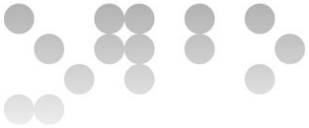


PAC3 - Tercera prueba de evaluación continuada

Presentación

Esta PEC se focaliza en los circuitos secuenciales. Los circuitos combinacionales nos permiten describir funcionalidades de un circuito, pero no nos permite guardar información. Mediante biestables y registros podemos guardar información en memoria y hacer circuitos más complejos. En este PEC practicaremos con este tipo de circuitos.

Competencias

- Entender el funcionamiento de los circuitos lógicos secuenciales y conocer y saber aplicar técnicas de diseño de sistemas secuenciales.

Objetivos

- Saber discernir, a partir de la funcionalidad que se quiere que tenga un circuito lógico, si el circuito tiene que ser de tipo secuencial o combinacional.
- Conocer el funcionamiento del biestable D y de todas las entradas de control que puede tener.
- Saber analizar un circuito secuencial.
- Saber realizar un cronograma a partir d´un circuito digital secuencial.
- Saber analizar un grafo de estados.
- Saber diseñar un circuito cualquiera a partir de la descripción de su funcionalidad mediante el modelo de Moore.

Recursos

Los recursos que se recomienda usar por esta PEC son los siguientes:

- **Básicos**: El módulo 4 de los materiales. En cada pregunta, se indica en qué sección de los materiales se puede encontrar la información para resolverlos. Notáis que también existe en los materiales una cantidad muy extensa de ejercicios para ver como se pueden resolver.
- **PEC anteriores**: En el aula de CANVAS, en recursos adicionales podéis encontrar PACS resueltas otros semestres.
- **Complementarios**: VerilCIRC, VerilCHART y el Wiki de la asignatura.

Criterios de valoración

- **Razonáis la respuesta en todos los ejercicios. Las respuestas sin justificación no recibirán puntuación.**
- **Los ejercicios realizados con IA generativa no recibirán puntuación.**
- **La valoración está indicada en cada uno de los apartados y subapartados.**

Uso de herramientas de IA

En esta actividad no está permitido el uso de herramientas de inteligencia artificial. Al plan docente y al web sobre integridad académica y plagio de la UOC encontraréis información sobre que se considera conducta irregular en la evaluación y las consecuencias que puede tener.

Formato y fecha de entrega

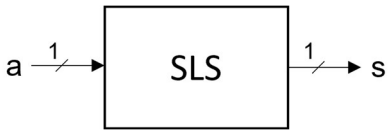
- Para dudas y aclaraciones sobre el enunciado, dirigíos al consultor responsable de vuestra aula.
- Hay que librar la solución en un fichero PDF usando una de las plantillas libradas conjuntamente con este enunciado.
- Se tiene que librar a través de la aplicación de **Entrega de la Actividad** correspondiendo del apartado **Contenidos** de vuestra aula.
- La fecha tope de entrega es lo **30 de abril** (a las 24 horas).

Descripción de la PEC a realizar

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 debe 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 múltiple de tres.

Cuando se detecta que el valor es múltiple 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 múltiple de cualquier número y, por lo tanto, es múltiple 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:

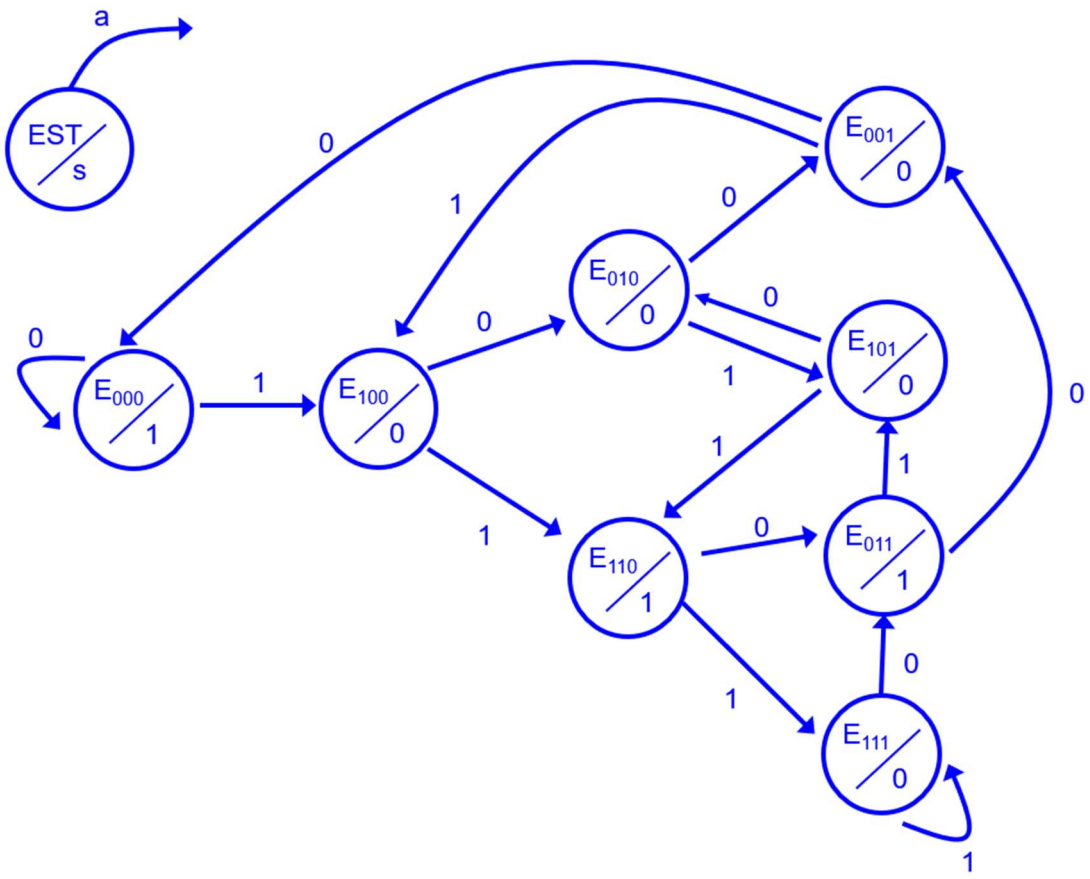
Entrada *a*	0	1	0	0	0	1	1	1	0	1	0	1	...
Salida *s*	1	1	0	0	0	1	0	1	0	1	0	0	0

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

Siguiendo la codificación descrita en el enunciado, a_i a_{i-1} a_{i-2}, para conseguir el funcionamiento deseado, el circuito debe tener los siguientes estados:

Estado	Descripción	Salida
E_{000}	Los últimos 3 ciclos han sido 000, es múltiple de 3	1
E_{100}	Los últimos 3 ciclos han sido 100, no es múltiple de 3	0
E_{010}	Los últimos 3 ciclos han sido 010, no es múltiple de 3	0
E_{110}	Los últimos 3 ciclos han sido 110, es múltiple de 3	1
E_{001}	Los últimos 3 ciclos han sido 001, no es múltiple de 3	0
E_{101}	Los últimos 3 ciclos han sido 101, no es múltiple de 3	0
E_{011}	Los últimos 3 ciclos han sido 011, es múltiple de 3	1
E_{111}	Los últimos 3 ciclos han sido 111, no es múltiple de 3	0

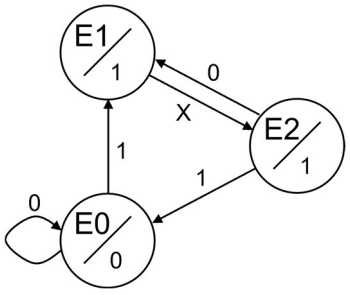
El grafo representa el comportamiento deseado:



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 debe tener la memoria ROM, de cuántos bits debe ser cada una, 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*	*010101*
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 si la solución es correcta y, en caso de error, indicad qué errores aparecen en la solución propuesta.

La solución propuesta por la IA es incorrecta por varias razones. Por un lado, vemos un error en el paso 4, ya que al implementar un circuito secuencial con una memoria ROM, la ROM debe tener una palabra para cada estado y cada posible combinación de variables de entrada. Es decir, una palabra por cada combinación de las señales (en este caso 2 bits para el estado y un bit de entrada). Por lo tanto, estas son las señales que llegarán a la entrada de direcciones de la ROM, que será de 3 bits. Por lo tanto, la ROM tendrá 2^3= 8 palabras.

Por otro lado, también vemos un error en el paso 5, el número de bits de cada palabra de la ROM es de 3 bits, dos para el estado futuro y uno para la salida, pero cuando define la codificación utiliza 6, con la dirección y la salida todo junto. Además, a pesar de que la tercera fila no dependa de la entrada, habrá que almacenarlo en la ROM.

A pesar de que el ejercicio no lo pide y no es necesario hacerlo, a continuación, vamos a resolver el ejercicio planteado de manera correcta:

Vemos que el circuito tiene 1 bit de entrada y un 1 bit de salida. Además, como que el circuito tiene 3 estados, necesitaremos 2 bits para representarlos, y por tanto necesitamos 2 biestables para poder almacenar estos 2 bits.

La memoria ROM que puede implementar este circuito tendrá 3 bits de dirección, 2 bits para el estado más 1 bit para la entrada La dimensión de las palabras de la ROM tendrá que ser de 3 bits, 1 bit para la salida y 2 bits para codificar el estado futuro. Por lo tanto, la ROM será de dimensiones 2^3 x 3

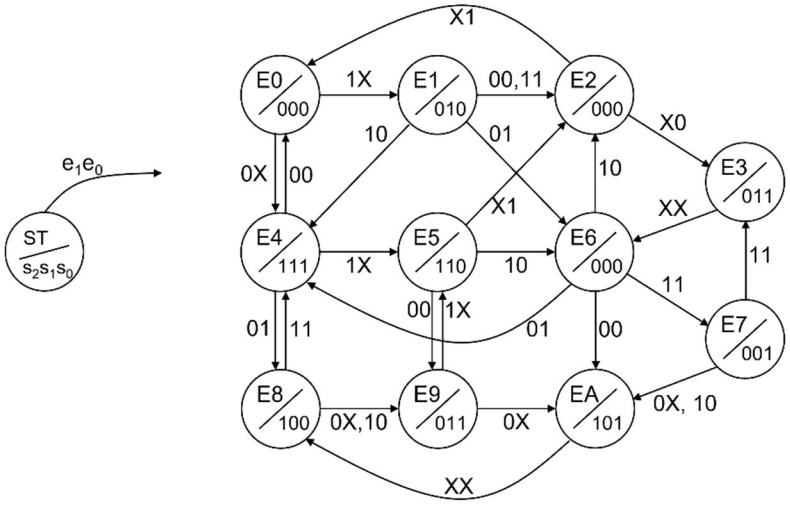
El contenido se determina a partir de la tabla de transiciones y salidas:

q_1	q_0	e	q_1+	q_0+	s
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	1
0	1	1	1	0	1
1	0	0	0	1	1
1	0	1	0	0	1
1	1	0	x	x	x
1	1	1	x	x	x

Y queda de la siguiente forma donde indicamos con el valor 0h los valores *don't care*:

@	hex
0	0h
1	2h
2	5h
3	5h
4	3h
5	1h
6	0h
7	0h

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

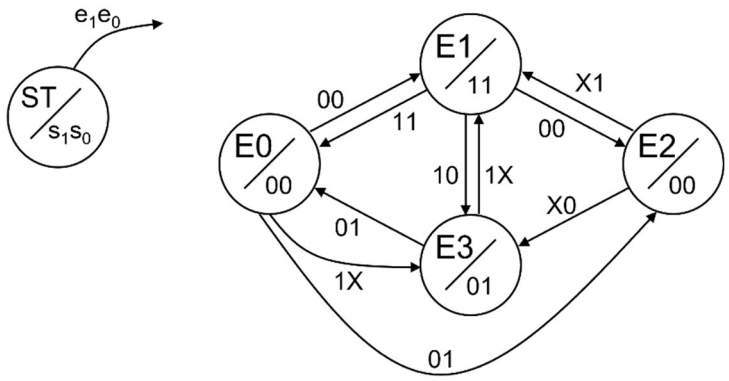


¿Cuántos bits de entrada tiene el circuito que implementa este grafo? ¿Cuántos bits de salida? ¿Cuál será el número mínimo de biestables para implementarlo? Si lo implementamos utilizando una memoria ROM, ¿cuántos bits de direcciones y cuántos bits de datos necesitará esta memoria?

Vemos que el circuito tiene 2 bits de entrada, e_1 y e_0, y 3 bits de salida, s_2, s_1 y s_0. Además, como que el circuito tiene 11 estados, $\log_2(11)\approx 3.46$, necesitaremos 4 bits para representarlos, y por tanto, necesitamos 4 biestables para poder almacenar estos 4 bits.

La memoria ROM que puede implementar este circuito tendrá 6 bits de dirección, 4 bits para el estado y 2 bits para la entrada. La dimensión de las palabras de la ROM será de 7 bits, 3 bit para la salida y 4 bits para codificar el estado futuro.

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. Para poder probar este ejercicio en VerilCHART tenéis que sustituir, si fuera el caso, los bits *don't care* por valores 0.

Dado que el grafo tiene 4 estados, harán falta 2 variables para codificarlos. Tal como es habitual, las denominamos q_1 y q_0 y definimos la codificación según su índice asociado tal como pide el enunciado.

Para obtener la tabla de transiciones observamos en el grafo a qué estado se llega desde cada estado con cada valor posible de las variables de entrada, e_1 y e_0.

q_1	q_0	e_1	e_0	q_1+	q_0+
0	0	0	0	0	1
0	0	0	1	1	0
0	0	1	0	1	1
0	0	1	1	1	1
0	1	0	0	1	0
0	1	0	1	X	X
0	1	1	0	1	1
0	1	1	1	0	0
1	0	0	0	1	1

1	0	0	1	0	1
1	0	1	0	1	1
1	0	1	1	0	1
1	1	0	0	X	X
1	1	0	1	0	0
1	1	1	0	0	1
1	1	1	1	0	1

Para rellenar la tabla de salidas solo hay que escribir el valor que tienen las variables de salida en cada estado.

q_1	q_0	s_1	s_0
0	0	0	0
0	1	1	1
1	0	0	0
1	1	0	1

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. Para poder probar este ejercicio en VerilCIRC tenéis que sustituir, si fuera el caso, los bits *don't care* por valores 0.

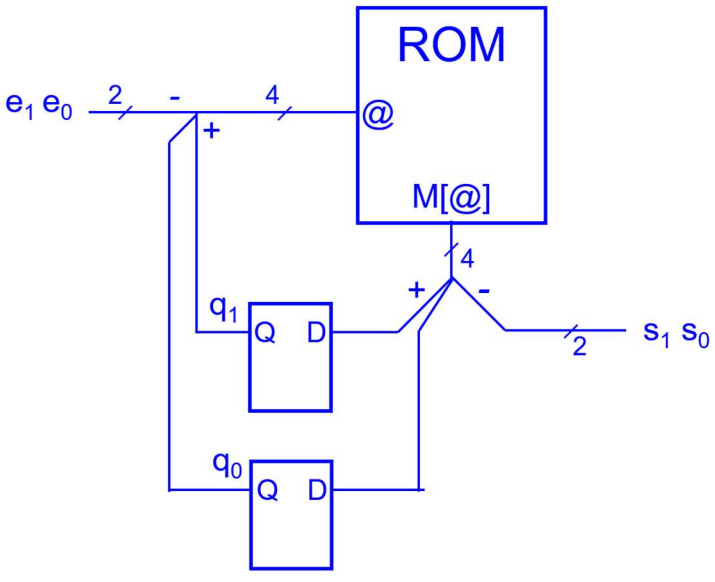
El circuito tendrá una memoria ROM con una entrada de direcciones de 4 bits (dos bits para codificar el estado y dos bits de entrada al circuito), y, por tanto, 16 palabras, que contendrán el valor del estado futuro (2 bits) y el valor de las 2 variables de salida a cada estado. Es decir, las palabras serán de 4 bits.

A continuación, se muestra el contenido de la ROM, donde los valores don't care se han sustituido por valores 0.

q_1	q_0	e_1	e_0	q_1+	q_0+	s_1	s_0	Hex
0	0	0	0	0	1	0	0	4h
0	0	0	1	1	0	0	0	8h
0	0	1	0	1	1	0	0	Ch

0	0	1	1	1	1	0	0	Ch
0	1	0	0	1	0	1	1	Bh
0	1	0	1	0	0	1	1	3h
0	1	1	0	1	1	1	1	Fh
0	1	1	1	0	0	1	1	3h
1	0	0	0	1	1	0	0	Ch
1	0	0	1	0	1	0	0	4h
1	0	1	0	1	1	0	0	Ch
1	0	1	1	0	1	0	0	4h
1	1	0	0	0	0	0	1	1h
1	1	0	1	0	0	0	1	1h
1	1	1	0	0	1	0	1	5h
1	1	1	1	0	1	0	1	5h

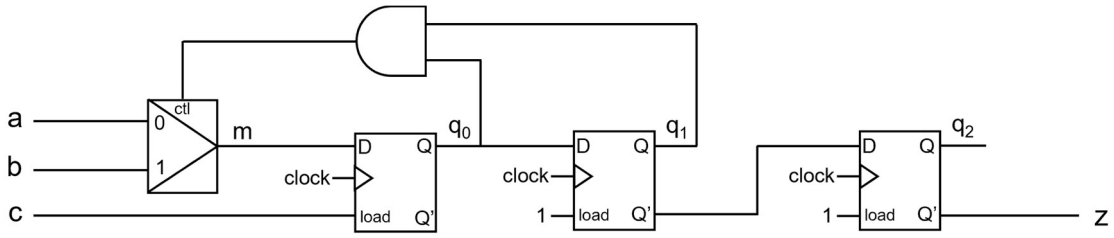
Además, el circuito tendrá 2 biestables que guardarán el estado actual. A continuación, se muestra el circuito completo.



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:

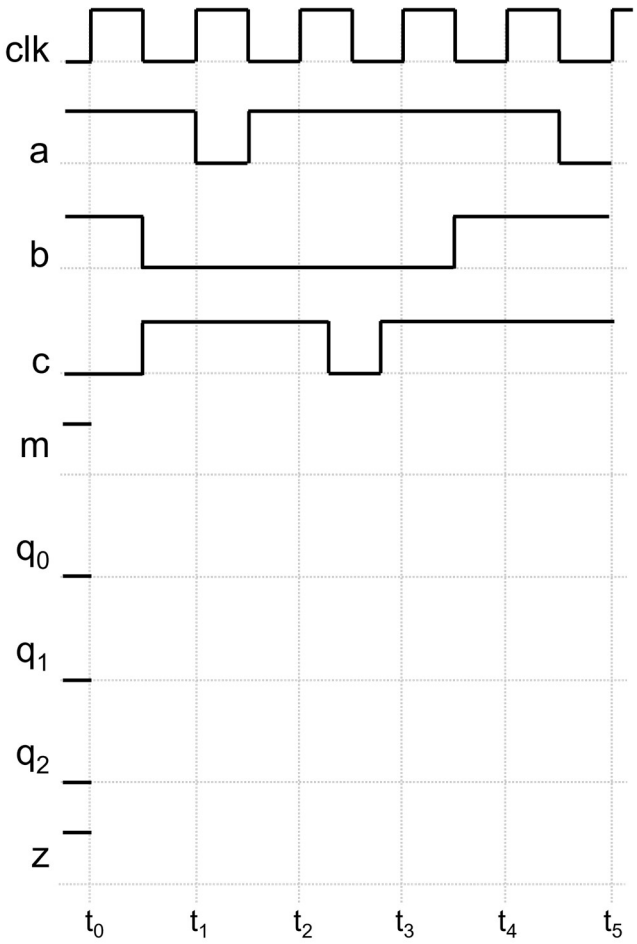
q_2	q_1	q_0	a	b	c	d_2	d_1	d_0	z
0	0	1	0	1	1				
0	1	0	1	0	1				
0	1	1	0	1	1				
1	0	0	0	1	1				
1	1	0	1	1	0				
1	1	1	1	0	1				

Para rellenar la tabla miramos a que corresponden los valores pedidos: vemos que la entrada del biestable d_2 corresponde a la salida negada q'_1, puesto que *load* está siempre a 1. La entrada del biestable d_1 corresponde a la salida q_0. La entrada del biestable d_0 corresponde a *m*, siempre que *load* esté a 1, el cual viene controlado por la entrada *c*. Para calcular *m* vemos que corresponde a la salida de un multiplexor con las entradas *a* y *b,* y controlado por *ctl* que viene dado por la salida de la puerta AND y será $q_0 \cdot q_1$. Finalmente, la salida *z* corresponde a la salida negada del biestable q'_2.

Por lo tanto, la tabla quedará de la siguiente manera,

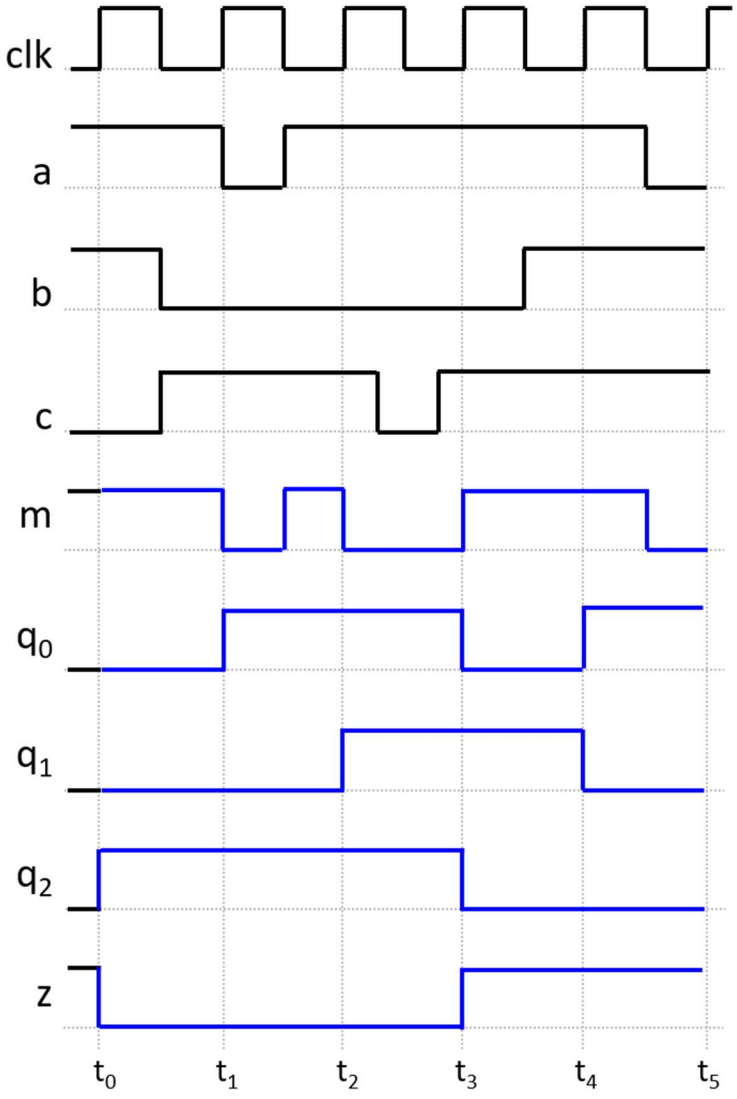
q_2	q_1	q_0	a	b	c	d_2	d_1	d_0	z
0	0	1	0	1	1	1	1	0	1
0	1	0	1	0	1	0	0	1	1
0	1	1	0	1	1	0	1	1	1
1	0	0	0	1	1	1	0	0	0
1	1	0	1	1	0	0	0	1	0
1	1	1	1	0	1	0	1	0	0

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



Nota: Tenéis disponible este ejercicio en VerilCHART.

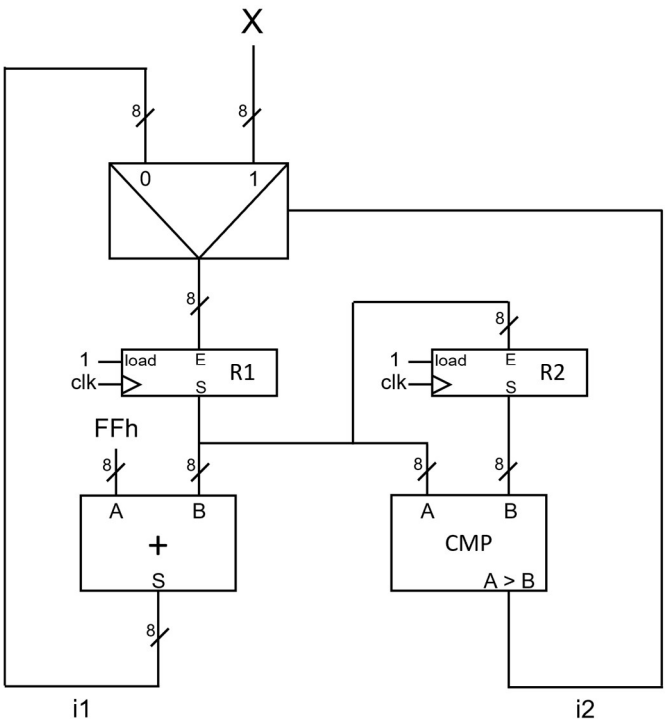
Siguiendo la explicación del apartado anterior podemos completar el cronograma, teniendo en cuenta a que corresponden los valores pedidos.


clk
a
b
c
m
q_0
q_1
q_2
z
t_0
t_1
t_2
t_3
t_4
t_5

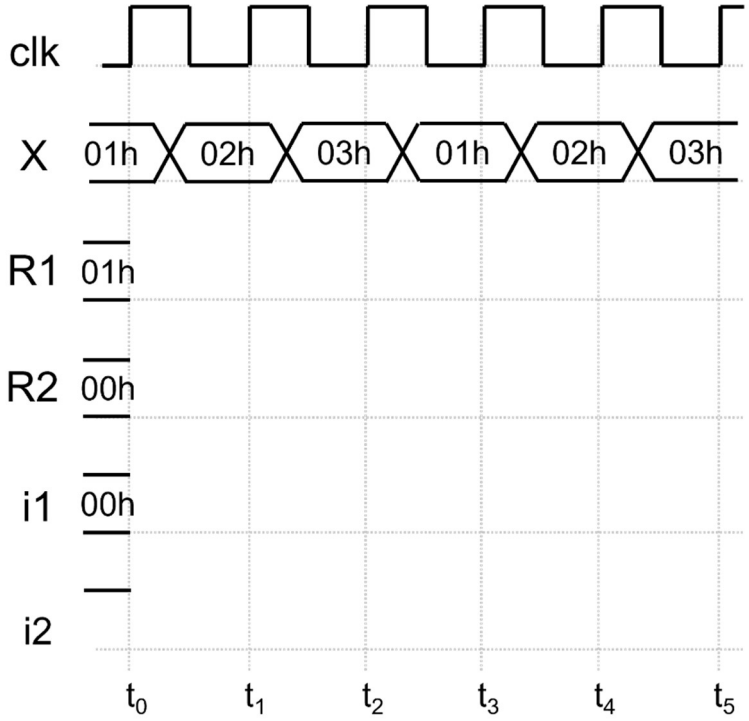
Ejercicio 4 [25 %]

(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 *i1* y *i2*.



Nota: Tenéis disponible este ejercicio en VerilCHART.

Para rellenar el cronograma miramos en qué momentos se cargan los registros y con qué valores.

En cuanto a los momentos, vemos que se cargan en cada flanco de reloj, porque a la entrada *load* de todos los registros hay conectado un 1.

En cuanto a los valores que se cargan, vemos que en la entrada de *R1* llega la salida de un multiplexor. En la entrada del MUX hay la entrada *X* y la salida de un sumatorio, *i1*. La entrada de control es la salida de un comparador, *i2*.

En *R2* se carga a cada flanco la salida *R1*.

La señal *i1*, es la salida del sumatorio que hace la suma de FFh con la salida del mismo registro *R1*. Por lo tanto, i1=FFh+R1

La señal *i2*, es la salida *A>B* de un comparador, al cual llegan los valores *R1* y *R2*. Por lo tanto, i2=(R1>R2).

Ahora ya podemos escribir todos los valores en el cronograma. Veamos qué pasa en el instante t_0. En el primer instante, i2=1, por lo tanto, la salida del MUX será *X*, que en este tiempo es 01h. Así pues, *R1* será 01h. *R2* será lo que había en *R1*, por lo tanto, 01h. En cuanto a *i1*, será la suma de FFh + 01h, es decir, 00h. Por último, *i2* será 0 puesto que no se cumple que R1>R2.

En el instante t_1, i2=0, por lo tanto, en la salida del MUX tendremos *i1*, esto quiere decir que en *R1* se cargará 00h. En el registro *R2* tendremos 01h. En cuanto a *i2* será la suma de FFh+00h, es decir FFh, y *i2* será 0 puesto que no se cumple que R1>R2.

Razonando de manera análoga rellenamos todo el cronograma, que queda tal como se muestra a continuación.