75.562 · Fundamentos de Computadores · 2024-25

PEC3 - Tercera prueba de evaluación continua

Apellidos: *López Henestrosa*
Nombre: José Carlos

Formato y fecha de entrega

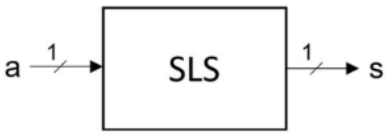
- Para dudas y aclaraciones sobre el enunciado debéis dirigiros al consultor responsable de vuestra aula.
- Hay que entregar la solución en un fichero PDF utilizando una de las plantillas entregadas conjuntamente con este enunciado.
- Se debe entregar a través de la aplicación de **Entrega de la Actividad** correspondiente del apartado **Contenidos** de vuestra aula.
- La fecha límite de entrega es el **30 de abril** (a las 24 horas).
- **Razonad la respuesta en todos los ejercicios. Las respuestas sin justificación no recibirán puntuación.**

Respuestas

Ejercicio 1 [25 %]

(Sección 4.2: Representación gráfica: grafos de estados)

Se quiere diseñar el grafo de estados de un circuito lógico secuencial llamado SLS. El circuito tiene una entrada de un bit, denominada `a`, y una salida de un bit, denominada `s`, según la estructura siguiente:



El circuito SLS va leyendo el valor de la entrada en cada ciclo. Denominamos a_i al valor en la entrada en el ciclo `i`. El circuito tiene que detectar en cada ciclo si el valor formado por los tres dígitos leídos en los últimos tres ciclos, $a_i\ a_{i-1}\ a_{i-2}$, es un número múltiplo de tres. Cuando se detecta que el valor es múltiplo de tres en el siguiente ciclo, se pone el valor `1` en la salida `s` durante un ciclo. En cualquier otro caso, la salida `s` vale `0`.

Tened en cuenta que el número cero es múltiplo de cualquier número y, por lo tanto, es múltiplo de tres. Inicialmente podéis considerar que el circuito se encuentra en el estado `000` (como si las últimas tres entradas fueran cero) y, por lo tanto, la salida es `1`.

Ejemplo de funcionamiento:

Entrada `a`	0	1	0	0	0	1	1	1	0	1	0	1	…
Salida `s`	1	1	0	0	0	1	0	1	0	1	0	0	0

Se pide que diseñéis el grafo de estados del circuito SLS, especificando claramente la leyenda del mismo.

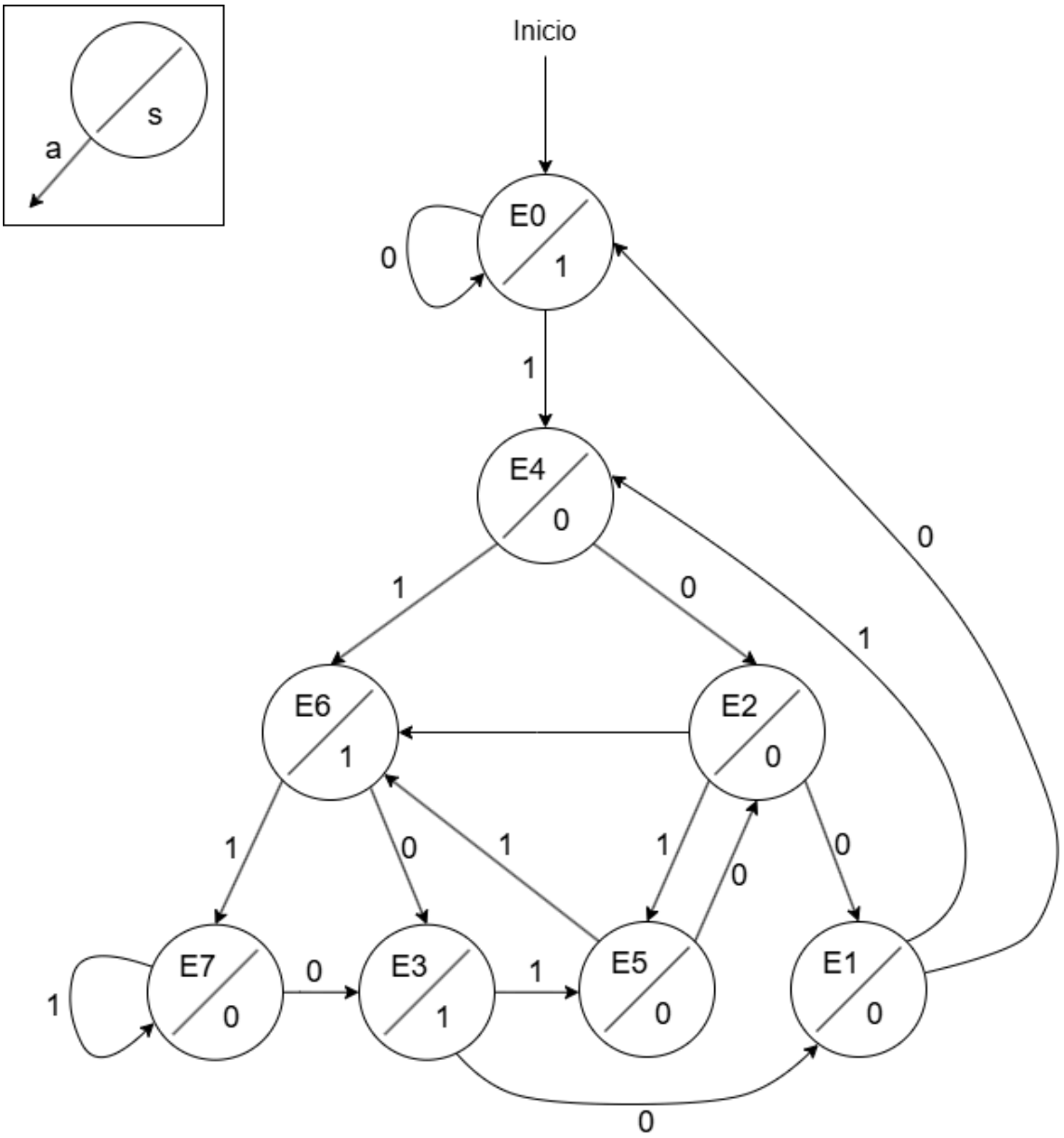
El valor de la salida `s` del circuito SLS es `1` cuando el circuito se encuentra en uno de los siguientes estados:

- `000` (`0` en decimal, el cual es múltiplo de 3).
- `011` (`3` en decimal, el cual es múltiplo de 3).
- `110` (`6` en decimal, el cual es múltiplo de 3).

Esta es la tabla de estados:

TABLA DE ESTADOS	
ESTADO	$a_i\ a_{i-1}\ a_{i-2}$
E0	000
E1	001
E2	010
E3	011
E4	100
E5	101
E6	110
E7	111

Si tenemos esto en cuenta junto con la información aportada por el enunciado del ejercicio, podemos formar el siguiente grafo de estados, cuya leyenda se encuentra en la parte superior izquierda:

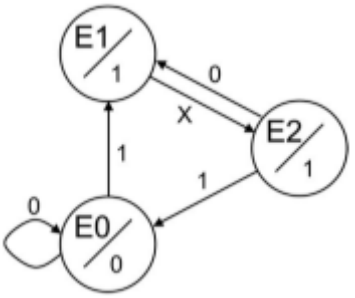


Grafo de estados del circuito SLS

Ejercicio 2 [25 %]

(Sección 4.4: Implementación (del modelo de Moore))

a) **[5 %]** El grafo siguiente describe el comportamiento de un circuito secuencial (no dibujamos su leyenda porque no es necesario, para este ejercicio). Suponiendo que este circuito se implementa de la forma habitual, con una memoria ROM y biestables, indicad cuántas palabras tiene que tener la memoria ROM, de cuántos bits tiene que ser cada uno, y el contenido de la memoria ROM.



Le hemos preguntado a una herramienta generativa que resuelva el ejercicio y ha dado el siguiente resultado:

Para determinar las características de la memoria ROM en la implementación del circuito secuencial, seguimos estos pasos:

1. ***Determinar la cantidad de estados***

   Observamos que el circuito tiene tres estados:

   - *E0*
   - *E1*
   - *E2*

   *Para codificar estos estados en la memoria, necesitamos **al menos 2 bits** (n) porque con 2 bits podemos representar hasta 4 estados (2^2=4), lo cual es suficiente para nuestros 3 estados.*

2. ***Determinar las entradas y salidas***

   La tabla de transición nos dice que:

   - *Hay **una única entrada X** (valores 0 o 1).*
   - *Hay **una única salida S** (valores 0 o 1).*

3. ***Formato de la memoria ROM***

   Cada palabra de la ROM tiene que contener:

   1. ***El estado siguiente E^+**, que está codificado en **2 bits**.*
   2. ***La salida S**, que ocupa **1 bit**.*

*Por lo tanto, **cada palabra de la ROM tendrá 2 + 1 = 3 bits**.*

4. ***Número de palabras en la ROM***

   Cada combinación de (E,X) representa una dirección en la ROM. Como:

   - *Hay **3 estados posibles** (E0, E1, E2).*
   - *Hay **2 posibles valores de X (0 o 1)**.*

   Entonces, el número total de combinaciones es:

   $$3 \times 2 = 6$$

   *Por lo tanto, la ROM es de **6 palabras**.*

5. ***Contenido de la ROM***

   Cada fila de la tabla de transición se traduce en una dirección de la ROM:

Estado Actual (E)	Entrada (X)	Estado Siguiente (E+)	Salida (S)	Codificación ROM
E0 (00)	*0*	*E0 (00)*	*0*	*000000*
E0 (00)	*1*	*E1 (01)*	*0*	*001001*
E1 (01)	*X*	*E2 (10)*	*1*	*010001*
E2 (10)	*0*	*E1 (01)*	*1*	*100011*
E2 (10)	*1*	*E0 (00)*	*1*	*101001*

Nota: La tercera fila se aplica a cualquier X, por lo cual se escribe una sola vez

Razonad la corrección de la solución y, en caso de error, indicad qué errores aparecen en la solución propuesta.

La respuesta de la herramienta generativa se está confundiendo con el valor X de la entrada del estado `E1`, ya que lo está considerando como la única salida del circuito. Lo importante en el paso 2 es identificar el número de bits de las entradas y de las salidas, el cual es 1.

Por otra parte, el paso 4 presenta también un error, ya que para calcular el número de palabras de la ROM se hace la siguiente operación:

$$número\ de\ palabras = 2^{número\ \ de\ bits\ de\ dirección}$$

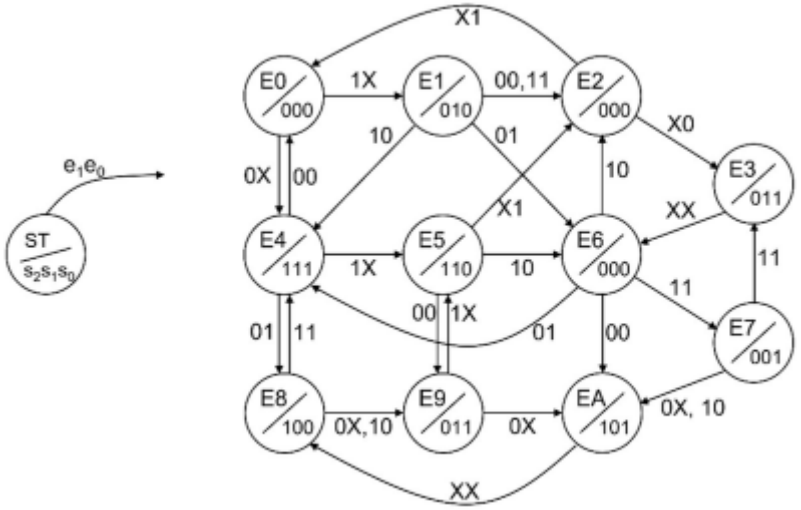
Para calcular el número de bits de dirección, sumamos el número de bits del estado (2) más el número de bits de la entrada (1), lo cual da como resultado 3. Hecho esto, calculamos $2^3 = 8$, lo cual representa el número de palabras de la ROM.

Por último, el paso 5 está mal, ya que la tabla de transición generada no es correcta.

Esta sería la tabla de transición corregida:

Estado Actual (E)	Entrada (X)	Estado Siguiente (E+)	Salida (S)	Codificación ROM
E0 (00)	0	E0 (00)	0	000
E0 (00)	1	E1 (01)	0	001
E1 (01)	0	E2 (10)	1	010
E1 (01)	1	E0 (00)	1	011
E2 (10)	0	E1 (01)	1	100
E2 (10)	1	E1 (01)	1	101

b) **[5 %]** Dado el grafo de estados siguiente:



¿Cuántos bits de entrada tiene el circuito que implementa este grafo?

Cada transición está etiquetada como e_1e_0, lo cual implica que hay **2 bits** de entrada.

¿Cuántos bits de salida?

Cada estado tiene un valor de salida etiquetado como $s_2s_1s_0$, lo cual significa que dicho valor es de **3 bits**.

¿Cuál será el número mínimo de biestables para implementarlo?

El grafo presenta 11 estados distintos. Para codificarlos, realizamos la siguiente operación para hallar el número de biestables necesarios para implementar el circuito:

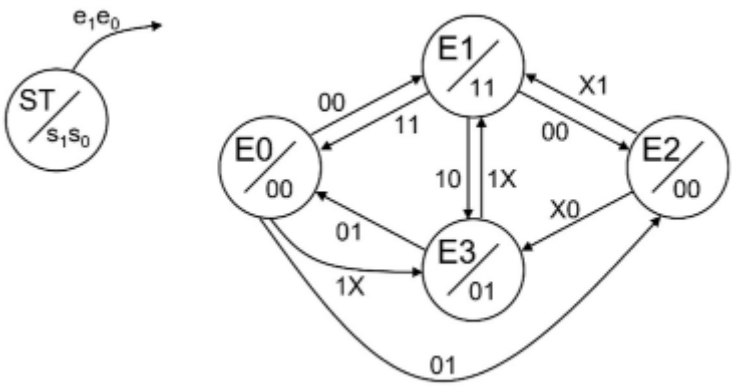
$$número\ de\ biestables = \lceil \log_2(11) \rceil = 4$$

Si lo implementamos utilizando una memoria ROM, ¿cuántos bits de direcciones y cuántos bits de datos necesitará esta memoria?

Las direcciones en la ROM corresponden a todas las combinaciones posibles del estado actual (4 bits) y de la entrada actual (2 bits). Por lo tanto, el total de bits de dirección es de 4 bits del estado + 2 bits de la entrada = **6 bits de dirección**.

En cuanto a los bits de datos, en cada dirección se debe almacenar el próximo estado (4 bits, porque hay 11 estados codificados en 4 bits) y la salida del sistema (3 bits). Como consecuencia, el total de bits de datos que necesitará la RAM es de 4 bits del próximo estado + 3 bits de la salida = **7 bits de datos**.

Dado el grafo de estados siguiente:



Se pide:

c) **[10 %]** Escribid la tabla de transiciones y la tabla de salidas del sistema representado por el grafo, codificando los estados según su índice asociado. Al escribir la tabla de transiciones, poned en primer lugar las variables que codifican el estado y a continuación las variables de entrada.

NOTA: Tenéis disponible el ejercicio a VerilChart (`20242_PAC3_2c_vch`). Para poder probar este ejercicio en VerilChart tenéis que sustituir, si fuera el caso, los bits *don't care* por valores 0.

A partir del grafo planteado en el ejercicio, realizamos la siguiente tabla con la codificación de estados:

ESTADO	q_1	q_0
E0	0	0
E1	0	1
E2	1	0
E3	1	1

Una vez tenemos la codificación de estados, procedemos a rellenar la tabla de transiciones y salidas:

Tabla de verdad

q1	q0	e1	e0	q1_plus	q0_plus	s1	s0
0	0	0	0	0	1	0	0
0	0	0	1	1	0	0	0
0	0	1	0	1	1	0	0
0	0	1	1	1	1	0	0
0	1	0	0	1	0	1	1
0	1	0	1	0	0	1	1
0	1	1	0	1	1	1	1
0	1	1	1	0	0	1	1
1	0	0	0	1	1	0	0
1	0	0	1	0	1	0	0
1	0	1	0	1	1	0	0
1	0	1	1	0	1	0	0
1	1	0	0	0	0	0	1
1	1	0	1	0	0	0	1
1	1	1	0	0	1	0	1
1	1	1	1	0	1	0	1

Captura de la tabla de transiciones y salidas realizado en VerilChart

d) **[5 %]** Diseñad el circuito que implementa el sistema representado por el grafo de estados utilizando una memoria ROM. Mostrad el contenido completo de la memoria ROM en hexadecimal.

NOTA: Tenéis disponible el ejercicio en VerilCirc (`20242_PAC3_2d_circ`). Para poder probar este ejercicio en VerilCirc tenéis que sustituir, si fuera el caso, los bits *don't care* por valores 0.

Las variables que codifican los estados q_i se guardan en **biestables**. En este caso, como tenemos 2 variables que codifican los estados (q_1 q_0), cada uno representará un biestable, por lo que el circuito tendrá **2**.

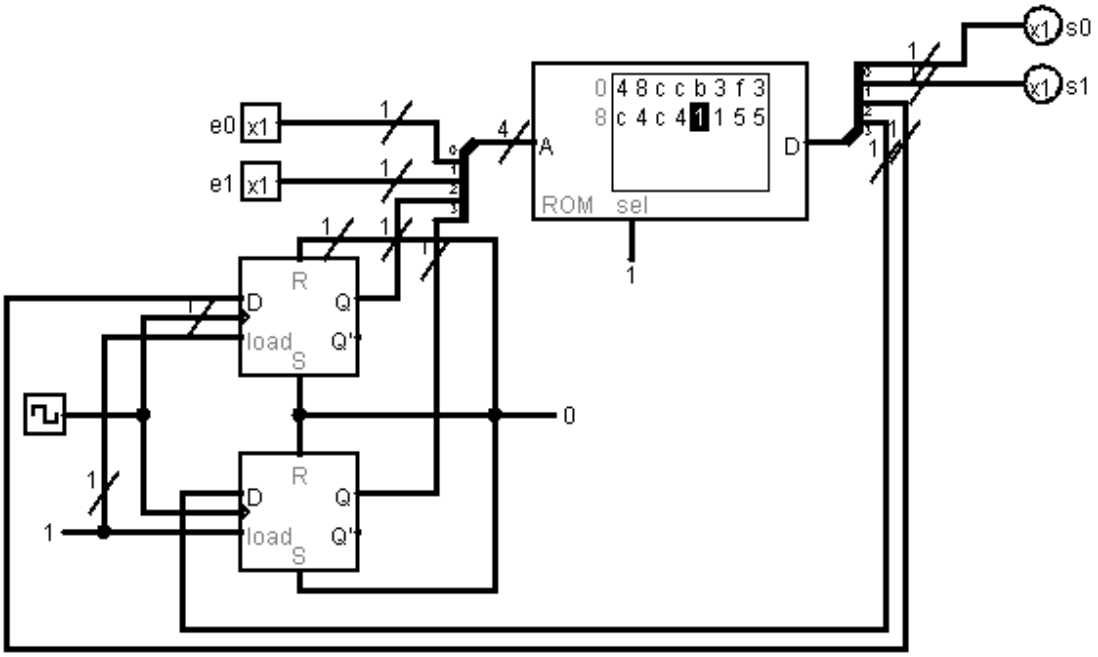
Las columnas q_1 y q_0 indican los valores de entrada (`D`) que deben tomar los biestables en el siguiente flanco de reloj. La siguiente **tabla de excitaciones** muestra estos valores (`d1 d0`), los cuales coinciden con los valores de `q1_plus` y `q0_plus`, respectivamente:

d1	d0	s_1	s_0
0	1	0	0
1	0	0	0
1	1	0	0
1	1	0	0
1	0	1	1
0	0	1	1
1	1	1	1
0	0	1	1
1	1	0	0
0	1	0	0
1	1	0	0
0	1	0	0
0	0	0	1
0	0	0	1
0	1	0	1
0	1	0	1

A partir de estos valores, podemos elaborar la siguiente tabla que representa el contenido de la memoria ROM:

DIRECCIÓN	d1	d0	s_1	s_0
0	0	1	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	0	0
4	1	0	1	1
5	0	0	1	1
6	1	1	1	1
7	0	0	1	1
8	1	1	0	0
9	0	1	0	0
10	1	1	0	0
11	0	1	0	0
12	0	0	0	1
13	0	0	0	1
14	0	1	0	1
15	0	1	0	1

A partir de los detalles dados anteriormente, procedemos a diseñar el circuito:

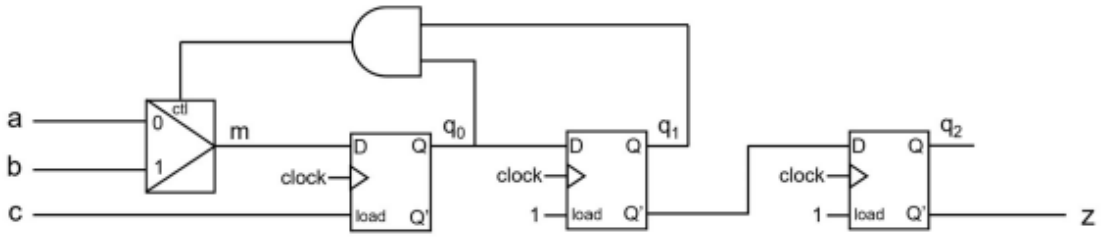


Captura de la tabla de transiciones y salidas realizado en VerilChart

Ejercicio 3 [25 %]

(Sección 2.3: Entradas asíncronas)

Dado el circuito siguiente:



a) **[10 %]** Analizad el circuito y rellenad la tabla siguiente por los valores de entrada y estados especificados:

El circuito sincroniza y controla el paso de datos a través de biestables D. Está formado por los siguientes componentes:

1. **Multiplexor:**

   La señal del control está determinada por el valor de la salida de la puerta lógica AND.

   - Si `c = 0`, la salida del multiplexor es el valor de la entrada `a`.
   - Si `c = 1`, la salida del multiplexor es el valor de la entrada `b`.

2. **Biestables D:**

   En total hay **tres** biestables D conectados en serie. Cada uno de ellos se activa por un reloj. Las características de cada biestable son las siguientes:

   - **Primer biestable (0):**
     - El valor de la entrada `load` se determina por el valor de la entrada `c`.
     - La entrada d_0 toma como valor la salida del multiplexor (`m`).
   - **Segundo biestable (1):**
     - El valor de la entrada `load` es una constante (`1`).
     - La entrada d_1 toma como valor la salida del primer biestable (q_0).
   - **Tercer biestable (2):**
     - El valor de la entrada `load` es una constante (`1`).
     - La entrada d_2 toma como valor la salida negada del segundo biestable (q_1').
     - La salida `z` es la negación del valor de entrada (d_2).

3. **Puerta lógica AND:**

   Toma como entradas q_0 (salida del primer biestable) y q_1 (salida del segundo biestable). Su salida se conecta a la señal de control del multiplexor.

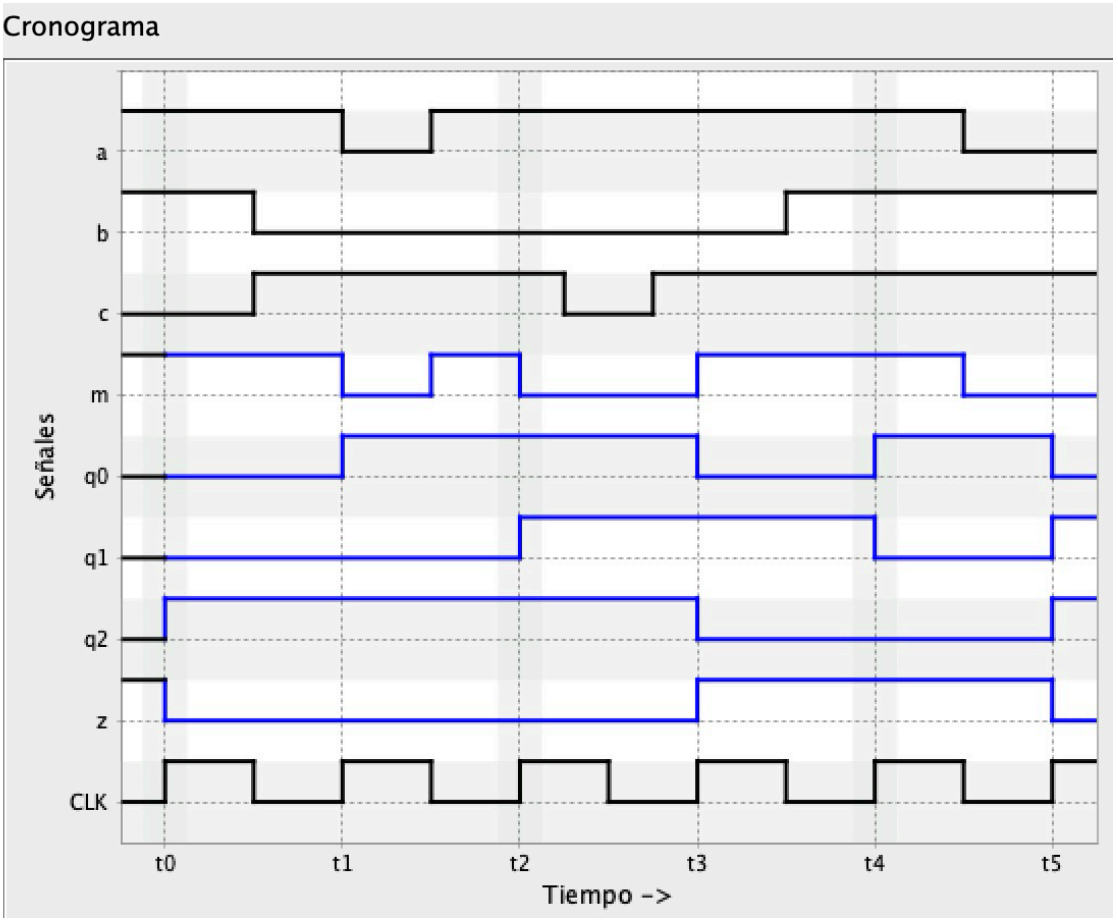
Ahora que sabemos el funcionamiento del circuito, procedemos a rellenar la tabla propuesta en el ejercicio:

q_2	q_1	q_0	a	b	c	d_2	d_1	d_0	z
0	0	1	0	1	1	1	1	0	1
0	1	0	1	0	1	0	0	1	1
0	1	1	0	1	1	0	1	1	1
1	0	0	0	1	1	1	0	0	0
1	1	0	1	1	0	0	0	1	0
1	1	1	1	0	1	0	1	0	0

b) **[15 %]** Completad el cronograma siguiente:

NOTA: Tenéis disponible este ejercicio en VerilChart (`20242_PAC3_3b_vch`).

Dada la descripción detallada del circuito en el apartado anterior, procedemos a completar el cronograma propuesto. Cabe destacar que el valor de `m` **NO** depende del reloj, ya que corresponde a la salida del multiplexor (bloque **combinacional**), cuyo valor cambia **instantáneamente**. Esto contrasta con el valor de las salidas de los biestables (q_0, q_1, q_2, z), que sí se actualizan en función de los flancos ascendentes de los ciclos de reloj.

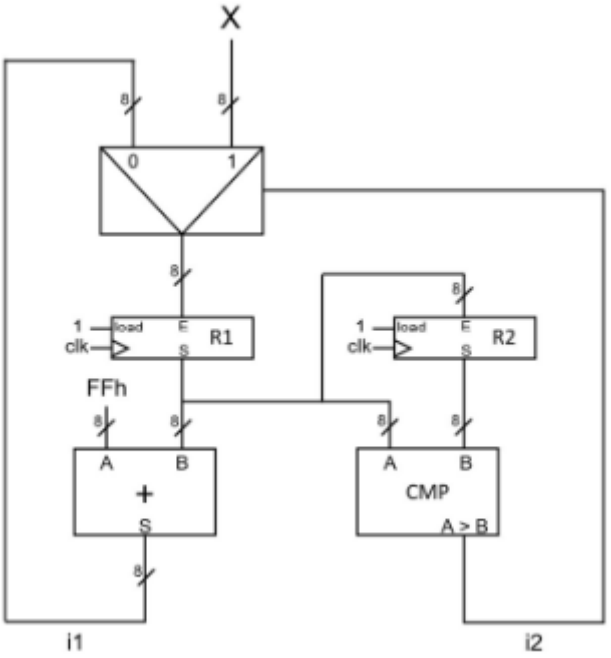


Captura del cronograma hecho en VerilChart

Ejercicio 4 [25 %]

(Sección 3.1. Registro)

Dado el circuito siguiente:



Completad el cronograma siguiente, poniendo los valores de los registros en hexadecimal, e incluyendo los valores intermedios i_1 y i_2.

NOTA: Tenéis disponible este ejercicio en VerilChart (`20242_PAC3_4_vch`).

Este circuito representa una estructura de procesamiento secuencial con dos registros, un sumador, un comparador y un multiplexor. Para realizar el cronograma, vamos a desglosar el circuito según los bloques que lo componen y sus conexiones:

1. **Multiplexor de 8 bits:**

   Bloque **combinacional** en el que la señal de control está determinada por el valor de la señal `i2`.

   - Si `i2 = 0`, la salida del multiplexor es el valor de la señal `i1`.
   - Si `i2 = 1`, la salida del multiplexor es el valor de la entrada `X`.

2. **Registros de 8 bits:**

   En total hay **dos** registros. Cada uno de ellos se activa mediante un reloj y el valor de la entrada `load` es siempre 1. Hay que tener en cuenta que los registros son bloques **secuenciales** que dependen del flanco ascendente del reloj.

Las características de cada registro son las siguientes:

- **Primer registro** (`R1`)**:** El valor de la entrada se determina por el valor de la salida del multiplexor.
- **Segundo registro** (`R2`)**:** El valor de la entrada se determina por el valor de la salida de `R1`.

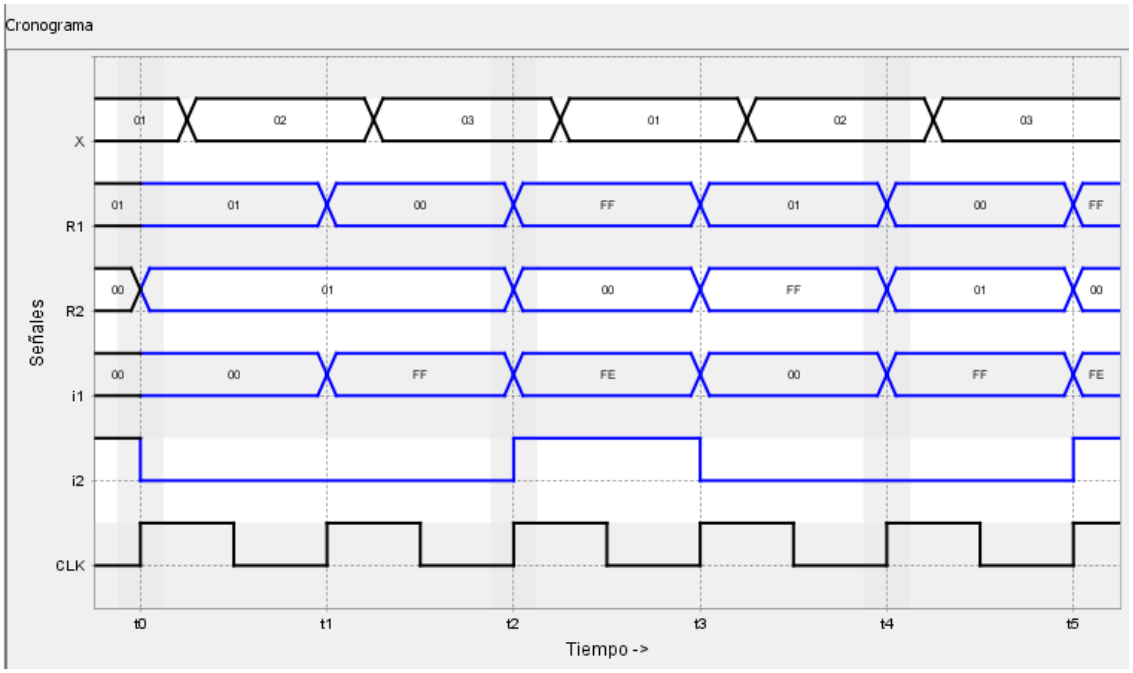
3. Sumador de 8 bits:

- Bloque **combinacional**.
- **Entrada A**: Constante `FF` (valor hexadecimal).
- **Entrada B**: Valor de salida de `R1`.
- **Salida**: Los 8 bits **menos significativos** de la suma entre A y B, que se cargan en la señal `i1` y representan la entrada 0 del multiplexor.

4. Comparador de 8 bits:

- Bloque **combinacional**.
- **Entrada A**: Valor de salida de `R1`.
- **Entrada B**: Valor de salida de `R2`.
- **Salida**: `i2`, la cual es la señal de control del multiplexor.
  - `i2 = 0` si `A <= B`
  - `i2 = 1` si `A > B`

Una vez tenemos este desglose, procedemos a completar el cronograma:



Captura del cronograma hecho en VerilChart

Con el fin de esclarecer los valores del sumador, lo cual puede resultar la parte más compleja de este ejercicio, realizamos un desglose de los valores de `i1` para cada intervalo de tiempo. Es importante tener en cuenta que el sumador es un bloque **combinacional** que no depende de los ciclos de reloj.

- `[t0, t1]: 01 + FF = 1`**`00`**
- `[t1, t2]: 00 + FF =` **`FF`**
- `[t2, t3]: FF + FF = 1`**`FE`**
- `[t3, t4]: 01 + FF = 1`**`00`**
- `[t4, t5]: 00 + FF =` **`FF`**
- `[t5, t6]: FF + FF =` **`FE`**