Instituto Politécnico Nacional

Escuela Superior de Cómputo

Práctica 10 - Pila Hardware 2

Unidad de aprendizaje: Arquitectura de Computadoras

Grupo: 3CV1

Alumno(a):
Ramos Diaz Enrique

Profesor(a):
Vega García Nayeli

17 de mayo 2020

Índice

1. Código de implementación

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_arith.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity Pila is
    generic ( m : integer := 16;
              n : integer := 3);
    Port ( PCin : in STD_LOGIC_VECTOR (m-1 downto 0);
           clk, clr, wpc, up, dw : in STD_LOGIC;
           PCout : out STD_LOGIC_VECTOR (m-1 downto 0);
           SPout : out STD_LOGIC_VECTOR (n-1 downto 0));
end Pila;

architecture Behavioral of Pila is
    type banco is array (0 to (2**n)-1) of STD_LOGIC_VECTOR(m-1 downto 0);
    signal aux : banco;
begin
    process(clk, clr, aux)
        variable SP : integer range 0 to (2**n)-1;
    begin
        if (clr = '1') then
            SP := 0;
            aux <= (others => (others => '0'));
        elsif (rising_edge(clk)) then
            if (wpc = '0' and up = '0' and dw = '0') then
                aux(SP) <= aux(SP) + 1;
            elsif (wpc = '1' and up = '0' and dw = '0') then
                aux(SP) <= PCin;
            elsif (wpc = '1' and up = '1' and dw = '0') then
                SP := SP + 1;
                if(SP = 2**n) then
                    SP := 0;
                end if;
                aux(SP) <= PCin;
            elsif (wpc = '0' and up = '0' and dw = '1') then
                SP := SP - 1;
                if(SP = -1) then
                    SP := (2**n)-1;
                end if;
                aux(SP) <= aux(SP) + 1;
            end if;
```

```
        end if;
        PCout <= aux(SP);
        SPout <= conv_std_logic_vector(SP, n);
    end process;
end Behavioral;
```

2. Código de simulación

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_arith.all;
use IEEE.STD_LOGIC_unsigned.ALL;
use IEEE.STD_LOGIC_TEXTIO.ALL;
use STD.TEXTIO.ALL;

entity test_bench is
end test_bench;

architecture Behavioral of test_bench is
    component Pila is
        Port ( PCin : in STD_LOGIC_VECTOR (15 downto 0);
               clk, clr, wpc, up, dw : in STD_LOGIC;
               PCout : out STD_LOGIC_VECTOR (15 downto 0);
               SPout : out STD_LOGIC_VECTOR (2 downto 0));
    end component;

    signal PCin : STD_LOGIC_VECTOR (15 downto 0) := (others => '0');
    signal clk, clr, wpc, up, dw : STD_LOGIC;
    signal PCout : STD_LOGIC_VECTOR (15 downto 0);
    signal SPout : STD_LOGIC_VECTOR (2 downto 0);
begin
    stack: Pila Port map (
        PCin => PCin,
        clk => clk,
        clr => clr,
        wpc => wpc,
        up => up,
        dw => dw,
        PCout => PCout,
        SPout => SPout
    );

    reloj : process begin
```

```
36          clk <= '0';
37          wait for 5 ns;
38          clk <= '1';
39          wait for 5 ns;
40      end process;
41
42      process
43          file arch_res : text;   --Apuntadores tipo
            ↪  txt
44          variable linea_res : line;
45          variable var_pc_out : STD_LOGIC_VECTOR (15 downto 0);
46
47          file arch_en : text; --Apuntadores tipo txt
48          variable linea_en: line;
49          variable var_pc_in : STD_LOGIC_VECTOR (15 downto 0);
50          variable var_clr : STD_LOGIC;
51          variable var_wpc : STD_LOGIC;
52          variable var_up : STD_LOGIC;
53          variable var_dw : STD_LOGIC;
54          variable cadena : string (1 to 2);
55      begin
56          --- PCIN CLR WPC UP DW
57          file_open(arch_en, "Estimulos.txt", READ_MODE);
58
59          --- SP PC
60          file_open(arch_res, "Resultado.txt", WRITE_MODE);
61
62          cadena := "SP";
63          write(linea_res, cadena, right, cadena'LENGTH+1);--ESCRIBE LA cadena
            ↪  "SP"
64          cadena := "PC";
65          write(linea_res, cadena, right, cadena'LENGTH+1);--ESCRIBE LA cadena
            ↪  "PC"
66
67          writeline(arch_res, linea_res);-- escribe la linea en el archivo
68
69          for i in 1 to 24 loop
70              readline(arch_en, linea_en); -- lee una linea completa
71              --- PCIN CLR WPC UP DW
72
73              --Lee PCIN
74              Hread(linea_en, var_pc_in);
75              PCin <= var_pc_in;
76
```

```
77              --Lee CLR
78              read(linea_en, var_clr);
79              clr <= var_clr;
80
81              --Lee WPC
82              read(linea_en, var_wpc);
83              wpc <= var_wpc;
84
85              --Lee UP
86              read(linea_en, var_up);
87              up <= var_up;
88
89              --Lee DW
90              read(linea_en, var_dw);
91              dw <= var_dw;
92
93              wait until rising_edge(clk); --ESPERA AL FLANCO DE SUBIDA
94              wait for 0.1 ns;
95              var_pc_out := PCout;
96
97              --- SP PC
98              Hwrite(linea_res, SPout, right, 3);--ESCRIBE EL CAMPO SP
99              Hwrite(linea_res, var_pc_out, right, 5);--ESCRIBE EL CAMPO PCOUT
100
101             writeline(arch_res, linea_res);-- escribe la linea en el archivo
102         end loop;
103         file_close(arch_en);  -- cierra el archivo
104         file_close(arch_res);  -- cierra el archivo
105         wait;
106     end process;
107 end Behavioral;
```

3. Simulación

3.1. Programa a "ejecutar"

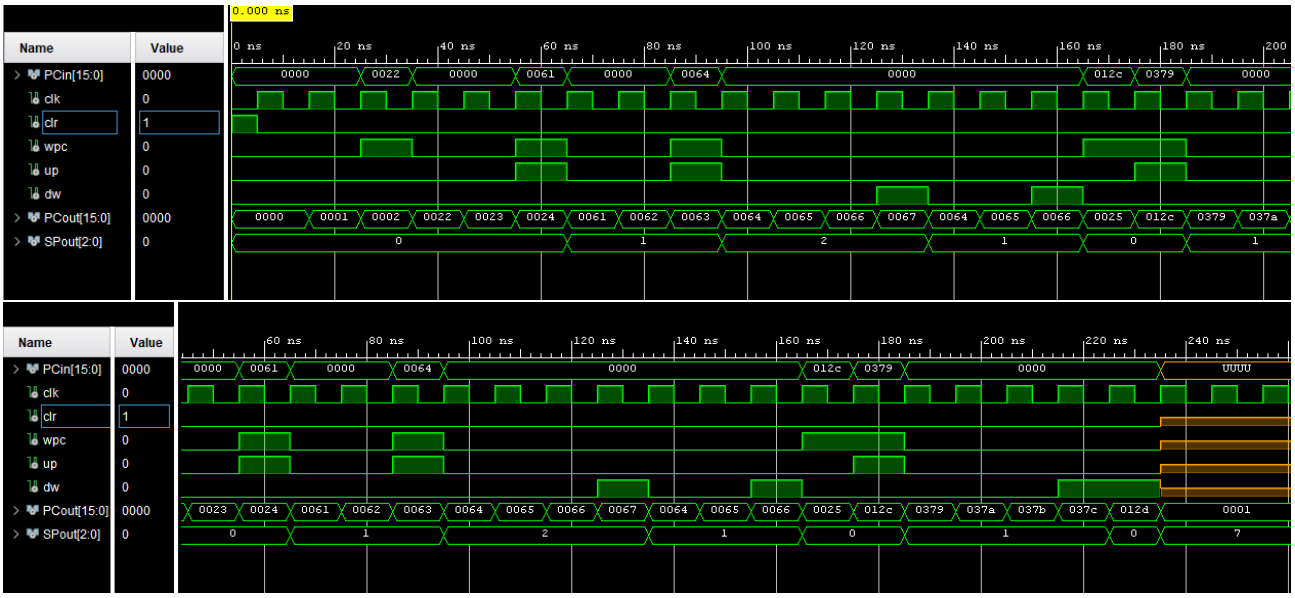
```
1  LI R6, #87
2  LI R8, #90
3  B 34
4  ADD R8, R2, R3
5  SUB R1, R2, R3
6  CALL 0x61
7  LI R6, #87
8  LI R8, #90
9  CALL 100
10 ADD R8, R2, R3
11 SUB R1, R2, R3
12 LI R6, #87
13 RET
14 SUB R1, R2, R3
```

```
LI R6, #87
RET
B 300
CALL 889
ADD R8, R2, R3
SUB R1, R2, R3
LI R6, #87
RET
RET
```

3.2. Archivo entrada: Estimulos.txt

```
0000 1 0 0 0
0000 0 0 0 0
0000 0 0 0 0
0022 0 1 0 0
0000 0 0 0 0
0000 0 0 0 0
0061 0 1 1 0
0000 0 0 0 0
0000 0 0 0 0
0064 0 1 1 0
0000 0 0 0 0
0000 0 0 0 0
0000 0 0 0 0
0000 0 0 0 1
0000 0 0 0 0
0000 0 0 0 0
0000 0 0 0 1
012c 0 1 0 0
0379 0 1 1 0
0000 0 0 0 0
0000 0 0 0 0
0000 0 0 0 0
0000 0 0 0 1
0000 0 0 0 1
```

3.3. Forma de onda de simulación

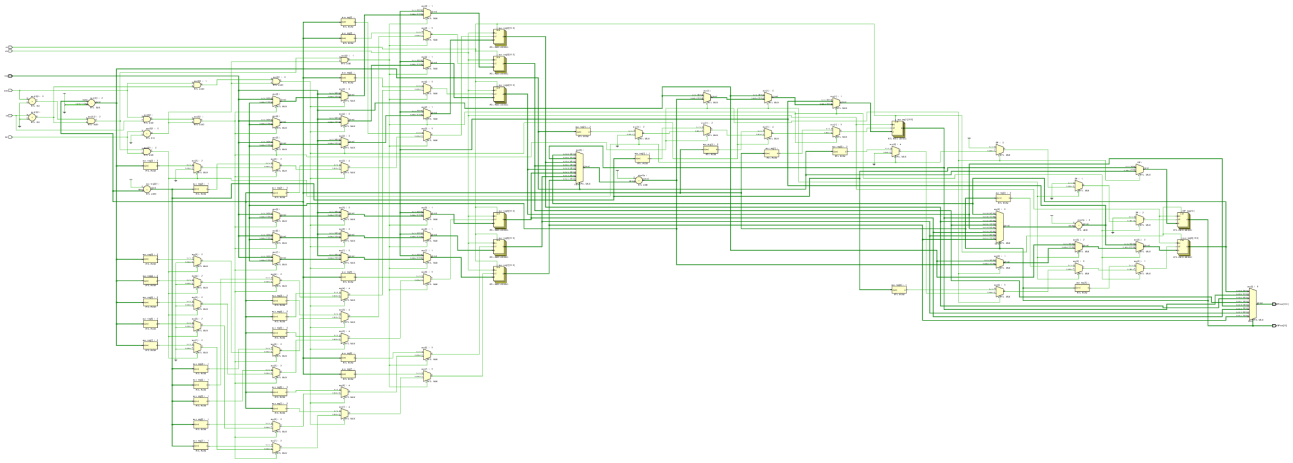


3.4. Archivo salida: Resultado.txt

```
 SP  PC
  0  0000
  0  0001
  0  0002
  0  0022
  0  0023
  0  0024
  1  0061
  1  0062
  1  0063
  2  0064
  2  0065
  2  0066
  2  0067
  1  0064
  1  0065
  1  0066
  0  0025
  0  012C
  1  0379
  1  037A
  1  037B
  1  037C
  0  012D
  7  0001
```

4. Diagramas RTL

4.1. Análisis RTL



4.2. Synthesis