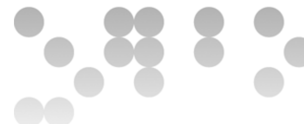


Práctica

Presentación

Por fin llegamos al final del curso. Esta práctica servirá para sintetizar todos los conocimientos del curso y ampliarlos con el diseño de circuitos más complejos. En este trabajo se os pedirá el diseño de un circuito a partir de un grafo de estados y el diseño de una máquina de estados en concreto. En la primera parte tendréis que aplicar los conocimientos que habéis adquirido en este curso, como, por ejemplo, sobre los mapas de Karnaugh o los sistemas de representación. Por lo tanto, se recomienda que repaséis los anteriores módulos de los materiales.

Competencias

— Conocer la organización general de un computador como circuito digital, y

— Conocer la arquitectura de Von Neumann.

Objetivos

— Conocer varios modelos de las máquinas de estados y de las arquitecturas de controlador con camino de datos.

— Haber adquirido una experiencia básica en la elección del modelo de máquina de estados más adecuado para la resolución de un problema concreto.

— Ser capaz de diseñar circuitos secuenciales a partir de grafos de transiciones de estados.

Recursos

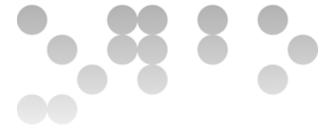
Los recursos que se recomienda utilizar para esta PEC son los siguientes:

Básicos:	El módulo 5 de los materiales.
Complementarios:	VerilCIRC, VerilCHART y el Wiki de la asignatura.

Criterios de valoración

— **Razonad la respuesta en todos los ejercicios: Las respuestas sin justificación no recibirán puntuación.**

— **La valoración está indicada en cada uno de los subapartados.**



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 y registro de EC** del apartado Evaluación de vuestra aula.

— La fecha límite de entrega es el **21 de diciembre** (a las 24 horas).

Solución

PRIMERA PARTE [60%]

Uno de los módulos de un controlador de un ascensor sigue el comportamiento que describe el EFSM siguiente:

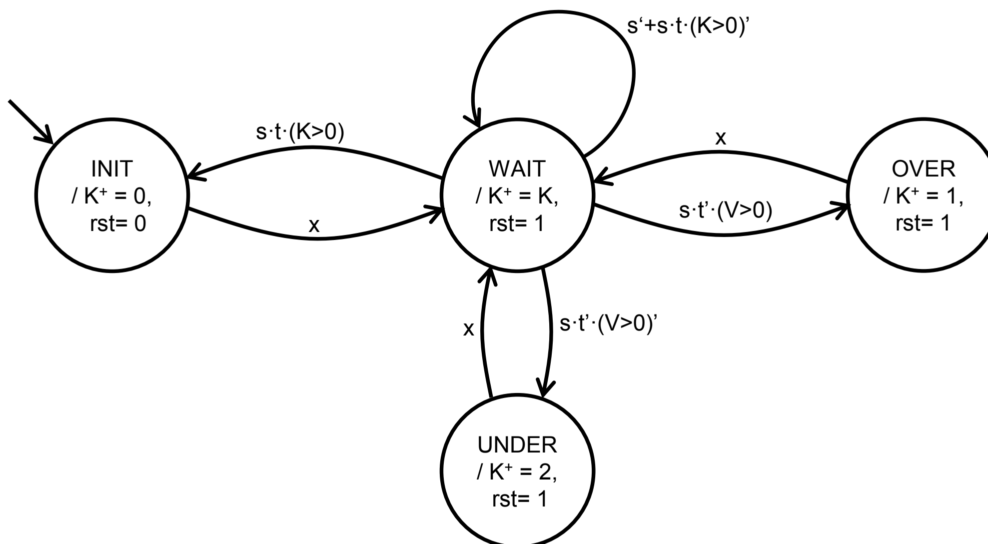
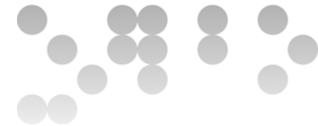


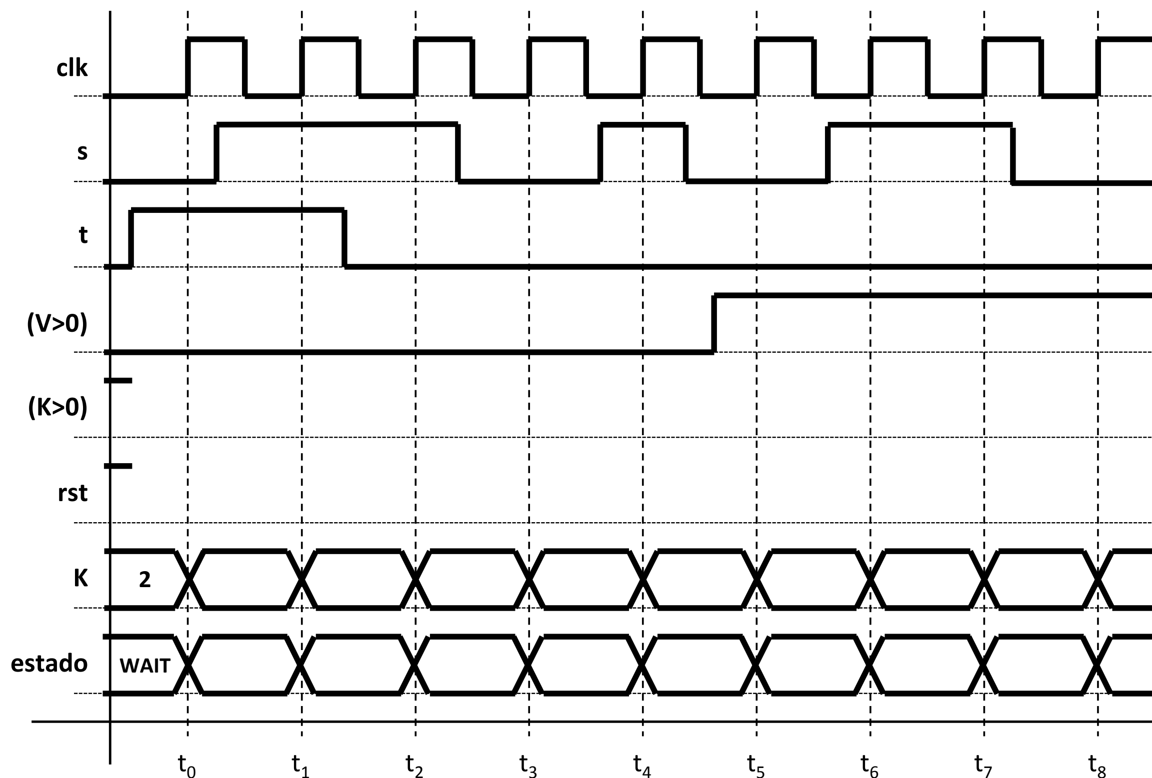
Fig. 1. EFSM del módulo de control que se ha elegido.

Se pide que:

a) [15%] A partir del grafo de la EFSM de la fig. 1 completéis, en el cronograma siguiente, la evolución de las señales (*K*>0) y *rst*, la evolución de los valores que toma *K*, y el estado en que se encuentra la máquina en cada ciclo de reloj.

Nota: Tenéis el ejercicio disponible en el VerilChart, donde el valores de la señal *K* se representan en hexadecimal. Tened presente que las señales (*K*>0) y (*V*>0) se corresponden con *K*0 y *V*0 en VerilChart, respectivamente.



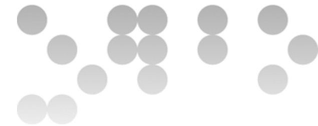


Para completar este cronograma conviene fijarse en que *rst* depende del estado y, por lo tanto, se puede esperar al final para determinar los valores que toma.

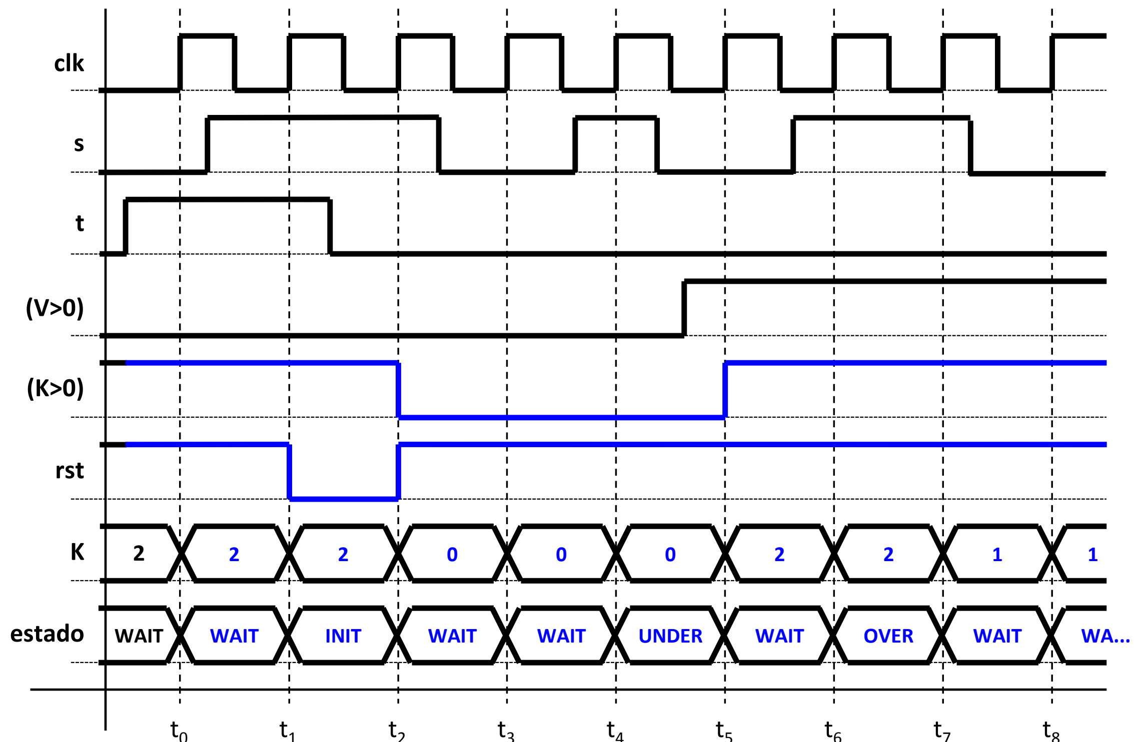
Así pues, periodo por periodo, primero se determinará el valor de (*K*>0) al final del periodo, es decir, hasta t_0 por la izquierda en el primer periodo, hasta t_1 en el segundo y continuar así hasta t_8. Una vez calculado el valor de (*K*>0) al final de cada periodo, se puede determinar cuál será el estado para el periodo siguiente, y también el valor de *K*.

Por ejemplo, al final del primer periodo, (*K*>0) es 1 y las señales *s*, *t* y (*V*>0) son 0, 1 y 0, respectivamente. Con esto, el estado siguiente continuará siendo WAIT. Y, en el periodo siguiente, *K* valdrá 2 porque en WAIT se indica que el valor de K para el periodo siguiente, K^+, es *K*.

Habiendo determinado los valores de (*K*>0), el estado y *K* para todos los periodos, el valor de la señal *rst* se puede deducir directamente a partir del valor que se le asigna a cada sido porque es una señal que depende, combinacionalmente, del estado actual.



Siguiendo este método, el cronograma queda de la manera siguiente:



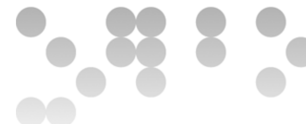
b) [15%] Obtened las tablas de transiciones y de salidas de la EFSM de la fig. 1, así como las codificaciones binarias de los estados y las salidas correspondientes.

Del grafo y del cronograma se observa que hay 4 señales de entrada a la FSM de control: *s*, *t*, (*V*>0) y (*K*>0), y salidas para determinar los valores de: *K* y *rst*. Así, las tablas de transiciones y salidas que se obtienen a partir del grafo son:

Estado actual	Entradas	Estado siguiente
INIT	X	WAIT
WAIT	s' + s·t·(K>0)'	WAIT
WAIT	s·t·(K>0)	INIT
WAIT	s·t'·(V>0)'	UNDER
WAIT	s·t'·(V>0)	OVER
OVER	X	WAIT
UNDER	X	WAIT

Estado	Salidas
INIT	K^+= 0, rst = 0
WAIT	K^+= K, rst = 1
OVER	K^+= 1, rst = 1
UNDER	K^+= 2, rst = 1

Para la codificación binaria, hay que asignar a cada estado un código binario individual. En este caso, se sigue el criterio más habitual, que es el de la numeración binaria, desde el 0 para el estado inicial hasta el número de estados que haya menos 1: 00 por INIT, 01 por WAIT, 10 por OVER y 11 por UNDER.



En el caso de la señal *rst* la codificación binaria la da la misma descripción de la EFSM.

Para la señal *K* hace falta un selector para elegir el valor siguiente de la variable correspondiente: *s_K*. Este selector tiene que elegir entre 0, 1, 2 y mantener el valor, es decir, tiene que ser de dos bits. Así pues, la codificación binaria del selector de operación puede ser la siguiente:

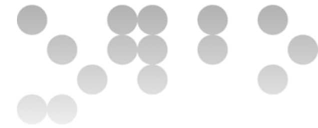
Operación	Selector s_K	Efecto sobre la variable
$K^+ = 0$	00	Ponerla a 0
$K^+ = 1$	01	Ponerla a1
$K^+ = 2$	10	Ponerla a 2
$K^+ = K$	11	Mantener el valor

Finalmente, las tablas de transiciones y de salidas con la codificación binaria correspondiente son:

Estado actual		Entradas				Estado+
Símbolo	q_1q_0	s	t	(V>0)	(K>0)	$q_1^+q_0^+$
INIT	00	x	x	x	x	01
WAIT	01	0	x	x	x	01
WAIT	01	1	1	x	0	01
WAIT	01	1	1	x	1	00
WAIT	01	1	0	0	x	11
WAIT	01	1	0	1	x	10
OVER	10	x	x	x	x	01
UNDER	11	x	x	x	x	01

Estado		Salidas	
Símbolo	q_1q_0	rst	s_K
INIT	00	0	00
WAIT	01	1	11
OVER	10	1	01
UNDER	11	1	10

Hay que tener en cuenta que la codificación binaria del selector *s_K* y de los estados puede ser diferente de la mostrada.



El controlador de un módulo contador específico sigue el comportamiento que describe la EFSM siguiente:

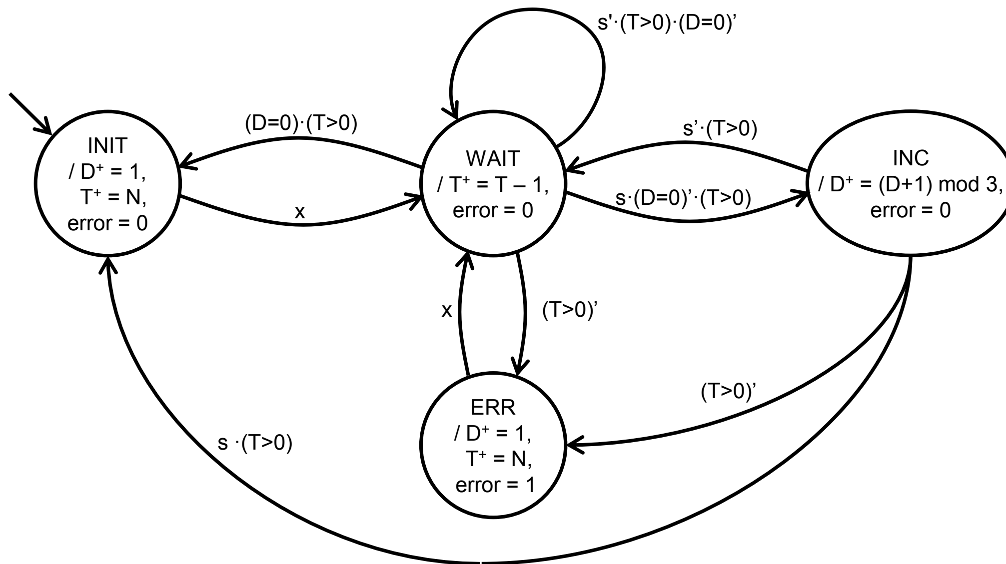


Fig. 2. EFSM del controlador del módulo contador.

Se pide que:

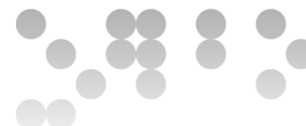
c) [10%] A partir de las tablas de transiciones y de salidas del grafo de la fig. 2 que se dan a continuación, implementéis la unidad de control usando ROM y los registros, los bloques combinacionales y las puertas lógicas que creáis convenientes, considerando que la entrada *N* es un valor dentro del rango [0,15]. Especificad y justificad las dimensiones de la/las ROM que uséis e indicad su contenido.

Estado actual	Entradas			$Estado^+$
	s	(D=0)	(T>0)	
INIT	x	x	x	WAIT
WAIT	x	1	1	INIT
WAIT	0	0	1	WAIT
WAIT	1	0	1	Inc
WAIT	x	x	0	ERR
Inc	1	x	1	INIT
Inc	0	x	1	WAIT
Inc	x	x	0	ERR
ERR	x	x	x	WAIT

Estado		Salidas		
Símbolo	q_1q_0	s_D	s_T	error
INIT	00	00	00	0
WAIT	01	10	01	0
Inc	10	01	10	0
ERR	11	00	00	1

La señal *s_D* y s_T se usan como selectores que eligen cuáles de los resultados se asignan a las variables *D* y *T*, respectivamente.

Nota: Tenéis el ejercicio disponible en VerilUOC, donde la señal (*D*=0) es *D*_0 y la señal (*T*>0) es *T*_0.



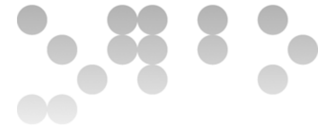
Para implementar la unidad de control con una ROM hace falta, primero, determinar su tamaño y, después, calcular el contenido a partir de la tabla de transiciones correspondiente.

El bus de direcciones de la ROM tiene que ser de 5 bits, puesto que la dirección se forma con los bits que codifican el estado (son 2, en este caso) y los de las señales de entrada (3 más). Por lo tanto, la ROM tiene que disponer de un total de 2^5 = 32 posiciones de memoria.

En cada posición se almacena el código del estado siguiente y el valor, en el estado actual, de cada una de las señales de salida. Por lo tanto, harán falta 2 bits para el estado y 5 bits para las salidas, cosa que hace un total de 7 bits. Así pues, la ROM tiene que ser de 32 posiciones de 7 bits.

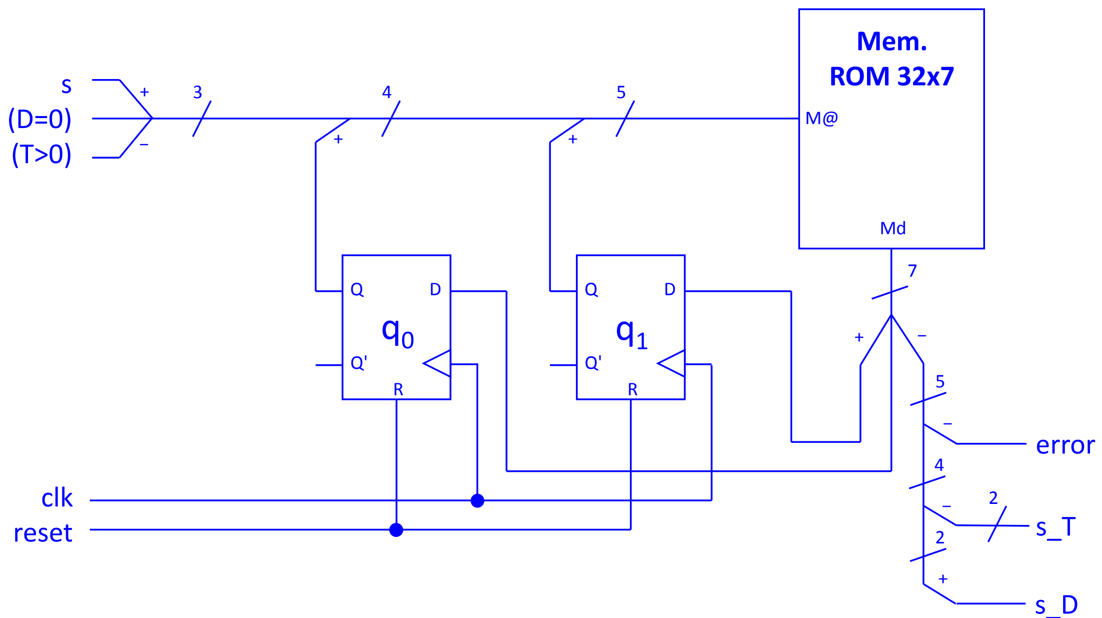
El contenido de la ROM se puede calcular a partir de las tablas de transiciones y salidas que se dan, combinando los códigos binarios correspondientes, sin casos *don't-care*. Así, pues, la ROM tiene que tener el contenido siguiente:

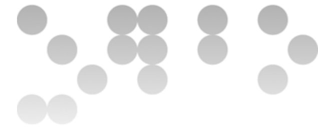
Posición de memoria				Contenido				Codificación hexadecimal
q_1q_0	s	(D=0)	(T>0)	$q_1^+q_0^+$	s_D	s_T	error	
00	0	0	0	01	00	00	0	20h
00	0	0	1	01	00	00	0	20h
00	0	1	0	01	00	00	0	20h
00	0	1	1	01	00	00	0	20h
00	1	0	0	01	00	00	0	20h
00	1	0	1	01	00	00	0	20h
00	1	1	0	01	00	00	0	20h
00	1	1	1	01	00	00	0	20h
01	0	0	0	11	10	01	0	72h
01	0	0	1	01	10	01	0	32h
01	0	1	0	11	10	01	0	72h
01	0	1	1	00	10	01	0	12h
01	1	0	0	11	10	01	0	72h
01	1	0	1	10	10	01	0	52h
01	1	1	0	11	10	01	0	72h
01	1	1	1	00	10	01	0	12h



Posición de memoria				Contenido				Codificación hexadecimal
q_1q_0	s	(D=0)	(T>0)	$q_1^+q_0^+$	s_D	s_T	error	
10	0	0	0	11	01	10	0	6Ch
10	0	0	1	01	01	10	0	2Ch
10	0	1	0	11	01	10	0	6Ch
10	0	1	1	01	01	10	0	2Ch
10	1	0	0	11	01	10	0	6Ch
10	1	0	1	00	01	10	0	0Ch
10	1	1	0	11	01	10	0	6Ch
10	1	1	1	00	01	10	0	0Ch
11	0	0	0	01	00	00	1	21h
11	0	0	1	01	00	00	1	21h
11	0	1	0	01	00	00	1	21h
11	0	1	1	01	00	00	1	21h
11	1	0	0	01	00	00	1	21h
11	1	0	1	01	00	00	1	21h
11	1	1	0	01	00	00	1	21h
11	1	1	1	01	00	00	1	21h

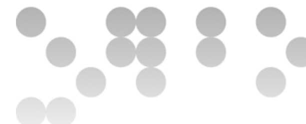
La figura siguiente muestra la implementación con una ROM de estas dimensiones. La entrada *M@* de la ROM se configura con el valor del estado actual (los 2 bits de mayor peso) y el valor de las entradas (los 3 bits de menor peso). Para guardar el estado actual se usa un registro de 2 bits.





La otra opción es usando dos ROM, una para el cálculo del estado siguiente y la otra, para las salidas. En este caso, la ROM de cálculo del estado siguiente tiene el mismo número de bits de dirección que el anterior (5) pero sólo dos bits para el contenido, puesto que sólo se almacena el código del estado siguiente:

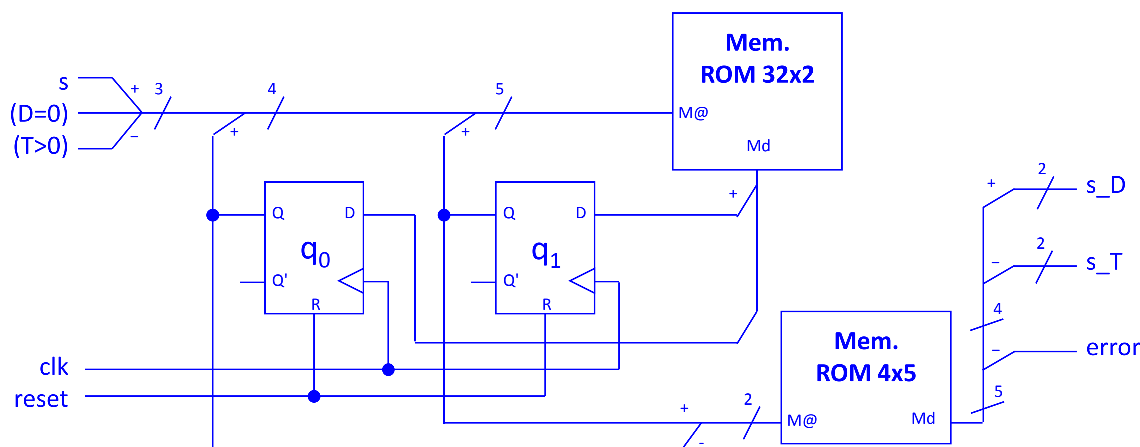
Posición de memoria				Contenido	Codificación hexadecimal
q_1q_0	s	(D=0)	(T>0)	$q_1^+q_0^+$	
00	0	0	0	01	01h
00	0	0	1	01	01h
00	0	1	0	01	01h
00	0	1	1	01	01h
00	1	0	0	01	01h
00	1	0	1	01	01h
00	1	1	0	01	01h
00	1	1	1	01	01h
01	0	0	0	11	03h
01	0	0	1	01	01h
01	0	1	0	11	03h
01	0	1	1	00	00h
01	1	0	0	11	03h
01	1	0	1	10	02h
01	1	1	0	11	03h
01	1	1	1	00	00h
10	0	0	0	11	03h
10	0	0	1	01	01h
10	0	1	0	11	03h
10	0	1	1	01	01h
10	1	0	0	11	03h
10	1	0	1	00	00h
10	1	1	0	11	03h
10	1	1	1	00	00h
11	0	0	0	01	01h
11	0	0	1	01	01h
11	0	1	0	01	01h
11	0	1	1	01	01h
11	1	0	0	01	01h
11	1	0	1	01	01h
11	1	1	0	01	01h
11	1	1	1	01	01h



La ROM del cálculo de las salidas tiene direcciones de dos bits (el código de estado) y contenidos de 5 bits, dos para *s_D*, dos para *s_T* y el que queda, para *error*:

Posición	Contenido			Codificación hexadecimal
q_1q_0	s_D	s_T	error	
00	00	00	0	00h
01	10	01	0	12h
10	01	10	0	0Ch
11	00	00	1	01h

Con estas dos ROM, el circuito final es:

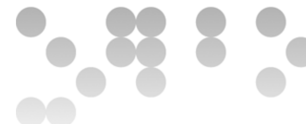


Con esta última versión se usan menos bits de memoria ROM: de 224 (32x7) con una única ROM a 84 (32x2 + 4x5).

d) [20%] Implementad el circuito completo del EFSM de la fig. 2: Construid el camino de datos usando las puertas lógicas y los bloques necesarios e incorporad la unidad de control obtenida al apartado anterior como un módulo más. Indicad y razonad, para cada bus, su dimensión en bits teniendo en cuenta que las señales de entrada ocupan el mínimo número de bits posible según su rango.

Nota: Tenéis el ejercicio disponible a VerilUOC. Recordad que la señal (*D*=0) se corresponde con *D*_0 y la señal (*T*>0) con T _0.

El camino de datos se tiene que ocupar, por un lado, de generar las señales (*D*=0) y (*T*>0) y, por el otro, de actualizar y almacenar el valor de las variables *D*, y *T*. Para lo último, tiene que usar las señales *s_D* y *s_T*, respectivamente.



Antes de determinar los circuitos que hacen falta para hacer los cálculos con *D* y *T* hay que determinar su formato.

La variable *D* puede tomar los valores 1 o (*D*+1) MOD 3 y, por lo tanto, siempre es menor que 3 y positiva. Así pues, basta con dos bits.

La variable *T* puede tomar los valores *N* o *T*–1. Dado que *N* es un valor entre 0 y 15, sólo hacen falta 4 bits para representar los valores que puede tomar. Dado que, según el modelo, cada vez que no se cumple que *T* sea mayor que 0 se pasa a un estado donde se vuelve a poner *a N*, *T* será siempre positivo. Así pues, basta con 4 bits para representar los valores de *T*.

Con todo, la señal (*D*=0) se puede calcular a partir de una *o negada* (NOR) de sus bits, y la señal (*T*>0), con un comparador con 0.

Para el cálculo de (*D*+1) MOD 3 hay muchas alternativas porque se trata de valores de 2 bits. La más simple es con un circuito con puertas generado a partir de la tabla de verdad siguiente:

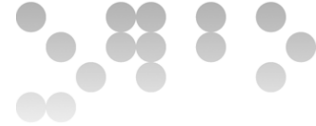
D			**F**		
valor	**d_1**	**d_0**	**(D+1) MOD 3**	**f_1**	**f_0**
0	0	0	1	0	1
1	0	1	2	1	0
2	1	0	0	0	0
3	1	1	X	x	x

Las funciones que se tienen que implementar son $f_1 = d_0$ y $f_0 = d_1' \cdot d_0'$. De hecho, aplicando el teorema de De Morgan, se obtiene $f_0 = (d_1+d_0)'$, es decir, coincide con la señal (*D*=0).

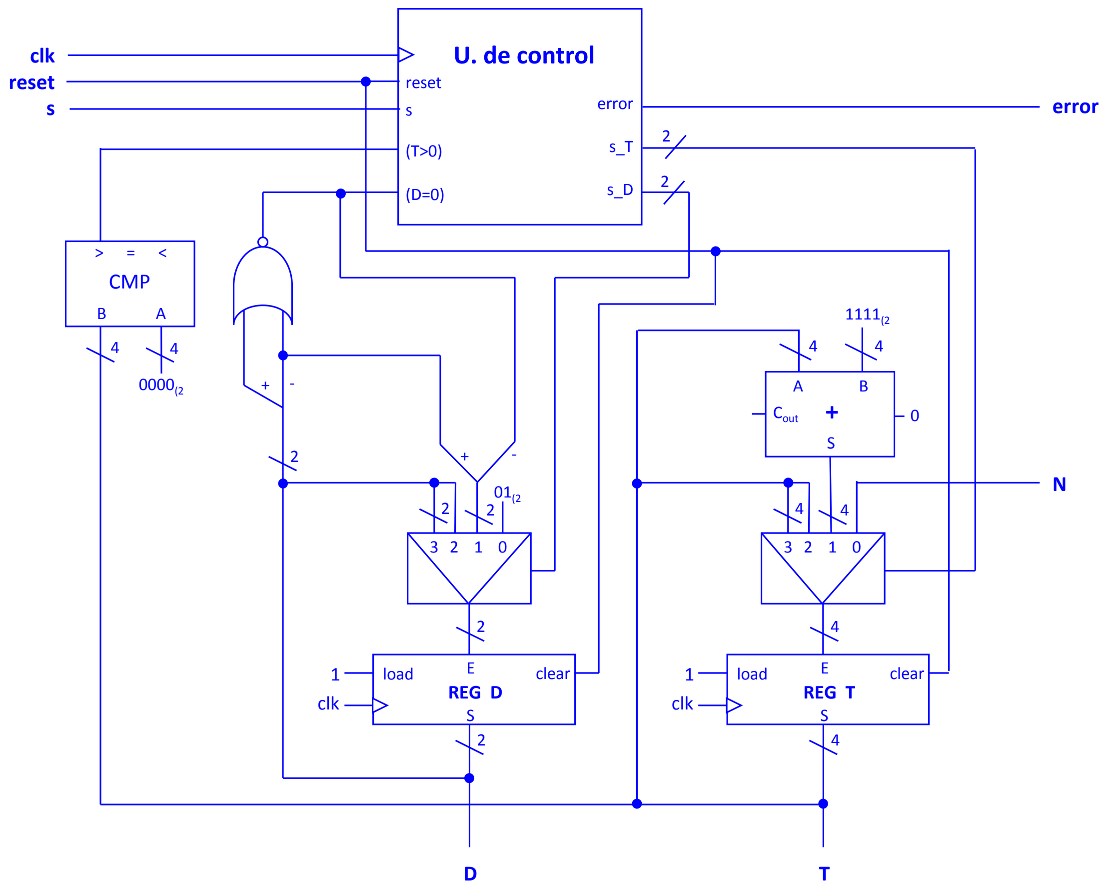
Para el cálculo de *T*–1 se puede usar un sumador de 4 bits de *T* y de $1111_{(2}$. Hay que tener presente que, matemáticamente, esto supone hacer la operación con números de 5 bits en complemento a 2: uno que sería *T* con un cero en la posición más significativa y otro que sería $11111_{(2}$ (–1). Ahora bien, como que el bit más significativo no interesa porque se sabe que el resultado siempre será positivo o cero, se puede descartar.

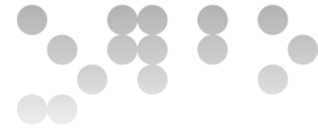
Finalmente, hacen falta dos multiplexores de buses para seleccionar qué valor se asigna en el estado siguiente a *D* y a *T* según las señales *s_D* y *s_T*, respectivamente.

Con todos los elementos anteriores ya se dispone del camino de datos necesario para la implementación de un circuito que se comporte como la EFSM de la fig. 2.



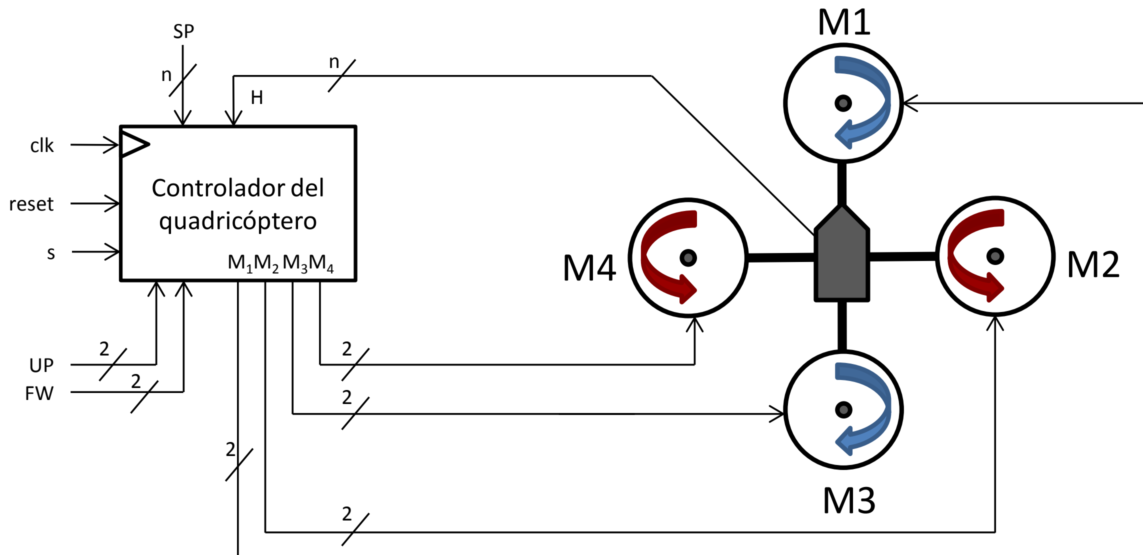
El circuito completo queda tal como se muestra a continuación:





SEGUNDA PARTE [40%]

Se quiere diseñar un controlador de movimiento básico y simplificado de un quadricóptero. El aparato dispone de cuatro motores, uno por cada hélice, de los cuales podemos controlar la velocidad de giro. Para conseguir que el aparato no gire sobre sí mismo, dos motores (M1 y M3) giran en sentido horario y dos, en sentido antihorario (M2 y M4), tal y como se muestra a la figura siguiente:

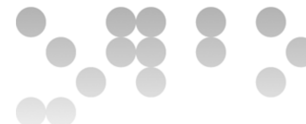


La velocidad de cada motor se da a través de las señales M_1, M_2, M_3 y M_4, en una escala de 0 a 3, en la que 0 indica 'motor parado', y 3, 'velocidad máxima'. El valor 2 en la señal de control de velocidad de los motores los hace girar a la velocidad necesaria para mantener el dispositivo en suspensión, sin subir ni bajar.

El controlador del quadricóptero mantiene los motores apagados hasta que recibe un pulso a 1 por la señal de entrada *s*. Este pulso puede durar más de un ciclo de reloj. Después del pulso, arranca los motores para ponerlos a velocidad de suspensión.

El quadricóptero dispone de un altímetro que proporciona la altitud *H* al controlador, en centímetros. Para alturas inferiores a 50 cm, sólo admitirá variaciones en altura, para poder permitir maniobras de despegue y de aterrizaje, exclusivamente. De 50 cm para arriba, el controlador admite todo tipo de órdenes de movimiento.

El aparato no podrá apagarse a menos que no se encuentre a una altura inferior a 5 cm. Por debajo de esta altura, si recibe un pulso a 1 por la entrada *s*, el controlador apagará los motores. Del mismo modo que para poner en marcha el aparato, los pulsos para pararlo también pueden durar más de un ciclo de reloj.



Así pues, el controlador básico a diseñar tiene que permitir “traducir” las órdenes de los usuarios a movimientos del quadricóptero.

Las órdenes de los usuarios llegan codificadas en binario a través de las señales siguientes: (Tened presente que las órdenes pueden estar activadas más de un ciclo de reloj)

— *UP*, señal de 2 bits que indica lo siguiente:
  00 → mantiene altura; 01 → sube; 10 → baja; 11 → no se usa.

  Para poder subir se tiene que incrementar, a la vez, la velocidad de todos los motores al nivel 3. Para bajar, todos los motores tienen que bajar a la velocidad 1.

— *FW*, señal de 2 bits que indica lo siguiente:
  00 → mantiene posición; 01 → adelante; 10 → atrás; 11 → no se usa.

  Para poder ir adelante, el motor de delante (M1) tiene que bajar de velocidad, a 1, y el posterior (M3) tiene que aumentar de velocidad, a 3. Para poder ir atrás, se configuran a la inversa, el de delante sube de velocidad y el posterior, baja.

— *SP*, señal de 2 bits que indica lo siguiente:
  00 → mantiene posición; 01 → giro a la derecha; 10 → giro a la izquierda, y
  11 → no se usa.

  Para conseguir girar el aparato en sentido horario, se tiene que aumentar la velocidad de los dos motores que giran en este sentido (M1 y M3) y bajar la de los otros dos. Para girar en sentido antihorario, se aumenta la velocidad de los otros dos motores (M2 y M4) y se baja la de los dos primeros. Para volver a mantener la posición todos los motores se ponen al nivel 2 de velocidad.

El controlador sólo admitirá pulsos de parada *s* a 1 con el aparato por debajo de los 5 cm.

Se pide diseñar el ASM del controlador de movimiento del quadricóptero que hace que se comporte de la forma que se ha especificado.

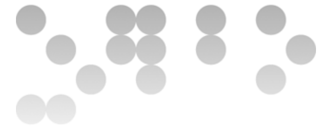
El controlador a diseñar tiene que estar en un estado de reposo, IDLE, hasta que *s* sea 1. Entonces pasa en un estado de espera, ON, a que *s* retorne a 0.

Cuando *s* vuelva a ser 0 de nuevo, se pondrán en marcha los motores, todos a velocidad 2, y se quedará a la espera de alguna orden del usuario.

Estando en marcha y en este nuevo estado de espera, HOVER, el controlador pasará a los estados de movimiento en función de la orden dada y de la altura.

Así pues, si la altura *H* es inferior a 5 cm, sólo comprobará la señal *s*. Si es 1, pasará en un estado de apagada, OFF, en el que los motores se apagarán y se esperará a que *s* vuelva a 0. Cuando lo haga, el controlador volverá al estado IDLE.

Continuando desde HOVER, comprobará si la señal *UP* es diferente de 00 e irá a los estados UP o DOWN, según el caso. Si es 00 y la altura *H* es igual o superior a 50 cm, también comprobará las señales *FW* y *SP*.



Volando por encima de los 50 cm, con *FW*=01 irá a FWD y con *FW*=10 a BCK. Si es 00, entonces, comprobará los valores de *SP* para ir a LEFT o RIGHT. En caso de que no haya ninguna orden, permanecerá en HOVER.

En los estados de movimiento (UP, DOWN, FWD, BCK, LEFT y RIGHT), las salidas de los motores se ajustarán a los valores para conseguir el efecto deseado y las condiciones de salida son las mismas que para HOVER.

El ASM resultando se muestra en el diagrama siguiente:

IDLE 0000
$M_1^+ = 0, M_2^+ = 0, M_3^+ = 0, M_4^+ = 0$
(s=1)?
ON 1000
$M_1^+ = 0, M_2^+ = 0, M_3^+ = 0, M_4^+ = 0$
(s=0)?
HOVER 1100
$M_1^+ = 2, M_2^+ = 2, M_3^+ = 2, M_4^+ = 2$
H < 5cm && (s=1)
OFF 0100
$M_1^+ = 0, M_2^+ = 0, M_3^+ = 0, M_4^+ = 0$
(s=0)?
UP=10?
UP=01?
H<50cm?
UP 1101
$M_1^+ = 3, M_2^+ = 3, M_3^+ = 3, M_4^+ = 3$
DWN 1110
$M_1^+ = 1, M_2^+ = 1, M_3^+ = 1, M_4^+ = 1$
FW=10?
FW=01?
SP=10?
SP=01?
BCK 1010
$M_1^+ = 3, M_2^+ = 2, M_3^+ = 1, M_4^+ = 2,$
FWD 1001
$M_1^+ = 1, M_2^+ = 2, M_3^+ = 3, M_4^+ = 2$
LEFT 0110
$M_1^+ = 1, M_2^+ = 3, M_3^+ = 1, M_4^+ = 3$
RIGHT 0101
$M_1^+ = 3, M_2^+ = 1, M_3^+ = 3, M_4^+ = 1$

Hay que tener presente que hay otras opciones de diseño que pueden ser igualmente válidas.