Pipeline

Ejecución en orden

Pipeline 5 etapas

Etapas del pipeline:

- Fetch: La instrucción se obtiene de memoria.
- Decode: Se determina el tipo de instrucción a ejecutar, junto con los operandos requeridos. Adicionalmente se pueden realizar cálculos simples.
- Execute: Se ejecuta la instrucción. Puede durar un ciclo o más, dependiendo del tipo de operación, y de la implementación de la ALU.
- Memory: Se accede a memoria para leer o escribir valores. Adicionalmente, se pueden realizar los cálculos de la dirección efectiva de memoria.
- Write-back: Se escribe el resultado de la instrucción en el banco de registros.

Ejercicio

Un procesador cuenta con un pipeline de instrucciones escalar de 5 etapas como el considerado en la práctica. Teniendo la secuencia de instrucciones y los datos del cuadro que aparecen más abajo:

1. Identificar todas las dependencias y conflictos que se producen. Indicando de que tipo son y el motivo por el cual se producen.
2. Desarrollar el diagrama de Gantt correspondiente a la evolución de la secuencia de instrucciones asumiendo ejecución en orden (considerando para Load/Store que la dirección de memoria se calcula en Decode), asumiendo un valor inicial de R5 = 3 y se utiliza el esquema de predicción branch tomado para los saltos. Asumiendo que en la etapa Decode se resuelve la dirección donde se va a saltar (PC target).
3. Para los conflictos identificados en el punto 1 indique que técnica se puede utilizar para su resolución, y de que forma resuelve el conflicto.

Load/Store (1 ciclo), mult (2 ciclos), add (1 ciclo)

Código

```
        R5 <- 3
        I1:  R1 <- R4*R7  (mult)
        I2: R3 <- M[R1]    (load)
        I3: R4 <- 1 + R5   (add)
        I4: JNZ R4, label  (branch)
        I5: R7 <- R4 * 1    (mult)
label I6: R8 <- R7 * 3  (mult)
```

Inciso 1

Identificar los conflictos del código anterior. Recordemos; los conflictos pueden ser de tres tipos, por un lado tenemos los conflictos de datos (RAW, WAW y WAR), por otro lado tenemos los conflictos de control (branch), y finalmente tenemos los conflictos estructurales (superposición de etapas).

En el contexto de un pipeline básico de 5 etapas, los conflictos WAR no pueden ocurrir, mientras que los RAW y WAW si, por lo tanto, debemos identificar cuando se dan en el código.

Inciso 1

I1: R1 <- R4*R7 (No hay conflictos)

I2: R3 <- M[R1] (Requiere que I1 finalice el cálculo de R1 - RAW)

I3: R4 <- 1 + R5 (No hay conflictos)

I4: JNZ R4, label (Requiere que I4 finalice el cálculo de R4 - RAW)

I5: R7 <- R4 * 1 (Requiere que I4 finalice el cálculo de R4 - RAW)

label I6: R8 <- R7 * 3 (Requiere que I5 finalice el cálculo de R7 - RAW)

Adicionalmente, no hay conflictos WAW.

Inciso 1

A la hora de identificar los conflictos de datos de un fragmento de código, ante la presencia de un branch, hay que tener en cuenta tanto la posibilidad de salto tomado, como la de salto no tomado. Identifiquen todos los conflictos!!!

Respecto a conflictos de control, podemos afirmar que habrá un conflicto por tener un branch (JNZ -> Jump Non Zero).

Finalmente, en esta etapa no podemos identificar conflictos estructurales.

Inciso 2 - Ciclo 1

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I1: R1 <- R4*R7	F													
I2: R3 <- M[R1]														
I3: R4 <- 1 + R5														
I4: JNZ R4, label														
I5: R7 <- R4 * 1														
label I6: R8 <- R7 * 3														

Inciso 2 - Ciclo 2

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I1: R1 <- R4*R7	F	D												
I2: R3 <- M[R1]		F												
I3: R4 <- 1 + R5														
I4: JNZ R4, label														
I5: R7 <- R4 * 1														
label I6: R8 <- R7 * 3														

Inciso 2 - Ciclo 3

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I1: R1 <- R4*R7	F	D	E*											
I2: R3 <- M[R1]		F	S											
I3: R4 <- 1 + R5														
I4: JNZ R4, label														
I5: R7 <- R4 * 1														
label I6: R8 <- R7 * 3														

Aclaración I: El enunciado especifica que la dirección efectiva de load/store se calcula en la etapa Decode, por lo tanto, como la dirección de memoria depende del valor de R1, la etapa Decode deberá esperar a que dicho valor está calculado.

Inciso 2 - Ciclo 4

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I1: R1 <- R4*R7	F	D	E*	E*										
I2: R3 <- M[R1]		F	S	S										
I3: R4 <- 1 + R5														
I4: JNZ R4, label														
I5: R7 <- R4 * 1														
label I6: R8 <- R7 * 3														

Inciso 2 - Ciclo 5

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I1: R1 <- R4*R7	F	D	E*	E*	M									
I2: R3 <- M[R1]		F	S	S	D									
I3: R4 <- 1 + R5					F									
I4: JNZ R4, label														
I5: R7 <- R4 * 1														
label I6: R8 <- R7 * 3														

Aclaración I (cont): Habiendo finalizado la etapa de ejecución de I1, el valor de R1 ya es conocido. Para poder utilizarlo en el cálculo de la dirección efectiva en I2, este valor se pasa a la etapa Decode vía forwarding.

Aclaración II: Marquen los forwardings!!! Ya sea con una flecha, o coloreando las celdas del Gantt.

Inciso 2 - Ciclo 6

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I1: R1 <- R4*R7	F	D	E*	E*	M	WB								
I2: R3 <- M[R1]		F	S	S	D	E								
I3: R4 <- 1 + R5					F	D								
I4: JNZ R4, label						F								
I5: R7 <- R4 * 1														
label I6: R8 <- R7 * 3														

Inciso 2 - Ciclo 7

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I1: R1 <- R4*R7	F	D	E*	E*	M	WB								
I2: R3 <- M[R1]		F	S	S	D	E	M							
I3: R4 <- 1 + R5					F	D	E+							
I4: JNZ R4, label						F	D							
I5: R7 <- R4 * 1							F							
label I6: R8 <- R7 * 3														

Aclaración III: Al momento de realizar el Decode del branch, el procesador descarta la instrucción recientemente obtenida de memoria (I5) y actualiza el PC asignando la dirección correspondiente a I6 (label).

Inciso 2 - Ciclo 8

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I1: R1 <- R4*R7	F	D	E*	E*	M	WB								
I2: R3 <- M[R1]		F	S	S	D	E	M	WB						
I3: R4 <- 1 + R5					F	D	E+	M						
I4: JNZ R4, label						F	D	E						
I5: R7 <- R4 * 1							F	-						
label I6: R8 <- R7 * 3								F						

Aclaración III (cont): En el ciclo siguiente, la instrucción I5 fue descartada, debido a la política de branch tomado, y se realiza el fetch de la instrucción I6, la cual corresponde al salto tomado del branch.

Aclaración IV: En este momento el procesador realiza la comparación de R4 con 0 para determinar si debe saltar o no.

Inciso 2 - Ciclo 9

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I1: R1 <- R4*R7	F	D	E*	E*	M	WB								
I2: R3 <- M[R1]		F	S	S	D	E	M	WB						
I3: R4 <- 1 + R5					F	D	E+	M	WB					
I4: JNZ R4, label						F	D	E	M					
I5: R7 <- R4 * 1							F	-	-					
label I6: R8 <- R7 * 3								F	D					

Aclaración IV (cont): Una vez evaluada la condición, el procesador determina que la condición de salto del branch se cumple, por lo tanto, reanuda su ejecución normalmente, pues, la predicción de branch resultó exitosa.

Inciso 2 - Ciclo 10

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I1: R1 <- R4*R7	F	D	E*	E*	M	WB								
I2: R3 <- M[R1]		F	S	S	D	E	M	WB						
I3: R4 <- 1 + R5					F	D	E+	M	WB					
I4: JNZ R4, label						F	D	E	M	WB				
I5: R7 <- R4 * 1							F	-	-	-				
label I6: R8 <- R7 * 3								F	D	E*				

Inciso 2 - Ciclo 11

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I1: R1 <- R4*R7	F	D	E*	E*	M	WB								
I2: R3 <- M[R1]		F	S	S	D	E	M	WB						
I3: R4 <- 1 + R5					F	D	E+	M	WB					
I4: JNZ R4, label						F	D	E	M	WB				
I5: R7 <- R4 * 1							F	-	-	-	-			
label I6: R8 <- R7 * 3								F	D	E*	E*			

Inciso 2 - Ciclo 12

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I1: R1 <- R4*R7	F	D	E*	E*	M	WB								
I2: R3 <- M[R1]		F	S	S	D	E	M	WB						
I3: R4 <- 1 + R5					F	D	E+	M	WB					
I4: JNZ R4, label						F	D	E	M	WB				
I5: R7 <- R4 * 1							F	-	-	-	-	-		
label I6: R8 <- R7 * 3								F	D	E*	E*	M		

Inciso 2 - Ciclo 13

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I1: R1 <- R4*R7	F	D	E*	E*	M	WB								
I2: R3 <- M[R1]		F	S	S	D	E	M	WB						
I3: R4 <- 1 + R5					F	D	E+	M	WB					
I4: JNZ R4, label						F	D	E	M	WB				
I5: R7 <- R4 * 1							F	-	-	-	-	-		
label I6: R8 <- R7 * 3								F	D	E*	E*	M	WB	

Inciso 3

- Conflictos de datos: Para solucionar conflictos de datos existen dos posibilidades (dependientes del procesador). Por un lado tenemos los ciclos stall, los cuales frenan el pipeline hasta tener el dato requerido escrito en el banco de registros, y la instrucción que lo requería sea capaz de obtenerlo. Por otro lado, la implementación de forwarding también soluciona los conflictos de tipo RAW.

- Conflictos estructurales: De producirse, basta con aplicar uno o más ciclos de stall sobre una de las instrucciones que busca apropiarse del recurso.

Inciso 3

- Conflictos de control: En materia de solución para conflictos de control, existen dos posibilidades que analizaremos a continuación:
  - Frenar el pipeline: En cuanto se detecta la presencia de un branch (etapa decode), el pipeline se frena hasta haber resuelto la condición de salto (resuelve en execute, en memory se retoma la ejecución del pipeline).
  - Branch retardado: El compilador busca instrucciones que sean independientes del branch y su dirección de salto, y las pone a ejecutar mientras se resuelve el destino del branch
  - Predicción de branch: El procesador asume que el branch será tomado (o no tomado) y al momento de detectar el branch actualiza el PC en consecuencia (tomado -> PC = label; no tomado -> PC = PC++). Si la predicción fallara, el pipeline debe descartar las instrucciones que tiene ejecutando (las cuales no corresponden al flujo de ejecución correcto del programa), reajustar el PC, y reanudar la ejecución a partir de la instrucción correcta.

Inciso 3 - alternativo

Calcular la cantidad de ciclos en promedio, considerando una duración de ciclo de 1ns.

Teniendo en cuenta que la ejecución final demoró 13 ciclos, y que la cantidad total de instrucciones ejecutadas es 5 (ojo, hay 6 instrucciones en total, pero por tratarse de un branch tomado, solo se ejecutaron 5), el tiempo promedio por ciclo lo calculamos como sigue:

$$CPI = \#ciclos / \# instrucciones = 13 / 5 = 2.6 * 1\ ns = \boxed{2.6\ ns}$$

ARRAY MULTIPLIER

Arquitectura de Computadoras

Abstracto

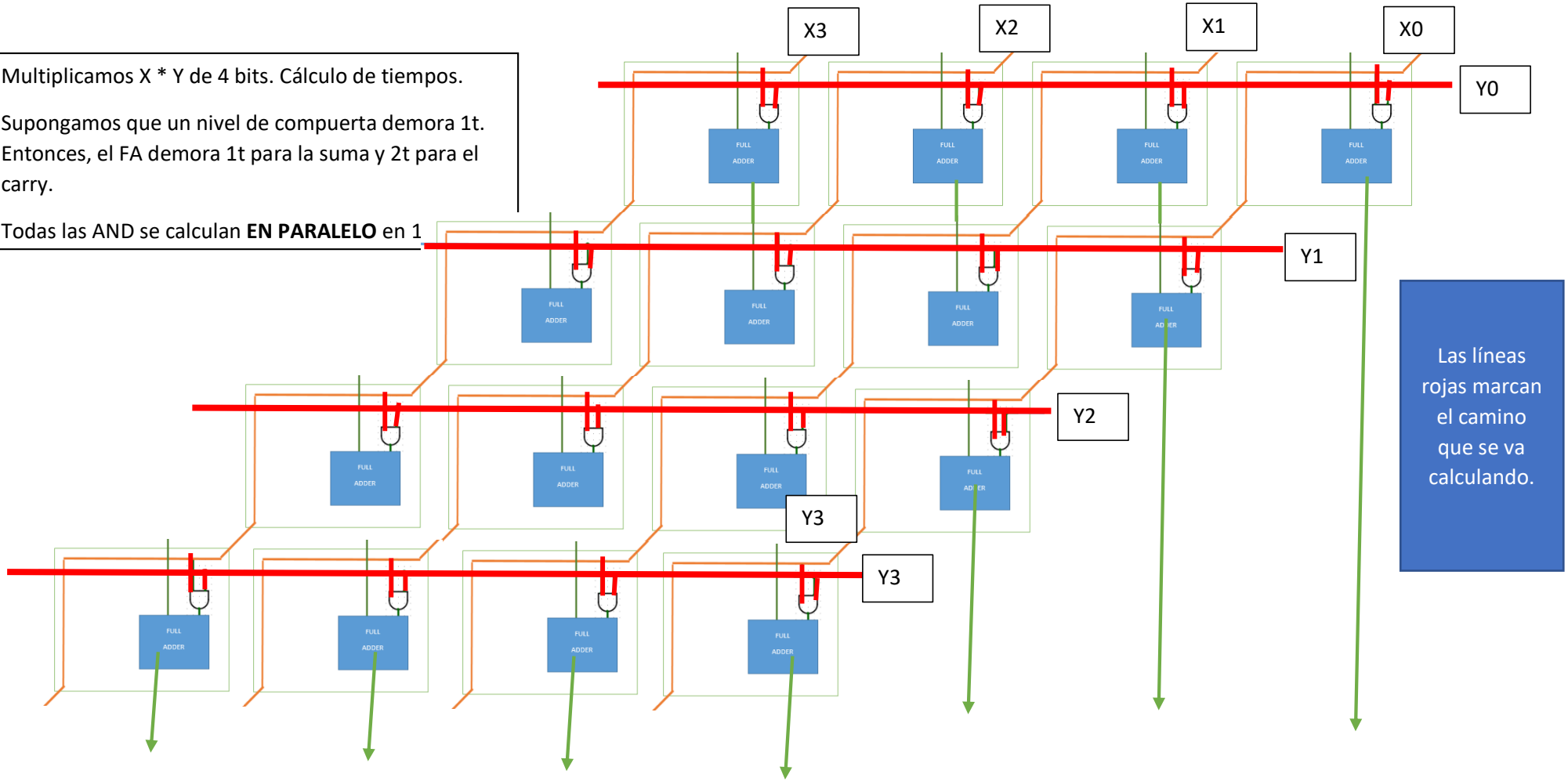
Se desarrolla el ejemplo de cálculo de tiempos para un esquema de Array Multiplier de 4 bits.

Universidad Nacional del Sur

Multiplicamos X * Y de 4 bits. Cálculo de tiempos.

Supongamos que un nivel de compuerta demora 1t. Entonces, el FA demora 1t para la suma y 2t para el carry.

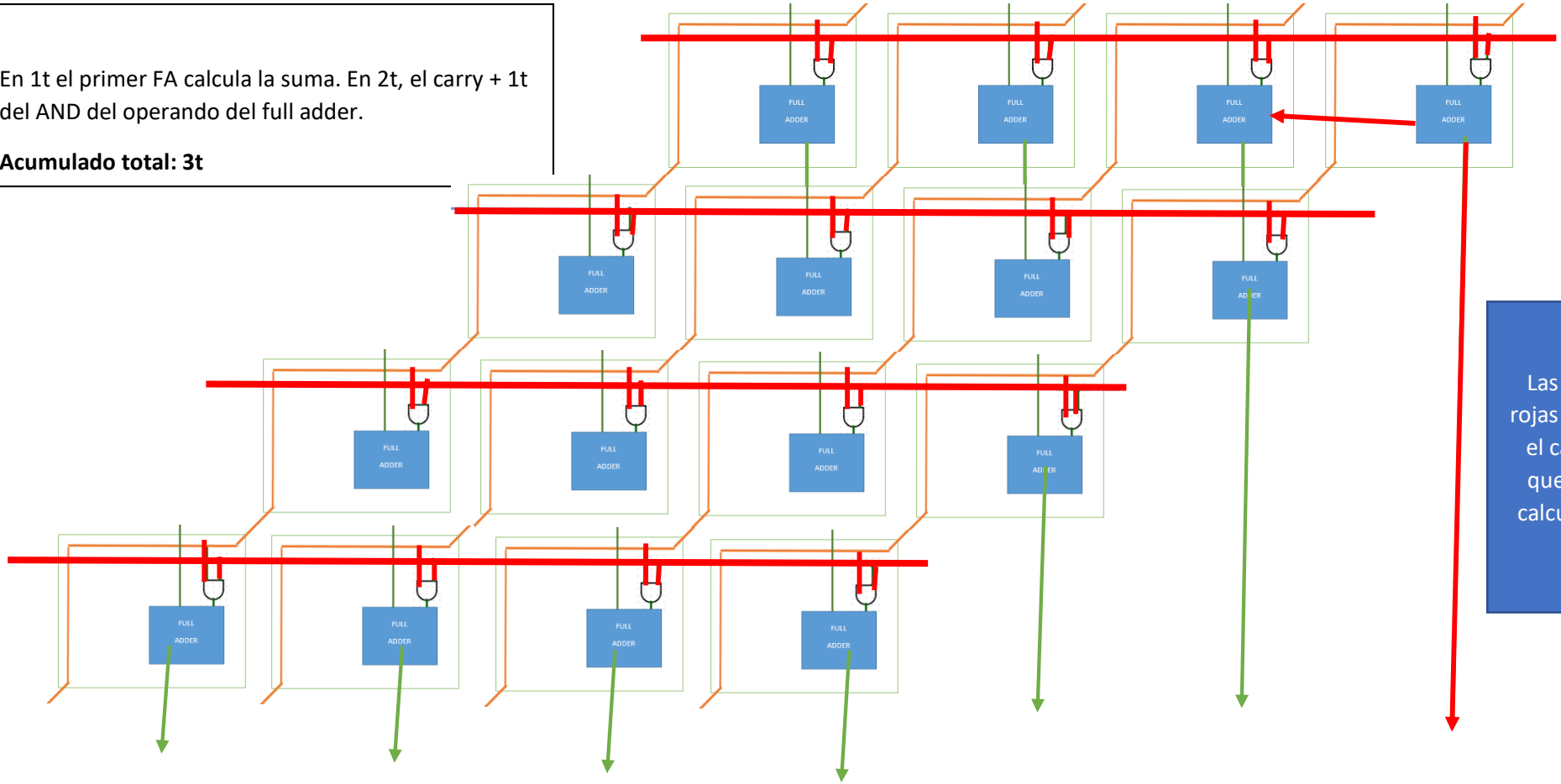
Todas las AND se calculan **EN PARALELO** en 1



Las líneas rojas marcan el camino que se va calculando.

En 1t el primer FA calcula la suma. En 2t, el carry + 1t del AND del operando del full adder.

Acumulado total: 3t

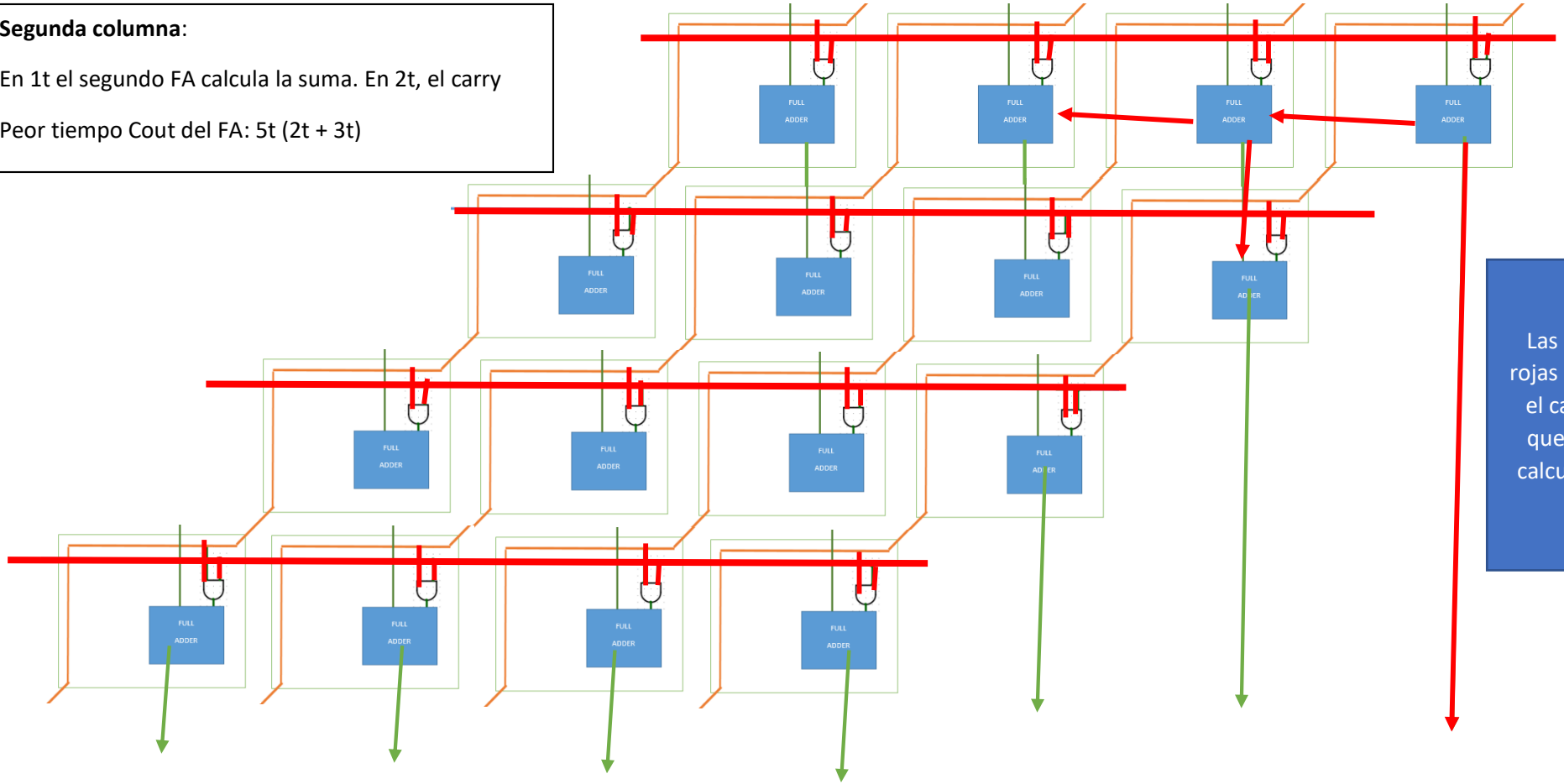


Las líneas rojas marcan el camino que se va calculando.

Segunda columna:

En 1t el segundo FA calcula la suma. En 2t, el carry

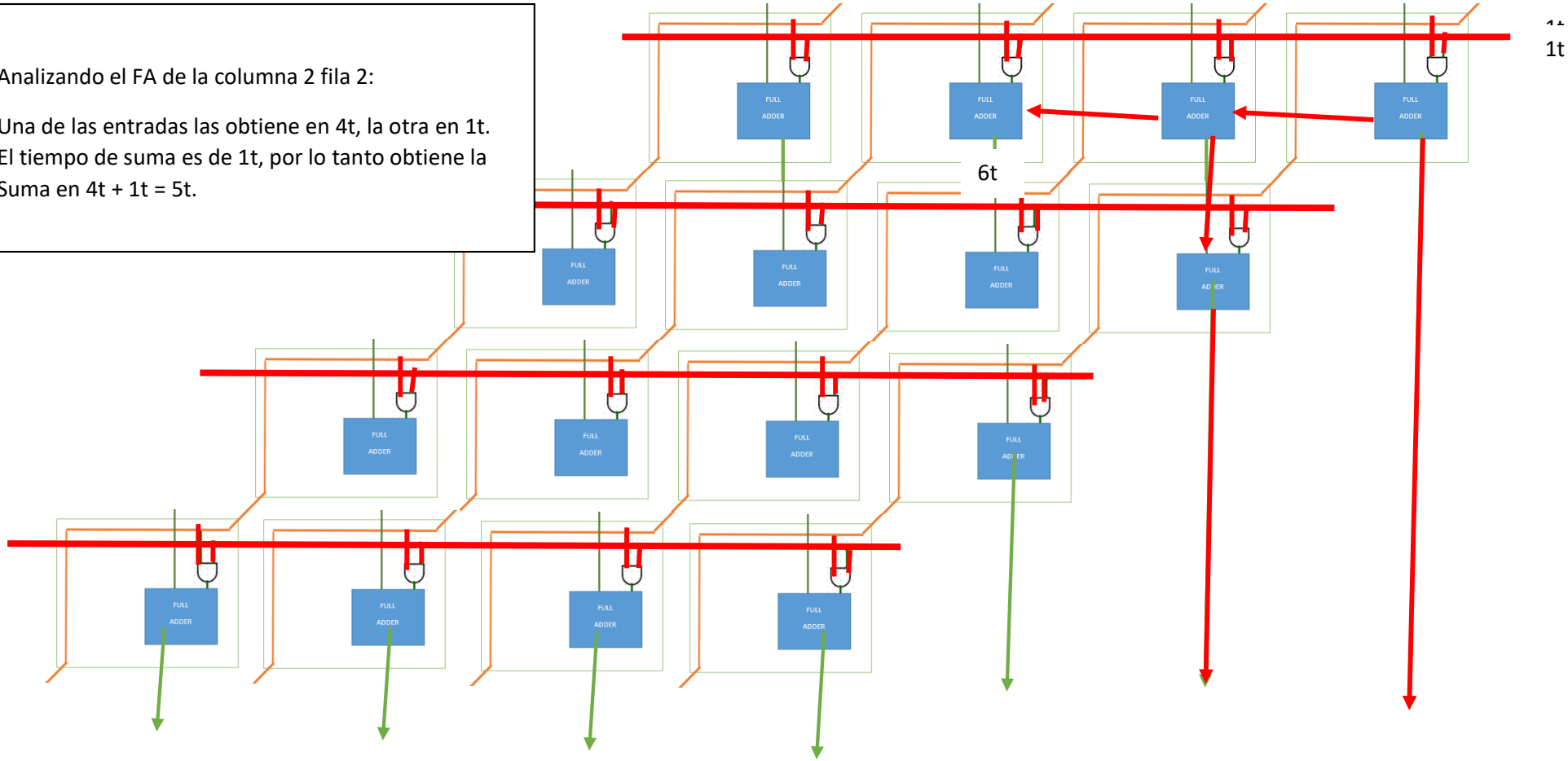
Peor tiempo Cout del FA: 5t (2t + 3t)



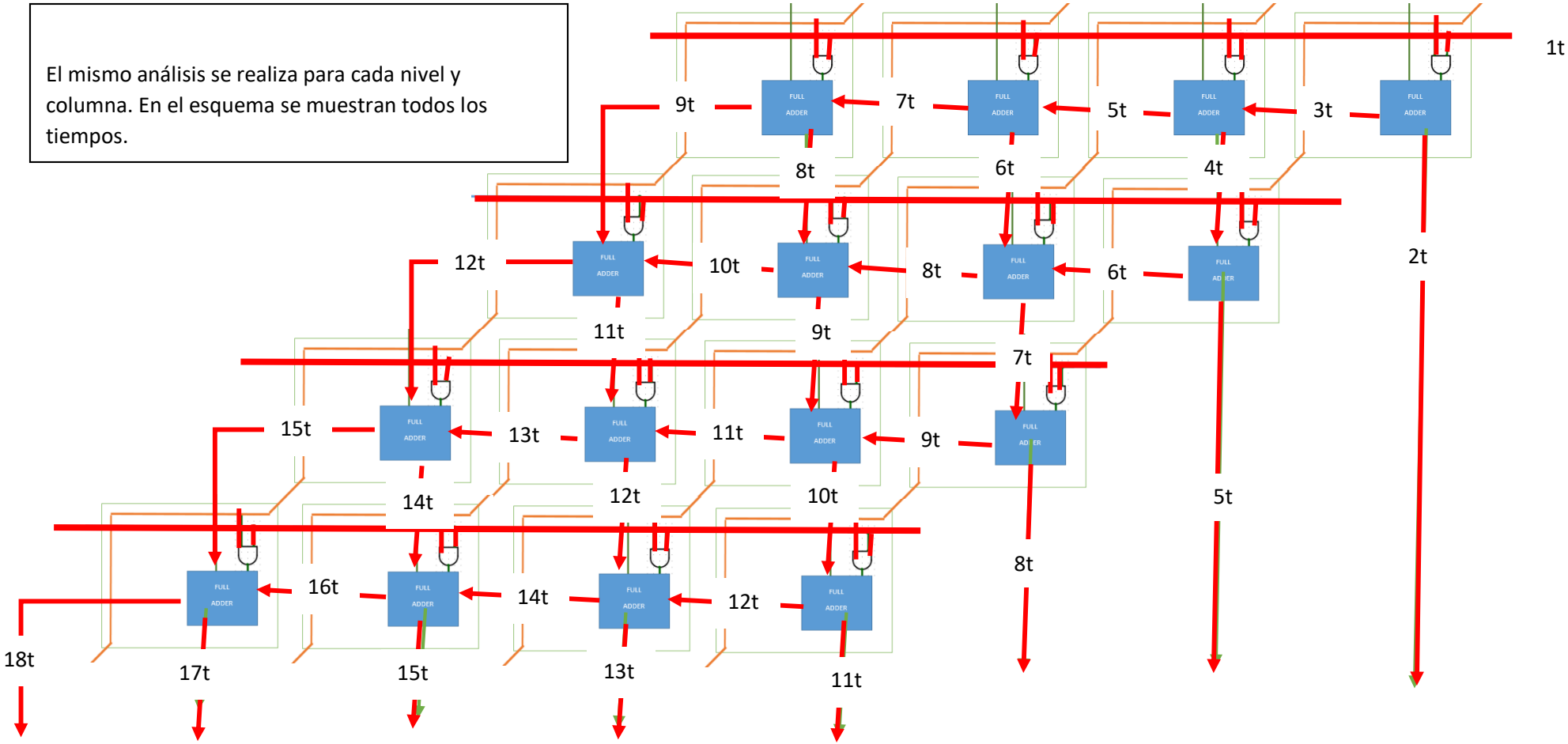
Las líneas rojas marcan el camino que se va calculando.

Analizando el FA de la columna 2 fila 2:

Una de las entradas las obtiene en 4t, la otra en 1t. El tiempo de suma es de 1t, por lo tanto obtiene la Suma en 4t + 1t = 5t.



El mismo análisis se realiza para cada nivel y columna. En el esquema se muestran todos los tiempos.

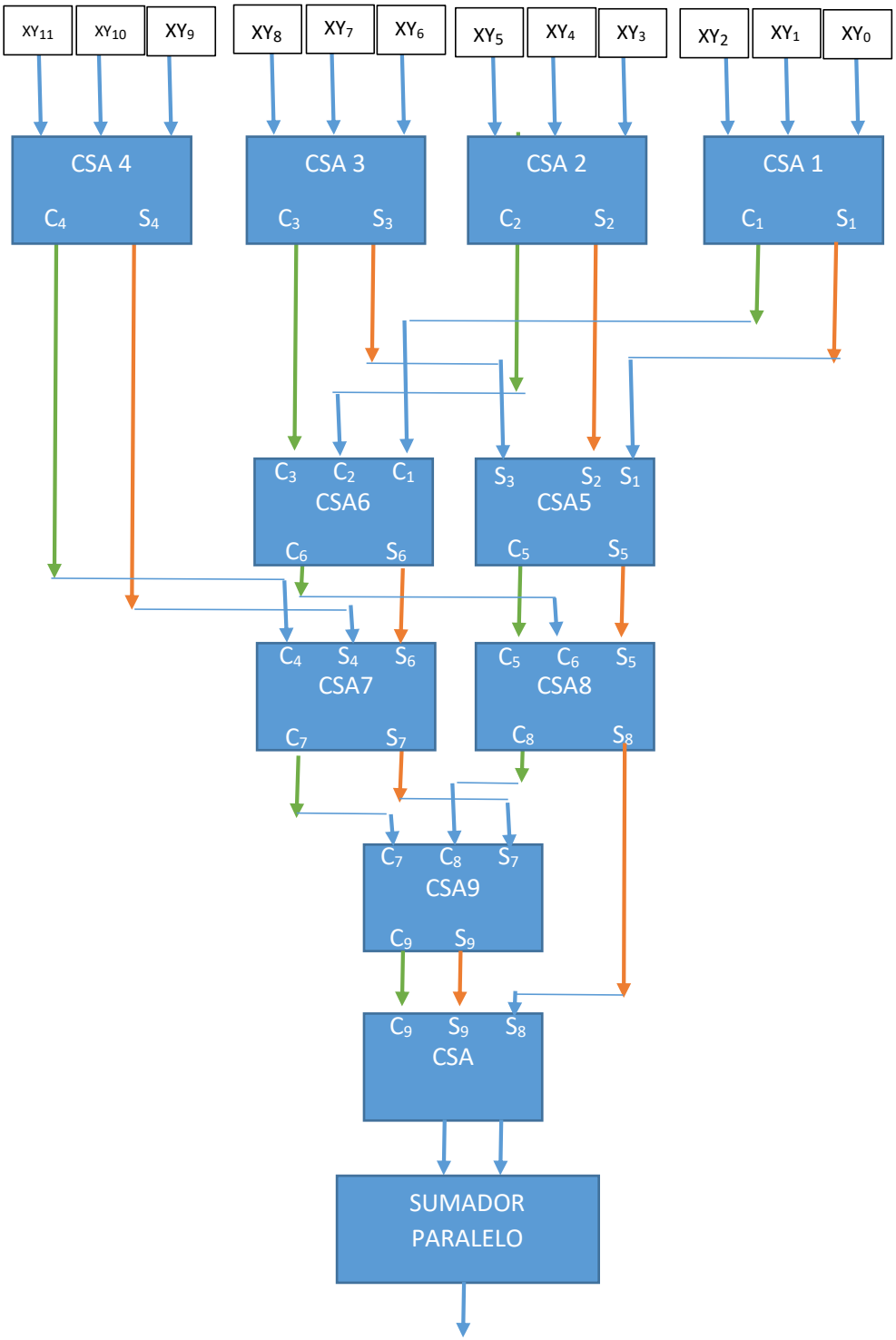


Esquematizar un **Wallace Tree** con **CSAs** para resolver el producto de dos operandos de 12 bits.

1. Indique los operandos de entrada al árbol al suponer X = 25 e Y = 43.
2. Determinar los operandos intermedios y finales a la salida de los CSA.
3. Efectuar la suma paralela a manera de verificación del resultado.
4. Suponiendo los siguientes tiempos, calcule el tiempo total que demora una operación de multiplicación:

    - Cada CSA demora: 4ns.
    - El sumador paralelo demora: 73ns

TIEMPO TOTAL= 4ns*5 + 73ns = 93ns



Realizar el siguiente producto empleando WALLACE TREE

X x Y **0001 1001 = 25 // 0010 1011 =43**

						0	**0**	**0**	**1**	1	0	0	1			
					x											
						0	**0**	**1**	**0**	1	0	1	1			
						0	**0**	**0**	**1**	1	0	0	1	XY0		
					0	**0**	**0**	**1**	1	0	0	1		XY1		
				0	**0**	**0**	**0**	**0**	**0**	0	0			XY2		
			0	**0**	**0**	**1**	1	0	0	1				XY3		
		0	**0**	**0**	**0**	**0**	**0**	**0**	**0**					XY4		
	0	**0**	**0**	**1**	1	0	0	1						XY5		
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**							XY6		
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**							XY7		
0	0	0	0	1	0	0	0	0	1	1	0	0	1	1		

			0	**0**	**0**	**1**	1	0	0	1	XY0									
	0	**0**	**0**	**1**	1	0	0	1	0	XY1										
0	**0**	**0**	**0**	**0**	**0**	0	0	0	0	XY2										
0	0	0	0	1	0	1	0	1	1	SS1										
0	0	0	0	1	0	0	0	0	0	SC1										
			0	**0**	**0**	**1**	1	0	0	1	0	0	0	XY3						
	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0	0	0	0	XY4							
	0	**0**	**0**	**1**	1	0	0	1	0	0	0	0	XY5							
	0	0	0	1	1	1	1	1	0	1	0	0	0	SS2						
	0	0	0	0	0	0	0	0	0	0	0	0	0	SC2						
0	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0	0	0	0	0	XY6						
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0	0	0	0	0	0	XY7						
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0	0	0	0	0	0	XY8						
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0	0	0	0	0	0	SS3						
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0	0	0	0	0	0	SC3						
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0	0	0	0	0	0	XY9						
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0	0	0	0	0	0	XY10						
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0	0	0	0	0	0	XY11						
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0	0	0	0	0	0	SS4						
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0	0	0	0	0	0	SC4						
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0	0	0	0	0	0	C3						
0	0	0	0	0	0	0	0	0	1	0	0	0	0	C1						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	C2						
0	0	0	0	0	0	0	0	0	1	0	0	0	0	S6						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	C6						

CSA6

0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	S1
0	0	0	0	0	1	1	1	1	1	0	1	0	0	0	S2
0	**0**	**0**	**0**	**0**	**0**	**0**	0	0	0	0	0	0	0	0	S3
0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	S5
0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	C5

0	**0**	**0**	**0**	**0**	**0**	**0**	0	0	0	0	0	0	0	0	S4
0	**0**	**0**	**0**	**0**	**0**	**0**	0	0	0	0	0	0	0	0	C4
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	S6
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	S7
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C7

0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	S5
0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	C5
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C6
0	0	0	0	0	1	1	1	0	0	1	0	0	1	1	S8
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	C8

0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	S7
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C7
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	C8
0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	S9
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C9

0	0	0	0	0	1	1	1	0	0	1	0	0	1	1	S8
0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	S9
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C9
0	0	0	0	0	1	1	0	0	0	1	0	0	1	1	S10
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	C10
0	0	0	0	1	0	0	0	0	1	1	0	0	1	1	Suma paralela

En el cuestionario global del día 17 de Junio del 2021, había 6 variantes para el ejercicio numero 1 relacionado con la resolución de circuitos eléctricos.

Entre estas 6 variantes se pedía modelar un diagrama de estados con:

- Dos válvulas y implementación con decoder
- Dos válvulas y implementación con PLA
- Dos válvulas y implementación con ROM
- Dos válvulas, luz y extractor con decoder
- Dos válvulas, luz y extractor con PLA
- Dos válvulas, luz y extractor con ROM

La resolución de todas estas variantes era bastante similar en varios aspectos, por lo cual (para esta explicación) se hará énfasis en una en particular y posteriormente se mencionaran algunas consideraciones especiales en el caso de querer implementar el resto.

"Suponiendo un diagrama de estados (Figura 1) que modela un proceso industrial gobernado por un reloj principal de 0.5Hz. El proceso consiste de 6 estados y las transiciones entre ellos ocurren con el pulso del reloj en función del valor de una señal A que modela si la potencia del motor es suficiente o no.

En función de estado en el que se encuentra, se deben cerrar o abrir dos válvulas V1 y V2. Además, mientras el sistema esté en el estado marcado en verde debe estar encendido un extractor de aire, mientras esté en el estado marcado en rojo debe encender una luz.

1. *Empleando un contador binario Up-Down (Figura 2 o el contador de Logisim) con carga paralela y compuertas:*
    a. *Implemente la transición entre los estados del sistema.*
    b. *Esquematice el diagrama del circuito*
    c. *Minimice las funciones implementadas con las compuertas mediante un k-map y escriba las funciones minimizadas resultantes.*
2. *Implemente las cuatro funciones de salida utilizando un* ***decoder de 3 a 8*** *y compuertas.*

Figura 1

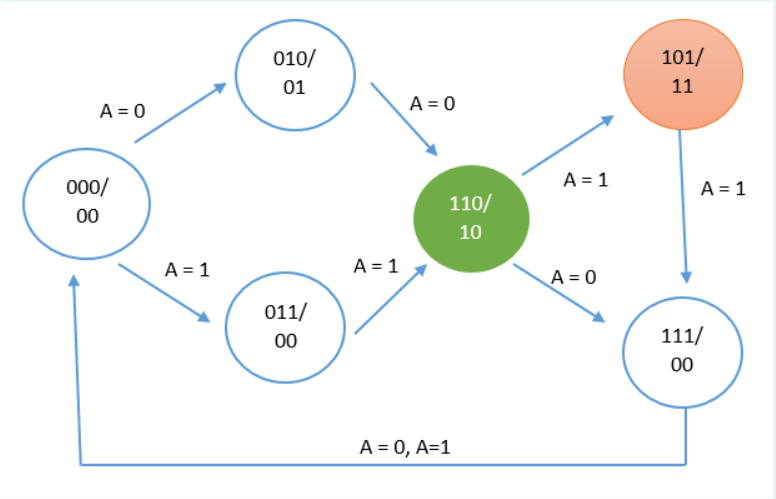
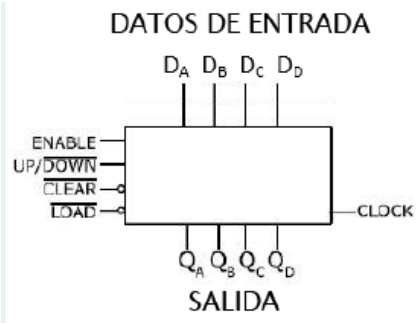


Figura 2



Línea	Descripción
Datos de entrada	Líneas de entrada de datos. Cuando Load=0, carga estos valores en el contador.
Enable	Enable = 1 habilita el circuito del contador Enable = 0 deshabilita la operación del circuito
Up/Down	Up/Down = 1, cuenta en forma ascendente. Up/Down = 0, cuenta en forma descendente.
Clear	Setea el contador en el estado 0000. **Es una línea asincrónica.**
Load	Load = 0, carga en el contador los datos de entrada (D_A, D_B, D_C, D_D)
Clock	Reloj del contador

Lo primero que hay que hacer es, en base a los estados, generar la tabla de verdad que modele el cambio entre estos estados. Esta tabla de verdad va a contar con 4 entradas (A,B,C,D) donde BCD representan la codificación del estado actual. En este punto no es necesario considerar las válvulas, la luz o el extractor ya que no influyen en el cambio de estados (su utilización solo es requerida en el ejercicio 2). Las salidas de esta tabla de verdad van a ser la codificación QB QC QD del nuevo estado.

A	B	C	D	QB	QC	QD
0	0	0	0	0	1	0
0	0	0	1	*	*	*
0	0	1	0	1	1	0
0	0	1	1	*	*	*
0	1	0	0	*	*	*
0	1	0	1	*	*	*
0	1	1	0	1	1	1
0	1	1	1	0	0	0
1	0	0	0	0	1	1
1	0	0	1	*	*	*
1	0	1	0	*	*	*
1	0	1	1	1	1	0
1	1	0	0	*	*	*
1	1	0	1	1	1	1
1	1	1	0	1	0	1
1	1	1	1	0	0	0

A partir de esta tabla de verdad, se pueden plantear los tres diferentes mapas de Karnaugh para las salidas QB, QC y QD.

Para el caso de QB, se tiene el siguiente mapa:

AB/CD	0 0	0 1	1 1	1 0
0 0	0	*	*	1
0 1	*	*	0	1
1 1	*	1	0	1
1 0	0	*	1	*

Tomando los términos de la siguiente manera, nos terminan quedando la siguiente expresión:

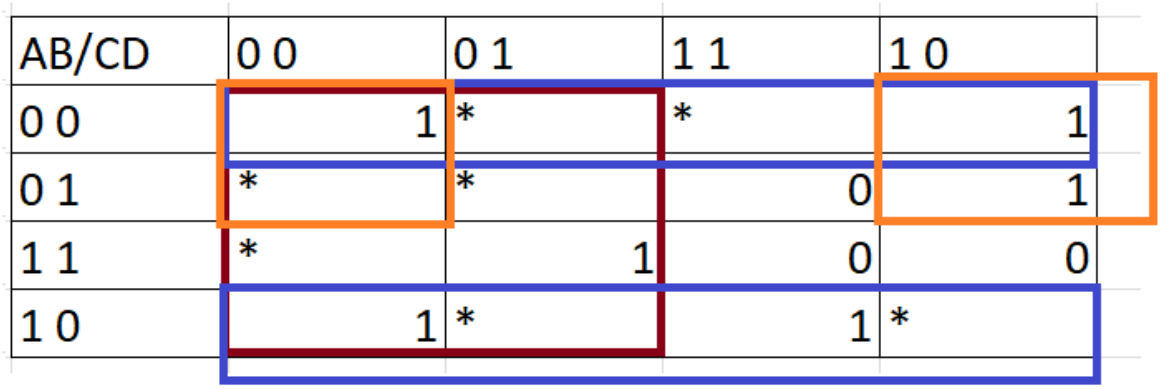
AB/CD	0 0	0 1	1 1	1 0
0 0	0	*	*	1
0 1	*	*	0	1
1 1	*	1	0	1
1 0	0	*	1	*

QB = C'D + CD' + B'C

Para el caso de QC, se tiene el siguiente mapa:

AB/CD	0 0	0 1	1 1	1 0
0 0	1	*	*	1
0 1	*	*	0	1
1 1	*	1	0	0
1 0	1	*	1	*

Tomando los términos de la siguiente manera, nos terminan quedando la siguiente expresión:

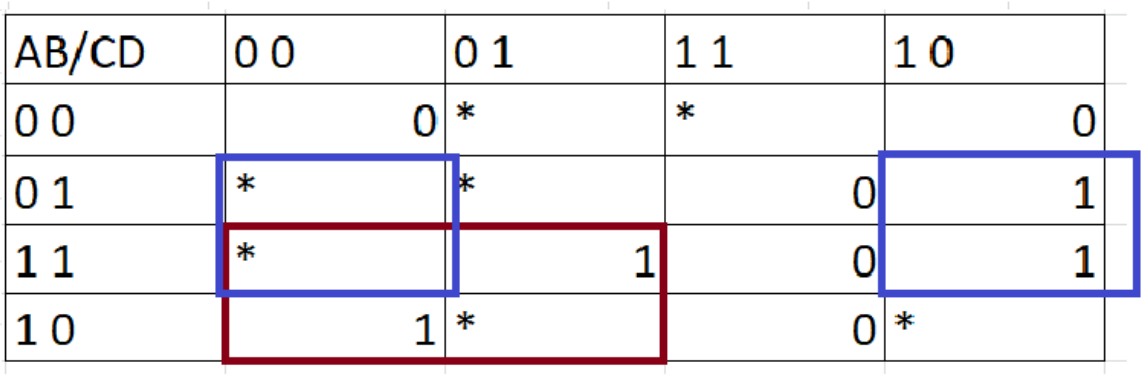


QC = C’ + B’ + A’D’

Por último, para el caso de QD se tiene el siguiente mapa:

AB/CD	0 0	0 1	1 1	1 0
0 0	0	*	*	0
0 1	*	*	0	1
1 1	*	1	0	1
1 0	1	*	0	*

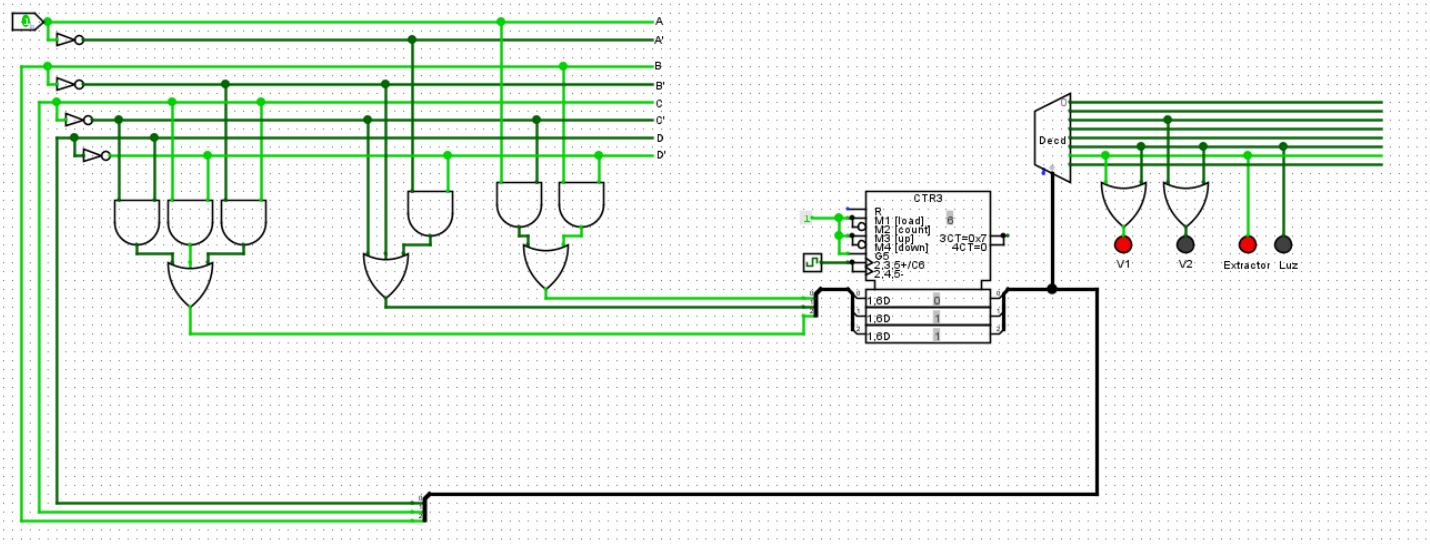
Tomando los términos de la siguiente manera, nos terminan quedando la siguiente expresión:



QD = AC’ + BD’

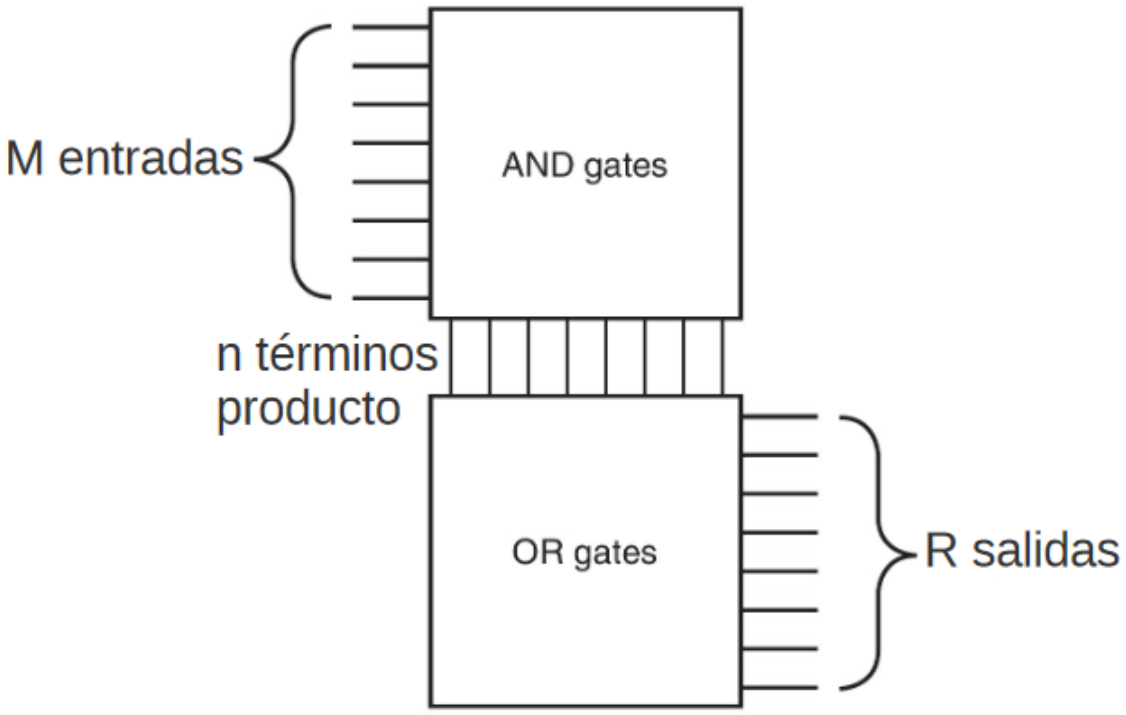
Hasta este punto, todas las variantes del ejercicio tenían que llegar a un resultado similar. Las principales variaciones en las respuestas pueden darse según como se manejen los mapas anteriores (y si se hace uso o no de los don’t care)

Para el punto 2, donde se requería la implementación del circuito, se podía optar tanto hacerlo en papel como en Logisim. En ambos casos, tenía que ser legible el circuito y se debería poder interpretar adecuadamente la codificación de los estados y la implementación del sub-circuito requerido (decoder, PLA o ROM). En el caso de que se implementara en Logisim, ustedes podían tener la certeza de si el circuito andaba o no, ya que lo podían probar.

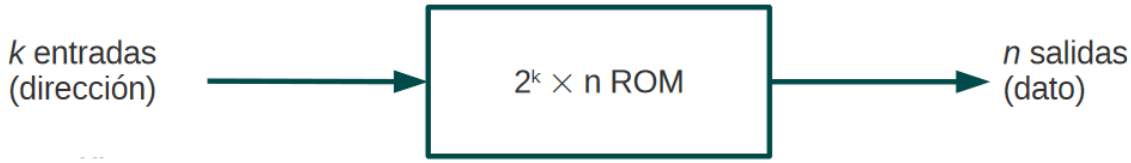


En el caso de que la implementación requerida sea con PLA o ROM, se podía usar la implementación que tiene algunas versiones del Logisim o hacerlo manualmente (esto último requería más tiempo). En caso de hacerlo en papel, se esperaba que se entregara la configuración de las funciones de ambos dispositivos, es decir:

- En el caso del PLA, se esperaba un diagrama codificando la función que permitía modelar las válvulas, luz y extractor (es decir, el diagrama con compuertas and y or)



- Para el caso de la ROM, se esperaba que se codificaran las salidas como funciones de la entrada.



Finalmente, el manejo o no de la luz y el extractor solo ocasionaba la inclusión de dos líneas y dos pines en todas las variantes del problema.