Reporte Tarea # 3

Brandon Jiménez Campos C33972

Resumen

En este reporte se presentará el diseño de un controlador de CPU, el cual realiza operaciones como cargar datos desde la memoria RAM, guardar datos en la memoria RAM, sumar, restar, la operación lógica and y or y activar una bandera de equal (igual). Para verificar el correcto funcionamiento de este diseño, se realizaron instrucciones al CPU provenientes de una memoria ROM, en donde se busca verificar el correcto funcionamiento de las operaciones y funciones del CPU. En donde gracias a estas pruebas se pudo confirmar a través del correcto comportamiento que el diseño del CPU es correcto y que cumple con todas las funcionalidades planteadas. Además, se realizó la síntesis del mismo, utilizando la libreria *cmos_cells.lib* mediante yosys y se obtuvo una correcta síntesis, por lo que en este reporte también se mencion cuántas compuertas y cuántos flip flops necesita para su creación y su correcto comportamiento ante las pruebas mencionadas anteriormente.

1. Descripción Arquitectónica

El diagrama de bloques utilizado para la creación de este diseño es el otorgado en el enunciado de la tarea al igual que la descripción de cada señal y cómo funciona. Por otro lado, el funcionamiento de la máquina de estados se encuentra en la figura 1, en la cual se detalla las condiciones para poder pasar de estados, el nombre de los estados y las salidas que se llegarán a activar en cada estado. A la hora de diseñar el controlador una de las partes que más dieron trabajo fue algunos problemas que se tuvieron debido al timing, la solución a esto, fue colocar registros internos para poder encender las señales en un ciclo de reloj, ya que la RAM y ROM son sincrónicas y cómo se estaba trabajando bastante en la lógica combinacional, sin esos registros internos se tenía algunos problemas ya que se encendían y las memorias al ser sincrónicas costaba más la transferencia de datos al CPU para que este realizara las instrucciones. Algunas cosas relevantes de este controlador es que se realizó muchas cosas en la combinacional y en la secuencial solamente se actualizan los valores de las señales y de las variables internas. En el cuadro 1 se encuentran los nombres y el número asignado al estado.

Nombre del estado	**Número asignado al estado**
Env pos ROM	0
Espera ROM	1
Leer datos	2
Load o Store	3
Espera RAM	4
Registro	5
Operaciones	6
Fin	7

Cuadro 1: Estados y su número asignado en el código de verilog.

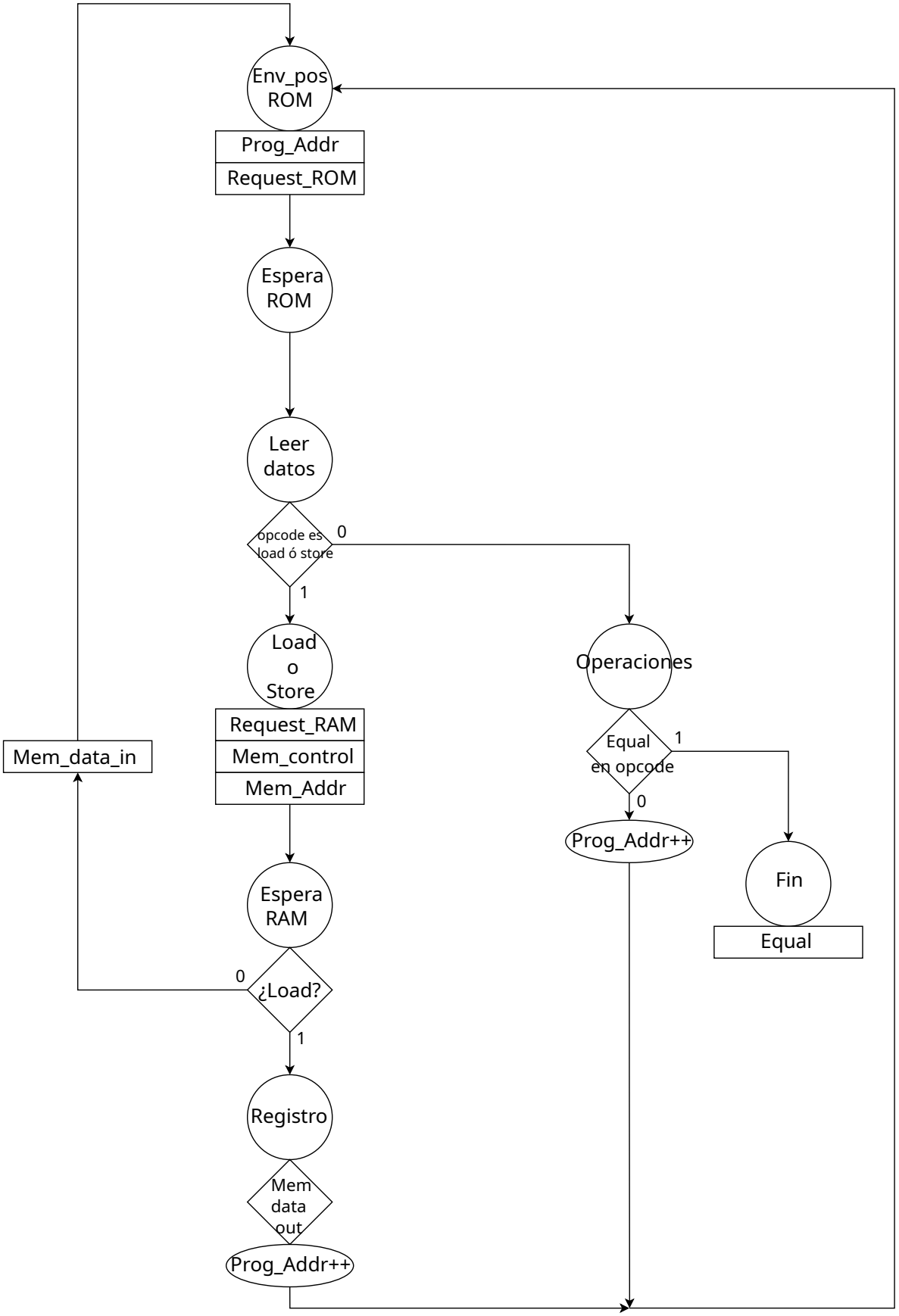

Figura 1: Diagrama ASM utilizado para el controlador del cajero automático.

Para este diseño de controlador su código de instrucciones es el presentado en el cuadro 2.

Operación	**OPCODE**
STORE	0x4
LOAD	0x3
ADD	0x9
SUB	0xA
AND	0x2
OR	0x1
EQUAL	0x5

Cuadro 2: Instrucciones para el CPU.

2. Plan de Pruebas

El controlador fue puesto bajo las siguientes pruebas con el fin de poder verificar su correcto funcionamiento ante las instrucciones para el cual fue diseñado, por ende, a continuación se presentan la serie de instrucciones en las cuales será sometido el controlador para poder verificar su funcionamiento:

Prueba I

Para esta prueba se desea cargar valores desde la memoria RAM a los valores de a y b, se desea realizar la suma de a y b y guardar este resultado en b, así como cargar los valores de a y b en la memoria, luego cargar datos nuevamente en a y b y finalmente utilizar un equal para poder finalizar el programa, por lo que el programa se vería como se observa en el cuadro 3

Step	**Code**
0	Load a from 005
1	Load b from 010
2	Add a + b →b
3	Store a at 015
4	Store b at 020
5	Load a from 020
6	Load b from 025
7	Equal a to b

Cuadro 3: Programa de prueba 1 para el CPU

Esta prueba fue realizada y completada exitosamente.

Prueba II

Para esta prueba, lo que se busca es realizar que se carguen valores provenientes desde la RAM a las variables a y b, poder realizar la resta de a y b y guardarlo en b, luego guardar en memoria el valor de b, restaurar a b y luego realizar la and entre a y b y guardarla en b y luego hacer la or de a y b y guardarla en a y guardar estos archivos en memoria y finalmente un equal para terminar el programa. Por lo que quedaría las instrucciones de la prueba como se presenta en el cuadro 4.

Step	Code
0	Load a from 001
1	Load b from 002
2	SUB a - b →b
3	Store b at 006
4	Load b from 002
5	AND a && b → b
6	OR a || b → a
7	Store a at 007
8	Store b at 010
9	Equal a to b

Cuadro 4: Programa de prueba 2 para el CPU.

Esta prueba fue realizada y completada de manera exitosa.

3. Instrucciones de utilización de la simulación

Para poder ejecutar la simulación se presentará a continuación la manera de poder ejecutarlo desde la terminal en linux. Para esto primero se debe de descargar los archivos nombrados como cpu.v, tester.v testbench.v, ram.v, rom.v, cpu.ys, *cmos_cells.lib*, *cmos_cells.v* y Makefile, se necesitan estos 9 archivos para poder ejecutar la simulación, se facilita la colocación de comandos ya que el archivo.ys, contiene todo lo necesario para la síntesis de la descripción conductual y también el colocar sólamente con las compuertas que se contienen en la biblioteca *cmos_cells.lib*, además de que se facilita un makefile el cual realizará todo, tanto realizar la síntesis, como la compilación y abrir el gtkwave, por ende se necesita solamente utilizar el comando presentado en el listado 1.

Listado 1: Colocación del comando make

```
user@ubuntu:~$ make
```

Si se desea realizar el proceso de síntesis solamente, sin la compilación ni la simulación, se utiliza el comando presentado en el listado 2, para esto es necesario tener los archivos *cmos_cells.lib cmos_cells.v cpu.v cpu.ys*

Listado 2: Colocación del comando para síntesis

```
user@ubuntu:~$ yosys -s cpu.ys
```

Este genera un archivo llamado *cpu_synth.v*, en el cual es donde se encuentra la descripción estructural.

Cuando se desee eliminar los archivos generados por el make se utiliza el comando mostrado en el listado 3

Listado 3: Comando para limpieza de archivos generados por el make.

```
user@ubuntu:~$ make clear
```

Además, el Makefile ejecuta la simulación con la síntesis, si se desea realizar la simulación con el código conductual lo único que se debe de realizar es eliminar que la línea de código del 'include "cpu.v", de manera que la parte de arriba del archivo testbench.v quede como se observa en la figura 2.

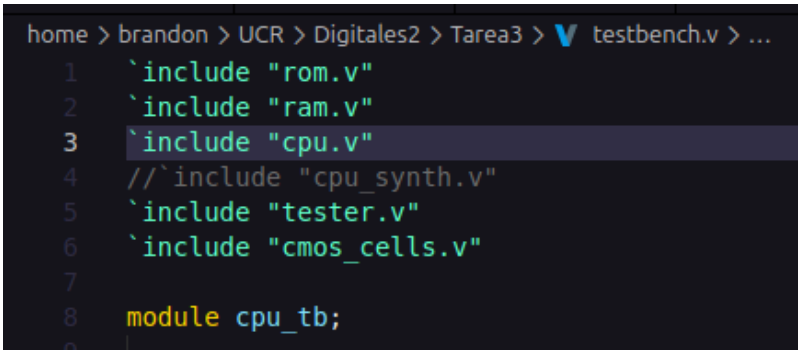

Figura 2: Modificación de testbench para ejecutar las pruebas en el verilog conductual.

4. Ejemplos de resultados

Al realizar el proceso de síntesis, se obtuvo que el diseño cuenta con la cantidad de compuertas lógicas NAND, NOR, NOT y flip flops presentados en el cuadro 5.

Elemento	Cantidad
Compuerta NAND	259
Compuerta NOR	276
Compuerta NOT	78
Flip FLop	65

Cuadro 5: Cantidad de compuertas lógicas y flip flops

Para comprobar el correcto funcionamiento del diseño del controlador se realizaron las 2 pruebas planteadas en el plan de pruebas, a continuación se presenta en las figuras 3 y **??** los resultados que se obtuvieron.

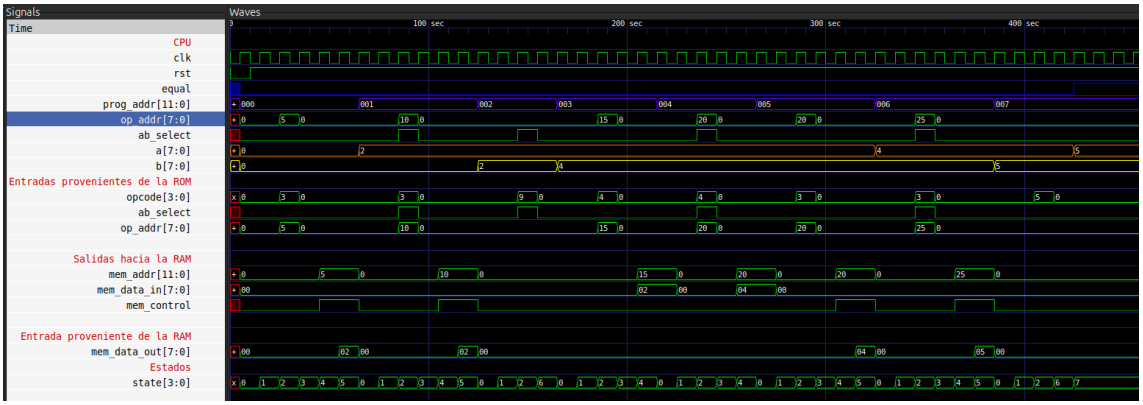

Figura 3: Programa para la prueba 1.

Al observar la figura 3, se puede observar que la señal morada es el prog addr, el cual indica en cuál instrucción se encuentra, también al observar las señales amarillas y naranjas, se puede observar el cómo toman valores las variables internas a y b, estos corresponden a los valores desde la posición 5 de la memoria RAM, y el valor de b de la posición 10 de la memoria RAM, luego se puede observar que luego el siguiente valor de b es 4, el cual es la suma de a y b, y este se

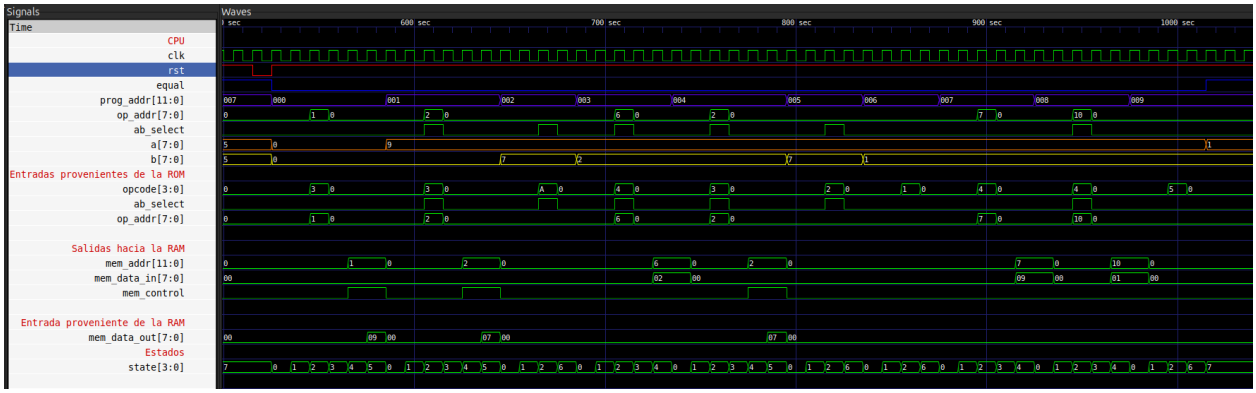

Figura 4: Programa para la prueba 2.

guardó en b, luego se guardaron los valores de a y b en la memoria RAM y finalmente se volvió a cargar el valor de a, el cual al menos en a es el último valor de b y luego el valor de b si es nuevo y finalmente se igualan y se puede observar en la señal azul que la bandera de equal se activa y ya no continúa más el programa. Al observar la figura 4, primeramente se puede observar la señal roja, en donde se activa la señal de reset, para poder comenzar un nuevo programa, y a su vez ya en este momento del tiempo la ROM ya tiene cargadas las instrucciones de la prueba 2, por lo que ya se comienza a realizar el programa de la prueba 2, en donde se observa que se cargan los valores para a y b y se realiza la resta y las and y las or, para verificar que se realice la correcta and y la or se presenta los cuadros 6 y 7.

Variable	Valor binario	Valor decimal
a	1001	9
b	0111	7
and	0001	1

Cuadro 6: Tabla en donde se realiza la operación lógica and

Variable	Valor binario	Valor decimal
a	1001	9
b	0001	1
or	1001	9

Cuadro 7: Tabla en donde se realiza la operación lógica or

5. Conclusiones y recomendaciones

A través de los resultados obtenidos en las pruebas se puede observar que tanto como el diseño conductual y estructural funcionan correctamente cumple con todas las instrucciones las cuales debe de poder realizar según las especificaciones, además de que permite el correcto funcionamiento en conjunto con la RAM y la ROM, gracias a las salidas de request, las cuales ayudan a mantener un orden sobre a quíen se le realiza la solicitud, si a la ROM o a la RAM, además de que se logra la correcta realización de operaciones tanto aritméticas como la suma y la resta como lógicas como lo son la AND y la OR. Un aspecto importante que se debe de considerar es el que las salidas del CPU estas ocurran en los flancos de reloj, ya que la RAM y la ROM al ser sincrónicas puede generar errores de timing, lo cual fue lo que ocurrió al inicio, sin embargo gracias a este problema

se pudo analizar y descubrir la importancia y la utilidad de los registros, y además de las variables auxiliares que en el código se presentan como las next.

Como recomendación se tiene que se realice el trabajo con un tiempo prudente, ya que se complica a la hora de conectar los 3 módulos para que funcionen en conjunto, también se debe de pensar con bastante tiempo el diseño del CPU para tratar de crear un CPU lo más optimizado posible, además de poder buscar una solución para poder colocar 2 programas en la ROM, la cual en este caso se realizó con un initial, además de que se recomienda crear un archivo .ys y un Makefile, el cual podrá ayudar bastante a no colocar tantos comandos en la terminal y así realizar la verificación de las pruebas de manera más rápida.