TALLER FSM: FSM COMO UN CIRCUITO ÁRBITRO

El propósito de la máquina es controlar el acceso de varios dispositivos a un recurso compartido en un sistema. Sólo uno a la vez puede utilizar el recurso. Supóngase que todas las señales en el sistema pueden cambiar los valores únicamente en el flanco positivo de la señal de reloj. Cada dispositivo proporciona una entrada a la FSM, llamada **solicitud**, y la FSM produce una salida independiente para cada dispositivo, denominada **concesión**.

Un dispositivo indica que necesita usar el recurso al validar su señal de solicitud. Siempre que el recurso compartido no está ya en uso, la FSM considera todas las solicitudes que estén activas. Con base en un **esquema de prioridad**, selecciona uno de los dispositivos que hacen una solicitud y valida su señal de concesión. Cuando el dispositivo termina de usar el recurso, invalida su señal de solicitud.

Supondremos que hay tres dispositivos en el sistema, llamados **dispositivo 1, dispositivo 2 y dispositivo 3.** Es fácil ver cómo la FSM puede ampliarse para manejar más dispositivos. Las señales de solicitud se llaman **r1, r2 y r3**, y las señales de concesión **g1, g2 y g3**.

Un nivel de prioridad se asigna a los dispositivos de tal manera que el dispositivo 1 tiene la mayor prioridad, el dispositivo 2 la prioridad siguiente en importancia y el dispositivo 3 la menor prioridad. Por tanto, si se valida más de una señal de solicitud cuando la FSM asigna una concesión, ésta se da al dispositivo que tiene la solicitud con mayor prioridad. Un diagrama de estado para la FSM buscada, diseñada como una máquina tipo Moore, puede observar en la figura 1.

Al principio, al inicializarla, la máquina se halla en el estado llamado **Idle (inactivo).** Ninguna señal de concesión se valida y el recurso compartido no está en uso. Hay otros tres estados, llamados **gnt1, gnt2 y gnt3**. Cada uno de ellos valida la señal para uno de los dispositivos.

La FSM permanece en el estado Idle siempre que todas las señales de solicitud sean 0. En el diagrama de estado la condición r1r2r3 = 000 se indica por medio del arco etiquetado 000.

Cuando una o más señales de solicitud se vuelven 1, la máquina pasa a uno de los estados de concesión, de acuerdo con el esquema de prioridad. Si r1 se valida, entonces el dispositivo 1 recibirá la concesión porque tiene la prioridad más alta. Esto se indica con el arco etiquetado 1xx que conduce al estado gnt1, el cual establece g1=1. El significado de 1xx es que la señal de solicitud r1 es 1, y los valores de las señales r2 y r3 son irrelevantes debido al **esquema de prioridad**. Como antes, se usa el símbolo x para indicar que el valor de la variable correspondiente puede ser 0 o 1.

La máquina permanece en el estado gnt1 mientras r1 sea 1. Cuando r1= 0, el arco etiquetado 0xx produce en el **siguiente flanco positivo del reloj** un cambio de regreso al estado Idle y g1 se invalida.

Si se activan otras solicitudes en este momento, entonces la FSM cambiará a un nuevo estado de concesión después del siguiente flanco activo del reloj.

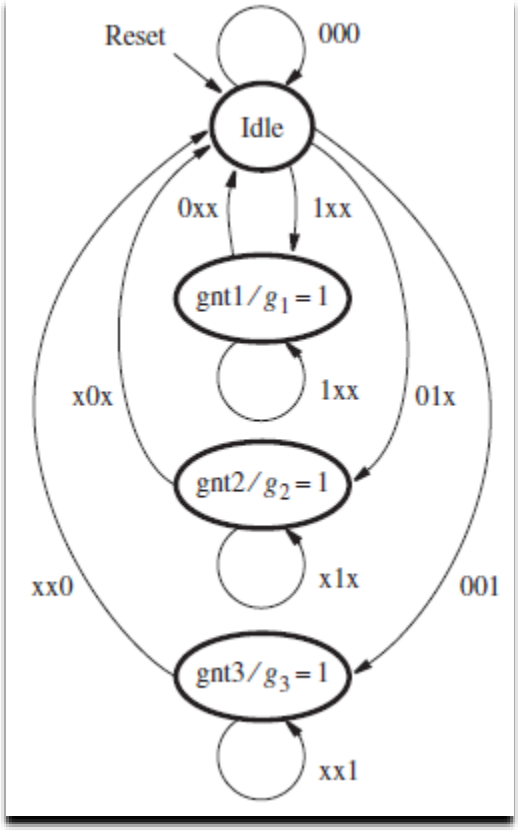

Figura 1. Diagrama de estado.

El arco que ocasiona un cambio en el estado gnt2 se llama 01x. Esta etiqueta se apega al esquema de prioridad porque representa la condición r2 =1, pero r1=0. De forma similar, la condición para introducir el estado gnt3 se da como 001, lo cual indica que la única señal de solicitud validada es r3.

El diagrama de estado se repite en la figura 2. La única diferencia entre este diagrama y el de la figura 1 es la forma en que se etiquetan los arcos. En la figura 2 se utiliza un esquema de etiquetado más simple, más intuitivo. Para la condición que lleva del estado Idle al estado gnt1, el arco se etiqueta r1 en vez de 1xx. Esta etiqueta significa que si r1=1, la FSM cambia al estado gnt1, independientemente de las otras entradas. El arco con la etiqueta r1r2 que lleva del estado Idle a gnt1 representa la condición r1r2=01, mientras que el valor de r3 es irrelevante. No existe un esquema estandarizado para etiquetar los arcos en los diagramas de estado. Algunos diseñadores prefieren el estilo de la figura 1; otros gustan más de un estilo parecido al de la figura 2.

En la figura 3 se presenta el código de VHDL para la máquina. Las tres señales de solicitud y de concesión se especifican como señales STD__LOGIC_ VECTOR de tres bits. La FSM se describe con una instrucción CASE. Como se muestra en la cláusula WHEN para el estado Idle, es fácil describir el

esquema de prioridad requerido. Si la instrucción IF especifica que si r1=1, entonces el estado siguiente para la máquina es gnt1. Si r1 no se valida, entonces la condición ELSIF se evalúa, la cual estipula que si r2 =1 entonces el estado siguiente será gnt2. Cada cláusula ELSIF sucesiva considera una señal de solicitud de la prioridad más baja sólo si todas las señales de solicitud de la prioridad más alta no se validan.

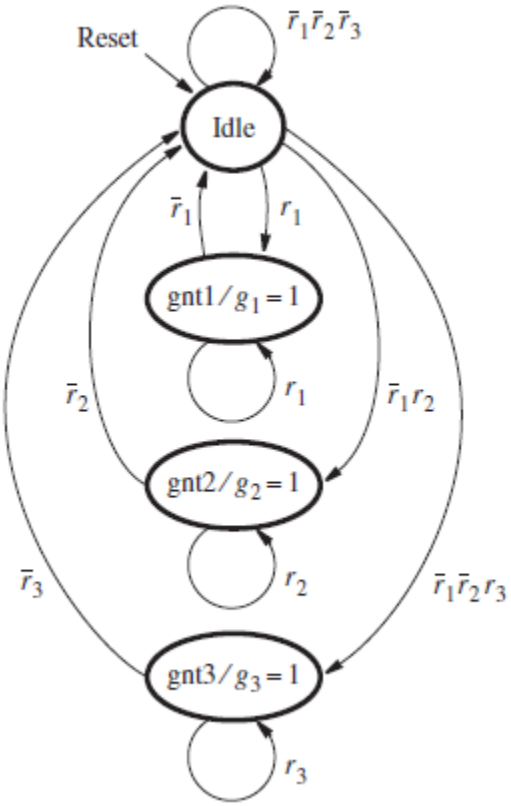

Figura 2. Estilo alternativo del diagrama de estado.

La cláusula WHEN para cada estado de concesión es sencilla. Para el estado gnt1 especifica que mientras r1 =1, el estado siguiente permanece en gnt1. Cuando r1=0, el estado siguiente es Idle. Los otros estados de concesión tienen la misma estructura. El código para las señales de concesión, g1, g2 y g3, se proporciona al final. Establece g1 en 1 cuando la máquina se halla en el estado gnt1; de otra forma, g1 se establece en 0. De manera similar, cada una de las otras señales de concesión es 1 sólo en el estado de concesión apropiado.

En vez de las tres instrucciones de asignación condicionales utilizadas para g1, g2 y g3, tal vez parezca razonable usar el proceso mostrado en la figura 4, el cual contiene una instrucción IF. Este código es incorrecto, pero el por qué? no es obvio. Cuando se usa una instrucción IF, si no hay una

cláusula ELSE o un valor predeterminado para una señal, entonces esa señal conserva su valor cuando la condición IF no se cumple. Esto se denomina **memoria implícita**.

En la figura 4 la señal g1 se establece en 1 cuando la FSM entra por primera vez en el estado gnt1, y luego g1 conservará el valor 1 sin importar a qué estado cambie la FSM. De igual modo, el código para g2 y g3 también es incorrecto. Si desea escribir el código que lleve una instrucción IF, entonces debería estructurarlo como se muestra en la figura 5. Para cada señal de concesión se asigna un valor predeterminado de 0, con lo que se evita el problema de la memoria implícita.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY arbiter IS
    PORT ( Clock, Resetn  : IN    STD_LOGIC ;
           r              : IN    STD_LOGIC_VECTOR(1 TO 3) ;
           g              : OUT   STD_LOGIC_VECTOR(1 TO 3) ) ;
END arbiter ;

ARCHITECTURE Behavior OF arbiter IS
    TYPE State_type IS (Idle, gnt1, gnt2, gnt3) ;
    SIGNAL y : State_type ;
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN y <= Idle ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            CASE y IS
                WHEN Idle =>
                    IF r(1) = '1' THEN y <= gnt1 ;
                    ELSIF r(2) = '1' THEN y <= gnt2 ;
                    ELSIF r(3) = '1' THEN y <= gnt3 ;
                    ELSE y <= Idle ;
                    END IF ;
                WHEN gnt1 =>
                    IF r(1) = '1' THEN y <= gnt1 ;
                    ELSE y <= Idle ;
                    END IF ;
                WHEN gnt2 =>
                    IF r(2) = '1' THEN y <= gnt2 ;
                    ELSE y <= Idle ;
                    END IF ;
                WHEN gnt3 =>
                    IF r(3) = '1' THEN y <= gnt3 ;
                    ELSE y <= Idle ;
                    END IF ;
            END CASE ;
        END IF ;
    END PROCESS ;
    g(1) <= '1' WHEN y = gnt1 ELSE '0' ;
    g(2) <= '1' WHEN y = gnt2 ELSE '0' ;
    g(3) <= '1' WHEN y = gnt3 ELSE '0' ;
END Behavior ;
```

Figura 3. Código de VHDL para el árbitro.

```
.
    PROCESS( y )
    BEGIN
        IF y = gnt1 THEN g(1) <= '1' ;
        ELSIF y = gnt2 THEN g(2) <= '1' ;
        ELSIF y = gnt3 THEN g(3) <= '1' ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Figura 4. Código de VHDL incorrecto para las señales de concesión.

```
    PROCESS( y )
    BEGIN
        g(1) <= '0' ;
        g(2) <= '0' ;
        g(3) <= '0' ;
        IF y = gnt1 THEN g(1) <= '1' ;
        ELSIF y = gnt2 THEN g(2) <= '1' ;
        ELSIF y = gnt3 THEN g(3) <= '1' ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Figura 5. Código de VHDL correcto para las señales de concesión.