Universidad Politécnica de Cartagena

Arquitectura Hardware de Comunicaciones

Informe Práctica 1

Ángel Truque Contreras
Isabel Bravo Sánchez

Octubre de 2024

Contenido

1. CÓDIGO IMPLEMENTADO

mult_sec.vhd Sat Sep 28 17:48:40 2024

```
  1  ----------------------------------------------------------------------------------
  2  -- Company:
  3  -- Engineer:
  4  --
  5  -- Create Date:    13:47:04 09/20/2024
  6  -- Design Name:
  7  -- Module Name:    mult_sec - Behavioral
  8  -- Project Name:
  9  -- Target Devices:
 10  -- Tool versions:
 11  -- Description:
 12  --
 13  -- Dependencies:
 14  --
 15  -- Revision:
 16  -- Revision 0.01 - File Created
 17  -- Additional Comments:
 18  --
 19  ----------------------------------------------------------------------------------
 20  library IEEE;
 21  use IEEE.STD_LOGIC_1164.ALL;
 22  use ieee.numeric_std.all;
 23
 24  entity mult_sec is
 25      Port ( -- inbus  : in std_logic_vector(7 downto 0);
 26             inbus_Q : in std_logic_vector(15 downto 0);
 27             inbus_M : in std_logic_vector(31 downto 0);
 28             -- outbus : out std_logic_vector(15 downto 0);
 29             outbus : out std_logic_vector(47 downto 0); -- nuevo resultado
 30             I      : in std_logic;
 31             -- weQ    : in std_logic;
 32             -- weM    : in std_logic;
 33             rst    : in std_logic;
 34             clk    : in std_logic);
 35  end mult_sec;
 36
 37  architecture Behavioral of mult_sec is
 38
 39     type type_state is (INICIO, SUMACU, DESDEC, CARGA_INICIAL, CARGA_SALIDA);
 40     signal state, nextstate: type_state;
 41
 42     -- signal CA, pp: std_logic_vector(8 downto 0);
 43     signal CA, pp: std_logic_vector(32 downto 0); -- un bit adicional por desbordamiento
 44     -- signal Q, M: std_logic_vector(7 downto 0);
 45     signal Q: std_logic_vector(15 downto 0);
 46     signal M: std_logic_vector(31 downto 0);
 47     signal init, ld, sh: std_logic;
 48     -- signal cnt: unsigned (2 downto 0);
 49     signal cnt: unsigned (3 downto 0); -- Vueltas que se van a hacer
 50     signal Z: std_logic;
 51     signal weQ, weM: std_logic; -- Las eliminamos como puertos de entrada y las
    ponemos como seÃ±ales equivalentes
 52
 53  begin
 54
 55     --Registro C y A:
 56     process(rst, clk)
```

Page 1

```
57      begin
58         if (rst='1') then
59            CA <= (others=>'0');
60         elsif rising_edge(clk) then
61            if (init='1') then
62               CA <= (others=>'0');
63            elsif (ld='1') then
64               CA <= pp;
65            elsif (sh='1') then
66               -- CA <= '0' & CA(8 downto 1);
67               CA <= '0' & CA(32 downto 1);
68            end if;
69         end if;
70      end process;
71
72      --Registro Q:
73      process(rst, clk)
74      begin
75         if (rst='1') then
76            Q <= (others=>'0');
77         elsif rising_edge(clk) then
78            if (weQ='1') then
79               -- Q <= inbus(7 downto 0);
80               Q <= inbus_Q(15 downto 0);
81            elsif (sh='1') then
82               -- Q <= CA(0) & Q(7 downto 1);
83               Q <= CA(0) & Q(15 downto 1);
84            end if;
85         end if;
86      end process;
87
88      --Registro M:
89      process(rst, clk)
90      begin
91         if (rst='1') then
92            M <= (others=>'0');
93         elsif rising_edge(clk) then
94            if (weM='1') then
95               -- M <= inbus(7 downto 0);
96               M <= inbus_M(31 downto 0);
97            end if;
98         end if;
99      end process;
100
101     --Sumador:
102     -- pp <= std_logic_vector(unsigned('0' & CA(7 downto 0)) + unsigned('0' & M));
103     pp <= std_logic_vector(unsigned('0' & CA(31 downto 0)) + unsigned('0' & M));
104
105     --Contador:
106     process(rst, clk)
107     begin
108        if (rst='1') then
109           cnt <= (others=>'0');
110        elsif rising_edge(clk) then
111           if (init='1') then
112              -- cnt <= "111";
113              cnt <= "1111"; -- 16 vueltas que se van a realizar (15 A 0)
```

```
114            elsif (sh='1') then
115               cnt <= cnt - 1;
116            end if;
117         end if;
118      end process;
119
120      --Detector de cero:
121      process(cnt)
122      begin
123         -- if (cnt="000") then
124         if (cnt="0000") then
125            Z <= '1';
126         else
127            Z <= '0';
128            end if;
129      end process;
130
131      --Unidad de control:
132      process(rst, clk)
133      begin
134         if (rst='1') then
135            state <= INICIO;
136         elsif rising_edge(clk) then
137            state <= nextstate;
138         end if;
139      end process;
140
141      process(state, I, Q(0), Z, CA)
142      begin
143         init <= '0'; ld <= '0'; sh <= '0';
144         nextstate <= state;
145         weQ <= '0'; weM <= '0'; -- para no tener que poner en cada estado
146         outbus <= (others => '0');
147
148         case state is
149
150            when INICIO =>
151               if (I='1') then
152                  nextstate <= CARGA_INICIAL;
153               else
154                  init <= '0';
155                  nextstate <= INICIO;
156               end if;
157
158            when CARGA_INICIAL =>
159                  init <= '1';
160                  nextstate <= SUMACU;
161                  weQ <= '1';
162                  weM <= '1'; -- a 1 para cargar en Q y M al principio
163
164            when SUMACU =>
165               if (Q(0)='1') then
166                  ld <= '1';
167               else
168                  ld <= '0';
169               end if;
170               nextstate <= DESDEC;
```

```
171
172            when DESDEC =>
173               if (Z='1') then
174                  nextstate <= CARGA_SALIDA;
175               else
176                  nextstate <= SUMACU;
177               end if;
178               sh <= '1';
179
180            when CARGA_SALIDA =>
181               outbus <= CA(31 downto 0) & Q;
182               nextstate <= INICIO;
183
184            when others =>
185               nextstate <= INICIO;
186         end case;
187      end process;
188
189   end Behavioral;
190
```

2. TEST BENCH

testbench_mult_sec.vhd Sat Sep 28 18:17:44 2024

```
1   --------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:   13:48:15 09/20/2024
6   -- Design Name:
7   -- Module Name:   /home/ise/P1ADH/testbench_mult_sec.vhd
8   -- Project Name:  P1ADH
9   -- Target Device:
10  -- Tool versions:
11  -- Description:
12  --
13  -- VHDL Test Bench Created by ISE for module: mult_sec
14  --
15  -- Dependencies:
16  --
17  -- Revision:
18  -- Revision 0.01 - File Created
19  -- Additional Comments:
20  --
21  -- Notes:
22  -- This testbench has been automatically generated using types std_logic and
23  -- std_logic_vector for the ports of the unit under test.  Xilinx recommends
24  -- that these types always be used for the top-level I/O of a design in order
25  -- to guarantee that the testbench will bind correctly to the post-implementation
26  -- simulation model.
27  --------------------------------------------------------------------------------
28  LIBRARY ieee;
29  USE ieee.std_logic_1164.ALL;
30
31  -- Uncomment the following library declaration if using
32  -- arithmetic functions with Signed or Unsigned values
33  --USE ieee.numeric_std.ALL;
34
35  ENTITY testbench_mult_sec IS
36  END testbench_mult_sec;
37
38  ARCHITECTURE behavior OF testbench_mult_sec IS
39
40      COMPONENT mult_sec
41     PORT(-- inbus  : IN std_logic_vector(7 downto 0);
42          inbus_Q : in std_logic_vector(15 downto 0);
43          inbus_M : in std_logic_vector(31 downto 0);
44          i       : IN std_logic;
45          -- weq    : IN std_logic;
46          -- wem    : IN std_logic;
47          rst     : IN std_logic;
48          clk     : IN std_logic;
49          -- outbus : OUT std_logic_vector(15 downto 0)
50          outbus : OUT std_logic_vector(47 downto 0));
51     END COMPONENT;
52
53     -- SIGNAL inbus :  std_logic_vector(7 downto 0):="00000000";
54     SIGNAL inbus_Q : std_logic_vector(15 downto 0):="0000000000000000";
55     SIGNAL inbus_M : std_logic_vector(31 downto 0):="00000000000000000000000000000000";
56     -- SIGNAL outbus :  std_logic_vector(15 downto 0);
57     SIGNAL outbus :  std_logic_vector(47 downto 0);
```

Page 1

```
58      SIGNAL i :  std_logic:='0';
59      -- SIGNAL weqm :  std_logic:='0';
60      -- SIGNAL wemm :  std_logic:='0';
61      SIGNAL rst :  std_logic:='1';
62      SIGNAL clk :  std_logic:='0';
63
64      constant periodo: time := 100 ns;
65
66    BEGIN
67
68    uut: mult_sec PORT MAP(
69          -- inbus => inbus,
70          inbus_Q => inbus_Q,
71          inbus_M => inbus_M,
72          outbus => outbus,
73          i => i,
74          -- weq => weqm,
75          -- wem => wemm,
76          rst => rst,
77          clk => clk  );
78
79      clk <= not clk after periodo/2;
80
81    -- Asignacion secuencial de estimulos
82      tb : PROCESS
83      BEGIN
84          rst <= '1';             --reset inicial
85          wait for 2*periodo;
86          rst <= '0';             --desactivamos el reset
87
88          -- PRUEBA
89
90          wait for 2*periodo;
91          inbus_M <= "00000000000000001000000000000001"; -- operando M = 32769 (65538 =
    2^16 + 2Â¹)
92          inbus_Q <= "0000000000000010"; -- operando Q = 2
93          wait for 1*periodo;
94          I <= '1';                 --inicio de la multiplicacion
95          wait for periodo;
96          I <= '0';
97          -- wait for 16*periodo; -- resultado aparece tras 16 ciclos
98          wait for 24*periodo;
99
100         wait;
101     end process;
102
103   END;
104
```

3. CAMBIOS RESPECTO A LA VERSIÓN ORIGINAL

```
-- inbus  : in std_logic_vector(7 downto 0);
inbus_Q : in std_logic_vector(15 downto 0);
inbus_M : in std_logic_vector(31 downto 0);
-- outbus : out std_logic_vector(15 downto 0);
outbus : out std_logic_vector(47 downto 0); -- nuevo resultac
I      : in std_logic;
-- weQ    : in std_logic;
-- weM    : in std_logic;
rst    : in std_logic;
clk    : in std_logic);
```

- **Inbus pasa de ser una señal de 8 bits a ser dos señales, una de 16 bits (inbus_Q -→ multiplicador) y otra de 32 bits →(inbus_M multiplicando)**
- **weQ y weM dejan de ser señales externas.**

```
-- signal CA, pp: std_logic_vector(8 downto 0);
signal CA, pp: std_logic_vector(32 downto 0); -- un bit adicional por desbordamiento
-- signal Q, M: std_logic_vector(7 downto 0);
signal Q: std_logic_vector(15 downto 0);
signal M: std_logic_vector(31 downto 0);
signal init, ld, sh: std_logic;
-- signal cnt: unsigned (2 downto 0);
signal cnt: unsigned (3 downto 0); -- Vueltas que se van a hacer
signal Z: std_logic;
signal weQ, weM: std_logic; -- Las eliminamos como puertos de entrada y las ponemos como seÃ±ales equivalente
```

- **Señales CA y pp (producto parcial) pasan a ser de 33 bits. Se le da un bit más para que no desborde la multiplicación.**
- **Señales Q y M pasan a ser del tamaño de inbus_Q y inbus_M respectivamente**
- **Señal cnt (contador) pasa a ser de 4 bits para poder contar 16 veces (de 15 a 0) de acuerdo al número de bits de Q.**

```
elsif (sh='1') then
   -- CA <= '0' & CA(8 downto 1);
   CA <= '0' & CA(32 downto 1);
end if;
```

- En el registro C y A cambia el tamaño de desplazamiento (32 bits más significativos en vez de 8).

```
if (weQ='1') then
   -- Q <= inbus(7 downto 0);
   Q <= inbus_Q(15 downto 0);
elsif (sh='1') then
   -- Q <= CA(0) & Q(7 downto 1);
   Q <= CA(0) & Q(15 downto 1);
```

- En el registro Q cambia el tamaño de la carga y el desplazamiento respecto al nuevo tamaño de Q.

```
if (weM='1') then
   -- M <= inbus(7 downto 0);
   M <= inbus_M(31 downto 0);
end if;
```

- inbus_M ha cambiado su tamaño a 32 bits.

```
--Sumador:
-- pp <= std_logic_vector(unsigned('0' & CA(7 downto 0)) + unsigned('0' & M));
pp <= std_logic_vector(unsigned('0' & CA(31 downto 0)) + unsigned('0' & M));
```

- **Cambio de tamaño de CA (escogemos los 32 bits menos significativos).**

```
if (init='1') then
   -- cnt <= "111";
   cnt <= "1111"; -- 16 vueltas que se van a realizar (15 A 0)
elsif (sh='1') then
```

- En el proceso del contador, ahora se van a realizar 16 repeticiones, de 15 a 0 (número de acuerdo al número de bits de Q – 16)

```
process(cnt)
begin
   -- if (cnt="000") then
   if (cnt="0000") then
```

- En el proceso de detector de cero, cnt ahora es de 4 bits (“0000”)

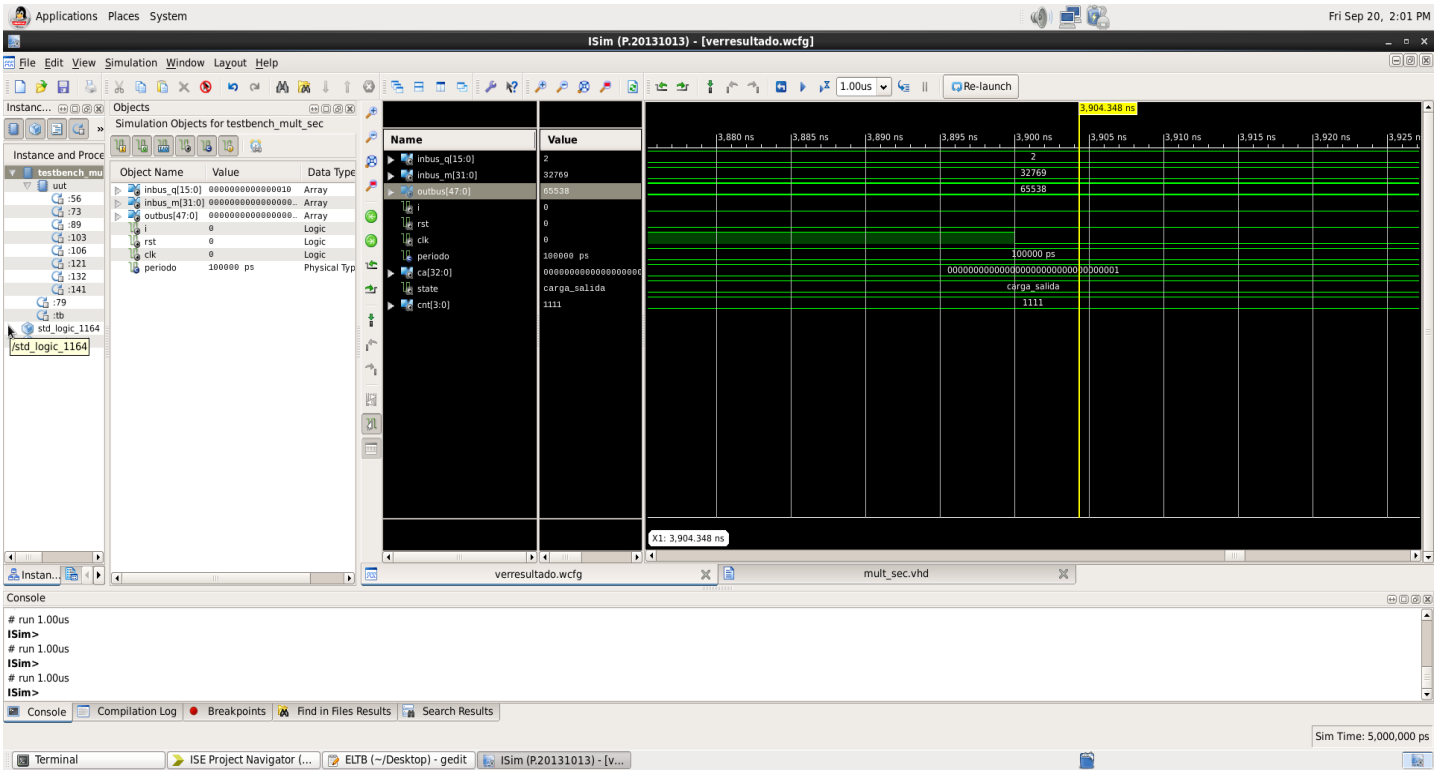
```
process(state, I, Q(0), Z, CA)
begin
   init <= '0'; ld <= '0'; sh <= '0';
   nextstate <= state;
   weQ <= '0'; weM <= '0'; -- para no tener que poner en cada estado
   outbus <= (others => '0');

   case state is

      when INICIO =>
         if (I='1') then
            nextstate <= CARGA_INICIAL;
         else
            init <= '0';
            nextstate <= INICIO;
         end if;

      when CARGA_INICIAL =>
            init <= '1';
            nextstate <= SUMACU;
            weQ <= '1';
            weM <= '1'; -- a 1 para cargar en Q y M al principio

      when SUMACU =>
           if (Q(0)='1') then
              ld <= '1';
           else
              ld <= '0';
           end if;
           nextstate <= DESDEC;

        when DESDEC =>
           if (Z='1') then
              nextstate <= CARGA_SALIDA;
           else
              nextstate <= SUMACU;
           end if;
           sh <= '1';

        when CARGA_SALIDA =>
           outbus <= CA(31 downto 0) & Q;
           nextstate <= INICIO;

        when others =>
           nextstate <= INICIO;
     end case;
  end process;

end Behavioral;
```

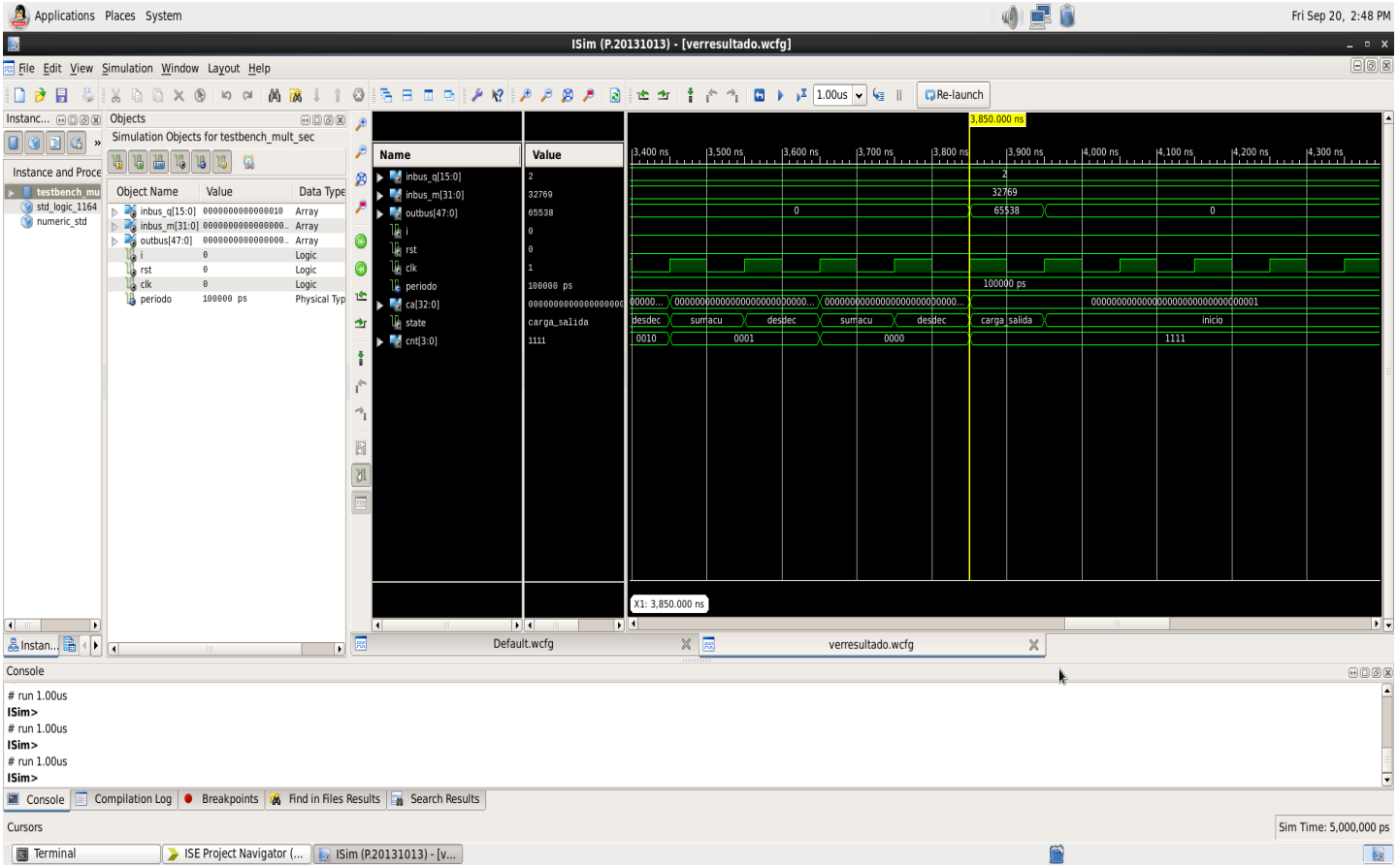
- En valores por defecto se han añadido tres señales: weQ, weM y outbus. Estas señales van a estar a cero por defecto excepto en estados concretos (CARGA_INICIAL → weQ, weM; CARGA_SALIDA → outbus).

Respecto a la creación de nuevos estados:

- Al principio pensamos en no crear ningún nuevo estado, introduciendo las nuevas señales de control en los flancos de transición. Pero esto nos causaba problemas con el reloj, pues la multiplicación empezaba con un ciclo de reloj de retraso al activarse los flancos de carga de buses e inicio de multiplicación simultáneamente (metaestabilidad).
- Para solucionarlo, creamos dos nuevos estados. Registros en los que se cargarán en un periodo de reloj los buses Q y M, y un periodo de reloj más tarde dará comienzo a la multiplicación. Hacemos lo mismo para la salida del resultado (estado CARGA_SALIDA).

4. SIMULACIÓN





El resultado de la multiplicación solo se muestra una vez que se ha obtenido el valor final:

32769*2 = 65538