Tecnológico de Monterrey

Campus Monterrey

Materia

Arquitectura de computadoras (Gpo 1)

Tema

Práctica #1. Componentes generales del CPU

Intergantes

- ➔ Jorge Besnier A01039882
- ➔ Cinthia Portillo A00811827

Maestro

Diego Valencia

Fecha
25/08/20

Introducción

Con esta práctica damos inicio al laboratorio de Arquitectura de computadoras, en el cual vamos a utilizar VHDL para simular el comportamiento de una computadora codificando cada uno de sus módulos. Esta práctica inicial, busca que retomemos los conocimientos de Sistemas Digitales Avanzados y VHDL comenzando por los módulos más sencillos. En este caso el documento incluye descripción del módulo, el código VHDL, y los resultados obtenidos. Por lo tanto el documento tiene la siguiente estructura de contenido:

Desarrollo

Módulo 1: Add

Este módulo deberá realizar una suma 'unsigned' entre dos números de 32 bits de modo que la salida tenga la misma dimensión i.e. A+B=C.

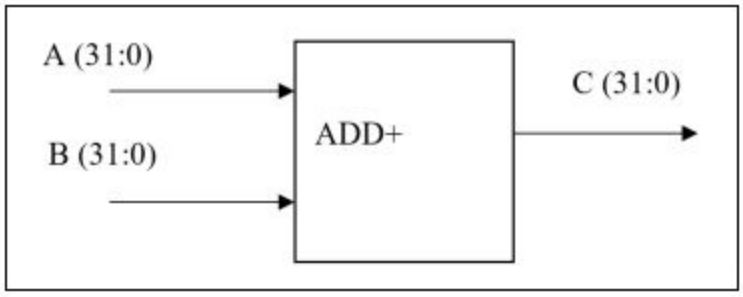

Figura 1.1. Módulo ADD para señales de 32 bits

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6
7
8  entity M1add is
9     Port ( a : in  STD_LOGIC_VECTOR(31 downto 0);
10           b : in  STD_LOGIC_VECTOR(31 downto 0);
11           c : out  STD_LOGIC_Vector(31 downto 0));
12 end M1add;
13
14 architecture Behavioral of M1add is
15
16 begin
17
18    c <= a + b;
19
20 end Behavioral;
```

Fig 1.2: Código Fuente Adder

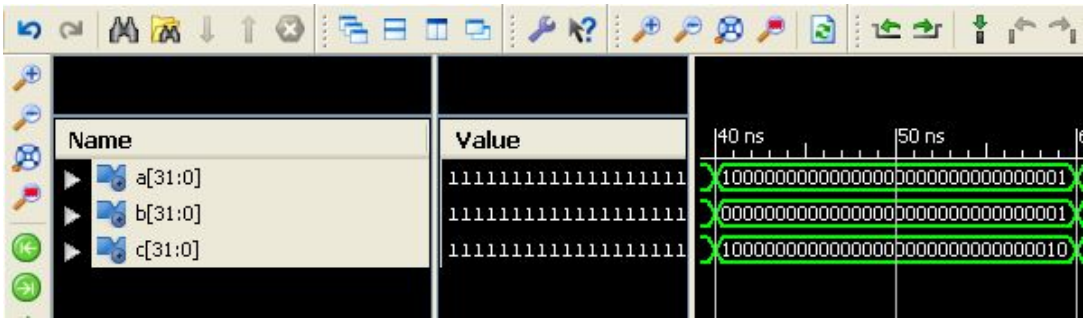

Fig 1.3: Simulación de Adder

Vemos en la simulación del funcionamiento en Fig 1.3 donde vemos el valor de a y el valor de b. Estos dos valores en binario de 32 bits hacen la suma de todos sus bits y la respuesta se almacena en la variable c.

Módulo 2: Shift Left 2 bits

Este módulo recibe un vector de 26 bits para hacer un "corrimiento lógico" de 2 posiciones y obtener un vector final de 28 bits.

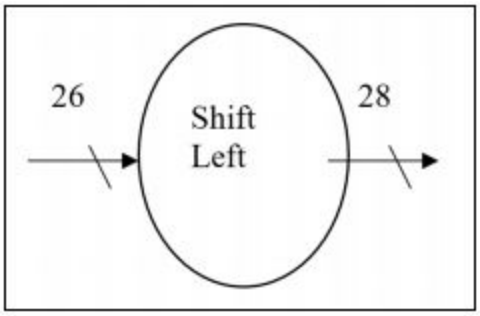

Figura 2.1. Módulo Shift Left con extensión para señales de 26 bits

```
19  ----------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24  use ieee.numeric_std.all;
25
26
27  entity mos2 is
28   Port (
29            Dint: in STD_LOGIC_VECTOR (25 downto 0);
30            Dout: out STD_LOGIC_VECTOR (27 downto 0)
31
32            );
33  end mos2;
34
35  architecture Behavioral of mos2 is
36
37  begin
38
39       Dout <= Dint & "00";
40
41  end Behavioral;
42
```

Fig 2.2: Código Fuente Shift Left con extensión

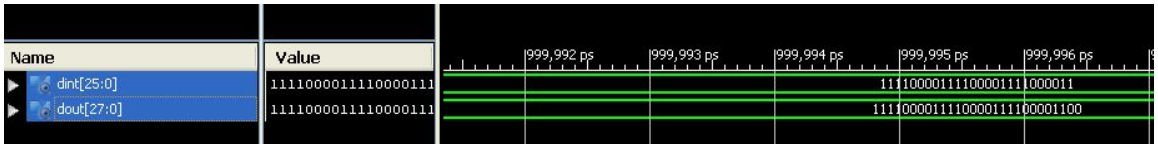

Fig 2.3: Simulación Shift Left

En la simulación Fig 2.3 podemos ver el resultado al ingresar un vector de 26 bits, el cual después de aplicar un corrimiento lógico de 2 posiciones a la izquierda nos da como resultado un vector final de 28, y los 2 nuevos bits son rellenados con '0'.

Módulo 3: Shift Left 2 bits para señales de 32 bits

A diferencia del módulo anterior, este corrimiento a la izquierda recibirá un vector de 32 bits y generará otro de 32 a la salida. Es decir, el corrimiento sigue el mismo concepto, pero el tamaño del vector de salida es el mismo que el de entrada.

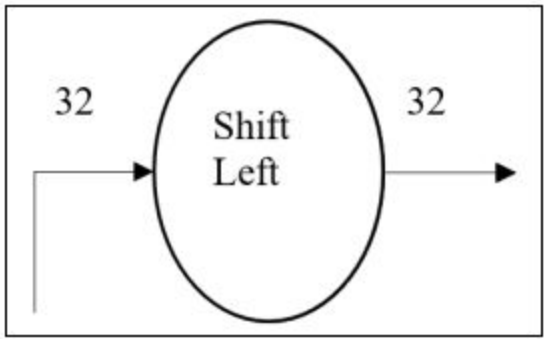

Figura 3.1 Módulo Shift Left para señales de 32 bits

```
19  ----------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24  use ieee.numeric_std.all;
25
26  entity moduleSL2bits is
27    Port (
28            Dint: in STD_LOGIC_VECTOR (31 downto 0);
29            Dout: out STD_LOGIC_VECTOR (31 downto 0)
30
31
32            );
33
34  end moduleSL2bits;
35
36  architecture Behavioral of moduleSL2bits is
37
38  begin
39
40        Dout <= (Dint(29 downto 0) & "00");
41
42  end Behavioral;
43
44
```

Fig 3.2: Código Fuente Shift left para 32 bits

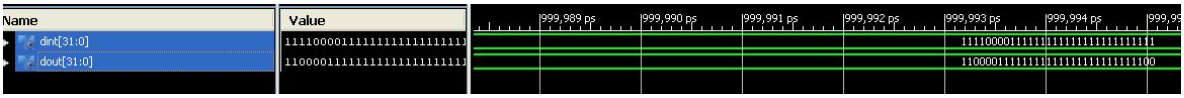

Fig 3.3: Simulación Shift left para 32 bits

En la simulación Fig 3.3 podemos ver el resultado al ingresar un vector de 32 bits, el cual después de aplicar un corrimiento lógico de 2 posiciones a la izquierda nos da como resultado el vector con el corrimiento y los 2 bits menos significativos son rellenados con '0'.

Módulo 4: Program Counter

Dada la siguiente entidad:

```
entity Program_counter is
   Port (D: in STD_LOGIC_VECTOR (31 downto 0 );
         Q: out STD_LOGIC_VECTOR (31 downto 0);
         RESET: in STD_LOGIC;
CLK:     in      STD_LOGIC);     end
Program_counter;
```

El 'program counter' no es más que un conjunto de Flip-Flop D's que, en cada transición negativa de reloj, Q obtiene lo que entre en D. La entrada asíncrona de RESET, pone en cero a todos los bits de la salida Q.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity M1ProgramCounter is
7      Port ( D : in  STD_LOGIC_VECTOR(31 downto 0);
8             Q : out  STD_LOGIC_VECTOR (31 downto 0);
9             RESET : in  STD_LOGIC;
10            CLK : in  STD_LOGIC);
11 end M1ProgramCounter;
12
13 architecture Behavioral of M1ProgramCounter is
14
15 begin
16
17    process(CLK, D, RESET)
18    begin
19       if RESET = '1' then Q <= "00000000000000000000000000000000";
20          elsif CLK'event and (CLK = '1') then Q <= D;
21       end if;
22    end process;
23
24 end Behavioral;
```

```
41
42    clk_process : process
43    begin
44       CLK<= '0';
45       wait for CLK_period/2;
46       CLK<= '1';
47       wait for CLK_period/2;
48    end process;
49
50
51    -- Stimulus process
52    stim_proc: process
53    begin
54
55       reset <= '1';
56
57       wait for 50ns;
58       reset <= '0';
59       D <= "00000000000000000000000000000000";
60
61       wait for 50ns;
62       reset <= '0';
63       D <= "11111111111111111111111111111111";
64
65       wait for 50ns;
66       reset <= '0';
67       D <= "00111111111111111111111111111111";
68
69       wait for 50ns;
70       reset <= '0';
71       D <= "00011111111100000000011111111111";
72
73
74       wait for 50ns;
75       reset <= '1';
76       D <= "00111111111111111111100000000000";
77       wait;
78    end process;
79
80 END;
```

Fig 4.1 y 4.2: Código fuente de Program Counter, y Testbench del Program Counter

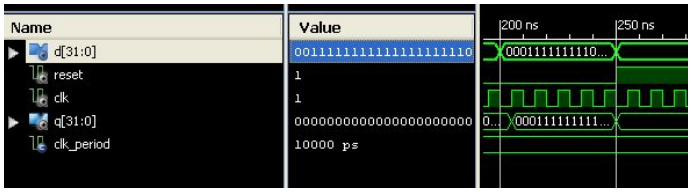

Fig 4.3: Simulación de Program Counter

En este caso, resulta un poco complejo simular el comportamiento porque depende de un reloj o CLK que va a determinar los cambios de estado. Por lo tanto lo clave para este simular un program counter debemos de iniciar un proceso como el que tenemos en el código de Testbench. Asimismo si se da un cambio en D la próxima transición de reloj positiva se verá reflejada en Q. Esto lo podemos ver en el figura 4.3 entra un valor en D y en la próxima transición se refleja ese valor en Q

Módulo 5: Sign extender

La extensión de signo es la operación en la cual se incrementa la cantidad de bits de un número preservando el signo y el valor del número original. Es llevada a cabo mediante agregación de dígitos del lado más significativo del número, siguiendo ciertos lineamientos dependiendo de la representación particular utilizada.

La función de este módulo es extender el tamaño de una variable al doble. Para esto, se recibe una señal de 16 bits que posteriormente pasa a 32 bits. La clave aquí es el bit más significativo del vector de entrada; si este bit está encendido (en '1'), entonces se agregan dieciséis 1's a la izquierda del vector de entrada para obtener uno de 32 a la salida. En caso contrario, si el MSB del vector de entrada se encuentra apagado (en '0'), ocurre lo mismo, pero con dieciséis 0's a la izquierda del vector de entrada.

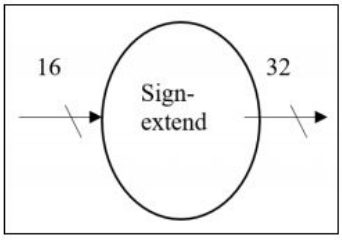

Figura 5.1. Módulo Sign Extender para señales de 16 bits

```
19  ----------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24  use ieee.numeric_std.all;
25
26
27
28  entity extender is
29  Port (
30            Dint: in STD_LOGIC_VECTOR (15 downto 0);
31            Dout: out STD_LOGIC_VECTOR (31 downto 0)
32
33            );
34  end extender;
35
36  architecture Behavioral of extender is
37
38  begin
39
40  process (Dint)
41
42  begin
43   if (Dint(15) = '1') then
44
45   Dout <= "1111111111111111" & Dint;
46   else
47   Dout <= "0000000000000000" & Dint;
48   end if;
49   end process;
50
51
52  end Behavioral;
```

Fig 5.2: Simulación de Program Sign Extender 16 bits

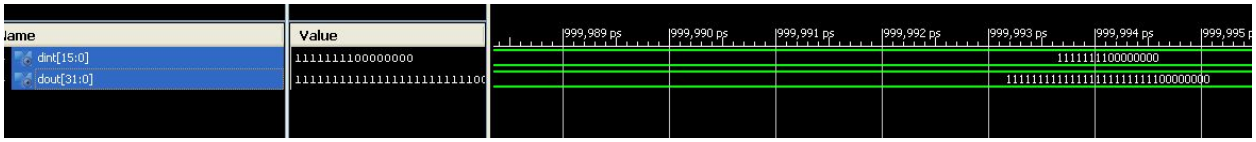

Fig 5.3: Simulación Sign Extender 16 bits con MSB en'1'

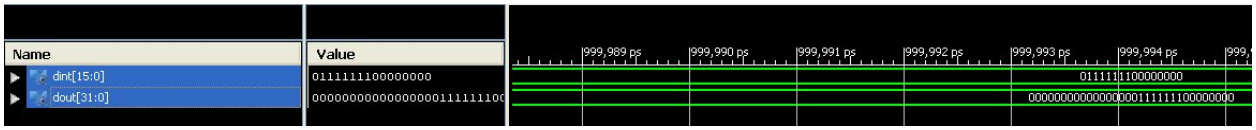

Fig 5.4: Simulación Sign Extender 16 bits con MSB en'0'

Para este módulo se extiende el tamaño de la variable de entrada al doble, recibimos una señal de 16 bits, y se genera una señal de 32 bits . El bit más significativo del vector de entrada nos dice si se agregan 1's o 0's a la derecha o a la izquierda.

En la fig 5.3 podemos ver que el MSB del vector de entrada es 1, por lo que se genera un vector de 32 bits, conservando el número original y agregando 16 1's a la izquierda. Por el contrario, en la fig 5.4 podemos ver que el BMS del vector de entrada es 1, por lo que se genera un vector de 32 bits, conservando el número original y agregando 16 0's a la izquierda.

Modulo 6: Multiplexores 2 a 1

Ahora, deberás diseñar dos multiplexores en módulos independientes. Si bien ambos contarán con una entrada selectora de un bit y dos más para datos, las entradas y la salida serán de 5 bits para uno y de 32 para el otro.

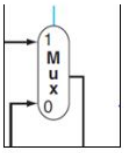

Figura 6.1 Módulo 5 multiplexor

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity M1Mux32bits is
    Port ( a : in  STD_LOGIC_VECTOR (31 downto 0);
           b : in  STD_LOGIC_VECTOR (31 downto 0);
           sel : in  STD_LOGIC;
           outMUX : out  STD_LOGIC_VECTOR (31 downto 0));
end M1Mux32bits;

architecture Behavioral of M1Mux32bits is

begin

   process (a,b,sel)
   begin
      case sel is
         when '0' => outMUX <=a;
         when '1' => outMUX <=b;
         when others => null;
      end case;
   end process;

end Behavioral;
```

Fig 6.2: Mux 32 bits secuencial

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity M1Mux32bits is
    Port ( a : in  STD_LOGIC_VECTOR (31 downto 0);
           b : in  STD_LOGIC_VECTOR (31 downto 0);
           sel : in  STD_LOGIC;
           outMUX : out  STD_LOGIC_VECTOR (31 downto
end M1Mux32bits;

architecture Behavioral of M1Mux32bits is

begin

   with sel select
      outMUX <= a when '0',
                b when others;

end Behavioral;
```

Fig 6.2: Mux 32 bits concurente

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity M1Mux5bits is
    Port ( a : in  STD_LOGIC_VECTOR (4 downto 0);
           b : in  STD_LOGIC_VECTOR (4 downto 0);
           sel : in  STD_LOGIC;
           outMUX : out  STD_LOGIC_VECTOR (4 downto 0));
end M1Mux5bits;

architecture Behavioral of M1Mux5bits is

begin

   process (a,b,sel)
   begin
      case sel is
         when '0' => outMUX <=a;
         when '1' => outMUX <=b;
         when others => null;
      end case;
   end process;

end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity M1Mux5bits is
    Port ( a : in  STD_LOGIC_VECTOR (4 downto 0);
           b : in  STD_LOGIC_VECTOR (4 downto 0);
           sel : in  STD_LOGIC;
           outMUX : out  STD_LOGIC_VECTOR (4 downto 0));
end M1Mux5bits;

architecture Behavioral of M1Mux5bits is

begin

   with sel select
      outMUX <= a when '0',
                b when others;

end Behavioral;
```

Fig 6.3: Mux 5 bits secuencial Fig 6.4: Mux 5 bits concurente

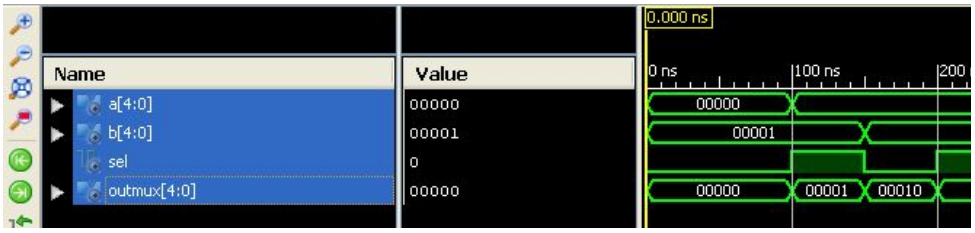

Fig 6.5: Simulación de Mux

En este módulo como en los anteriores es importante que el comportamiento sea concurrente, para evitar problemas más adelante con el desarrollo de la arquitectura computacional. Se presentan ambos casos porque se hizo la observación en revisión de implementación Por otra parte podemos ver que en la figura 6.5. Si se encuentra en 0 selecciona el valor de a, por el contrario si es 1 se selecciona el valor de b. Finalmente, se ve reflejado en outmux.

Conclusión

Jorge

Regresar a vhdl es muy complicado cuando llevas más de tres semestres sin utilizarlo. Recomiendo volver a los proyectos y tareas de Sistemas Digitales avanzados, sobre todo para recordar cómo hacer testbenches. Además, recomiendo el libro de la profesora de Tec Monterrey Norma Roffe "Diseño de sistemas digitales a través de diseños esquemáticos y VHDL. Volumen 1 Edición" para retomar de manera más eficiente la programación en VHDL. Finalmente, la recomendación más importante es tener cuidado con los recursos de internet realmente son soluciones que no te van a servir, nuevamente la recomendación es retomar las notas de sistemas digitales avanzados y verificar que sus diseños sean concurrentes.

Cinthia

Para esta práctica nos tomó un poco de tiempo volver a recordar cómo usar y programar en VHDL, ya que tanto mi compañero como yo llevamos varios semestres sin utilizarlos, pero con investigación pudimos refrescar nuestros conocimientos. En el desarrollo de los módulos fui recordando diferentes conceptos básicos para programar en VHDL, aprendí que existe un operador sll (shift left logical), pero para nuestra práctica no fue utilizado y desarrollamos los módulos del shift de una manera más sencilla para nosotros. Puedo decir que realizar esta actividad despertó mi interés en ir conociendo un poco más a detalle la arquitectura del procesador, es decir, cómo funciona cada módulo o el rol que tiene cuando vemos el diagrama del procesador.