Practica 4. El Banco de Registros

Paul Vazquez
Ingenieria en Sistemas Digitales y Robotica
Tecnologico de Monterrey
Laboratorio de Arquitectura de computadoras A00819877

Rodolfo Cuan
Ingenieria en Sistemas Digitales y Robotica
Tecnologico de Monterrey
Laboratorio de Arquitectura de computadoras A01233155

1. Introducción

Quizás el elemento mas común a través de distintos tipos de arquitecturas de computadoras son los registros, que nos permiten realizar operaciones en dos operandos y guardar el resultado para posterior uso. Dichos registros son elementos de memoria del tipo SRAM (Static Random Access Memory) y se trata de la memoria con el acceso mas rápido debido a su simplicidad, tamaño y cercanía al ALU. En la arquitectura MIPS todas las instrucciones lógico-aritméticas requieren de dos operandos y un operador. Dichos operandos pueden tratarse de contantes o la dirección de algún registro. El hecho de que siempre sean dos operandos requiere dos salidas del modulo de registros, que se especificara mas adelante. Para acceder a la información la dirección especificada se tiene que decodificar con un ”decoder” y establecer si se desea leer o escribir en dicha dirección. Al tratarse de una arquitectura de 32 bits, cada registro tiene un tamaño fijo de 32 bits, por lo que el acceso a dichos componentes es llamada por ”palabra” a diferencia de la memoria principal cuyo acceso es por ”byte”.

2. Desarrollo

Como se indica en el manual, el objetivo final de la practica es desarrollar un banco de registros o Register File [1]. Para esto, se deben crear 3 módulos: un Mux, un Decoder y los registros. A continuación se presenta el desarrollo de cada uno de los módulos en VHDL y al final como se interconectaron para crear el banco de registros.

2.1. Mux

El mux es un componente con el cual se puede seleccionar una señal de varias. En este caso, el mux desarrollado, tiene como entrada 32 señales de 32 bits y una señal de 5 bits, llamada seleccionador. El modulo tiene una salida de 32 bits, la cual corresponderá a el valor de la entrada que el seleccionador indique.

Para crear el modulo, primero se pusieron en orden las 32 señales en un arreglo, llamado *DATA_IN*. Después, dentro de un proceso, se lee el valor del seleccionador, que en nuestro caso definimos como *REG_FILE_MUX_READ_REG*. Este valor se convierte en un entero sin signo y se utiliza como índice en el arreglo previamente creado (Listing 1).

```
REG_FILE_MUX_READ_DATA <=
DATA_IN(
to_integer(unsigned(REG_FILE_MUX_READ_REG)));
```

Listing 1: Asignación de Salida de Mux

2.2. Decoder

Con el decoder convertimos el valor que se introduce en REG_FILE_WRITEREG_SEL encendiendo uno de los posibles 32 enables que se encuentran en la salida cuando se tiene el enable de escritura seleccionado. Para lograr esto hacemos un shift al valor de 0x01 equivalente a la conversión a entero de REG_FILE_WRITEREG_SEL. Por ejemplo, si se tiene el valor 0x00011, se hace un shift left equivalente a 3 posiciones a 0x01, para lograr que el bit numero 3 (el cuarto bit) de la salida se encienda. Lo anterior se logra con un par de lineas de código:

```
initial_value(0) <= REG_FILE_DECODER_REGWRITE_ENABLE;

REG_FILE_DECODER_ENABLES <=
std_logic_vector(shift_left
(initial_value,
to_integer(unsigned(REG_FILE_DECODER_WRITEREG_SEL))));
```

Listing 2: Asignación de Salida de Decoder

2.3. Registers

Para generar esta sección del componente se declaro un arreglo de 32 espacios con cada bit_vector de tamaño 32. De esta forma para seleccionar al registro ”0” se seleccionaba la posición ”0” de dicho arreglo. Considerando que la salida del decoder son 32 lineas, se calcula con un for que posición del arreglo es la que se desea acceder y se escribe en dicha dirección. Para evitar sobreescribir el registro 0, se mantiene conectado en GND asignándole permanentemente el valor de 0 en el código sin importar su linea de enable.

```
process(CLK, ENABLES, WRITE_DATA)
variable register_selected : integer := 0;
begin
OUT_DATA(0)<=x"00000000";
if CLK'event and CLK = '0' then
        for k in 1 to 31 loop
                if ENABLES(k) = '1' then
                OUT_DATA(k) <= WRITE_DATA;
                end if;
        end loop;
end if;
end process;
```

Listing 3: For de registros

2.4. Register File

Como se estableció previamente, los 3 componentes deben interconectarse para el correcto funcionamiento del banco de registros. En este modulo, primeramente se definen como componentes el mux, decoder y los registros. Después se declaro el componente *REG_FILE*, el cual internamente estará compuesto por los otros componentes. Lo anterior puede ser observado gráficamente en la Fig. 1.

Observando la Fig. 1, se pueden detectar 6 entradas y 2 salidas, enlistadas a continuación:

- REG_FILE_WRITEREG_SEL : Entrada de 5 bits para seleccionar registro
- REG_FILE_REGWRITE_ENABLE : Entrada de 1 bits para seleccionar lectura o escritura
- WRITE_DATA : Entrada de 32 bits de datos para escribir
- CLK : Entrada de Reloj
- REG_FILE_MUX_READ_REG_1 : Entrada de 5 bits para seleccionar salida 1
- REG_FILE_MUX_READ_REG_2 : Entrada de 5 bits para seleccionar salida 2
- REG_FILE_READ_DATA_1 : Salida de 32 bits de datos del registro seleccionado con REG_FILE_MUX_READ_REG_1
- REG_FILE_READ_DATA_2 : Salida de 32 bits de datos del registro seleccionado con REG_FILE_MUX_READ_REG_2

Adicionalmente se requiere la declaración de 2 señales tipo *signal*, una de 32 bits y una del tipo creado *REGISTROS*. Estas señales son las conexiones entre los componentes.

Teniendo declaradas las entradas, salidas y señales internas, se procedió a la inicialización de los componentes. Un decoder, un arreglo de registros y dos muxes. Al inicializar cada componente las entradas y señales se van conectando acorde la declaración de componentes y el esquema (Fig. 1).

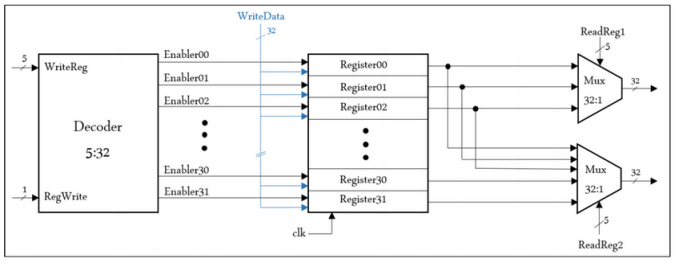

Figure 1: Esquema general del Banco de Registros mostrada en [1].

3. Resultados y Discusión

Para probar los módulos se creo un testbench para cada uno.

3.1. Mux

El testbench del modulo mux inicia poniendo los registros 0,10, y 20, con los valores x"ffffffff", x"f0f0f0f0" y x"aaaaaaaa", respectivamente. Después se leyó la salida *REG_FILE_MUX_READ_DATA* al seleccionar los diferentes registros con la entrada *REG_FILE_MUX_READ_REG*. Los resultados son mostrados en la Fig. 2.

Figure 2: Resultado del test bench del modulo mux.

3.2. Decoder

Para probar el decoder fue necesario comprobar que solo activara las lineas de enable cuando el enable de todo el decoder estuviera en '1' lógico. Posteriormente se probo seleccionar '0x00' en la entrada de 5 bits para que se encendiera la primera linea de los enables, correspondiente a un valor de "0x00000001" en el bitvector de salida. Cuando se introdujo el valor de 0x10 en el select en la salida se reflejaba la linea 16 correspondiente a un valor de "0x00010000". Cuando elegíamos un valor de 0x1F en el select, la ultima linea de los enables se encendía, correspondiente a "0x80000000".

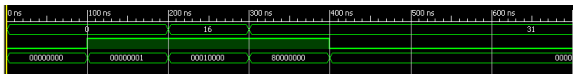

Figure 3: Resultado del test bench del modulo decoder.

3.3. Registers

Ser escribieron en tres localidades de memoria, encendiendo el enable 0, el enable 3 y el enable 31, con los datos a escribir: 0x00000000, 0x00001111, 0x1000110.

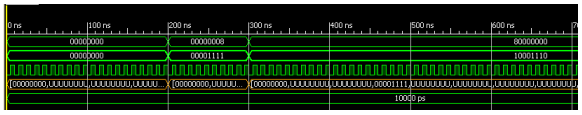

Figure 4: Resultado del test bench del modulo registers.

Una vez realizada la escritura, se revisó el contenido de los registros en el memory view y concluimos que la escritura había sido exitosa en las tres localidades:

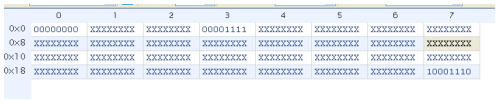

Figure 5: Resultado de la memoria en el test bench del modulo registers.

3.4. Register File

El register file se probo escribir y leer diferentes valores. Los resultados son mostrados en la Fig. 6.

1) (a) Se pone en modo lectura y se lee el registro 0 y se muestra ambas salidas.
2) (b) Se pone en modo escritura y escribe x"f0f0f0f0" en el registro 2.
3) (c) Se mantiene en modo escritura y escribe x"ffff0000" en el registro 4.
4) (d) Se vuelve a poner en modo lectura y se lee el registro 2 y el registro 4.

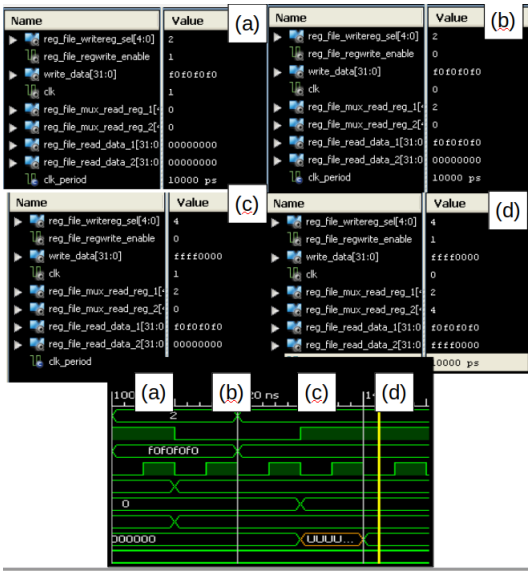

Figure 6: Resultado del test bench del modulo register file. Siguiendo los pasos del listado anterior

4. Conclusión

4.1. Paul Vazquez

El tener que elaborar tres componentes diferentes y conectarlos en una sola practica represento un gran reto que no se había visto con anterioridad en el curso. Gracias a que tuvimos practicas anteriores para recordar los conceptos básicos de VHDL pudimos realizarlo con éxito. Considero que con estos componentes la arquitectura MIPS va tomando mas forma, y mantenemos contemplado en todo momento la implementación final.

4.2. Rodolfo Cuan

Definitivamente esta es la practica mas compleja que hemos tenido hasta ahora. Sin embargo, el objetivo se cumplió. Las practicas anteriores nos ayudaron a recordar aspectos básicos VHDL, pero en esta nos recordó específicamente a como interconectar módulos previamente desarrollados. Lo bueno de la practica fue que estaba dividida en módulos, esto facilito a la hora de desarrollarlos e interconectarlos. Con cada practica podemos observar como nuestra arquitectura se esta formando poco a poco.

References

[1] Diego Fernando Valencia Martínez. *Manual del laboratorio*. Escuela de Ingeniería y Ciencias Departamento de Ciencias Computacionales, 2020.