**Práctica 6:** Secuenciador básico

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 del secuenciador básico, el cual es una parte fundamental del procesador

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

1. Desarrollo

Para poder implementar la carta ASM dada en el secuenciador básico es necesario llenar nuestra memoria de acuerdo al contenido de la misma. 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, ya que el secuenciador se basa en el direccionamiento implícito. Nuestra codificación quedó de la siguiente manera (La variable vale 1 por defecto).

|  |  |
| --- | --- |
| **Entradas** | **Código** |
| X | 00 |
| Y | 01 |
| INT | 10 |
|  | 11 |

Figura 1. Representación binaria de entradas

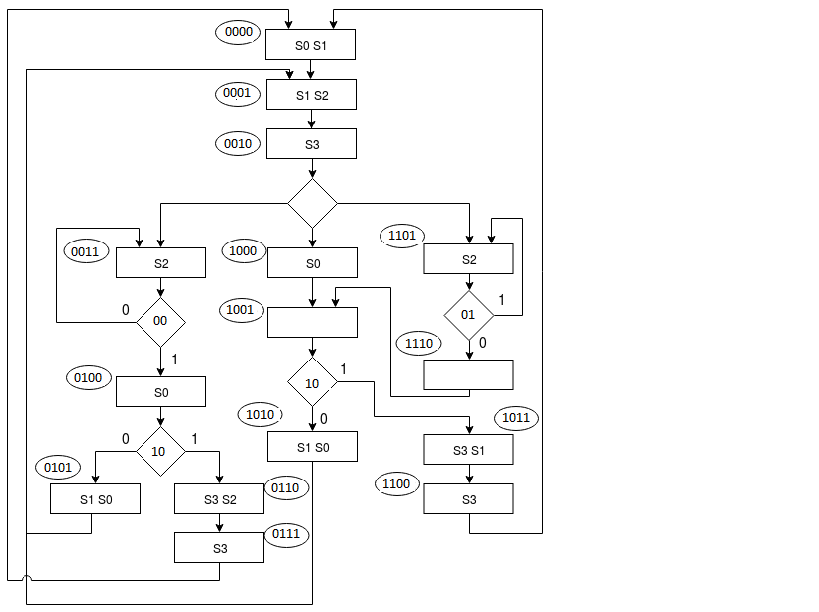


Figura 2. Carta ASM con la representación binaria de estados y entradas

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. Ya que no tenemos salidas condicionales, no es necesario agregar un campo extra a la memoria.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Presente** | | | | **Prueba** | | **VF** | **MI** | | **Liga** | | | | **Salidas** | | | |
| P3 | P2 | P1 | P0 | K1 | K0 |  | I1 | I0 | L3 | L2 | L1 | L0 | S3 | S2 | S1 | S0 |
| 0 | 0 | 0 | 0 | \* | \* | \* | 0 | 0 | \* | \* | \* | \* | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | \* | \* | \* | 0 | 0 | \* | \* | \* | \* | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | \* | \* | \* | 1 | 0 | \* | \* | \* | \* | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | \* | \* | \* | \* | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | \* | \* | \* | 0 | 0 | \* | \* | \* | \* | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | \* | \* | \* | 0 | 0 | \* | \* | \* | \* | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | \* | \* | \* | \* | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | \* | \* | \* | 0 | 0 | \* | \* | \* | \* | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 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, MI, los bits de liga y la salida podemos obtener el contenido que tendrá la memoria en la dirección apuntada por los bits correspondientes al estado presente.

Para saber de dónde tomará el estado siguiente es necesario definir cómo se activarán las salidas de acuerdo a las entradas, para ello definimos la lógica interna como se muestra a continuación.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **I1** | **I0** |  | **Selector** |  |  |  |
| 0 | 0 | \* | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | \* | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 |

Figura 4. Lógica interna con compuerta XOR

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.

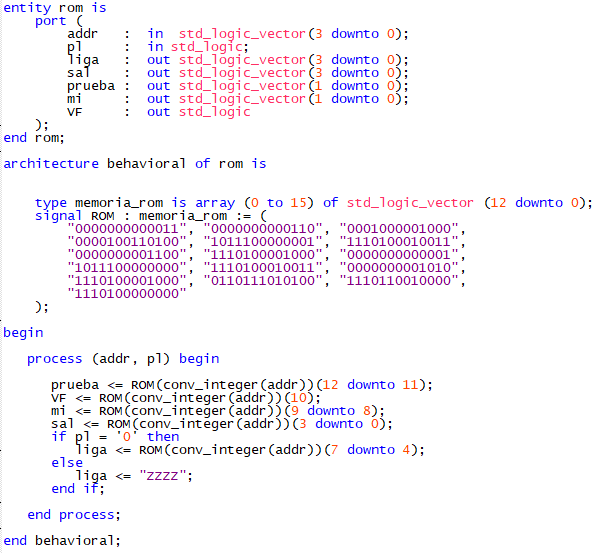


Figura 5. Código del comportamiento de la memoria

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

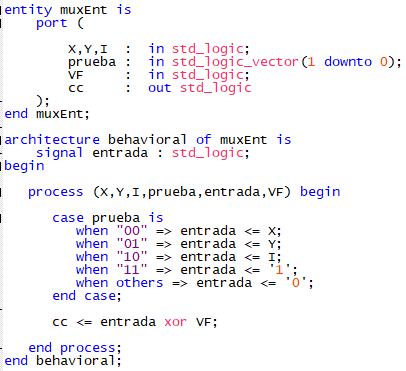


Figura 6. Código del comportamiento del multiplexor para las entradas y el selector

Con el selector, resultado de hacer XOR entre VF y el valor dado por la prueba, multiplexamos el estado siguiente, si es 0, entonces el estado presente será el incremento, sino entonces será el que venga por el bus de datos, que está conectado a todos lo registrosg.

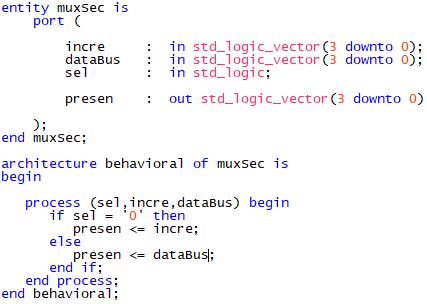


Figura 7. Código del multiplexor para el estado siguiente

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.

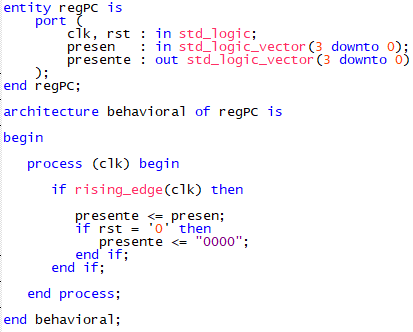
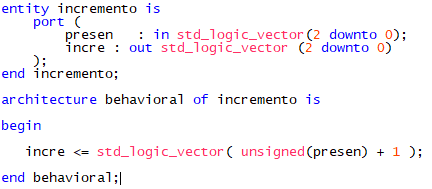
 

Figura 8. Código del registro de microprograma Figura 9. Código del registro de incremento

Además de los registros anteriores, es necesario agregar el registro de interrupción y el registro de transformación los cuales estarán conectados directamente al bus de datos del multiplexor. Éstos registros activaran sus salida con su bandera, si ésta se activa manda su valor, sino se pone en un estado de alta impedancia.

Los valores de estos registros son procesados por otra unidad independiente al secuenciador, pero en esta práctica los valores los tomará de switches.

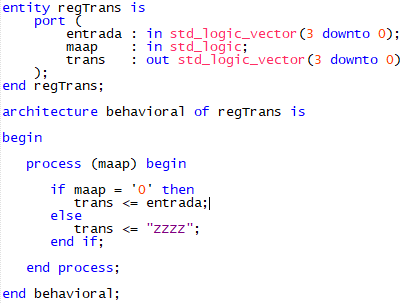
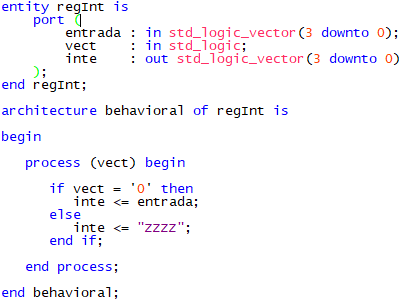
 

Figura 10. Código del registro de transformación Figura 11. Código del registro de interrupción

Para determinar cuál es el valor que se mandará por el bus de datos conectado al multiplexor, nos apoyamos de la lógica interna que, basado en el selector y el código de microinstrucción, nos dirá qué banderas se activarán y por consiguiente de dónde se tomará el estado siguiente.

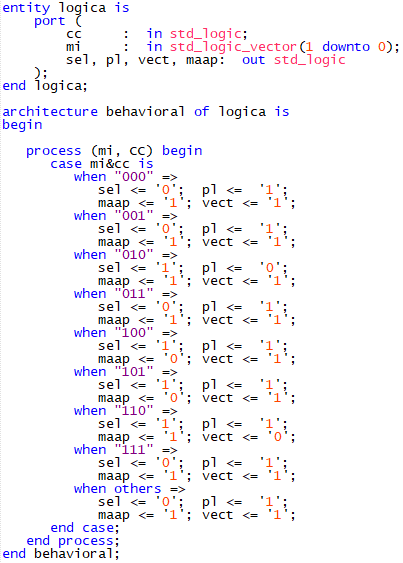


Figura 11. Código del registro de interrupción

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

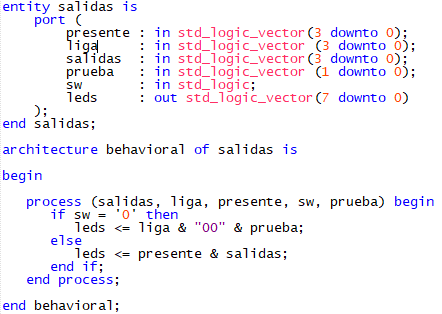


Figura 12. Código para visualizar las salidas

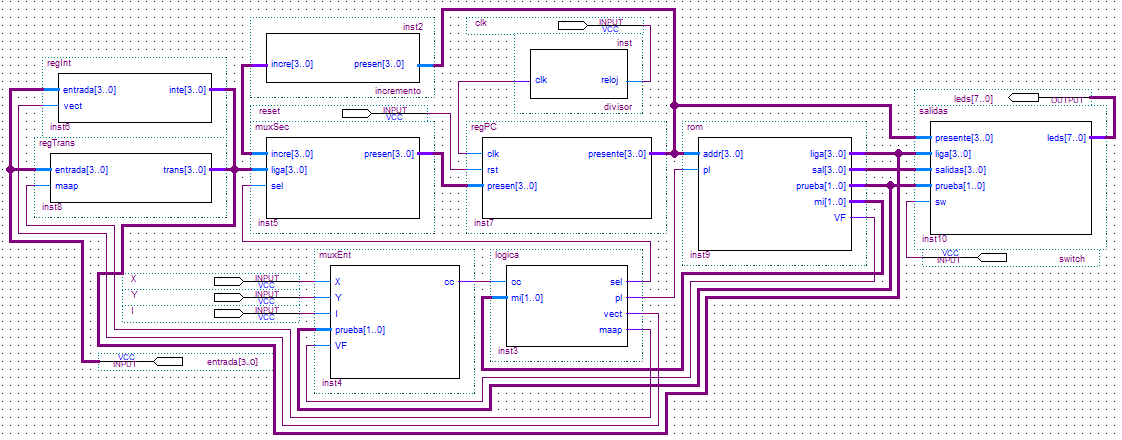
Con los bloques anteriores podemos formar todo nuestro circuito, agregando un bloque extra, un divisor de frecuencia para poder visualizar adecuadamente las salidas

Figura 13. Diagrama de bloques del sistema

1. Simulaciones

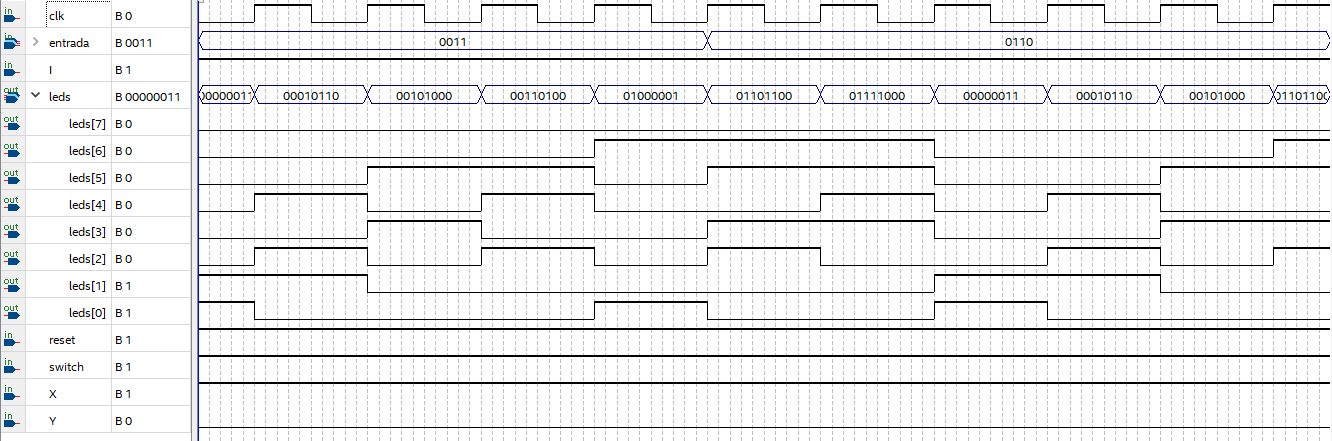


Figura 12. Simulación del sistema

Debido a que la entrada del switch está en alto, en los leds de salida se mostrará el estado presente y las salidas, que corresponde a los leds 7-4 y 3-0 respectivamente.

Antes del primer flanco de subida a la salida podemos ver que se inicia en el estado 0000 y se activan las salidas 1 y 0. Después del primer flanco de subida pasamos al estado 0001 activando sus salidas correspondientes y después al estado 0010 siguiendo el mismo proceso. Llegando a este estado se hace un salto de transformación tomando el estado siguiente del registro dedicado a este, el cual está representado por ***entradas***, así que salta al estado 0011 y activa las salidas correspondientes.

Después del estado 0011 da un salto condicional, correspondiente a la entrada **X**, como ésta está en alto, entonces pasa al estado 0100. Después de este estado hace un salto de interrupción, por lo que toma hace la condición dependiendo de la variable **I,** como está en alto, entonces toma el estado siguiente del registro de interrupción, que igual está definida por ***entradas***, por lo que salta al estado 0110, posterior a esto hace pasos continuos hasta regresar al estado 0000.

1. Conclusiones

Guzmán Sánchez José Emmanuel

Es importante comprender el funcionamiento del secuenciador, ya que éste nos dará pasó a comprender cómo se maneja la memoria en una arquitectura más compleja. Lo más importante es comprender cómo funcionan las banderas de activación ya que éstas son las que determinaran el flujo de nuestros datos. De igual manera, se debe de mantener un registro para cada componente, ya que cuando avancemos a la construcción de la arquitectura CISC, esto será de suma importancia para poder almacenar registros y variables.

Mejia Ortiz Aarón Enrique

Sáenz Barragán Ricardo