**Práctica 5:** Construcción de máquinas de estado usando memoria con direccionamiento implícito

Laboratorio de Organización y Arquitectura de Computadoras

**Grupo:** 2

**Alumnos:**

* Guzmán Sánchez José Emmanuel
* Mejía Ortiz Aarón Enrique
* Sáenz Barragán Ricardo

Objetivo

Familiarizar al alumno en el conocimiento de construcción de máquinas de estados usando direccionamiento de memorias con el método de direccionamiento implícito.

**Dispositivo**: MAX10 DE-Lite 10M50DAF484C7G

1. Desarrollo

Para poder implementar el direccionamiento implícito es necesario llenar nuestra memoria de acuerdo a nuestra carta ASM. Lo primero que hicimos fue definir el código de estados y el código de entradas de nuestro sistema, es decir, la representación binaria de cada uno dentro del mismo. Este paso es de suma importancia, ya que cuando se tienen entradas condicionales se deben de codificar los estados de modo que en al menos un rama sólo se haga un incremento. Nuestra codificación quedó de la siguiente manera (La variable vale 1 por defecto).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Estados** | **Código** |  |  |  |
| EST0 | 000 |  | **Entradas** | **Código** |
| EST1 | 100 |  | A | 00 |
| EST2 | 010 |  | B | 01 |
| EST3 | 001 |  | C | 10 |
| EST4 | 011 |  |  | 11 |

Figura 1. Representación binaria de estados y entradas

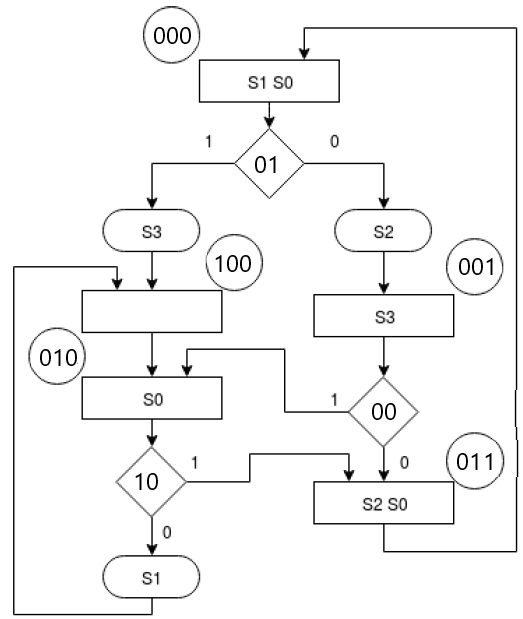
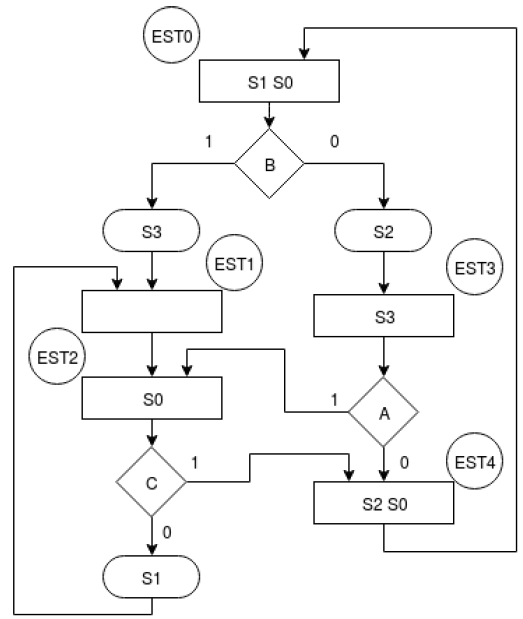


Figura 2. Carta ASM con la representación binaria de estados y entradas

Una vez que definimos la representación de nuestras entradas y estados, fue necesario llenar una tabla de transición de estados, para poder llevar la interacción entre entradas y estados a un nivel más lógico.

Debido a que tenemos salidas condicionales, es necesario agregar un campo extra a la memoria, el cual será la salida cuando la variable sensada sea falsa, para tomar en cuenta ambas ramas de la condición. Se escogerá una rama u otra con un multiplexor el cual tendrá como señal de entrada . La tabla con esta información se muestra a continuación.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Presente** | | | **Prueba** | | **VF** | **Liga** | | | **Salidas F.** | | | | **Salidas V.** | | | |
| P2 | P1 | P0 | K1 | K0 | L2 | L2 | L1 | L0 | S3 | S2 | S1 | S0 | S3 | S2 | S1 | S0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figura 3. Tabla de transición de estados

Una vez que llenamos la tabla, concatenado los bits de prueba, VF, los bits de liga y la salida tanto en su rama falsa como verdadera, podemos obtener el contenido que tendrá la memoria en la dirección apuntada por los bits correspondientes al estado presente.

Con lo anterior, podemos definir el comportamiento que tendrá la memoria ROM en VHDL. Como se observa, primero obtenemos todo el contenido de la memoria de acuerdo al valor del estado presente. Una vez obtenido, podemos separar cada uno de sus elementos.

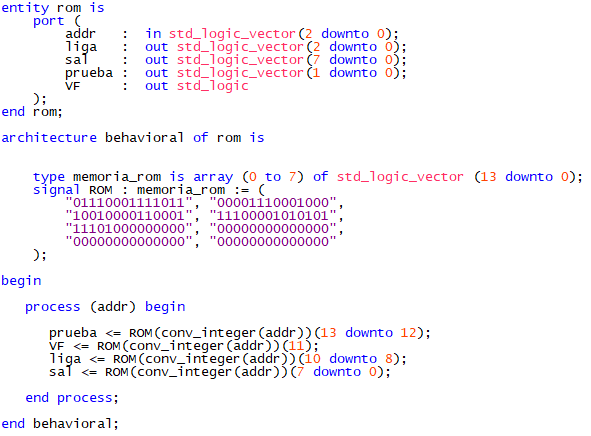


Figura 4. Código del comportamiento de la memoria

Con el contenido de la memoria obtenemos el valor de la prueba y de VF. Con estos valores ya podemos multiplexar las entradas y además obtener el selector que nos dirá si el siguiente valor será de la liga o del incrementador.

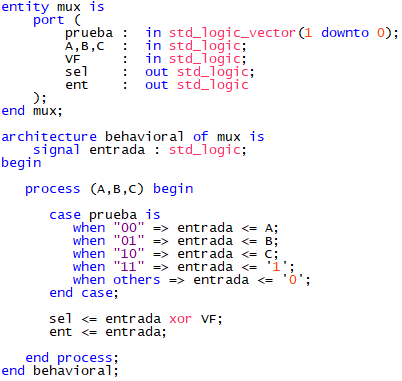


Figura 5. Código del comportamiento del multiplexor para las entradas y el selector

Con el valor obtenido de la prueba podemos multiplexar las salidas, debido a que acepta salidas condicionales, habrá una salida para la rama falsa y una para la rama verdadera. Con el selector, resultado de hacer XOR entre VF y el valor dado por la prueba, multiplexamos el estado siguiente, si es 1, entonces el estado presente será el incremento, sino entonces será el de la liga.

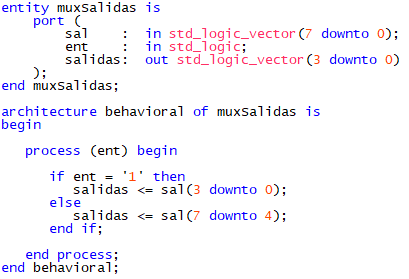
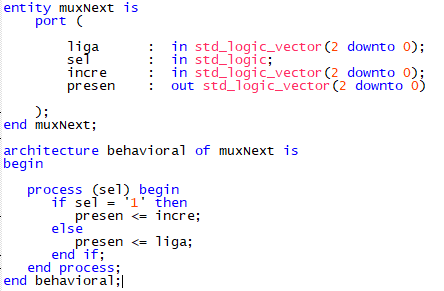
 

Figura 6. Código del multiplexor de salidas Figura 7. Código del multiplexor para el estado siguiente

Para poder agregar el *reset*, es necesario un bloque de registro, que mantenga su valor y cuando se active el *reset* ponga el estado presente a 0. Este mismo bloque alimentará al incrementador, el cual sólo leerá el valor del estado presente y lo incrementará en una unidad.

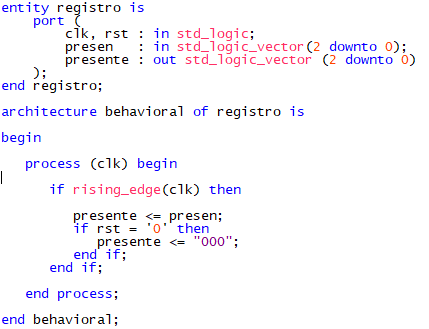
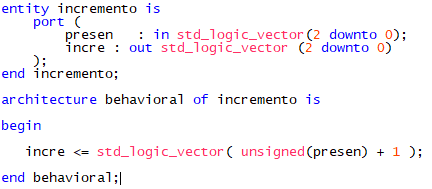
 

Figura 8. Código del multiplexor de salidas Figura 9. Código del multiplexor para el estado siguiente

Finalmente, para poder mostrar las salidas es necesario agregar un bloque que nos permita conmutar entre las salidas deseadas, ya sea visualizar el contenido de la memoria (liga y salidas efectivas) o el valor del estado presente y la prueba

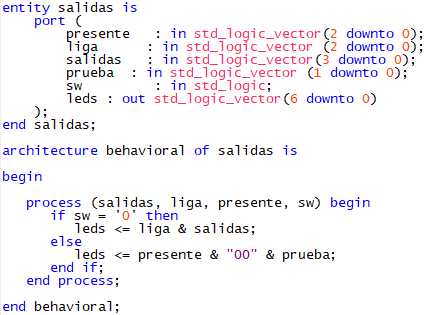


Figura 10. Código para visualizar las salidas

Con los bloques anteriores podemos formar todo nuestro circuito, agregando un bloque extra, un divisor de frecuencia para poder visualizar adecuadamente las salidas

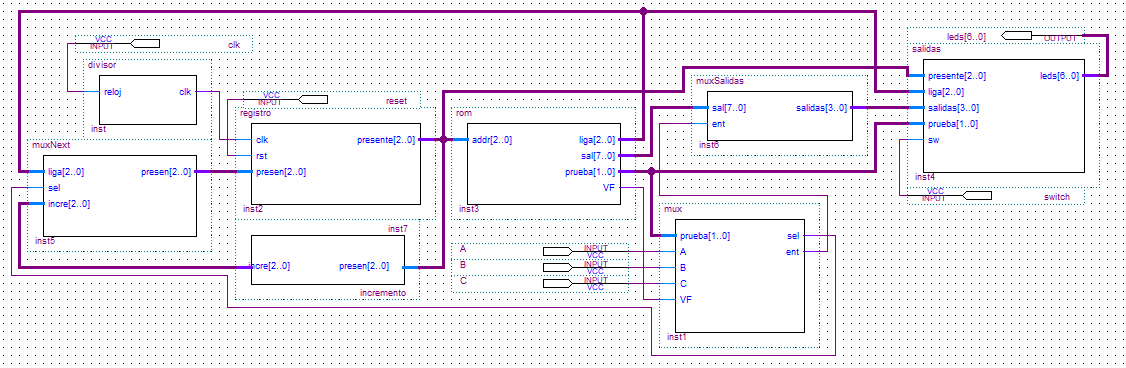


Figura 11. Diagrama de bloques del sistema

1. Simulaciones

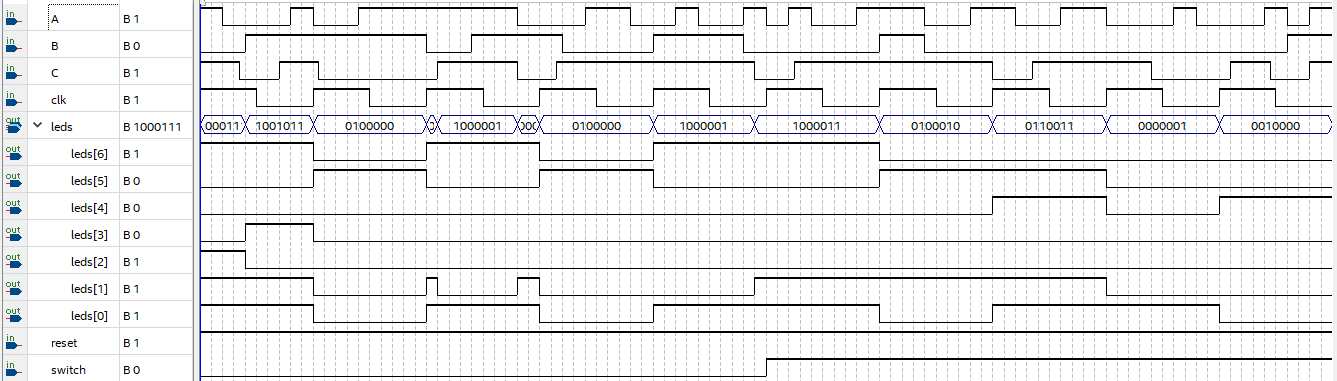


Figura 12. Simulación del sistema

En los primeros ciclos de reloj, debido a que la entrada del switch está en 0, las salidas mostradas en los leds serán las referentes al contenido de la memoria (la liga, 6-4, y salidas efectivas, 3-0).

Iniciamos en el estado 000, por lo tanto la liga es 100, y debido a que el valor de B es 1, entonces el estado siguiente es el de la liga. Podemos apreciar esto, ya que en el siguiente ciclo de reloj, el valor de la liga es 010 y todas las salidas están apagadas.

Cuando activamos el switch, en los leds se muestran el estado actual (leds 6-4) y el valor de la prueba (leds 1 y 0). Podemos apreciar que cuando el switch se activa se muestra el estado 000 y la prueba 11, que es la correspondiente a la variable auxiliar para hacer los saltos forzados. No se muestra el valor que corresponde de la prueba debido a que muestra los del estado anterior, ya que proviene del estado 011, donde su liga es 000 y la prueba 11, pero al siguiente ciclo de reloj se muestra los valores que deben de ser.

1. Conclusiones

Guzmán Sánchez José Emmanuel

Usar el direccionamiento implícito nos da una ventaja respecto al de entrada-estado debido a que logramos eliminar una de las ligas de la memoria, ya que ésta se sustituye por un incremento, sin embargo se mantienen las salidas para ambas ramas. Lo complicado de este tipo de direccionamiento es encontrar el adecuado código de estados que nos permita utilizar este tipo de direccionamiento. Este proceso se complica conforme el sistema va escalando y se vuelve más grande.

Mejia Ortiz Aarón Enrique

Este direccionamiento es muy útil ya que reduce la cantidad de memoria que se necesita. Pero a cambio, requiere de un mayor análisis y planeación para construir la carta ASM en la que se basará, ya que necesita tener uno de los siguientes estados como incremento. Tener cartas ASM muy grandes puede representar una inversión grande de tiempo, sobre todo si hay muchos saltos conectados.

Sáenz Barragán Ricardo

Con está práctica comprendí como al usar el direccionamiento implícito se simplifica más la forma en la que se guarda la memoria y en los cambios de estados, de igual manera comprendí como es posible simplificar el tiempo de ejecución de las instrucciones. Aunque ahora vemos un aumento en el hardware que se usa, la simplificación de los procesos que se realizan es mucho mayor y el funcionamiento que se logra para la interpretación de las cartas ASM es mucho mejor. Se puede ver como uno de los problemas con esta implementación es intentar escalara a sistemas más complejos.