Escuela de Ingeniería Electrónica
Licenciatura en Ingeniería Electrónica
EL5811 Verificación funcional de circuitos integrados
I Semestre 2025

TEC | Tecnológico de Costa Rica

Proyecto 1: Ambiente de Verificación para Banco de Registros de 8088

Sebastián Cruz Solano 2020042177

Índice

1. Resumen Ejecutivo

El presente informe describe el proceso de diseño, implementación y verificación funcional de un banco de registros para el microprocesador Intel 8088. El proyecto fue desarrollado utilizando el lenguaje de descripción de hardware Verilog para la implementación, y SystemVerilog para la construcción del entorno de verificación. La estructura del banco de registros realizada para el proyecto incluye unicamente a los registros de propósito general (*GPR's*) y garantizar el correcto funcionamiento.

El diseño del módulo aborda tanto la arquitectura de registros de 8 y 16 bits, como su validación funcional mediante un entorno de pruebas aleatorias automatizadas. La estructura de los registros de propósito general del banco de registros se divide en dos grupos: registros de datos como AX, BX, CX y DX y sus sub-registros *High* y *Low*; y registros de dirección de puntero e índice SP, BP, SI y DI. Para la construcción del circuito en Verilog se utilizaron módulos como multiplexores, decodificadores de escritura, módulos de registros de 8 y 16 bits y un buffer triestado; que se interconectan en un *Top* para su integración mediante un bus bidireccional.

Para el proceso de verificación funcional del modulo del banco de registros de 8088 se desarrollaron dos *testbenches* en SystemVerilog para comprobar el correcto funcionamiento del modulo. El primero, de tipo básico para observar el comportamiento de las señales internas del modulo ante operaciones de lectura y escritura en todo los registros. El segundo, de tipo más complejo y automatizado para crear pruebas aleatorias con secuencias de lectura y escritura sobre todos los registros, utilizando datos de 16 bits generados de forma dinámica (*tester*). Además, se implementó un *scoreboard* encargado de actuar como modelo de referencia para comparar los resultados leídos del diseño con los valores esperados previamente almacenados. Dentro del *scoreboard* se incluyeron mecanismos para contabilizar la cantidad de errores y aciertos en un archivo de texto.

Durante el proceso de verificación con el *tester* y el *scoreboard* se logró identificar fallas funcionales menores en cuanto a la lógica de selección de los registros altos y bajos, y la sincronización del reloj en la lectura de datos. Estos errores fueron corregidos en el diseño e implementación del modulo con lo que se logró obtener un funcionamiento estable y correcto del banco de registros en todas las operaciones de lectura y escritura, tanto para los registros de 16 bits como para los de 8 bits.

Finalmente, se logró desarrollar el diseño de un banco de registros del microprocesador 8088 con un sistema completamente funcional y una plataforma de verificación capaz de validar el comportamiento del modulo mediante pruebas automatizadas y aleatorias. La integración del entorno de verificación permitió detectar errores lógicos en la etapa de diseño y garantizar que todas las combinaciones de registros y operaciones de lectura y escritura posibles fueran cubiertas.

2. Introducción

El Intel 8088, sucesor del 8086, es un microprocesador de 16 bits desarrollado por Intel en 1979, conocido por haber sido utilizado por IBM en la primera computadora personal IBM PC. El 8088 es idéntico en cuanto a su arquitectura y *set* de instrucciones al 8086, a excepción de que el bus de datos externo es de 8 bits en el 8088, mientras que en el 8086 era de 16 bits.

En el diseño de la arquitectura de un procesador, el banco de registros es un modulo indispensable que permite, de forma temporal, almacenar datos y direcciones. Los registros funcionan como un tipo de memoria muy rápida pero con poca capacidad de almacenamiento para el procesador para realizar instrucciones y operaciones en la unidad aritmética lógica (*ALU*). En la arquitectura del 8088, se tienen cuatro registros de propósito general de 16 bits que pueden ser segmentados y accedidos como ocho registros de 8 bits, parte alta y baja.

Para este proyecto se implementó un banco de registros de la arquitectura del 8088 que permite el acceso a los registros de 16 bits y a su división de 8 bits con un bus bidireccional. Se validó mediante un entorno de pruebas utilizando *tester* y *scoreboard* con pruebas automatizadas en SystemVerilog.

3. Marco Teórico

3.1. Banco de Registros

El banco de registros de un procesador es la unidad de almacenamiento más cercana a los núcleos de procesamiento y por ende la más rápida. Como se muestra en la Figura 1, se ubican en la parte superior de la jerarquía de memoria, por encima de la memoria caché, la *random access memory (RAM)* y los dispositivos de almacenamiento como discos de estado solido *(SSD)* y discos duros *(HDD)*. Esta posición superior en la jerarquía de memoria permite que el procesador ejecute instrucciones sin retardos significativos.

Los registros son un tipo de memoria volátiles, lo que significa que pierden su contenido al apagarse o reiniciarse el sistema. Están diseñados para almacenar valores de manera temporal, tales como resultados de operaciones, direcciones de memoria, y datos intermedios que deben utilizarse rápidamente.

En comparación con otros medios de almacenamiento, los registros operan a velocidades considerablemente mayores. Mientras que la memoria RAM puede tener latencias del orden de nanosegundos, los registros se acceden en una fracción del ciclo de reloj del procesador. Por lo que, los registros se usan para contener los datos con que se está trabajando puesto que el acceso a los registros es mucho más rápido que los accesos a memoria. Esta diferencia de velocidad es de suma importancia para mantener un buen rendimiento del sistema.

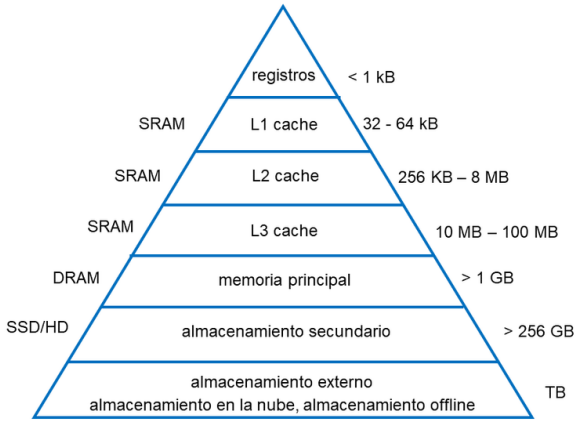

Figura 1: Pirámide de jerarquía de memoria.

3.2. Banco de registros del 8088

La arquitectura del microprocesador Intel 8088 está separada en dos secciones principales: la unidad de ejecución y la unidad de interfaz del bus.

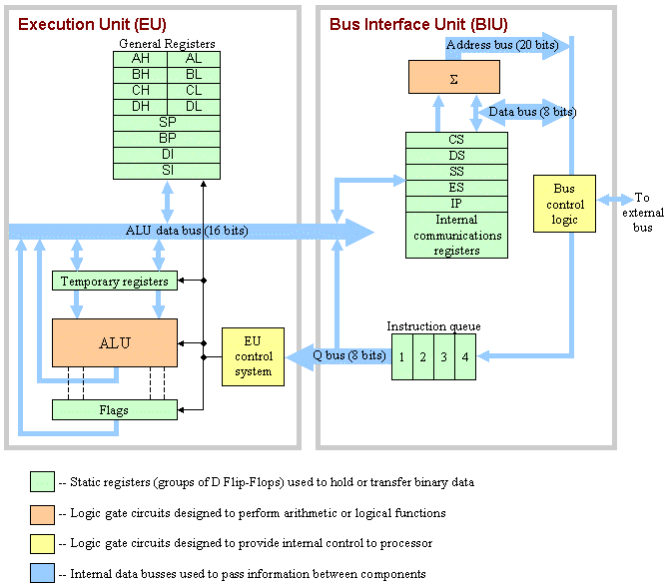

Figura 2: Arquitectura del microprocesador Intel 8088

En la unidad de ejecución se encuentran los 14 registros de 16 bits divididos en dos grupos: los de propósito general y direcciones. Existen cuatro registros de propósito general *AX, BX, CX y DX*, los cuales se pueden subdividirse en dos registros de 8 bits: una parte baja (*AL, BL, CL y DL*) y una parte alta (*AH, BH, CH y DH*). Esto permite direccionar en forma de 8 y 16 bits, y obtener 12 registros para operaciones. La codificación de estos registros está definida mediante una combinación de la señal de selección (SEL) y una señal adicional (W) que define si el acceso es a la parte baja o alta. De esta forma se puede seleccionar de forma dinámica si se quiere acceder a los registros de 8 bits o al registro çompleto"de 16 bits. Loa registros de direcciones del 8088 son SP (stack pointer), BP (base pointer),

SI (source index) y DI (destination index). Estos registros, a diferencia de los de propósito general, sólo se pueden acceder en su totalidad como registros de 16 bits. La codificación utilizada para el acceso a los registros se muestra en la Figura 4

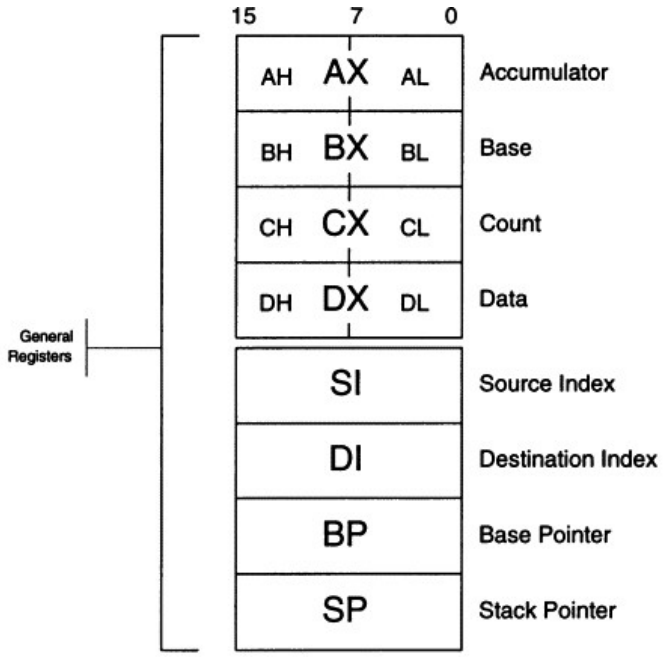

Figura 3: Registros de la unidad de ejecución.

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

Figura 4: Codificación de acceso a los registros.

3.3. Tester

El *testing* en un entorno de verificación es el proceso de evaluar un sistema o componentes con herramientas de simulación para determinar la funcionalidad de un diseño. Los *testbenches* desarrollados en SystemVerilog son pruebas de usabilidad que generan estímulos al dispositivo bajo prueba (*DUT*) para validar el funcionamiento correcto. Se llevan a cabo planes y casos de prueba que simulen todas las posibilidades de uso de un DUT para verificar las respuestas obtenidas mediante monitores o *scoreboards*.

3.4. Scoreboard

El *scoreboard* es un componente de la verificación funcional que actúa como un modelo de referencia al comparar los datos recibidos del DUT con los resultados esperados. Es decir, recibe las operaciones realizadas por el *tester* al DUT y las compara con un modelo de referencia. El scoreboard permite verificar de forma automática el DUT mediante reportes de aciertos y errores. El flujo fundamental de un scoreboard se basa en: Los elementos enviados al DUT se agregan al scoreboard y se almacenan en su base de datos. Luego, los elementos devueltos por el DUT, recolectados por monitores, se agregan al scoreboard para ser comparados con los elementos enviados.

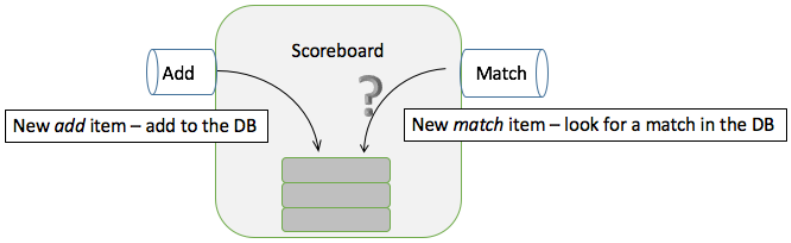

Figura 5: Flujo fundamental de un Scoreboard.

4. Descripción de la solución

4.1. Diseño del Banco de Registros del 8088

Se implementó un diseño del banco de registros del 8088 que contiene la arquitectura de registros de 8 y 16 bits. La estructura de los registros de datos de propósito general consiste en cuatro registros de 16 bits AX, BX, CX y DX; que pueden ser accesados como dos registros de 8 bits, la parte alta (AH, BH, CH y DH) que corresponden a los 8 bits más significativos y la parte baja (AL, BL, CL y DL) los 8 bits menos significativos. Los registros de dirección de puntero e índice (SP, BP, SI y DI) son registros de 16 bits que se acceden como registros completos y no tienen separación.

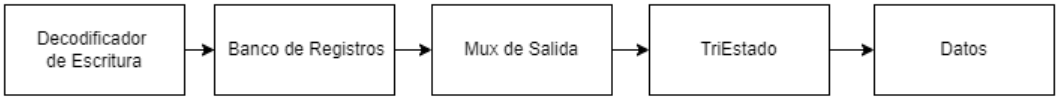

Figura 6: Diagrama de nivel 1 del módulo.

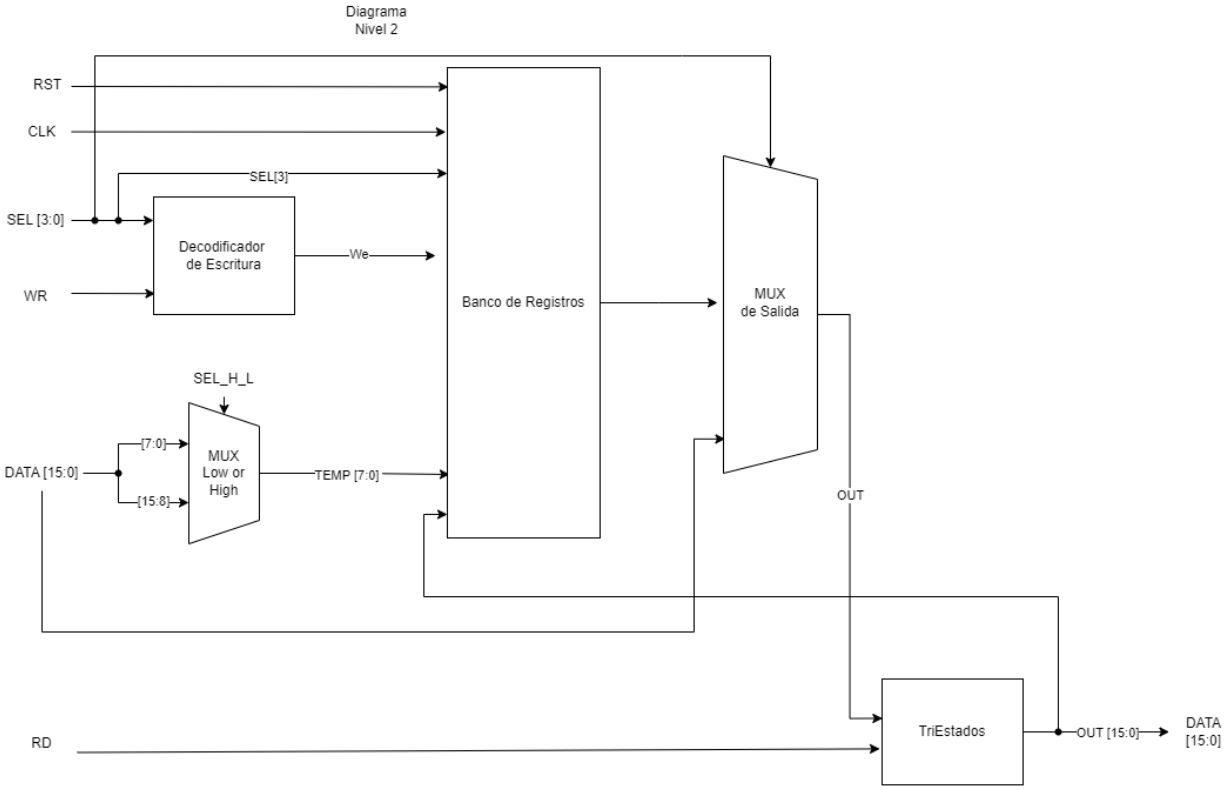

Figura 7: Diagrama de nivel 2 del módulo.

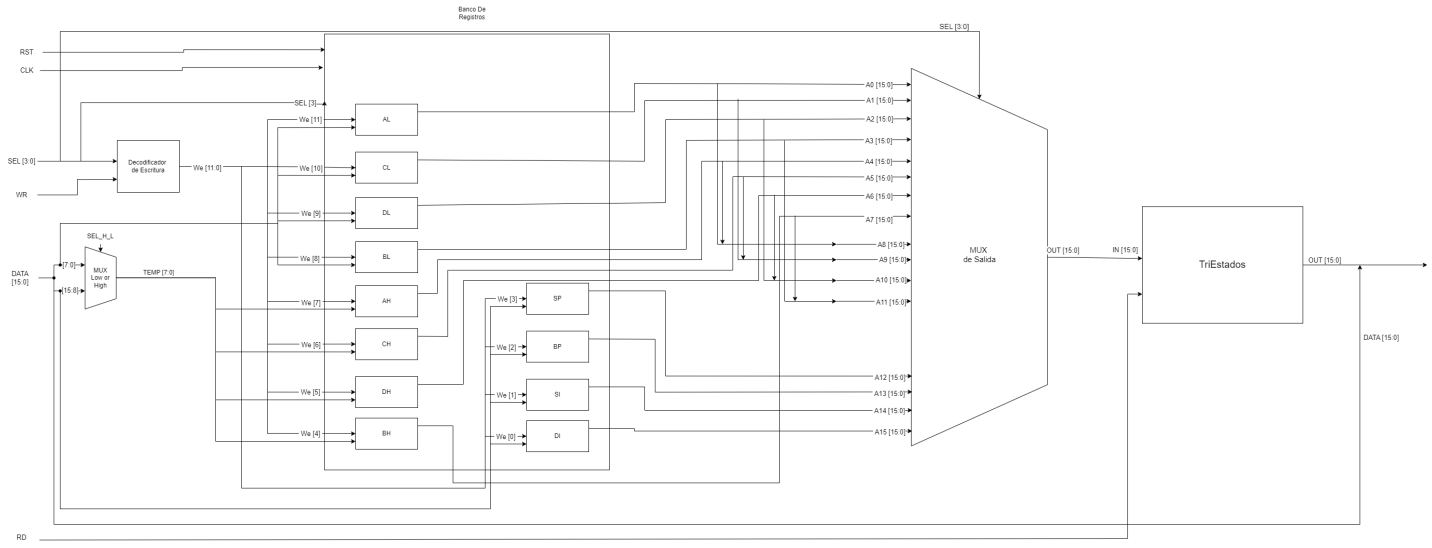

Figura 8: Diagrama de nivel 3 del módulo.

El objetivo de diseño del banco de registros es que utilizando la codificación de la Figura 4 se pueda acceder a un registro en especifico, ya se de 8 o 16 bits, para realizar una operación de lectura o escritura. Por lo que, cada registro es una instanciación de un módulo de 8 o 16 bits que se habilitan para lectura o escritura mediante una señal *WE* proveniente de un decodificador. La selección del registro se basa en la concatenación de dos señales de control: la señal de selección (*SEL*) de 4 bits y *w* para la selección entre parte alta y baja en los registros de datos. El acceso al bus compartido de datos se controla mediante un buffer triestado. A continuación se describen los módulos creados:

4.1.1. Decodificador de Escritura

Entradas:

- SEL [3:0]: Selección del registro
- WR: Señal de escritura

Salidas:

- WE [11:0]: Señal de habilitación de escritura

El decodificador de escritura recibe como entrada una señal de 5 bits proveniente de la concatenación de la señal de escritura *WR* y la selección del registro *SEL*. La salida es una señal *WE* de 12 bits que se utiliza para activar el bit correspondiente al registro seleccionado para escritura. Para la codificación de la señal de *SEL* y *WE* se utiliza la Figura 4. Por ejemplo, una combinación 5'b10000 ($WR = 1$ y $SEL = 0000$) activa el bit correspondiente al registro *AL*.

4.1.2. Registros de 8 y 16 bits

Entradas:

- CLK: Señal de reloj
- RST: Reset
- WriteEnable: Habilitación de escritura
- D: Dato de entrada

Salidas:

- Q: Dato almacenado en el registro

Los módulos de registros de 8 y 16 bits son los bloques de almacenamiento de datos, en donde dependiendo del valor de la señal de entrada *WriteEnable* se guarda el dato de entrada *D* en el dato de salida *Q*. El módulo *Reg8bits* funciona para modelar los registros de 8 bits, mientras que *Reg16bits* es utilizado para los de 16 bits.

4.1.3. Integración de los Registros

Entradas:

- CLK: Señal de reloj
- RST: Reset
- SEL: Selección del registro
- SEL_H_L: Selección entre parte baja o alta
- DATA [15:0]: Dato de entrada
- WE [11:0]: Señal de habilitación de escritura

Salidas:

- AL a DL [7:0]: registros bajos
- AH a DH [7:0]: registros altos
- SP a DI [15:0]: registros de 16 bits

Este módulo se encarga de instanciar todos los registros de 8 y 16 bits, manejar su interconexión, determinar en cual registro se va a escribir según el valor de *WE* y enviar el dato correcto según parte baja, parte alta o registro completo de acuerdo con las señales SEL y SEL_H_L.

4.1.4. Multiplexor 2 a 8

Entradas:

- ENT1: Byte bajo
- ENT2: Byte alto
- SEL_H_L: Selección entre parte baja o alta

Salidas:

- OUT: Byte seleccionado.

El multiplexor de 2 a 8 es utilizado para separar el dato de entrada en byte bajo y byte alto y seleccionar, basado en la señal *SEL_H_L* la parte baja o alta de los registros de 16 bits divididos. De esta forma se puede seleccionar los registros *AL, BL CL y DL* y almacenar los bits menos significativos de la señal de entrada *DATA*, es decir *DATA*[7:0]. Si se selecciona un registro alto como *AH, BH, CH y DH* se almacenan los bits más significativos, *DATA*[15:8].

4.1.5. Multiplexor 16 a 1

Entradas:

- A0 a A15 [15:0]: registros de entrada
- SEL [3:0]: Selección del registro

Salidas:

- OUT [15:0]: Dato de salida del registro seleccionado

El módulo de multiplexor de 16 a 1 sirve como un selector final para, basado en la codificación de la señal de *SEL*, seleccionar uno de los 16 registros de salida para presentarlo como salida general del banco de registros.

4.1.6. Buffer Triestado

Entradas:

- IN [15:0]: Datos desde el registro seleccionado.
- SEL: Habilitación del buffer.

Salidas:

- OUT [15:0]: Bus de datos triestado.

El módulo de buffer triestado se utuliza para controlar el acceso del banco de registros al bus de datos para que solo un registro conduzca en un momento dado. Si la señal de *SEL* está en alto, el dato de entrada IN se conduce en el bus hacia la salida *OUT*; si no, se "desconecta"del bus y la salida se mantiene en alta impedancia *Z*.

4.1.7. Top

Entradas:

- CLK: Señal de reloj
- RST: Reset
- RD: Señal de lectura
- WR: Señal de escritura
- SEL [3:0]: Selección del registro

Salida bidireccional:

- DATA [15:0]: Salida del banco de registros

El módulo Top es el módulo principal del sistema que integra todos los módulos anteriores y se encarga de ordenar las señales de lectura, escritura, selección de registros y conexión y desconexión con el bus de datos mediante el buffer triestado con base en el valor de la señal *RD*.

4.2. Verificación Funcional del Banco de Registros del 8088

Basado en el *SPEC* del banco de registros del 8088 implementado, se diseñaron y ejecutaron dos *testbenches* en SystemVerilog para validar el correcto funcionamiento del dispositivo.

4.2.1. Verificación Básica

El primer testbench realizado es una prueba básica en la que, de forma manual, se realizan operaciones explícitas de lectura y escritura en todos los registros del banco. El testbench diseñado itera sobre todos los valores posibles de la señal *SEL*, de 0 a 15, para acceder a cada registro del banco; a los de 8 bits como AL, AH, BL, BH, CL, CH, DL y DH y a los de 16 bits como AX, BX, CX, DX SP, BP, SI y DI. A cada registro se le escribió un valor especifico y conocido, y después de cada escritura se realizó una operación de lectura en el mismo registro. Esto se hizo inmediatamente después de la escritura para poder tener una referencia para comparar el dato obtenido de la lectura de los registros con el dato específico escrito, y poder verificar el funcionamiento correcto del módulo en ambas operaciones.

4.2.2. Verificación con Tester y Scoreboard

El segundo testbench realizado es más complejo y automatizado utilizando los métodos de tester y scoreboard. El tester permite generar secuencias aleatorias de operaciones de lectura o escritura con datos dinámicos de 16 bits en diferentes registros. Para poder obtener las operaciones aleatorias, los datos dinámicos y definir el registro a utilizar se crearon funciones en SystemVerilog que retornan estos datos. Para el caso de obtener el tipo de operación se creó una función llamada:

```
obtener_operacion_RD_WR()
```

Esta función genera un bit de forma aleatoria y lo retorna. Si es 1 se hace una operación de lectura ($RD = 1$), y si no se hace una escritura ($RD = 0$).

Para obtener el dato a escribir en el registro cuando se realiza una operación de escritura se creó la función:

```
obtener_valor()
```

Esta función crea una variable de 3 bits llamada *ceros_y_unos* y se le asigna un valor aleatorio con el objetivo de tener una probabilidad de $1/8$ de que el dato a escribir sea *16'h0000* o *16'hFFFF*. Si esta variable toma cualquier otro valor, la función retorna un dato aleatorio de 16 bits.

La tercera función creada es:

```
obtener_registro()
```

Esta retorna un valor aleatorio de 4 bits que es el valor que se le asigna a la variable *SEL* para definir el registro en el que se va a realizar la operación.

El testbench realizado también incluye un scoreboard que permite almacenar y comparar los datos obtenidos del DUT con un modelo de referencia por lo que, en el caso de que se realice una operación de escritura el tester escribe en el registro seleccionado por *SEL* y posteriormente realiza una lectura para confirmar que el valor leído corresponde al dato escrito. El dato escrito se guarda en un arreglo llamado *expected* que permite, cuando se hace una operación de lectura, comparar el dato de salida del DUT con el dato almacenado en este arreglo. Si los datos coinciden, el scoreboard aumenta un contador de aciertos de lectura o escritura, dependiendo de la operación. Si los datos no son los esperados el scoreboard lanza un error y se aumenta un contador de errores. Al finalizar la cantidad de iteraciones establecidas para el tester, el scoreboard imprime un resumen de la cantidad de lecturas y escrituras correctas y errores.

5. Análisis de Resultados

5.1. Resultados de la Verificación Básica

En la ejecución del testbench básico se obtuvo todos los resultados correctos para ambas operaciones en todos los registros. Para cada registro se logró, de forma individual, hacer operaciones de lectura y escritura y obtener el mismo dato. Además, en el *waveform* de la Figura 9 se pueden observar los cambios en las señales de *SEL* para cambiar de registro, *WR* y *RD* dependiendo de la operación a realizar, *SALIDA* el dato de salida del módulo y el valor de todos los registros del banco.

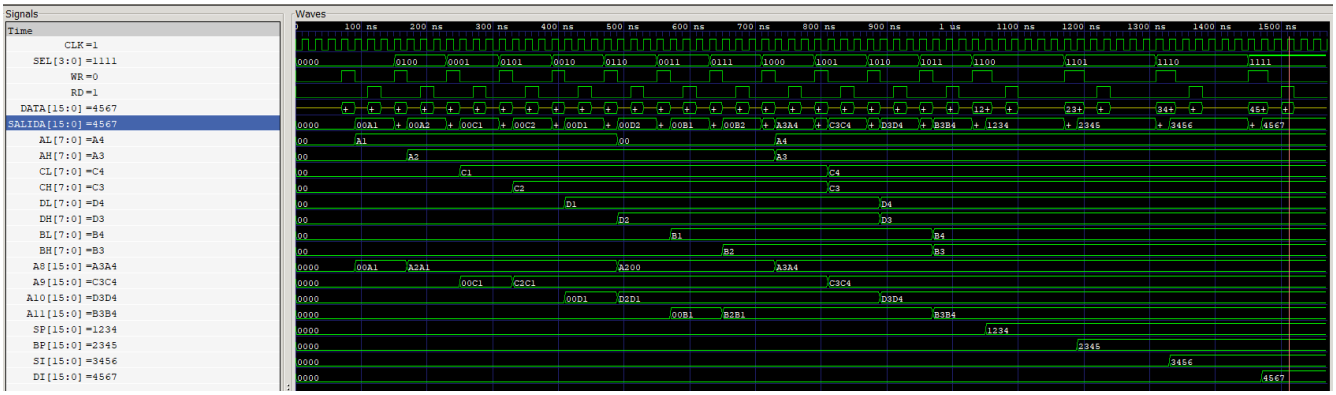

Figura 9: *Waveform* obtenido de la verificación básica.

Para el caso de los registros bajos de 8 bits como AL, se escribió el dato *16'h00A1* y se obtuvo como resultado:

```
Lectura desde AL: 00a1
```

Lo que confirma un direccionamiento correcto para los registros bajos de 8 bits en operaciones de lectura y escritura.

Para el caso de los registros altos de 8 bits como AH, se escribió el dato *16'hA200* y se obtuvo como resultado:

```
Lectura desde AH: 00a2
```

Lo que confirma un direccionamiento correcto para los registros altos de 8 bits en operaciones de lectura y escritura.

Para los registros completos de 16 bits como AX, se escribió el dato *16'hA3A4* y se obtuvo como resultado:

```
Lectura desde AX: a3a4
```

Lo que confirma un direccionamiento correcto para los registros çompletos"de 16 bits en operaciones de lectura y escritura.

Para los registros de dirección e indice de 16 bits como SP, se escribió el dato *16'h1234* y se obtuvo como resultado:

```
Lectura desde SP: 1234
```

Lo que confirma un direccionamiento correcto para los registros de dirección e indice de 16 bits en operaciones de lectura y escritura.

Este testbench confirma el correcto funcionamiento del módulo del banco de registros para ambas operaciones en todos los registros.

5.2. Resultados de la Verificación con Tester y Scoreboard

En segundo testbench con tester y scoreboard se obtuvo resultados que confirman el buen funcionamiento del banco de registros. Haciendo 20 iteraciones del tester se obtiene el siguiente resumen del scoreboard:

```
Tester terminado.
Resumen final:
Escrituras Totales Correctas: 8
Escrituras Totales Erroneas: 0
Lecturas Totales Correctos: 10
Lecturas Totales Erroneas: 1
```

Para 100 iteraciones del tester se obtiene el siguiente resumen del scoreboard:

```
Tester terminado.
Resumen final:
Escrituras Totales Correctas: 52
Escrituras Totales Erroneas: 0
Lecturas Totales Correctos: 29
Lecturas Totales Erroneas: 18
```

A pesar de que el scoreboard muestra errores en las lecturas, si se analiza de forma más detallada los resultados obtenidos, se puede puede afirmar que estos errores no son debido a un mal funcionamiento del módulo del banco de registros, sino a cómo se actualiza el arreglo de referencia *expected*. Ya que cuando se escribe en un registro de 8 bits, el modelo *expected[SEL]* no actualiza el registro de 16 bits. Esto se puede observar en los resultados del scoreboard, en donde en la repetición 97 se arroja un error.

```
Repeticion: 97
Valor de RD: 1 (LECTURA DE REGISTROS)
Leyendo registro 11, dato_leido = df00
Leyendo registro 11, expected[SEL] = df28

SCOREBOARD lectura ERROR: REG = 11 | Dato Leido = df00 | Dato esperado = df28
```

Este es el registro de 16 bits BX y vemos que el dato leído fue *df00* y en el arreglo *expected[SEL]* se tenia que el dato esperado era *df28*. Sin embargo, si analizamos la repetición 95, vemos que:

```
Repeticion: 95
Valor de RD: 0 (ESCRITURA DE REGISTROS)
Escribiendo 0000 en registro 3
Lectura desde reg 3: 0000
```

```
Valor de SEL: 0011:
Valor de DATA_drive: 0000:
Valor de dato_leido: 0000:
SCOREBOARD escritura OK: REG = 3 | Dato de entrada = 0000 | Leido = 0000
```

En esta iteración se está escribiendo *0000* en el registro BL y como el registro BX es la concatenación de BH BL, el registro BX tiene un valor de *df00*, por lo que la lectura de la repetición 97 es correcta y el módulo funciona de manera esperada.

6. Conclusiones

- Se comprobó que el diseño del banco de registros implementado funciona de manera correcta, permitiendo operaciones de lectura y escritura en los registros altos y bajos de 8 bits y los registros de 16 bits.
- Se logró verificar por dos métodos distintos, uno manual y otro automatizado, el correcto funcionamiento de los registros en situaciones de lectura y escritura directas y aleatorias.
- Se validó el camino de datos del módulo, la habilitación de señales de escritura como *WE*, la selección del registro con la señal *SEL*, la correcta decodificación del decodificador y el buen funcionamiento del buffer triestado.
- Se evidenció la importancia de conocer el *SPEC* del DUT para poder analizar de forma detallada los resultados del scoreboard y poder detectar falsos negativos.
- Se mostró la importancia de un tester automatizado con un scoreboard para el reporte de resultados, para poder generar gran cantidad de pruebas y secuencias sin la necesidad de hacerlas de forma manual.

7. Repositorio del Proyecto

Todos los archivos de diseño y verificación se pueden encontrar en el repositorio del proyecto. Este se encuentra público y se puede acceder **haciendo click aquí**.

8. Enlace a Video de Defensa

El video de la defensa del proyecto se encuentra público en la plataforma de YouTube y se puede acceder **haciendo click aquí**.

Referencias

[1] Barry B. Brey. *Microprocesadores Intel*. Prentice Hall, 2009.

[2] Cadence. Building Efficient Scoreboards. URL: https://community.cadence.com/cadence_blogs_8/b/fv/posts/building-efficient-scoreboards#:~:text=A%20%E2%80%9Cscoreboard%E2%80%9D%20is%20a%20verification,matched%20against%20the%20items%20sent.. (accessed: 15.04.2025).

[3] David Tarnoff. *COMPUTER ORGANIZATION AND DESIGN FUNDAMENTALS Examining Computer Hardware from the Bottom to the Top*. Lulu.com, 2007.

[4] James W. Coffron. *PROGRAMMING THE 8086/8088*. SYBEX Inc., 1983. ISBN: 0-89588-12()'9.

[5] San Hlaing Oo, Khin Soe y Kyaw Khaing. TEACHING OF 8088/86 PROGRAMMING WITH 8086 ASSEMBLY EMULATOR, 2020.

[6] John L. Hennessy David A. Patterson. Computer Organization and Design RISC-V Edition, 2021.