

Práctica VHDL – Implementación de un banco de registros

Xabier Iglesias Pérez (xabier.iglesias.perez@udc.es)
Alonso Rodriguez Iglesias (alonso.rodriguez@udc.es)

Banco de registros: (registersBank.vhd)

El primer fichero es el correspondiente a la implementación del banco de registros.

Se compone de una entidad *registersBank* con los elementos:

- clk y reset como líneas de entrada binarias,
- rs, rt y rd como los selectores de registro (arrays lógicos de 5 bits),
- rdValue como el valor a escribir (array lógico de 32 bits),
- EscrReg como entrada binaria que indica si estamos ante una operación de escritura,
- regA y regB como arrays binarios de 32 bits que contienen la salida de una lectura sobre el banco.

```
entity registersBank is
  Port (
    clk, reset : in  std_logic;
    rs, rt, rd  : in  std_logic_vector(4 downto 0);
    rdValue     : in  std_logic_vector(31 downto 0);
    EscrReg     : in  std_logic;
    regA, regB  : out std_logic_vector(31 downto 0)
  );
end entity registersBank;
```

Internamente contiene un array de registros (arrays binarios de 32 bits) definido de la siguiente manera:

```
type registros_array is array (0 to 31) of std_logic_vector (31 downto 0);
signal regs : registros_array;
```

El algoritmo que modela su comportamiento tiene en cuenta un factor clave:

La salida de los registros A y B está retrasada con respecto al flanco de subida que inicia una operación de lectura.

```
begin
  -- ponemos las salidas al valor de los registros (async)
  regA <= regs(to_integer(unsigned(rs)));
  regB <= regs(to_integer(unsigned(rt)));

  sync:process(clk, reset)
  begin
```

Si no tuviéramos en cuenta este efecto, el resultado de una operación de escritura y lectura consecutivas vendría retrasado un ciclo como mínimo.

En el orden de operación, para cada flanco de subida hacemos:

1. Ponemos el registro 0 a 0
2. Si la línea de reset está a 1
 1. Inicializamos cada registro como su número de índice. Ej: reg1 = 1, reg2 = 2, reg3 = 3, etc.
3. Si no, si la línea EscrReg está a 1 y no estamos intentando escribir en el registro 0
 1. Escribimos el valor en el registro correspondiente. Ej: reg(5) = 2

```

sync:process(clk, reset)
begin
    -- ponemos el registro $0 a 0
    regs(0) <= "00000000000000000000000000000000";

    if rising_edge(clk) then
        -- reset síncrono
        if reset = '1' then
            for I in 1 to 31 loop
                regs(I) <= std_logic_vector(to_unsigned(I, regA'length));
            end loop;
        else
            if EscrReg = '1' then
                -- no debemos sobrescribir el $0
                if rd /= "00000" then
                    regs(to_integer(unsigned(rd))) <= rdValue;
                end if;
            end if;
        end if;
    end if;
end process;

```

De manera asíncrona, fuera del proceso, escribimos en los registros de salida los valores indicados por los selectores adecuados.

```

-- ponemos las salidas al valor de los registros (async)
regA <= regs(to_integer(unsigned(rs)));
regB <= regs(to_integer(unsigned(rt)));

```

TestBench (TB_registersBank.vhd)

En el caso del testbench, emplearemos un archivo de datos que contiene, en este orden (rs, rt, rd, rdValue, EscrReg, Valor esperado para A, Valor esperado para B)

Con este sistema, podemos cargar pruebas nuevas con suma facilidad. (solo hay que abrir tb.dat y añadir una línea con los datos separados por espacios)

0	0	0	2	1	0	0
0	0	1	5	1	0	0
0	0	2	30	1	0	0
1	2	2	10	0	5	30
1	0	0	0	0	5	0
0	1	0	0	1	0	5
1	2	0	0	0	5	30
2	1	0	4	1	30	5
1	2	2	4	1	5	30
2	2	0	0	0	4	4

En tanto a la implementación, es bastante sencilla:

Contiene una estructura análoga a la que ya vimos en el apartado anterior. A mayores, contiene variables necesarias para el manejo de ficheros y los resultados hardcodedos, así como algunas variables contador o límite.

```

signal clk : STD_LOGIC := '0';
signal rst, EscrReg : STD_LOGIC;
signal rs, rt, rd : std_logic_vector(4 downto 0);
signal rdValue, regA, regB : std_logic_vector(31 downto 0);
constant NUM_CICLOS : integer := 20;

constant NUM_COL : integer := 7;
type t_integer_array is array(integer range <>) of integer;
file file_handler : text open read_mode is "tb.dat";

signal temp1, temp2 : std_logic_vector(31 downto 0) := "00000000000000000000000000000000";

```

En tanto a las funciones que cumple el TestBench, estas son:

1: Simula el ciclo de reloj, con un tiempo de ciclo de 1 ns.

```

reloj: process
begin
    clk <= not clk;
    wait for 0.5 ns;
end process;

```

2: Envía la señal de reset.

```

-- creando el reset
ini: process
begin
    rst<='1';
    wait until clk'event and clk='1';
    rst<='0' after 2 ns;
    wait;
end process;

```

3: Para cada línea de tb.dat, lee los parámetros y se los envía al banco de registros, luego compara los resultados y muestra un aviso si difieren de lo esperado.

El flujo completo resumido es:

Envía ceros a todas las entradas

```

rs <= "00000";
rt <= "00000";
rd <= "00000";
rdValue <= "00000000000000000000000000000000";
EscrReg <= '0';

```

Lee el fichero tb.dat e itera por cada línea guardándolo su contenido en un array

```

readline(file_handler, row);

for i in 1 to NUM_COL loop
    read(row, v_data_read(i));
end loop;

```

Guarda en las variables correctas los valores leídos:

```
rs <= std_logic_vector(to_unsigned(v_data_read(1),rs'length));
rt <= std_logic_vector(to_unsigned(v_data_read(2),rt'length));
rd <= std_logic_vector(to_unsigned(v_data_read(3),rd'length));
rdValue <= std_logic_vector(to_unsigned(v_data_read(4),rdValue'length));

EscrReg <= to_unsigned(v_data_read(5), 1)(0);

temp1 <= std_logic_vector(to_unsigned(v_data_read(6),regA'length));
temp2 <= std_logic_vector(to_unsigned(v_data_read(7),regB'length));
```

Compara los valores de la salida del banco de registros con los esperados:

```
assert (temp1 = regA)
  report "Error en el valor de regA"
  severity error;

assert (temp2 = regB)
  report "Error en el valor de regB"
  severity error;
```

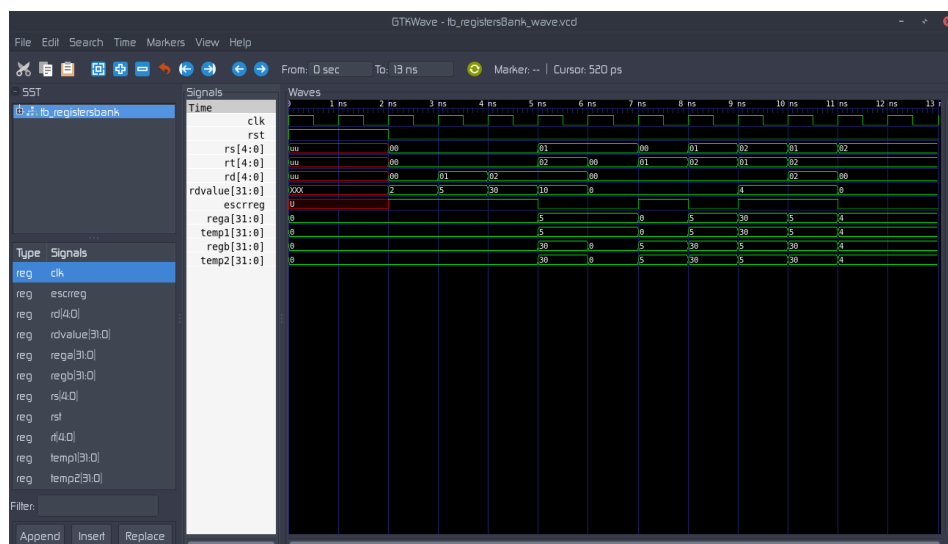
Resultados:

La salida resultante de ejecutar el programa es:

(Alonso, ejecuta y captura de la terminal)

La falta de mensajes de error, indica que todas las comparaciones se realizaron con éxito y por ende el banco de registros funciona correctamente.

La salida, vista desde GTKWave del TestBench es:



(No hace falta forzar la vista, la captura png en alta resolución está anexa como gtkwave.png)

Podemos observar resultados consecuentes con lo especificado en tb.dat