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 esta práctica 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. Para la primera parte, tendréis que aplicar los conocimientos que habéis adquirido en este curso como, por ejemplo, los mapas de Karnaugh o los sistemas de representación de la información. Por lo tanto, se recomienda que repaséis los módulos correspondientes.

Competencias

- Conocer la organización general de un computador como circuito digital.
- Conocer los rasgos distintivos de 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 usar por esta práctica son los siguientes:

- **Básicos:** El módulo 5 de los materiales.
- **Prácticas anteriores:** En el apartado de recursos adicionales del aula de CANVAS podéis encontrar prácticas otros semestres con ejemplos de resolución.
- **Complementarios:** VerilCIRC, VerilCHART y el Wiki de la asignatura.

Criterios de valoración

- **La valoración se indica en cada uno de los subapartados.**
- **Razonad la respuesta en todos los ejercicios.**
- **Las respuestas sin justificación no recibirán puntuación.**
- **Los ejercicios realizados con IA generativa no recibirán puntuación.**
- **Los ejercicios que se detecten como plagio mediante la herramienta de plagio de la universidad no recibirán puntuación.**

Uso de herramientas de IA

En esta actividad no está permitido el uso de herramientas de inteligencia artificial. En el plan docente y en la web sobre integridad académica y plagio de la UOC encontraréis información sobre qué 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 entregar la solución en un documento PDF, usando una de las plantillas adjuntas a este enunciado.
– Se tiene que 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 **28 de mayo,** a las 24 horas.

Ejemplo de solución

PRIMERA PARTE [65%]

Dada la máquina de estados siguiente:

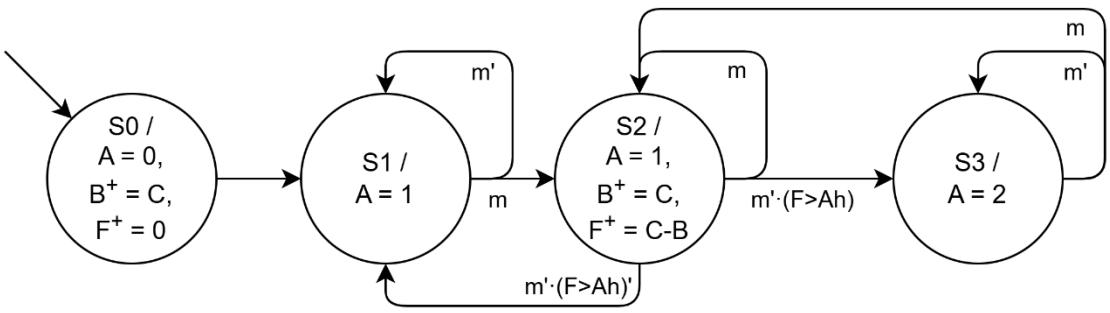
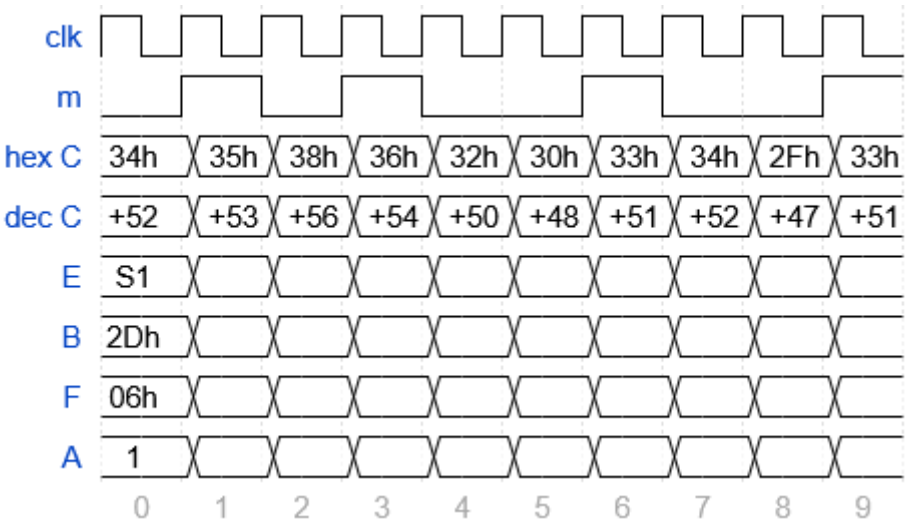


Fig. 1. EFSM de un controlador simplificado de un detector de flujo de personas.

Esta máquina de estados tiene, como entradas, una señal *m* de un bit que indica el paso de un minuto y una señal de datos *C*, que indica cuántas conexiones a la red móvil de datos hay en un momento determinado, y, como salidas, una señal de dos bits (*A*) de aviso de exceso de entrada de personas (se supone que cada conexión es una persona) y el valor calculado del flujo de personas por minuto *F*. Por simplicidad, tanto la señal de entrada *C* como las variables *B* y *F* son números en complemento a 2 de 8 bits. Sabiendo todo esto, se pide que:

a) [15%] A partir del grafo del EFSM de la Fig. 1 completéis, en el cronograma siguiente, la evolución del estado (*E*) y de las señales *A*, *B* y *F* a cada ciclo de reloj.
Nota: Tenéis el ejercicio disponible en VerilChart, donde el estado es *E* y los valores de las señales numéricas se representan en hexadecimal. De hecho, en VerilChart, la señal de entrada *C* aparece solo en hexadecimal

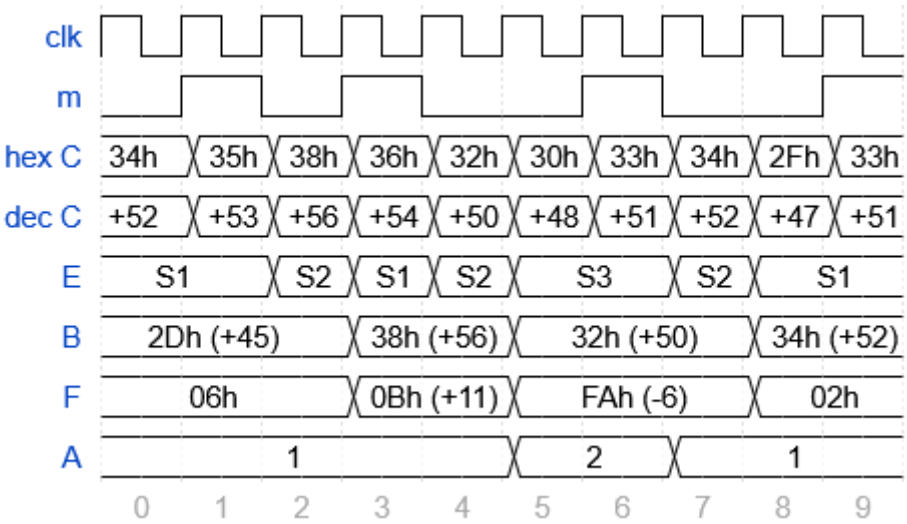


De acuerdo con el cronograma, la máquina de estados está en el estado S1 y, dado que *m*=0, el estado siguiente, en el ciclo número 1, será el mismo S1, sin cambios ni en *B* ni en *F*. En el ciclo siguiente (ciclo número 2), dado que *m*=1, pasará a S2.

En el ciclo número 2, estando en S2, la máquina pasará al ciclo número 3 actualizando *B* y *F* a los valores de *C* (38h o +56) y *C*–*B* (0Bh o +11) del ciclo número 2. Dado que *m*=0 y que no se cumple que *F*>0Ah, pasará al estado S1.

De manera similar se pueden determinar los valores para cada ciclo de reloj a partir de los valores del ciclo anterior y se puede ir repitiendo este procedimiento hasta completar el cronograma, tal como se muestra en la figura siguiente.

La salida *A* se puede rellenar al final, según el valor del estado en cada ciclo.



b) [15%] Obtén las tablas de transiciones y de salidas de la EFSM de la Fig. 1, así como las codificaciones binarias de estados y salidas correspondientes. Para simplificar la notación, (*F*>0Ah) se denominará *F*10.

En el grafo hay las señales de entrada *m* y *F*10 que condicionan las transiciones y una entrada de datos *C* que solo se utiliza en el cálculo del valor futuro de la variable *B*. Para las variables *B* y *F* hay que determinar cuál será el valor al ciclo siguiente, del mismo modo que hay que determinar los valores de la salida *A* según el estado.

Las tablas de transiciones y salidas que se obtienen a partir del grafo son:

Estado actual	Entrada	Estado siguiente
S0	--	S1
S1	m	S1
S1	m	S2
S2	m' ·F10'	S1
S2	m' ·F10	S3
S2	m	S2
S3	m'	S3
S3	M	S2

Estado	Salidas
S0	A=0, B^+ = C, F^+ = 0
S1	A=1, B^+ = B, F^+ = F
S2	A=1, B^+ = C, F^+ = C-B
S3	A=2, B^+ = B, F^+ = F

Para la codificación binaria, hay que asignar en cada estado un código binario individual. En este caso, se sigue el criterio más habitual, que es el de la numeración binaria, desde el 00 (S0) hasta el 11 (S3).

La variable *B* tiene suficiente con un selector de un bit (*s_B*), puesto que solo puede tener como valores futuros su mismo contenido o *C*. La variable *F*, en cambio, necesita un selector de dos bits (*s_F*), puesto que tiene tres posibles valores futuros (0, *F* y *C*–*B*).

Estado	Código	A	s_B	B^+	s_F	F^+
S0	00	00	0	C	00	0
S1	01	01	1	B	01	F
S2	10	01	0	C	10	C-B
S3	11	10	1	B	01/11	F

El valor de la salida *A* se determina directamente a partir del código del estado.

El selector *s_B* coincide con el bit menos significativo del estado (q_0) y el selector *s_F* se puede hacer coincidir con el código de estado para simplificar la implementación del

circuito correspondiente y, por eso, hay la doble opción de codificación (01 o 11) en su columna. Aun así, una solución sin estas simplificaciones también es correcta.

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

Estado actual		Entrada		Estado⁺
Símbolo	q_1q_0	m	F10	$q_1^+q_0^+$
S0	00	x	x	01
S1	01	0	x	01
S1	01	1	x	10
S2	10	0	0	01
S2	10	0	1	11
S2	10	1	x	10
S3	01	0	x	11
S3	01	1	x	10

Estado			s_F
Símbolo	q_1q_0	A	s_1s_0
S0	00	00	00
S1	01	01	01
S2	10	01	10
S3	11	10	01/11

Una etiqueta pasiva de RFID envía datos a un dispositivo lector a partir del momento que ha conseguido recibir suficiente energía de la señal que emite el lector. Los datos las emite variando la dispersión de la señal del lector. Esto lo hace alternando la conexión de la antena al suelo (*y*=1) o dejándola desconectada (*y*=0). Para emitir un cero, la antena se desconecta durante un periodo de tiempo (*dt*) y se vuelve a conectar durante el mismo periodo (*dt*). En el caso de la emisión de los unos, el periodo de conexión dura el doble (*dt*).

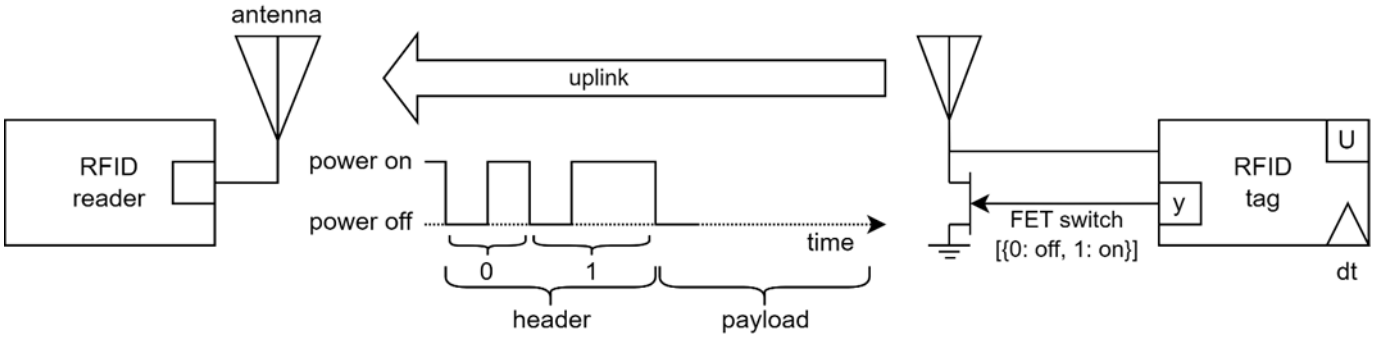


Fig. 2. La etiqueta pasiva (derecha) emite un código *U* cada vez que recibe suficiente energía a través de la antena.

Por simplicidad, se considerará que el código *U* es un código constante de 8 bits y que ya incluye un 01 inicial. En nuestro caso, será A6h.

El comportamiento de esta etiqueta pasiva se representa en el diagrama de estados de la Fig. 3.

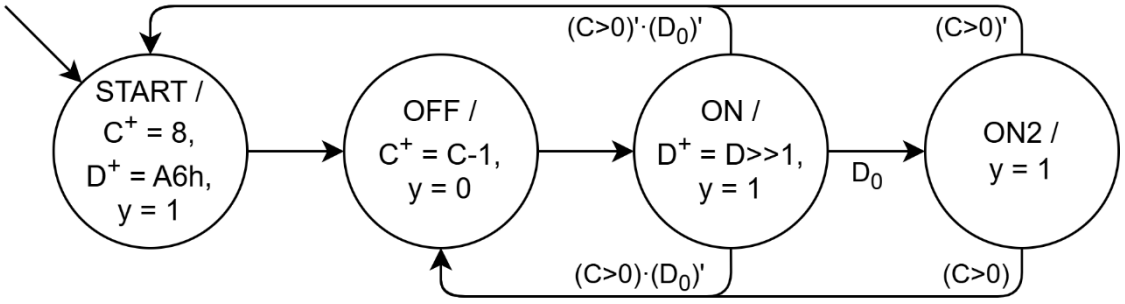


Fig. 3. EFSM de un emisor de la identificación de una etiqueta pasiva.

A la vista de la EFSM correspondiente, se pide que:

c) [15%] A partir de las tablas de transiciones y de salidas del grafo de la Fig. 3 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. Especificad y justificad las dimensiones de la/las ROM que uséis e indicad su contenido en binario, especificando a qué corresponde cada bit, y en hexadecimal.

Estado actual	Entradas de control		Estado+
	D_0	(C>0)	
START	x	x	OFF
OFF	x	x	ON
ON	0	0	START
ON	0	1	OFF
ON	1	x	ON2
ON2	x	0	START
ON2	x	1	OFF

Estado		Salidas		
Símbolo	q_1q_0	y	s_C	s_D
START	00	1	00	00
OFF	01	0	01	01
ON	10	1	10	10
ON2	11	1	10	01

Nota: Tenéis el ejercicio disponible en VerilCIRC y, si queréis comprobar previamente la corrección de una ROM válida con palabras de 7 bits, la tenéis disponible en VerilChart. Tened en cuenta que hay otras opciones igualmente válidas con ROM de diferentes dimensiones. En este entorno, el nombre de la señal de estado es *E* y el de las entradas D_0 y (*C*>0) son *D*0 y *Cno*0, respectivamente.

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

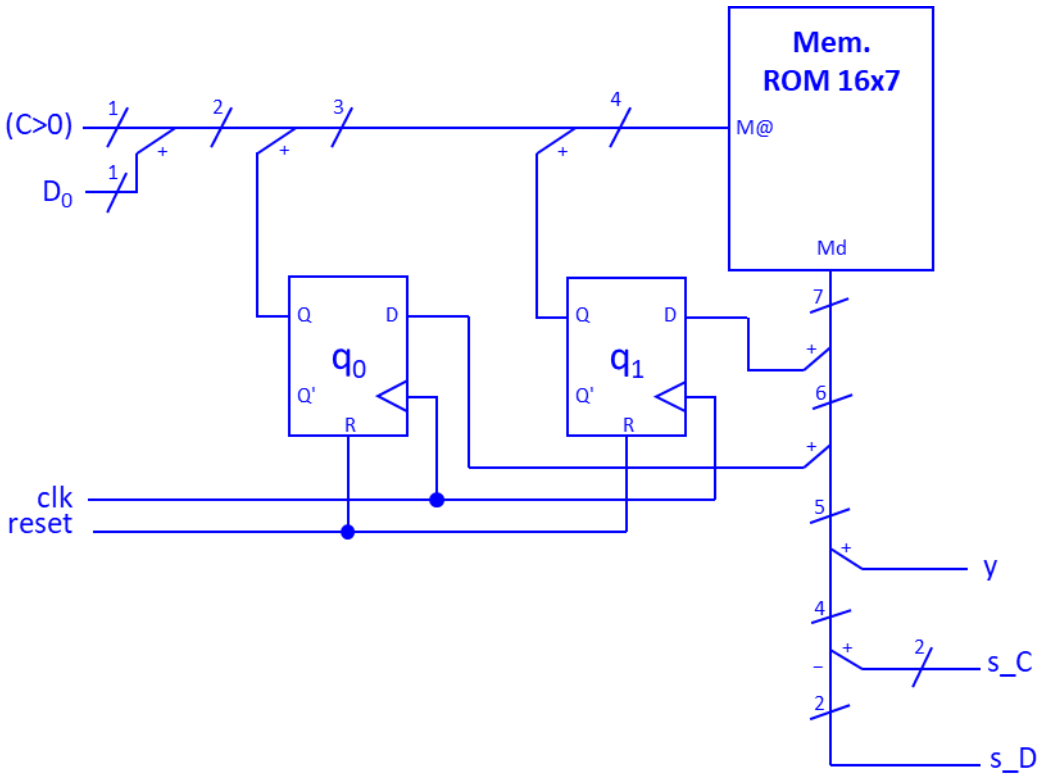
El bus de direcciones de la ROM tiene que ser de 4 bits, puesto que la dirección se forma con los bits que codifican el estado (son 2, en este caso) y los de las señales de entrada (2 más). Por lo tanto, la ROM tiene que disponer de un total de $2^4 = 16$ posiciones de memoria.

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, hacen falta 2 bits para el estado y harían falta 5 bits para las salidas (*y*, *s_C* y *s_D*), lo que hace un total de 7 bits.

Así pues, si se usa una única ROM, tiene que ser de 16x7 bits. El contenido de la ROM se determina a partir de la tabla de transiciones y la de salidas:

Posición de memoria			**Contenido**				**Codificación hexadecimal**
q_1q_0	**D_0**	**(C>0)**	**$q_1^+q_0^+$**	**Y**	**s_C**	**s_D**	
00	0	0	01	1	00	00	30h
00	0	1	01	1	00	00	30h
00	1	0	01	1	00	00	30h
00	1	1	01	1	00	00	30h
01	0	0	10	0	01	01	45h
01	0	1	10	0	01	01	45h
01	1	0	10	0	01	01	45h
01	1	1	10	0	01	01	45h
10	0	0	00	1	10	10	1Ah
10	0	1	01	1	10	10	3Ah
10	1	0	11	1	10	10	7Ah
10	1	1	11	1	10	10	7Ah
11	0	0	00	1	10	01	19h
11	0	1	01	1	10	01	39h
11	1	0	00	1	10	01	19h
11	1	1	01	1	10	01	39h

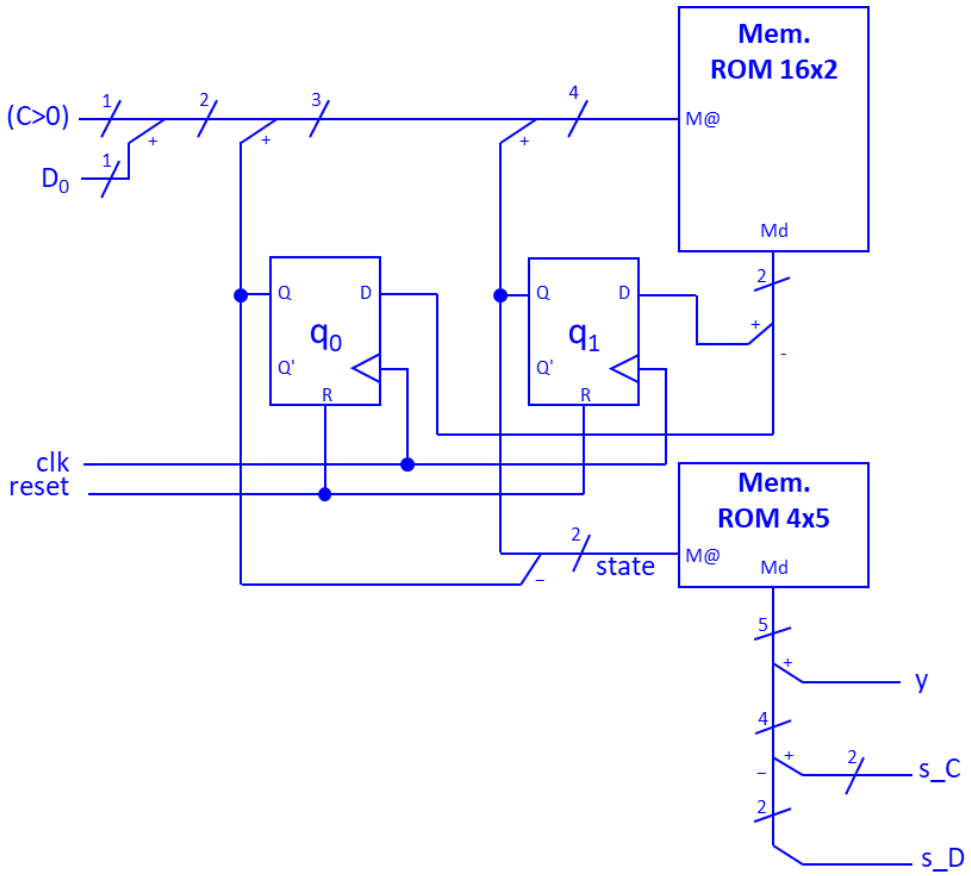
La figura siguiente muestra la implementación con la ROM correspondiente. La entrada *M@* de la ROM se configura con el valor del estado actual (los 2 bits de más peso) y los valores de las señales de entrada (los bits de menos peso). Para guardar el estado actual se usan 2 biestables.



Con dos ROM, una para la tabla de transiciones y otra para la de salidas, la primera sería de 16x2 (es la misma que la anterior, con las palabras de memoria que solo contienen la codificación del estado siguiente) y la segunda, de 4x5, tal como se muestra a continuación.

Posición de memoria	Contenido			Codificación hexadecimal
q_1q_0	y	s_C	s_D	
00	1	00	00	10h
01	0	01	01	05h
10	1	10	10	1Ah
11	1	10	01	19h

El circuito correspondiente queda como se muestra a continuación.



También se podrían sustituir las ROM por puertas lógicas, pero queda fuera de lo que se pide al ejercicio.

d) [20%] Implementad el circuito completo del EFSM de la Fig. 3: Construid el camino de datos usando las puertas lógicas y los bloques necesarios e incorporad la unidad de control obtenida en el apartado anterior. Tened en cuenta que la variable *D* es de 8 bits. Indicad y razonad, para cada bus, su dimensión en bits.

Nota: Tenéis el ejercicio disponible en VerilCIRC. En el circuito, las señales *C* y *D* también son de salida para poder comprobar el funcionamiento más fácilmente. Tened en cuenta que VerilCIRC no puede verificar subcircuitos diseñados por el usuario y, por lo tanto, tendréis que copiar en vuestra solución el circuito de la unidad de control que hayáis diseñado en el apartado anterior.

Por un lado, el camino de datos se tiene que ocupar de generar las señales *C*>0 y *D*0 para la unidad de control. De la otra, tiene que hacer todos los cálculos posibles de los nuevos valores de las variables *C* y *D*, y almacenarlos en los registros correspondientes según las señales de control que toquen. En este caso, además, hay una señal de salida inmediato (*y*), sin que se tenga que almacenar en ninguna parte.

La dimensión del bus que transporta la señal *C* y del registro que lo almacena tiene que ser de 4 bits, puesto que tienen que poder representar del 0 al 8. En el caso de la señal *D*, tal como se indica en el enunciado, tiene que ser de 8 bits.

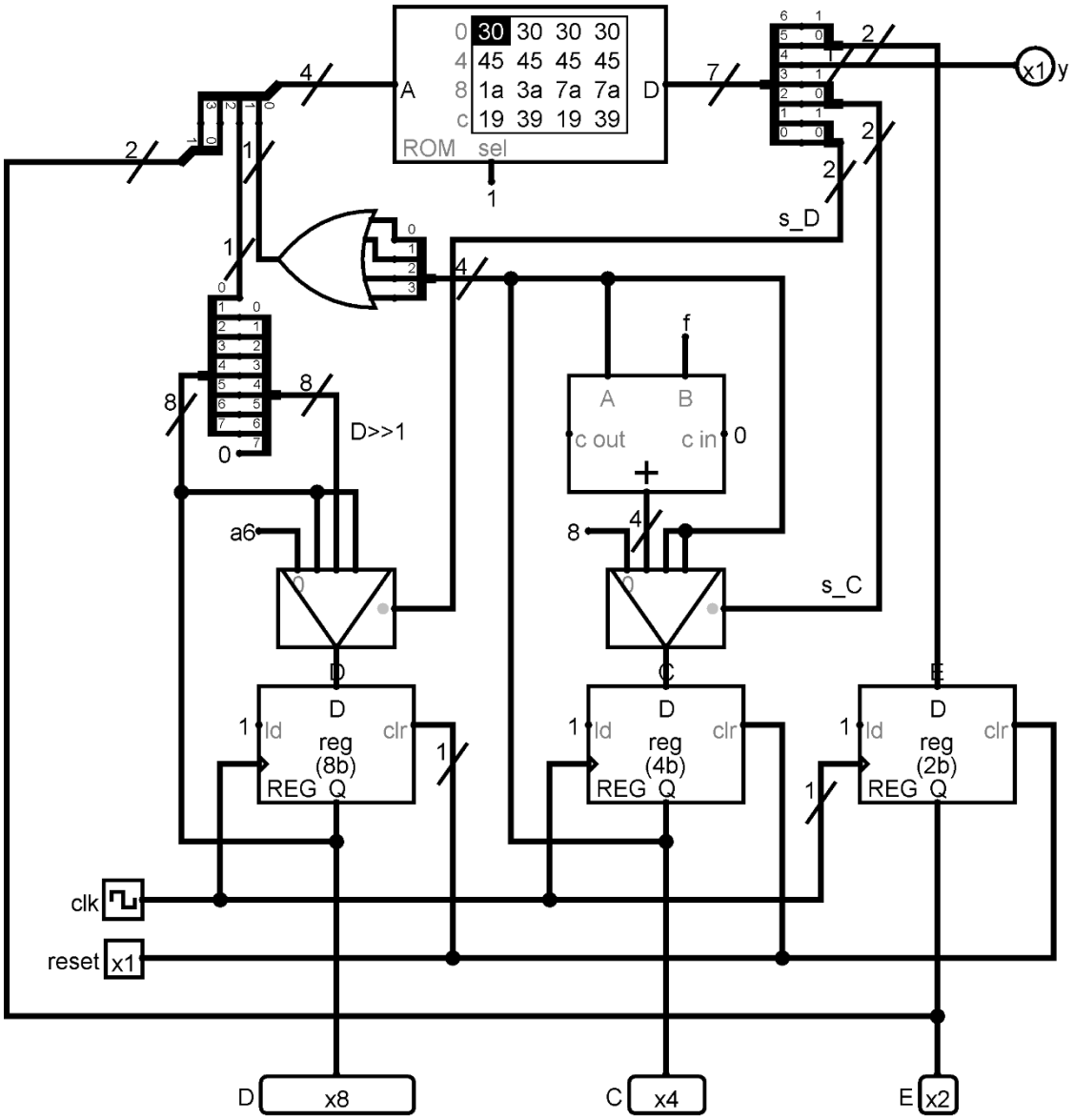
Los selectores *s_C* y *s_N* son de dos bits y controlan qué valor se carga en los registros que almacenan la variable correspondiente según la tabla de salidas del circuito.

Como que a *C* se le puede asignar A6h, *C*-1 y *C*, hay que calcular la resta de uno con un sumador que le sume –1, que es $1111_{(2}$. En este sentido, formalmente, la operación se tendría que hacer con 5 bits, puesto que para representar +8 en complemento a 2 hacen falta 5. Sin embargo, como que solo nos interesan los valores positivos y *C* no puede ser nunca –1, se puede dejar en 4, considerando el bit de signo implícito.

Para determinar la señal *C*>0 basta con ver si contiene algún 1, cosa que se hace con una puerta OR, pero también se puede hacer con un comparador de 4 bits.

En el caso de *D*, se le puede asignar A6h, *D*>>1 o *D* y, por lo tanto, la operación que se tiene que implementar es un desplazamiento a la derecha que, se puede hacer con un par de separadores de buses de forma que los bits se aparejen según este desplazamiento o bien con un decalador. El bit de más a la derecha (D_0) da directamente el valor de la señal *D*0.

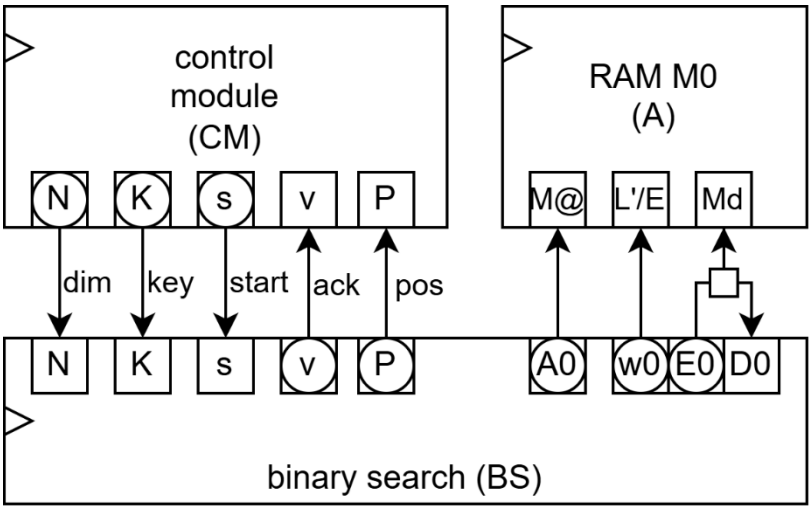
Con todo esto, el circuito completo para el EFSM de la Fig. 3 queda de la forma siguiente:



Hay que tener presente que el circuito mostrado es una posible solución, pero hay otros de igualmente válidas con dos ROM, o que usen bloques lógicos combinacionales equivalentes, o que incorporen alguna optimización.

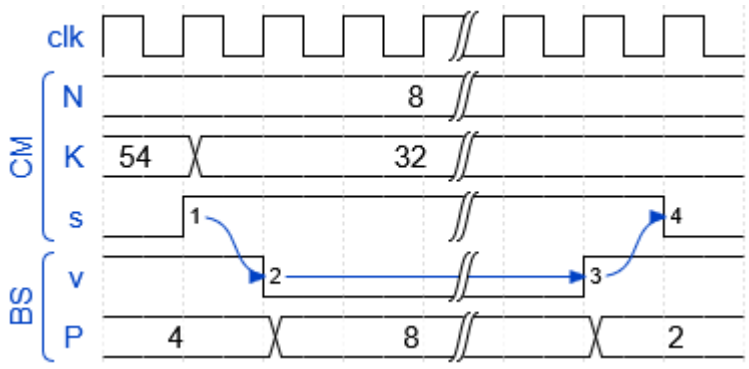
SEGUNDA PARTE [35%]

Se tiene que diseñar el modelo de un módulo de búsqueda binaria (BS) que usa una memoria (M0) que contiene los datos ordenados (*A*), en las que se tiene que buscar un determinado elemento (*K*) que proporciona un módulo de control (CM).



El módulo BS, además de la señal de entrada *K*, tiene otra (*N*) para saber la longitud del vector de datos *A* y otra para saber en qué ciclo tiene que iniciar la búsqueda (*s*). Por simplicidad, se supone que cada dato de *A* ocupa una única palabra de memoria.

BS tiene las salidas *P* y *v*, que contienen la dirección del elemento *K* en la memoria M0 y el aviso que este valor ya es válido, respectivamente. La señal *P* será *N* en caso de que no se haya encontrado el elemento (las direcciones válidas son al rango [0, *N*–1]) y *v* se tiene que mantener a 0 desde el ciclo siguiente en qué *s*=1 y hasta el ciclo en que se actualice *P* con el resultado de la búsqueda, tal como se puede ver en el cronograma siguiente.



Las entradas *N*, *K* y *s* se mantienen constantes desde que *s*=1 hasta que *v* vuelva a 1, como mínimo.

El algoritmo que implementa el módulo BS desde que se le ha indicado que empiece la búsqueda (ciclo en qué *s* pasa a 1 desde 0) hasta que la acaba (ciclo en qué *v* vuelve a 1 desde 0) se describe en el siguiente programa en C.

```
#include <stdio.h>
int main()
{
    // example array
    int A[] = {10, 21, 32, 43, 54, 65, 76, 87};
    int N = 8;
    // variable initialization
    int P = N;         // position of key, N if not found
    int L =  0;        // leftmost position
    int R =  N-1;      // rightmost postition
    int M = (N-1)>>1; // middle position
    int K = 0;         // search key
    // input key from user and store it into variable K
    printf("Search key = "); scanf("%d", &K);
    // while the sublist from L to R is not empty and
    // the middle element is not K (not found)
    while(L<=R && A[M]!=K) {
        if(A[M]<K) {
            L = M + 1;
        } else {
            R = M - 1;
        } // if
        M = (L+R)>>1;
    } // while
    // if sublist is not empty, then
    // the key has been found and its position is stored into P
    if(L<=R) { P = M; }
    // output search outcome
    printf("Position = %d (=%d: not found)\n", P, N);
    return 0;
} // main
```

En el modelo que se tiene que diseñar hay que tener en cuenta la comunicación con la memoria M0, que se hace con la salida que contiene las direcciones de memoria (*A*0), la señal de salida que indica si se hace una lectura o una escritura (*w*0), la salida que contiene los datos a escribir (*E*0) y la entrada que proporciona lo que se lee (*D*0). En este caso, *w*0 siempre será 0 y *E*0 se puede dejar también a 0, puesto que no se harán escrituras.

Diseña la ASM correspondiente al comportamiento del módulo BS que se ha descrito teniendo en cuenta que todas sus salidas son variables de la máquina de estados.

El diagrama de estados de la ASM se puede hacer siguiendo el programa de ejemplo, empezando por una inicialización de las variables del programa, que también lo serán de la máquina de estados. A esta inicialización se tiene que añadir la de las variables de relación con la memoria y con el módulo de control. En este sentido, la variable del programa (*M*) que contiene la posición central del vector *A* es la misma que la dirección de la memoria *A*0, puesto que M0 contiene *A* desde la posición 0 hasta *N*–1.

El estado inicial tiene que ser un estado de espera (WAIT) a que *s*=1. En el ciclo en qué esto pase, la máquina de estados tiene que empezar a ejecutar el algoritmo que se indica

en el programa. Para esto, pasa a BEGIN, donde se inicializan los índices izquierda (*L*), central (*A*0) y derecho (*R*) de la lista de valores en los que hacer la búsqueda. Además, se tiene que poner la salida *v* a 0 para indicar que el valor de *P* no es válido y también se inicializa el valor de *P* a *N* de forma que el valor por omisión de *P* sea la indicación que no se ha encontrado el elemento *K*.

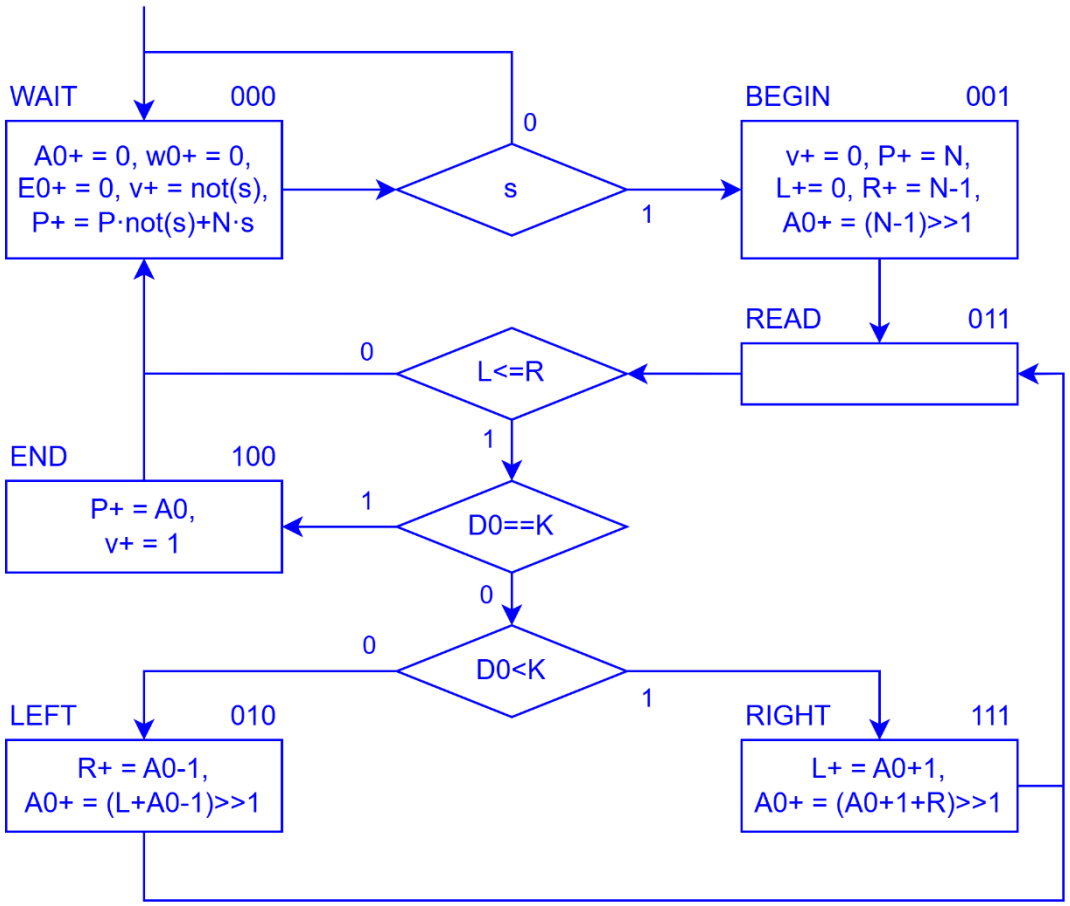
Dado que las asignaciones de los valores en el ciclo siguiente de *v* y de *P* dependen de si la máquina se queda en el estado WAIT (*s*=0) o pasa a BEGIN (*s*=1), estas variables se han de calcular en función de *s*, tal y como aparece en la caja del estado WAIT en el diagrama. De todas maneras, por simplicidad, también se considera correcto hacer v^+=1 y hacer las asignaciones a *v* y a *P* que constan en BEGIN, a pesar de que, en este caso, pasarán dos ciclos entre *s*=1 y *v*=0.

De BEGIN se tiene que pasar a un estado READ para que la entrada *D*0 tenga el valor de la posición indicada a *A*0. En este estado se tiene que comprobar si el elemento *K* está en la parte izquierda o derecha de la lista y actualizar los índices correspondientes. Para hacerlo, la máquina de estados pasa a LEFT o a RIGHT, según sea el caso.

Evidentemente, si el elemento que se ha leído (*D*0) se corresponde al elemento a buscar (*K*), de READ pasa a END, donde actualiza el valor de *P* a A0 y de *v* a 1.

Las comparaciones entre *D*0 y *K* solo son válidas si la lista de valores entre *L* y *R* tiene alguno. Si la lista está vacía (*L*>*R*), esto implica que no se ha encontrado el valor y que ya se puede pasar en el estado de espera WAIT.

La figura siguiente muestra la ASM correspondiente a este comportamiento:



Hay que tener presente que hay varias soluciones válidas posibles y que el diagrama que se ha presentado no sigue exactamente los pasos que hay en el programa en C, que contiene una comparación adicional entre *L* y *R*.