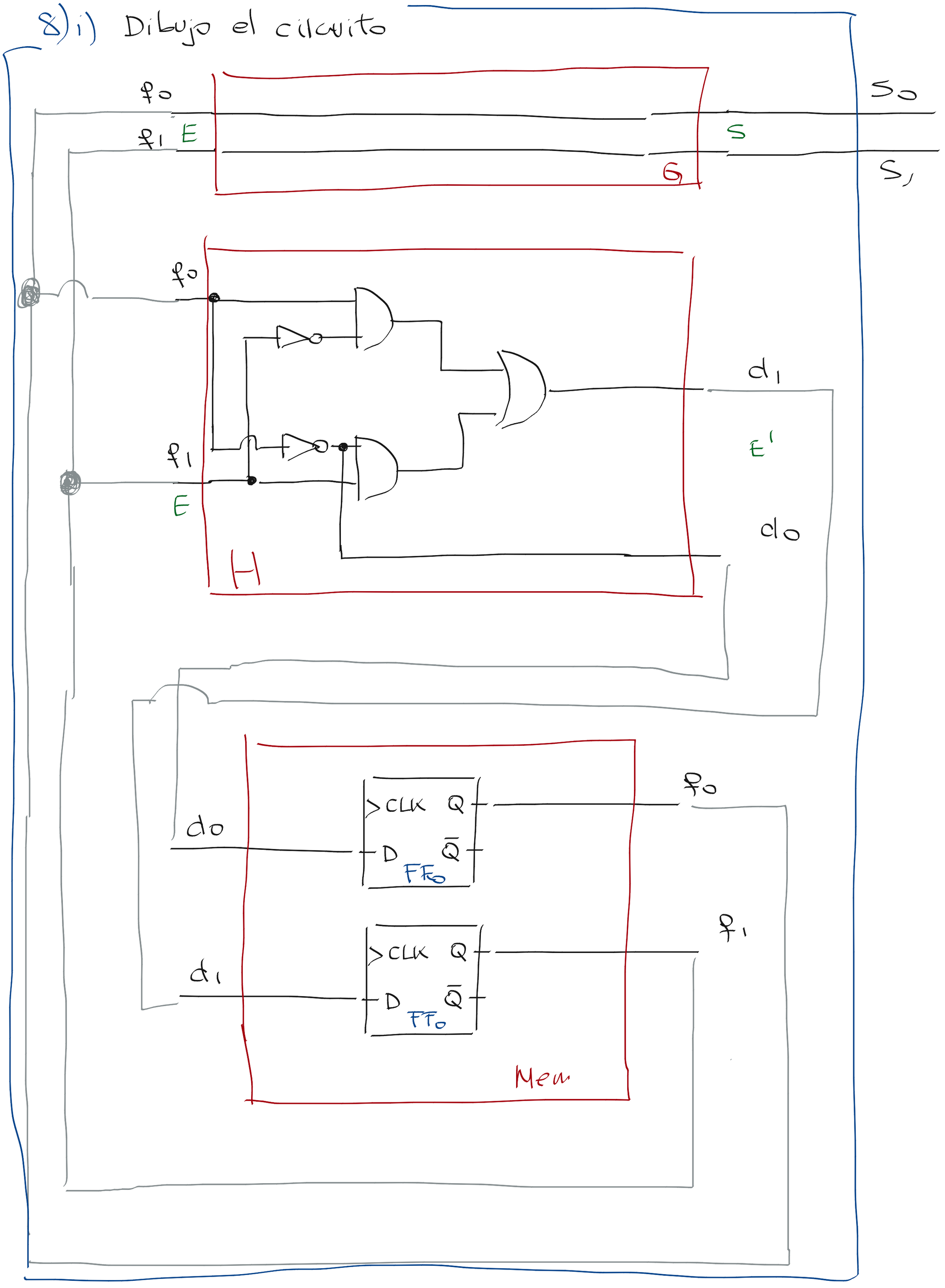
$$d_1 = f_0 \cdot \overline{f_1} + \overline{f_0} \cdot f_1$$

→ d_0

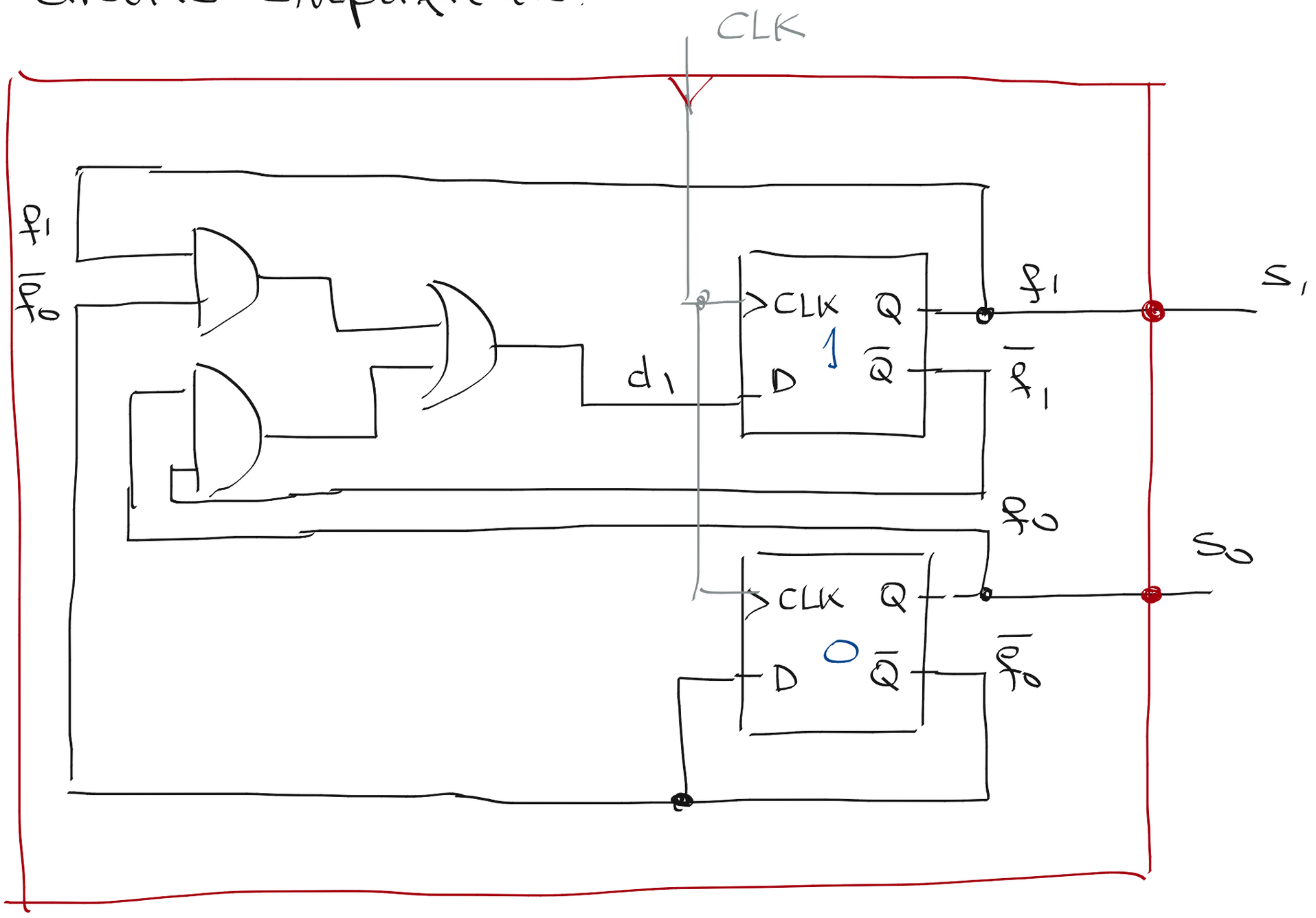
f_1 \ f_0	0	1
0	1	0
1	1	0

$$d_0 = \overline{f_0}$$

8) i) Dibujo el circuito



8) ii) Circuito simplificado.



Anexos

Materiales extra