75.562 · Fundamentos de Computadores · 2024-25

Práctica

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 **28 de mayo** (a las 24 horas).
- **Razonad la respuesta en todos los ejercicios. Las respuestas sin justificación no recibirán puntuación.**

Respuestas

PRIMERA PARTE [65 %]

Dada la máquina de estados siguiente:

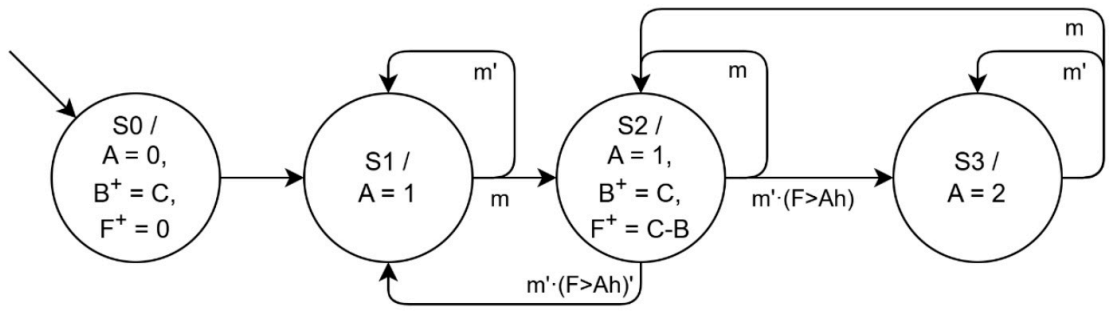


Fig. 1. EFSM de un controlador simplificado de un detector de flujo de personas.

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 lo siguiente:

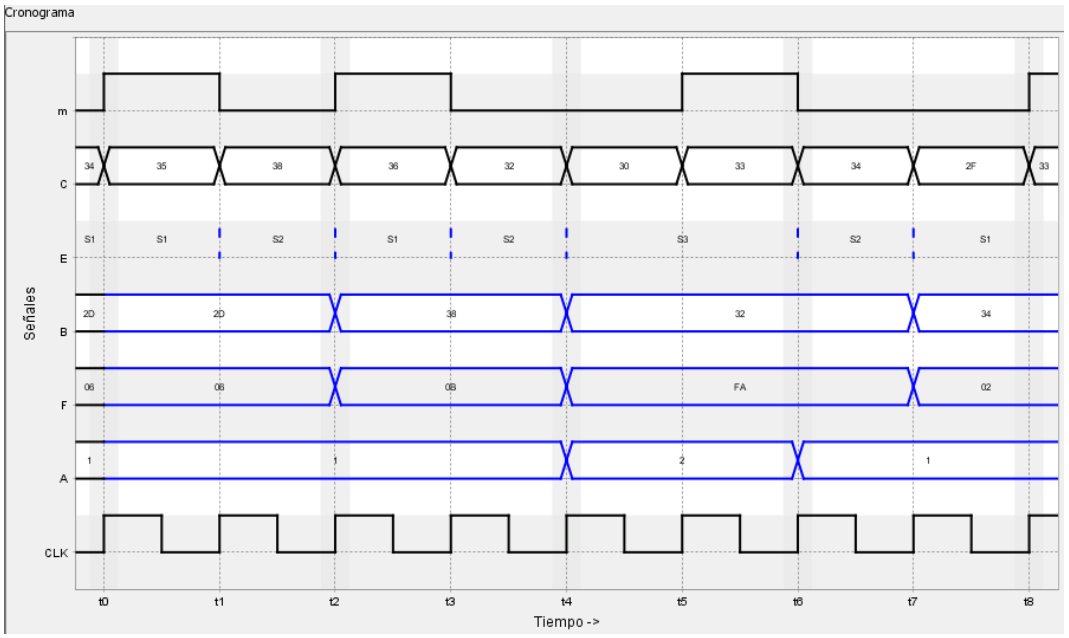
a) [15%] A partir del grafo del EFSM de la Fig. 1, completad en el cronograma siguiente la evolución del estado (E) y de las señales A, B y F en cada ciclo de reloj.

NOTA: Tenéis disponible el ejercicio en VerilChart (20242_PR_1a_vch) 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.

Al hacer el seguimiento del grafo a partir de las condiciones iniciales que nos indica el cronograma, podemos saber cómo evolucionan el estado, la señal A y las variables B y F.

En cada ciclo de reloj se calcula el valor que tomarán las variables B y F según la situación determinada por el estado. Hay que tener en cuenta que estas variables se actualizan al final del ciclo, en el flanco ascendente, con los valores de las variables justo antes del flanco. Por otro lado, el valor de la señal B depende directamente del estado en el que se encuentra la máquina y, a su vez, se determina en el mismo ciclo teniendo en cuenta el estado actual.

Con estas indicaciones, y siguiendo el diagrama de estados, el cronograma queda de la siguiente manera:



Captura del cronograma realizado en VerilChart

b) [15 %] Obtened 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 F10.

En el grafo está la señal de entrada m, que condiciona las transiciones. También están las variables B y F, de las cuales F determina las transiciones en función de sus valores. En dicho grafo, la condición del valor de F es F>Ah, la cual puede ser 0 (*false*) o 1 (*true*).

TABLA DE ESTADOS		
ESTADO ACTUAL	**ENTRADA**	**ESTADO SIGUIENTE**
S0	x	S1
S1	m'	S1
S1	m	S2
S2	m	S2
S2	m' · F10	S3
S2	m' · F10'	S1
S3	m'	S3
S3	m	S2

TABLA DE SALIDAS	
ESTADO	**SALIDAS**
S0	A=0, B^+=C, F^+=0
S1	A=1
S2	A=1, B^+=C, F^+=C-B
S3	A=2

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 0 para el estado inicial hasta el número de estados que haya menos 1: 00 para S0, 01 para S1, 10 para S2 y 11 para S3.

Para elegir el valor siguiente que tenemos que asignar a la variable B necesitamos un selector, que denominaremos s_B. En la tabla siguiente tenemos los valores entre los que se elige y una posible codificación binaria de cada uno de ellos:

TABLA DE OPERACIONES DE B		
OPERACIÓN	**SELECTOR s_B**	**EFECTO SOBRE LA VARIABLE**
$B^+=C$	0	Inicializarla al mismo valor que la señal C.
$B^+=B$	1	Mantenimiento del contenido.

De forma similar, usaremos un selector, s_F, para seleccionar el valor siguiente de la variable F:

TABLA DE OPERACIONES DE F		
OPERACIÓN	**SELECTOR s_F**	**EFECTO SOBRE LA VARIABLE**
$F^+=0$	00	Inicializarla a 0.
$F^+=F$	01	Mantenimiento del contenido.
$F^+=C-B$	10	Inicializarla al resultado de restar la señal C menos la variable B.

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

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

TABLA DE TRANSICIONES				
ESTADO ACTUAL		**ENTRADA**		**ESTADO⁺**
SÍMBOLO	$q_2q_1q_0$	**F10**	**m**	$q_2^+q_1^+q_0^+$
S0	00	x	x	01
S1	01	x	0	01
S1	01	x	1	10
S2	10	x	1	10
S2	10	1	0	11

S2	10	0	0	01
S3	11	x	1	10
S3	11	x	0	11

TABLA DE SALIDAS				
ESTADO ACTUAL		SALIDAS		
SÍMBOLO	$q_2q_1q_0$	s_B	s_F	A
S0	00	0	0	00
S1	01	x	x	01
S2	10	0	1	01
S3	11	x	x	10

Es importante mencionar que, según la codificación elegida por los estados y por los selectores, el resultado de las tablas anteriores puede variar.

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`).

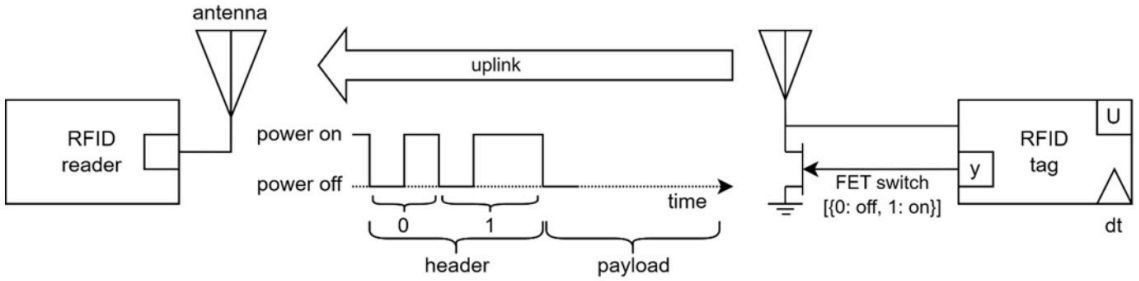


Fig. 2. La etiqueta pasiva (derecha) emite un código `U` cada vez que recibe suficiente energía a través de la antena.

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.

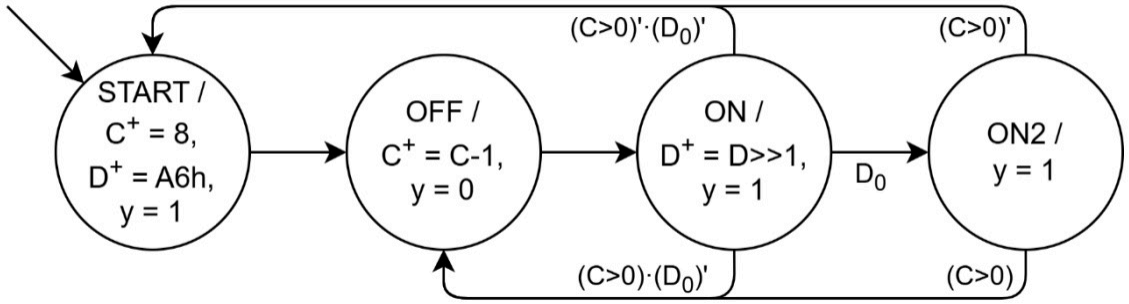


Fig. 3. EFSM de un emisor de la identificación de una etiqueta pasiva.

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	DÓNDE
DÓNDE	0	0	START
DÓNDE	0	1	OFF
DÓNDE	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
DÓNDE	10	1	10	10
ON2	11	1	10	01

NOTA: Tenéis disponible el ejercicio en VerilCirc (`20242_PR_1c_circ`), y, si queréis comprobar previamente la corrección de una ROM válida con palabras de 7 bits, la tenéis disponible en VerilChart (`20242_PR_1c_TB`). 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 `D0` y (`C>0`) son `D0` y `Cno0`, 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, puesto que hay cuatro estados) y los de las señales de entrada (2 más por las señales D_0 y `(C>0)`). 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 un lado, hay una salida directa (`y`). Además, se tienen que definir las salidas que controlarán las operaciones a realizar con las variables `s_C` y `s_D`.

Para elegir el valor siguiente que tenemos que asignar a la variable `C` necesitamos un selector, que denominaremos `s_C`. En la tabla siguiente tenemos los valores entre los que se elige y una posible codificación binaria de cada uno de ellos:

TABLA DE OPERACIONES DE `C`		
OPERACIÓN	**SELECTOR `s_C`**	**EFECTO SOBRE LA VARIABLE**
$C^+=8$	00	Inicializarla a 8.
$C^+=C-1$	01	Restarle 1.

C⁺=C	10	Mantenimiento del contenido.

De forma similar, usaremos un selector, s_D, para seleccionar el valor siguiente de la variable D:

TABLA DE OPERACIONES DE D		
OPERACIÓN	**SELECTOR s_D**	**EFECTO SOBRE LA VARIABLE**
D⁺=A6h	00	Inicializarla a A6h.
D⁺=D	01	Mantenimiento del contenido.
D⁺=D>>1	10	Desplaza los bits de D una posición a la derecha.

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 suma un total de 7 bits.

Así pues, si se usa una única ROM, tendrá que ser de 16x7 bits. El contenido de la ROM se determina a partir de la tabla de transiciones y la de salidas (los casos con *don't cares* los hemos considerado como valores 0):

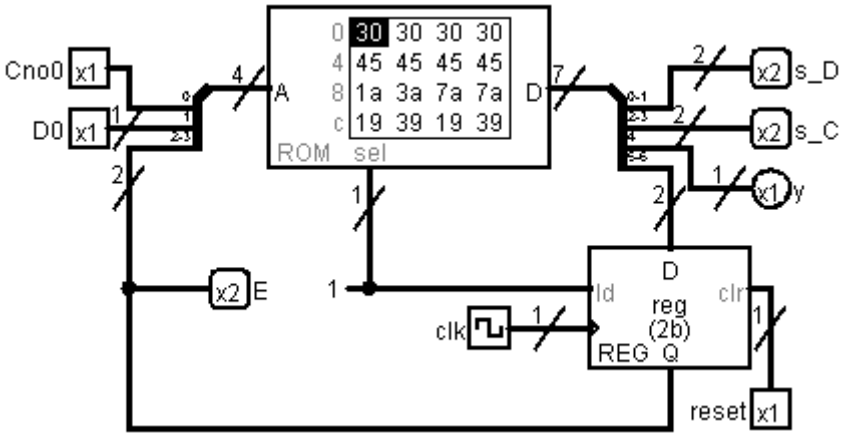
Tabla de verdad

E	D0	Cno0	s_E	y	s_C	s_D
START	0	0	OFF	1	0	0
START	0	1	OFF	1	0	0
START	1	0	OFF	1	0	0
START	1	1	OFF	1	0	0
OFF	0	0	ON	0	1	1
OFF	0	1	ON	0	1	1
OFF	1	0	ON	0	1	1
OFF	1	1	ON	0	1	1
ON	0	0	START	1	2	2
ON	0	1	OFF	1	2	2
ON	1	0	ON2	1	2	2
ON	1	1	ON2	1	2	2
ON2	0	0	START	1	2	1
ON2	0	1	OFF	1	2	1
ON2	1	0	START	1	2	1
ON2	1	1	OFF	1	2	1

Captura de la tabla de la ROM hecha en VerilChart

La figura siguiente muestra la implementación con la ROM correspondiente. La entrada A 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 usa un registro de 2 bits. También se podrían haber usado 2 biestables en su lugar.



Captura del circuito hecho en VerilCirc

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 disponible el ejercicio en VerilCirc (`20242_PR_1d_circ`). 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.

A partir del diseño del apartado anterior, podemos deducir las señales necesarias para las transiciones y para actualizar y almacenar el valor de las variables que se usan.

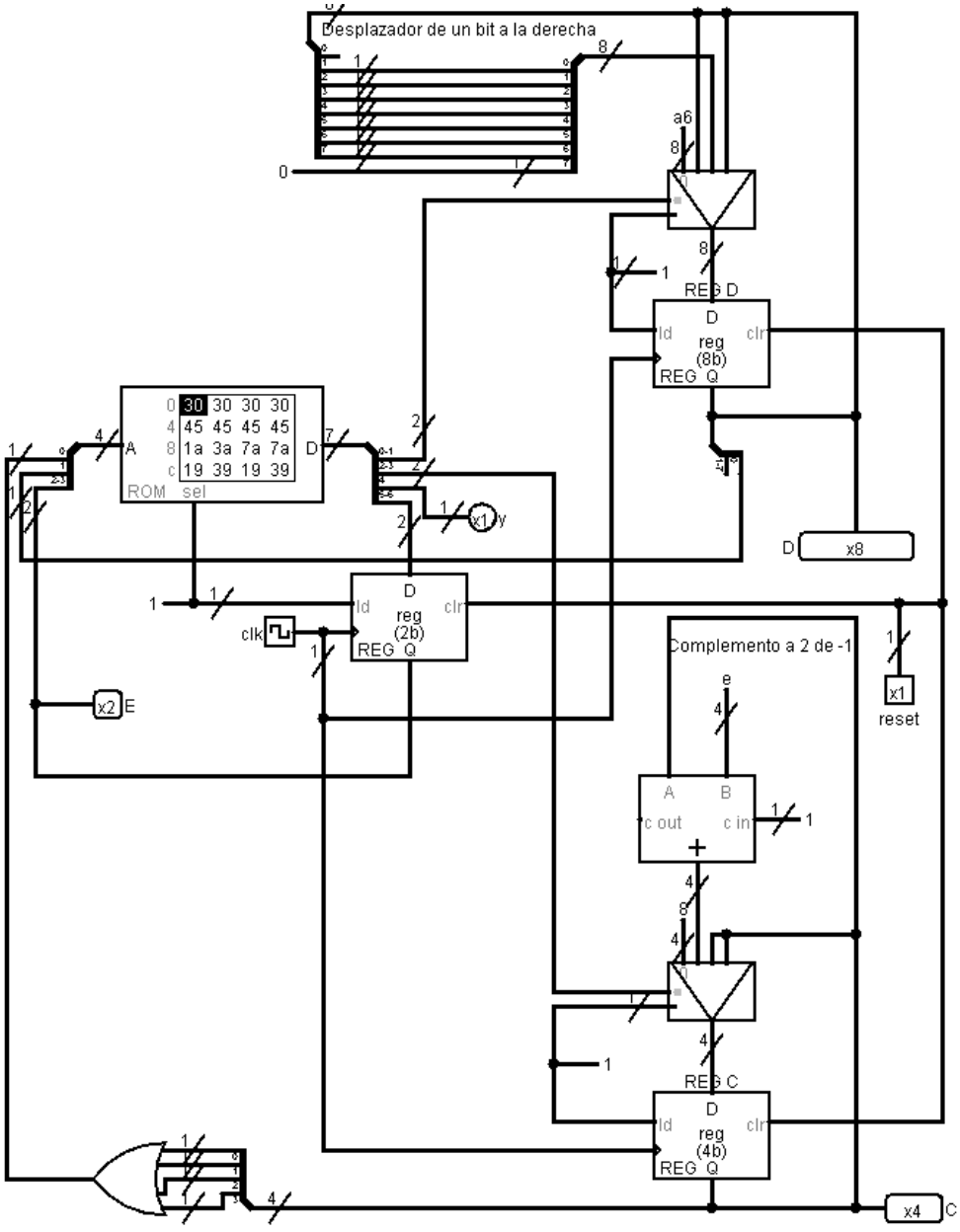
La variable `C` debe tener una medida de 4 bits, puesto que es un contador con un valor máximo de 8 ($1000_{(2}$). Sobre esta variable `C`, tal como se determina en el apartado anterior, se establecen tres operaciones posibles: disminuir su valor en una unidad, inicializarla a 8 o mantener su valor previo. Por tanto, la implementación debe utilizar un selector (MUX) para determinar el valor que se almacena en el registro de `C`, en función del valor de salida `s_C` determinado por la unidad de control. Este selector elegirá una de las operaciones mencionadas anteriormente. Cabe destacar que la resta se ha implementado como una suma del valor de `C` más `Eh`, el cual representa el número -1 en complemento a 2.

Finalmente, y puesto que la unidad de control utiliza la señal de entrada (`C>0`), hay una puerta lógica OR con cuatro entradas, donde cada una de ellas es un bit de `C`. Si la salida del OR es 1, significa que uno de los bits de `C` es 1, lo que implica que el valor de `C` es mayor que 0.

De manera similar, la variable `D` debe tener una medida de 8 bits, puesto que es un contador de valor máximo `A6h` ($10100110_{(2}$). Las operaciones sobre esta variable son inicializarla a `A6h`, mantener su valor anterior o desplazar los bits una posición a la derecha. Por eso, y con un MUX de tres entradas de selección controladas por la salida `s_D` de la unidad de control, y tal como se ha establecido en el apartado anterior, seleccionarán entre las operaciones posibles mencionadas anteriormente.

Es importante señalar que, en lugar de utilizar un desplazador, se han utilizado dos separadores para desplazar los bits de `D` un bit a la derecha para simplificar el circuito.

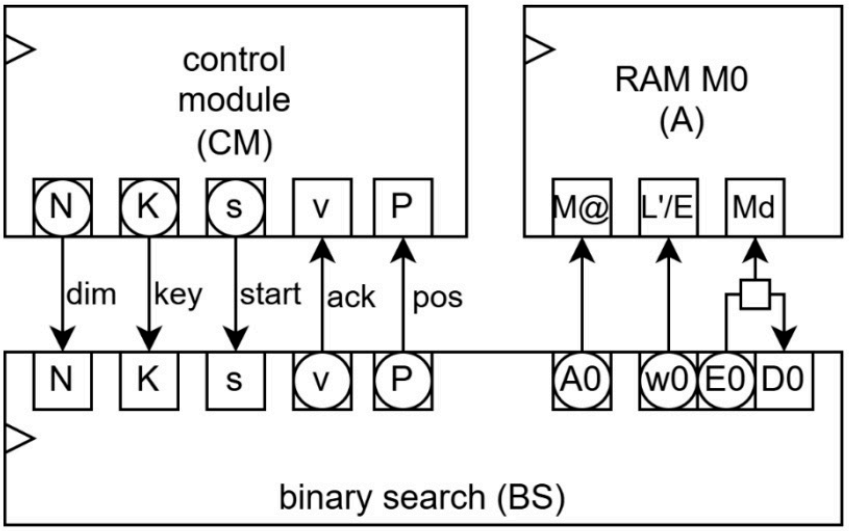
El circuito correspondiente queda como se muestra a continuación:



Captura del circuito hecho en VerilCirc

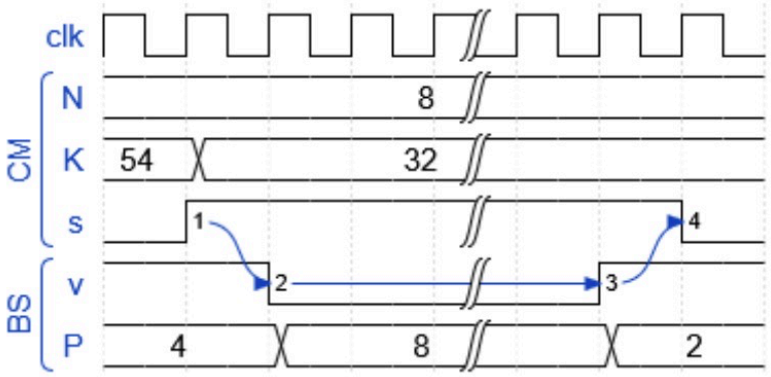
SEGUNDA PARTE [35 %]

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 de 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 que `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 que `s` pasa a 1 desde 0) hasta que lo acaba (ciclo en el que `v` vuelve a 1 desde 0) se describe en el siguiente programa en C:

```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 position
    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 (`A0`), la señal de salida que indica si se hace una lectura o una escritura (`w0`), la salida que contiene los datos a escribir (`E0`) y la entrada que proporciona lo que se lee (`D0`). En este caso, `w0` siempre será 0 y `E0` 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.

Inicialmente, es necesario esperar a que la variable `s` se active y, hasta que esta no sea `s=1`, permanecer en el estado `WAIT`. En este estado, la variable `v`, que indica cuándo el valor de `P` es válido, vale 1. Además, las variables `w0` y `E0` se inicializan a 0, ya que no se realizarán escrituras en la memoria.

El resto de las inicializaciones se realizan en el estado `INIT`, en el que se inicializan las variables y se indica el comienzo de la búsqueda binaria. Se inicializa la variable `P` a `N`, ya que es el valor por defecto que tendrá si no encuentra la posición de `K` dentro del vector `A`. Por otro lado, también se inicializan las siguientes variables de control:

- `L`: Límite izquierdo de la sublista donde se busca. Se inicializa a 0 y se actualiza cuando se descarta la izquierda.
- `R`: Límite derecho de la sublista donde se busca. Se inicializa a `N-1` y se actualiza cuando se descarta la derecha.
- `M`: Posición media del rango actual de búsqueda: `M=(L+R)>>1`. Se usa como dirección de lectura.

Una vez iniciada la búsqueda, se solicita una lectura en la memoria para obtener el valor del vector en la posición `M`, cuyo estado denominaremos `READ_M`. En este estado, el valor de `A0` es el de `M`, ya que contiene la dirección de la memoria que se va a leer.

A continuación, se compara el valor leído desde la memoria `D0` con el elemento `K`. Si ambos valores son iguales, significa que se ha encontrado el valor de `K` dentro del array `A`, por lo que pasamos al estado `FOUND`, en el que `P` se iguala a la posición en memoria (`M`) en la que se encuentra dicho elemento. Si `D0` y `K` no son iguales y `L` es menor que `R`, se comprueba si `L` es mayor que `R`. En caso de ser así, esto significa que ya no quedan más elementos por comprobar, por lo que pasamos al estado `NOT_FOUND`. En este estado, se vuelve a inicializar `P` con el valor de `N`, a pesar de que ya se hace en el estado `INIT`, con la intención de dejar claro que, si no se encuentra a `K` en el array `A`, el valor de `P` es `N`.

Cuando `L` es menor que `R`, comprobamos si `D0` es menor que `K` para saber si tenemos que avanzar en el límite izquierdo, el cual tiene el estado `UPDATE_L` (`L = M + 1`), o retroceder en el límite derecho, que tiene el estado `UPDATE_R` (`R = M - 1`).

Una vez se ajusta el límite correspondiente, pasamos al último estado, `UPDATE_M`, en el que se recalcula la posición media en función de los valores de los nuevos límites (`M = (L + R) >> 1`). Tras esto, se vuelve al estado `READ_M` para realizar una nueva iteración sobre el array y continuar con el proceso descrito anteriormente hasta

llegar al estado `FOUND` o `NOT_FOUND`. Cuando se llega a uno de esos dos estados, la búsqueda finaliza y vuelve al estado inicial `WAIT`.

Este es el diseño de la ASM con todos los estados e inicializaciones descritos: