22 de Septiembre de 2020

El banco de registros “Register File”

Laboratorio de arquitectura de Computadoras

ITESM Campus Monterrey

Profesor
Diego Fernando Valencia Martínez

Miguel Morales de la Vega
Javier Mondragón Martín del Campo

A00821541
A01365137

Introducción

El “Register File” como lo dice su nombre es un arreglo de registros, los cuales son accesados por medio de una entrada de 5 bits, se tiene una entrada para escribir en el registro y contiene dos salidas correspondientes a los datos dentro del arreglo de registros. El “Register File”, es uno de los módulos esenciales para los procesadores RISC, este módulo provee a la ALU de los datos necesarios para realizar operaciones.

En el desarrollo de esta práctica observaremos los componentes que integran al “Register File” tales son un decoder para la sección de registros a escribir, un arreglo de 32 registros de 32 bits cada uno y dos multiplexores para leer el arreglo de registros. Uno de los conceptos claves que usaremos para el desarrollo del arreglo de registros es la creación de un paquete de datos “PORT32OF32”, este paquete será accedido y usado como un tipo int, double, etc dentro de la declaración de la señal dentro módulo de registros. El otro concepto clave es el uso del ciclo “for” para la creación de los diferentes canales de datos que irán del banco de registros a las entradas de los multiplexores.

Procedimiento

Multiplexor

Para este módulo se requiere generar un multiplexor de 32 entradas de 32 bits, 5 bits de control y 1 salida de 32 bits. Para ello se tomó como referencia el multiplexor de la práctica uno (imagen 2.1.1).

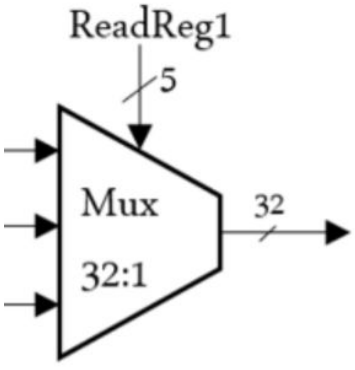

Imagen 2.1.1
Diagrama del multiplexor 32:1

Como este requería 32 entradas de 32 bits, se requiere utilizar la siguiente definición: “is array(31 downto 0) of STD_LOGIC_VECTOR (31 downto 0)” de la entrada, ya que, VHDL no acepta la definición directamente se genero un paquete (o “package”) con el nombre “array_port” donde tendría esta definición definida en un “type” para asignarla a la entrada llamada PORT32OF32.

Paquete:

```
package array_port is

	type port32of32 is array(31 downto 0) of STD_LOGIC_VECTOR (31 downto 0);

end array_port;
```

Una vez definido esto se utilizó como un arreglo:

```
DataOut <= DataIn(to_integer(unsigned(ReadReg)));
```

Donde utilizamos la señal de control como “ReadReg” para redirigir una señal de entrada del multiplexor traducida de hexadecimal a entero como lo requiere en VHDL por medio de la función unsigned y to_integer utilizado en anteriores prácticas a la salida. El resultado mostrado en la imagen 2.1.2 es el esperado.

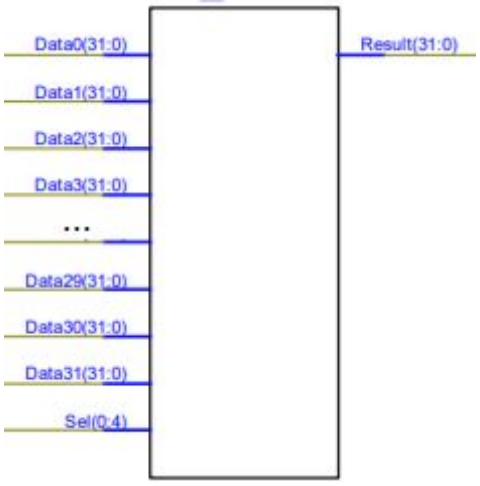

Imagen 2.1.2
Diagrama esquemático del multiplexor 32:1 generado por VHDL

Decoder 5x32

Este módulo es un demultiplexor de 5 entradas a 32. Se requiere una entrada de 5 bits, un habilitador y las 32 salidas de un bit. En caso que el habilitador tenga un registro de alto, la salida tendrá una de las 32 habilitadas definida por la entrada de los 5 bits, registrando un número hexadecimal para habilitar una de las 32 (imagen 2.2.1).

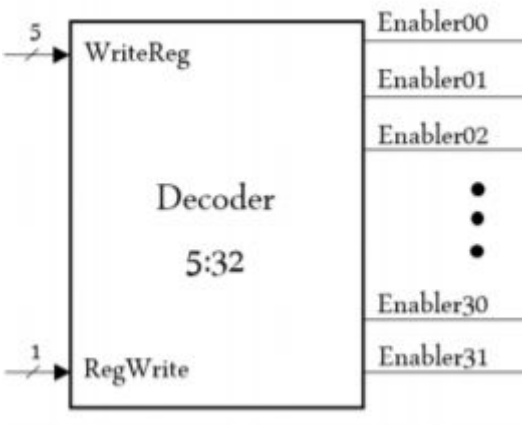

Imagen 2.2.1
Diagrama del Decoder 5:32

Para una fácil implementación, entendimiento y expandir su tamaño en un futuro si es que se requiere, se utilizó la función “for generate” para generar cada uno de los módulos donde se habilitará si y sólo si la entrada habilitadora estaba activada y la dirección correspondiente en hexadecimal (donde lo traduce a un entero) estaba seleccionada para cada salida, en todo caso arroja en la salida un estado bajo:

```
dem: for i in 0 to 31 generate
        begin
                Enabler(i) <=  '1' when i = to_integer(unsigned(WriteReg)) AND RegWrite = '1'
                               else '0';
        end generate;
```

El resultado es el esperado, donde VHDL lo traduce al diagrama de la imagen 2.2.2:

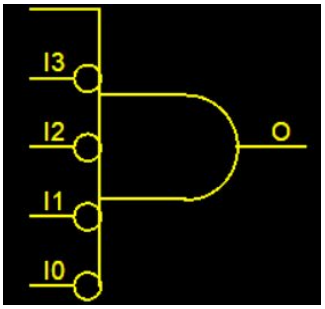

Imagen 2.2.2
Diagrama esquemático generado por VHDL del demultiplexor 5:32 con habilitador

Registro

El módulo consta de 32 entradas de 1 bit, 1 entrada de 32 bits, 1 entrada de reloj y 32 salidas de 32 bits almacenando lo que está en la entrada de 32 bits en alguna de las salidas mediante su habilitador correspondiente conectado a una de las 32 entradas de 1 bit. Exceptuando el primer registro que siempre debe de estar en 0 (imagen 2.3.1).

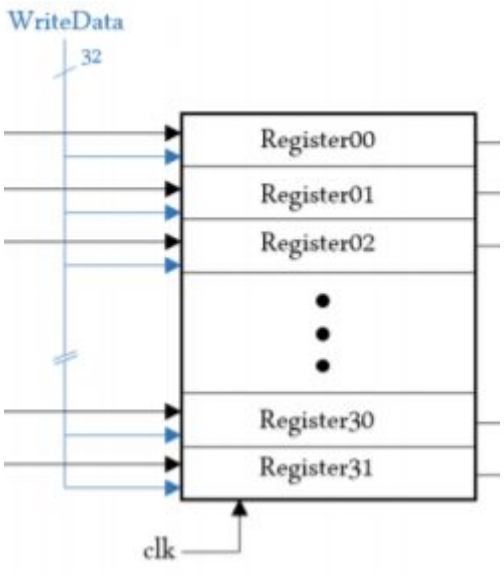

Imagen 2.3.1
Diagrama del Registro

Para esto, se puede realizar generando 31 flip flops tipo D con habilitador de reloj y conectando la primera entrada a GND. Para una fácil implementación de igual manera se utilizó el for generate, almacenando en la salida lo que hay en la entrada de 32 bits, si su respectivo habilitador y el reloj hace una transición de bajo a alto:

```
con: for i in 1 to 31 generate
begin
       DATA(i) <= WriteData when Enabler(i) = '1' AND rising_edge(CLK);
end generate;
DATA(0) <= (others => '0');
DataOut <= DATA;
```

Siendo Data de tipo PORT32OF32 mencionado anteriormente para el multiplexor y WriteData como la entrada de 32 bits. Finalmente se asigna a la salida también de tipo PORT32OF32. El diagrama esquemático generado por VHDL se puede visualizar en la imagen 2.3.2.

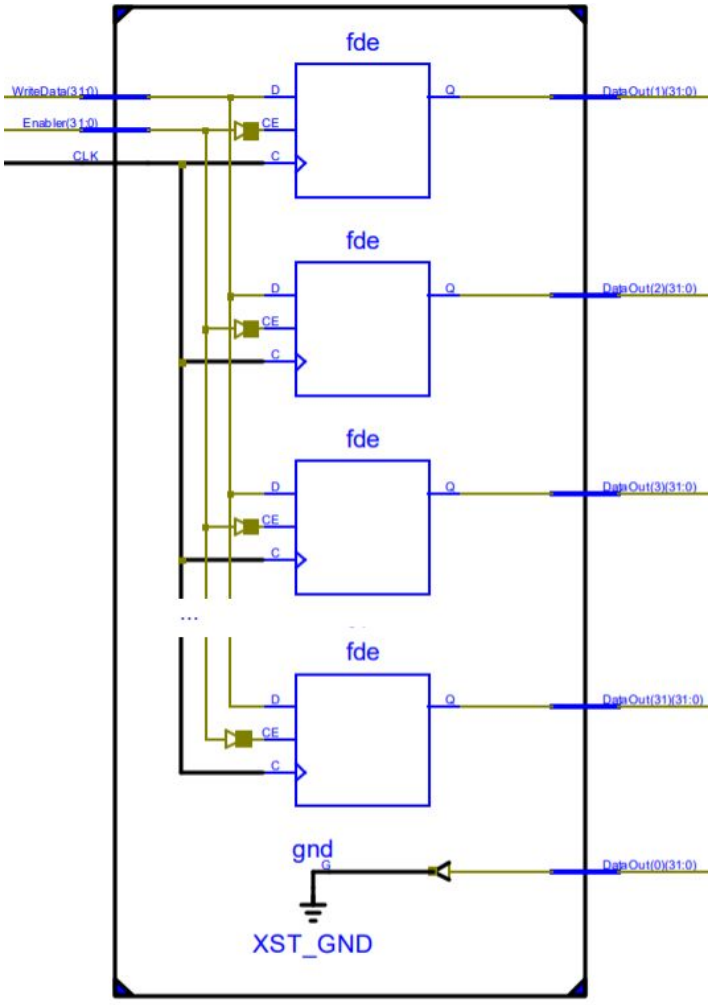

Imagen 2.3.2
Diagrama esquemático generado por VHDL de la memoria de registro con el primero en bajo siempre

Register File

Este módulo es una conexión de los 3 anteriores módulos para funcionar como uno solo teniendo de módulos 2 multiplexores, un decoder 5x32 y un registro (imagen 2.4.1).

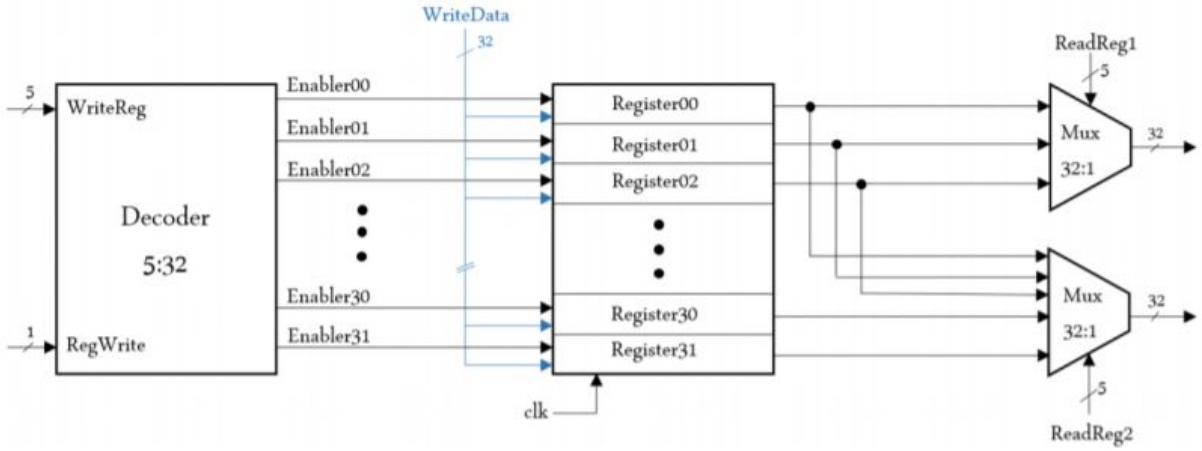

Imagen 2.4.1

Diagrama del Register File

Las señales que se conectaban entre módulos (signals) fueron dos, ENBus conectando la salida del decoder a las entradas de enable al register y PORT32OF32 conectando la salida de el registro a las entradas de 32 bits de cada multiplexor. El código de VHDL para lograr este módulo fue el siguiente:

```
architecture Behavioral of FileReg is

        signal MUXIN : PORT32OF32;
        signal ENBus : STD_LOGIC_VECTOR(31 downto 0);

        component MUX32to1
        port ( DataIn : in  PORT32OF32;
            ReadReg : in  STD_LOGIC_VECTOR (4 downto 0);
            DataOut : out  STD_LOGIC_VECTOR (31 downto 0));
        end component;

        component Reg
        port ( WriteData : in STD_LOGIC_VECTOR (31 DOWNTO 0);
                        Enabler : in STD_LOGIC_VECTOR (31 DOWNTO 0);
                        CLK : in STD_LOGIC;
                        DataOut : out PORT32OF32);
        end component;

        component Decoder
        port ( WriteReg : in  STD_LOGIC_VECTOR (4 downto 0);
                        RegWrite : in STD_LOGIC;
                        Enabler : out  STD_LOGIC_VECTOR (31 downto 0));
        end component;

begin

        M1: MUX32to1
        port map(
                DataOut => ReadOut1,
                ReadReg => ReadReg1,
                DataIn => MUXIN
        );

        M2: MUX32to1
        port map(
                DataOut => ReadOut2,
                ReadReg => ReadReg2,
                DataIn => MUXIN
        );

        R: Reg
        port map(
                WriteData => WriteData,
                Enabler => EnBus,
                CLK => CLK,
                DataOut => MUXIN
```

```
        );

        D: Decoder
        port map(
                WriteReg => WriteReg,
                RegWrite => RegWrite,
                Enabler => EnBus
        );

end Behavioral;
```

Teniendo como resultado obtenido y esperado lo que se muestra en la imagen 2.4.2.

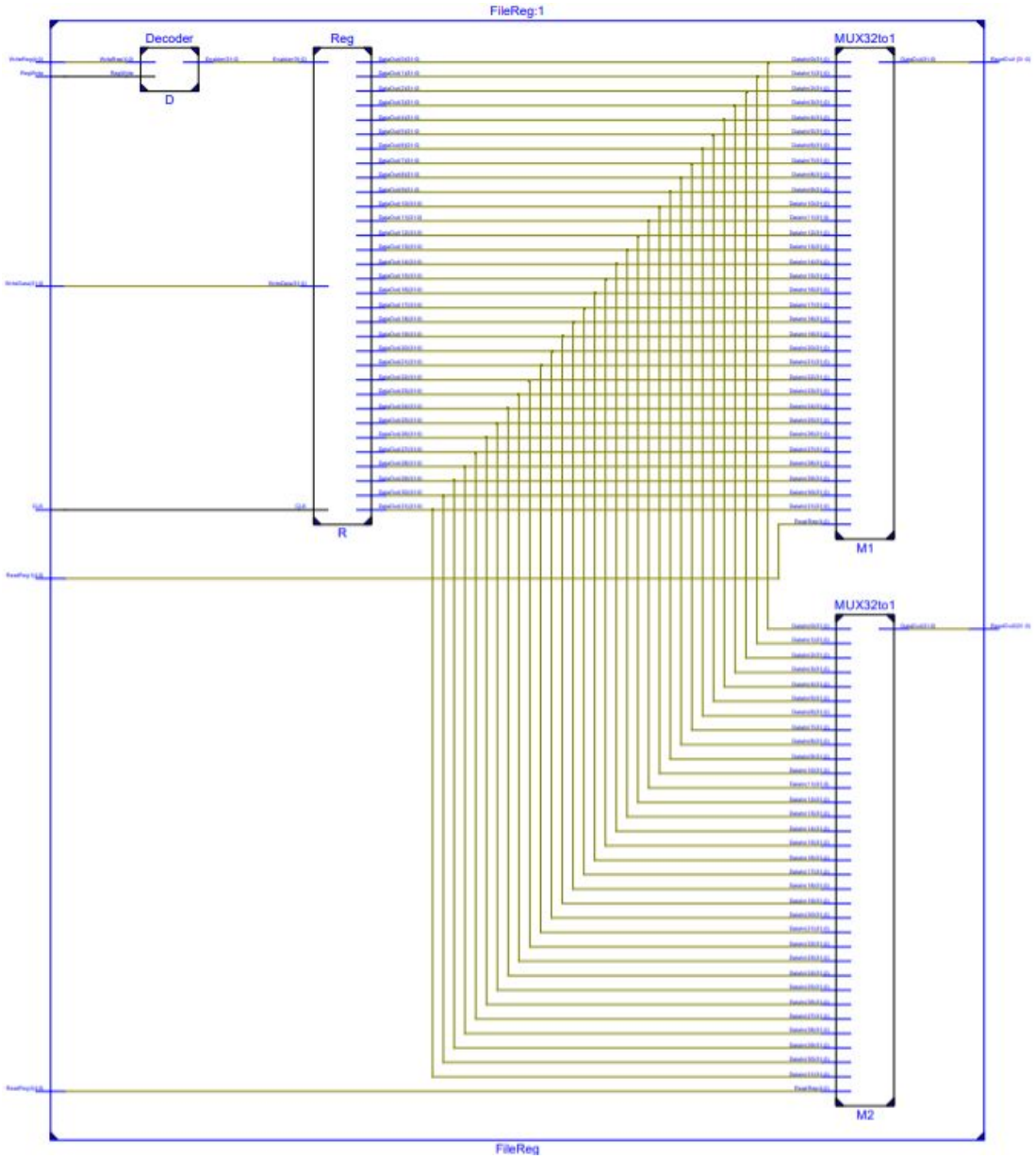

Imagen 2.4.2
Diagrama esquemático generado por VHDL del Register File

Donde se pueden apreciar los módulos conectados como están mostrados en el diagrama de la imagen 2.4.1.

Resultados

Se realizaron diferentes pruebas para cada uno de los módulos y todos funcionando en conjunto en el Register file.

Multiplexor

Para el módulo se guardaron los valores en las entradas (datain) 0: x"11111111", 1: x"22222222", 2: x"33333333", 17: x"17171717", 31: x"FFFFFFFF" y en los registros restantes 0.
Se leyeron las salidas de los registros 0, 1, 2, 3, 17, 31 insertando estos valores en la entrada de control (readreg) de manera secuencial en la salida dataout, en la imagen 3.1 se puede apreciar los resultados.

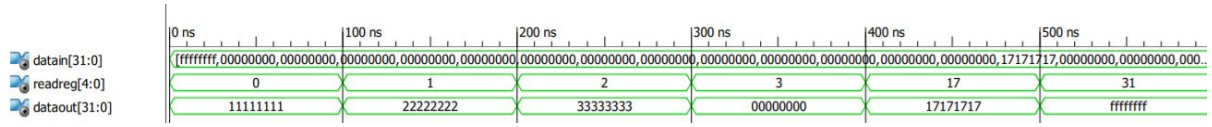

Imagen 3.1
Resultados de la prueba del multiplexor

Podemos apreciar que los resultados fueron exactamente los esperados.

Decoder 5x32

Para el módulo se hizo la prueba intentando habilitar las salidas 0 y 1 (mediante la entrada WriteReg) con el habilitador (RegWrite) desactivado esperando siempre un estado bajo en la salida e intentando habilitar las salidas 0, 1 y 31 con el habilitador activado. Los resultados se pueden apreciar en la imagen 3.2.

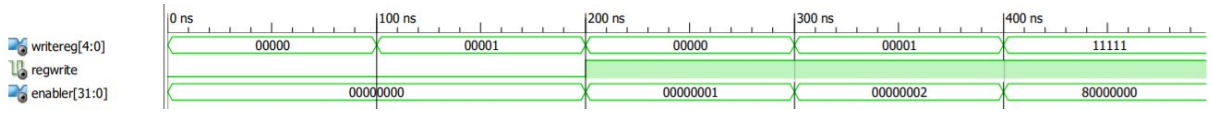

Imagen 3.2
Resultados de la prueba del Decoder 5x32

Podemos apreciar que los resultados fueron exactamente los esperados.

Registro

Para el módulo se hicieron intentos de escritura con distintos datos (mediante la entrada WriteData) en los registros 0, 1, 2, 31 con sus habilitadores (Enabler) encendidos y apagados y un intento de sobreescritura en el segundo registro, esperando cuando esta el habilitador desactivado que no haya una escritura y que en el primer registro siempre su salida sea un estado bajo en todas sus salidas. Los resultados se pueden apreciar en la imagen 3.3.

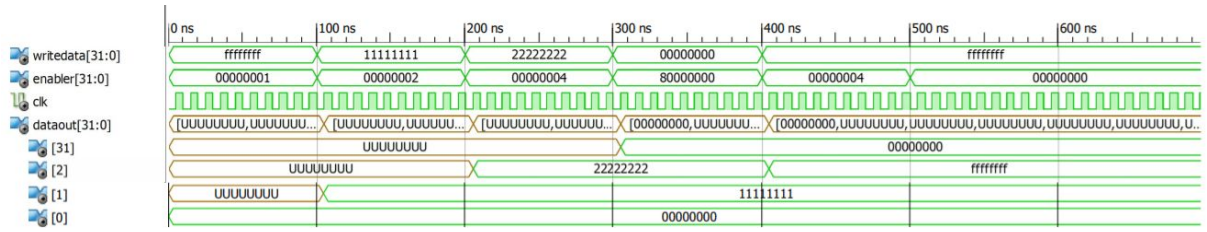

Imagen 3.3
Resultados de la prueba del Registro

Podemos apreciar que los resultados fueron exactamente los esperados.

Register File

Para este módulo se intentó registrar en la entrada de 32 bits (WriteData) el dato x"F0F0F0F0" mientras que el habilitador (RegWrite) estaba activado y desactivado en los registros 0, 1 y 3 intentando leer (en la salida readout) el mismo registro en el primer y segundo multiplexor (mediante ReadReg) a la vez y distintos en el mismo tiempo. Esperando que en el registro 0 no se guarde ningún dato y siempre su salida sea 0, en el 1 y 3 teniendo la salida x"F0F0F0F0" después de su escritura y sin escritura mientras que el habilitador está desactivado. Los resultados se pueden apreciar en la imagen 3.4.

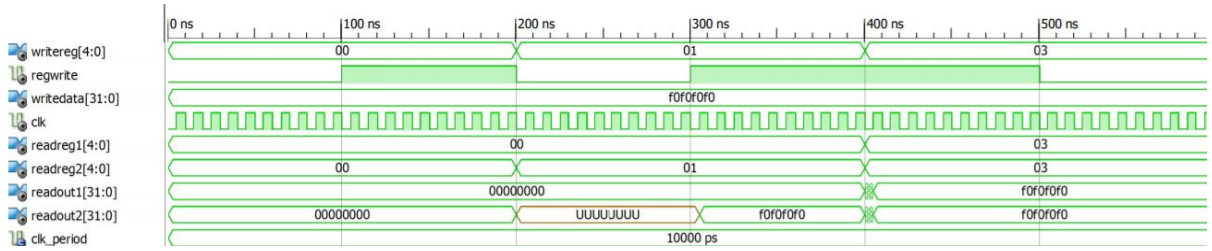

Imagen 3.4
Resultados de la prueba del Register File

Podemos apreciar que los resultados fueron exactamente los esperados.

Conclusión

Mediante el método de for generate y los paquetes podemos generar código bastante comprensible y pequeño, esto nos muestra que VHDL tiene herramientas muy fuertes que simplifican mucho la escritura del código pero se tiene que investigar y leer la documentación del mismo para llegar a estas soluciones. Gracias a estos métodos su implementación fue rápida y sencilla haciendo la funcionalidad del diagrama en un tiempo reducido y código compacto.

Este módulo que es utilizado como medio de memoria utiliza módulos más pequeños para su funcionalidad y la habilidad de segmentar para repartir los distintos módulos junto con la habilidad de juntarla definitivamente hace una diferencia para repartir el trabajo en un equipo y explotar las habilidades de los integrantes para ejecutar un proyecto de manera rápida utilizando herramientas como VHDL que nos permiten este tipo de ejecuciones. De igual manera sirve para encontrar errores, probar los módulos de manera individual y si es

que se comparte, entender los diferentes módulos por partes y así hacer su comprensión más fácil.

Gracias a las técnicas mencionadas anteriormente no se tuvo problema con ningún módulo.

Reflexion Individual

Miguel Morales:

> En esta práctica pude entender uno de los componentes esenciales en la arquitectura computacional, también ver cómo hemos creado hasta ahora muchos módulos que cada vez se entrelazan más entre sí. Uno de los mayores obstáculos enfrentados en la práctica fue la creación del arreglo de registros, aunque esto fue posible por la declaración de un tipo de señal la cual es un arreglo.
> También cabe mencionar que la forma en cómo se crearon los canales de comunicación entre módulos para el “Register File” fue una nueva forma de ver como se puede cablear de manera lógica varios canales en una sola declaratoria.

Javier Mondragon:

> Gracias a la comprensión de las técnicas que VHDL nos ha facilitado por medio de funciones y paquetes se facilita la generación de código repetido secuencialmente o la utilización de paquetes para tener un amplio rango de puertos sin tener que escribir cada uno de manera individual y tener el acceso de manera sencilla. En este proyecto se tuvo más trabajo de equipo que los anteriores ya que se utilizaron métodos que uno u el otro no comprendía al cien por ciento y el explicar la funcionalidad de cada uno mejoraba la comprensión de cada uno del módulo más grande y nos ayudó a comprender de qué otras maneras se pueden resolver los problemas de VHDL generando circuitos con otros métodos vistos en Sistemas Digitales Avanzados.

Referencias

- https://stackoverflow.com/questions/28468334/using-array-of-std-logic-vector-as-a-port-type-with-both-ranges-using-a-generic
- https://stackoverflow.com/questions/28468334/using-array-of-std-logic-vector-as-a-port-type-with-both-ranges-using-a-generic
- https://jnjsite.com/vhdl-multiplexor-de-4-a-1-con-seleccion-de-2-bits/
- https://www.hdlworks.com/hdl_corner/vhdl_ref/VHDLContents/PortMap.htm