Relatório 8 - Multiplicador de 64 bits

Patrik Loff Peres (20103830)
Universidade Federal de Santa Catarina (UFSC)
Departamento de Engenharia Elétrica e Eletrônica (DEEL)

I. Introdução

Neste laboratório foi projetado um circuito com estrutura *datapath* e controle que calcula o resultado **M** (128 bits) da multiplicação de dois números **A** (64 bits) e **B** (64 bits), utilizando a estratégia de soma e deslocamento[1]. O circuito tem uma entrada **go** (1 bit) que quando ativada indica o começo da operação, um **reset** (1 bit) e uma saída **idle** (1 bit) que deve ser ativa quando a operação estiver completa.

II. Descrição do Projeto

Inicialmente foi projetado o controle e *datapath* do circuito de acordo com as especificações e referência, com algumas alterações no proposto, principalmente na maquina de estados, mas mantendo a mesma lógica para realizar a operação de multiplicação. O diagrama da figura 1 mostra uma visão mais abrangente do circuito e os diagramas das figura 2 e 3 mostram mais detalhadamente o controle e *datapath*, respectivamente.

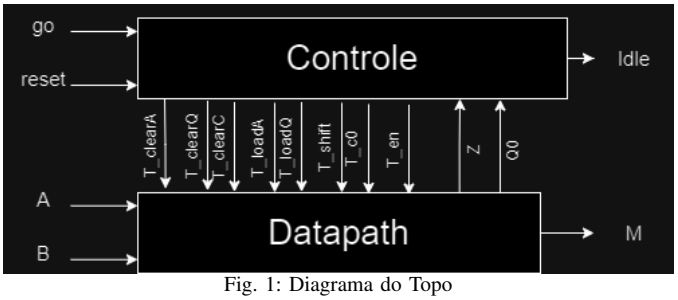

Fig. 1: Diagrama do Topo

Com os diagramas prontos foi implementado os códigos VHDL dos componentes necessários, são eles: Contador, Registrador de 64 bits com shift, Flip-Flop tipo D e Somador, além dos blocos de *datapath*, controle e o código topo, juntando os dois. Também, foi feito o *testbench* para simulação. Todos os VHDL estão no final do documento.

O Controle recebe os sinais de entrada **go** e **reset** e tem como saída oito sinais de controle para o *datapath* e uma saída de controle **idle**, e é composto por uma maquina de estados, que possui 6 possíveis estados: start, M0, MQ0, MQ10, MQ11 e Idle.

No estado start o circuito reseta e espera o sinal **go** para passar para o próximo estado, M0. No estado M0 o circuito avalia o valor de Q0, um sinal de status que indica se deve ser feita soma e deslocamento a direita (estados MQ10 e MQ11) ou somente o deslocamento a direta (estado MQ0). No estado

[1]D. Capson, "An example of ASM design: A binary multiplier", McMaster University

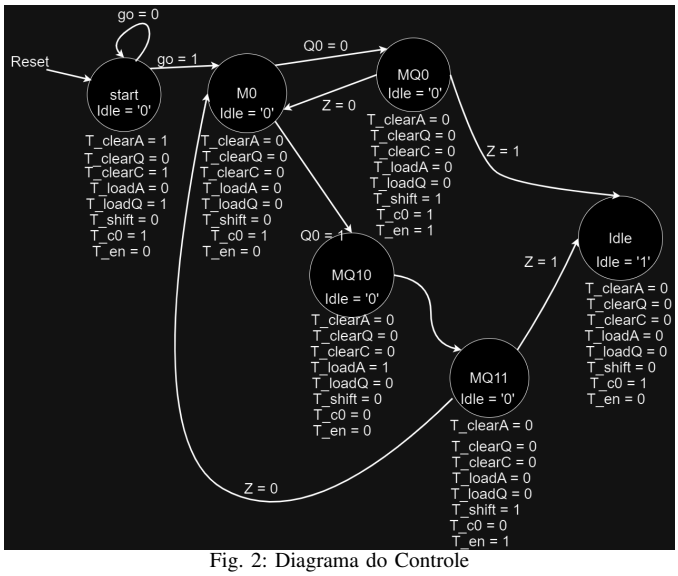

Fig. 2: Diagrama do Controle

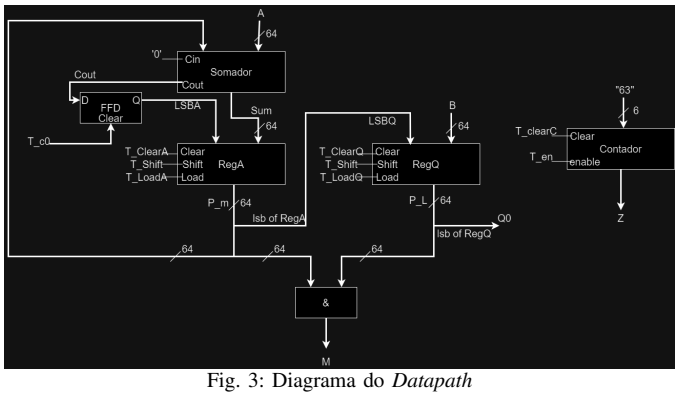

Fig. 3: Diagrama do *Datapath*

MQ10 é feita a soma do que está registrado em RegA com a entrada **A** e carregado no registrador, no estado MQ11 é feito o deslocamento a direta com a entrada serial de RegA sendo o **Cout** do somador. No estado MQ0 é feito o deslocamento a direta com a entrada serial de RegA sendo zero. Por fim, quando o sinal de *status* **Z** for 1, indicando que o contador chegou ao fim, o circuito passa para o estado Idle, em que ele mantém a resposta na saída e o sinal de controle **Idle** igual a um. O circuito possui *reset* que pode ser ativado em qualquer etapa de funcionamento.

O *datapath* recebe as entradas **A** e **B** e os sinais vindo do controle, e tem como saída dois sinais de *status* **Q0** que indica qual deve ser a operação realizada e **Z** que indica que a multiplicação foi realizada e uma saída de dados **M**.

O somador soma o valor da entrada **A** com o valor ar-

mazenado no registrador RegA, tendo como saida o resultado da soma (64 bits) e o Cout (1 bit).

O Flip-Flop tipo D é necessário para determinar se o bit serial que entra em RegA é zero ou o Cout.

Os registradores recebem sinais de load, shift e clear, além da entrada e da entrada serial. Quando acionado o sinal de shift o Registrador desloca o valor registrado 1 bit a direta e insere o valor da entrada serial a esquerda. A saída dos registradores são concatenadas para gerar a saída da multiplicação **M**.

O contador recebe o valor 63 na entrada e subtrai 1 a cada ciclo de relógio em que enable estiver ativado. O sinal clear faz o valor registrado internamente do registrador receba o valor de entrada, que é sempre 63. O sinal de enable é necessário pois um ciclo da operação de soma e shift demora mais que um ciclo de relógio.

Para o circuito funcionar é necessário manter a entrada **A** estável durante o cálculo e a saída **M** só fica estável enquanto o multiplicador não estiver fazendo cálculos, poderiam ser adicionados registradores na entrada e saída para tornar o circuito mais versátil.

III. Simulação *Zero-Delay*

Para verificar o funcionamento adequado do circuito foi realizada um simulação com atraso zero (figura 4).

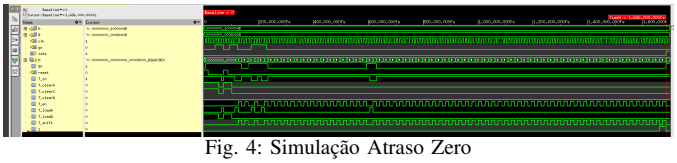

Fig. 4: Simulação Atraso Zero

Como é possível notar, o circuito pode ser resetado no meio do calculo, e o resultado está correto.

IV. Síntese Lógica

Com isso, foi realizada a síntese lógica do circuito, que resultou em 540 *standard cells*, cujo informações de área e *timing* estão nas figuras 5 e 6 respectivamente. Destaca-se nesses dados, a área total ocupada de $11859,723 \mu m^2$ e atraso critico de $10335 ps$, sendo ele o caminho de *carry-in* e *carry-out* do somador.

```
============================================================
  Generated by:           Genus(TM) Synthesis Solution 21.17-s066_1
  Generated on:           Jul 06 2024  03:37:57 pm
  Module:                 topo
  Technology libraries:   PnomV180T025_STD_CELL_7RF
                          physical_cells
  Operating conditions:   _nominal_
  Interconnect mode:      global
  Area mode:              physical library
============================================================

  Instance    Module   Cell Count  Cell Area  Net Area   Total Area
------------------------------------------------------------------
topo                        540  22853.914 11631.347    34485.260
  u_Controle controle        17    466.637   205.371      672.008
  u_Datapath datapath       523  22387.277  6913.140    29300.417
    Counter  contador        26    745.114   339.242     1084.356
    RegA     reg_64         134   6389.914  1617.106     8007.019
    RegQ     reg_64_1       132   6337.229  1284.202     7621.431
    flipflop FFD              2     75.264     0.000       75.264
    sum1     somador        229   8839.757  3019.967    11859.723
```

Fig. 5: Informações sobre área ocupada

```
  add_29_19_g1524__6260/COUT      ADDF_E          1 19.8  109  +152    9109 R
  add_29_19_g1523__5107/CIN                                 +0    9109
  add_29_19_g1523__5107/COUT      ADDF_E          1 19.8  109  +152    9262 R
  add_29_19_g1522__2398/CIN                                 +0    9262
  add_29_19_g1522__2398/COUT      ADDF_E          1 19.8  109  +152    9414 R
  add_29_19_g1521__5477/CIN                                 +0    9414
  add_29_19_g1521__5477/COUT      ADDF_E          1 19.8  109  +152    9566 R
  add_29_19_g1520__6417/CIN                                 +0    9566
  add_29_19_g1520__6417/COUT      ADDF_E          1 19.8  109  +152    9718 R
  add_29_19_g1519__7410/CIN                                 +0    9718
  add_29_19_g1519__7410/COUT      ADDF_E          1 18.5  104  +150    9868 R
  add_29_19_g1518__1666/B                                   +0    9868
  add_29_19_g1518__1666/Z         XOR2_C          2 28.2  232  +120    9987 R
  inc_add_29_31_g981__6783/B                                +0    9987
  inc_add_29_31_g981__6783/Z      XNOR2_C         1 18.1  166  +147   10134 R
sum1/S[63]
RegA/D[63]
  g3244__8428/C1                                            +0   10134
  g3244__8428/Z                   AO222_E         1 13.5  112  +148   10282 R
  aux_reg[63]/D             <<<   DFF_E                     +0   10282
  aux_reg[63]/CLK                 setup                 0   +53   10335 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clk)                       capture                         12000 R
-----------------------------------------------------------------------------
Cost Group   : 'clk' (path_group 'clk')
Timing slack :    1665ps
Start-point  : u_Datapath/RegA/aux_reg[0]/CLK
End-point    : u_Datapath/RegA/aux_reg[63]/D
```

Fig. 6: Informações sobre *timing*

V. Simulação Atraso Unitário

Em seguida foi realizada a simulação de atraso unitário para verificar que o circuito funciona adequadamente mesmo se entradas e saídas não mudarem instantaneamente após cada transição. O resultado encontra-se na figura 7.

Fig. 7: Simulação com atraso unitário

O circuito funciona como esperado e tem como saída o resultado correto da operação

VI. Síntese Física

Com a simulação de atraso unitário concluída, foi realizada a síntese física do circuito, que resultou no leiaute da figura 8. O leiaute passou nas verificações DRC e *Process Antenna* como mostram as figura 9 e 10.

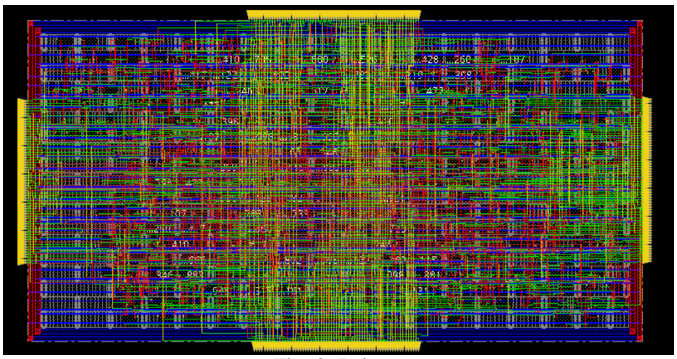

Fig. 8: Leiaute

VII. Simulação com atraso preciso

Com todas as informações do circuito é possível fazer uma simulação considerando o atraso mais próximo do real para o circuito sintetizado, considerando os atrasos das portas e do roteamento. A figura 11 mostra que o circuito continua funcionando como especificado.

```
*** Starting Verify DRC (MEM: 2035.7) ***

 VERIFY DRC ..... Starting Verification
 VERIFY DRC ..... Initializing
 VERIFY DRC ..... Deleting Existing Violations
 VERIFY DRC ..... Creating Sub-Areas
 VERIFY DRC ..... Using new threading
 VERIFY DRC ..... Sub-Area: {0.000 0.000 137.280 140.000} 1 of 2
 VERIFY DRC ..... Sub-Area : 1 complete 0 Viols.
 VERIFY DRC ..... Sub-Area: {137.280 0.000 268.240 140.000} 2 of 2
 VERIFY DRC ..... Sub-Area : 2 complete 0 Viols.

 Verification Complete : 0 Viols.

*** End Verify DRC (CPU: 0:00:00.1  ELAPSED TIME: 0.00  MEM: 264.1M) ***
```

Fig. 9: Verificação DRC

```
******* START VERIFY ANTENNA ********
Report File: topo.antenna.rpt
LEF Macro File: topo.antenna.lef
Verification Complete: 0 Violations
******* DONE VERIFY ANTENNA ********
(CPU Time: 0:00:00.1  MEM: 0.000M)
```

Fig. 10: Simulação com atraso unitário

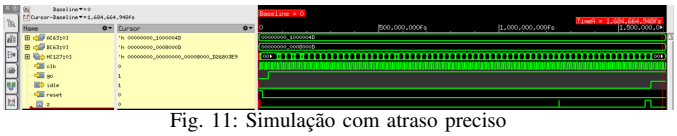

Fig. 11: Simulação com atraso preciso

Considerando os resultados obtemos a FoM = Área ocupada / frequência do relógio, portanto:

$$FoM = \frac{11859,723\mu m^2}{83,33MHz} = 142,32\mu m^2/MHz \qquad (1)$$

Topo

```
--
    ----------------------------------------------------

-- Arquivo Topo (controle + datapath)
--
    ----------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use work.all;

--
    ----------------------------------------------------


entity topo is

port( reset, go,clk:  in std_logic;
  A,B:    in std_logic_vector(63 downto 0);
  M:    out std_logic_vector(127 downto 0);
  idle:   out std_logic
);
end topo;

--
    ----------------------------------------------------


architecture behavior of topo is
signal T_clearA,T_clearQ,T_clearC,T_loadA,
    T_loadQ,T_shift,T_c0,T_en,z,Q0: std_logic;

component datapath is
port(   A:                  in
    std_logic_vector(63 downto 0);
  B:                  in std_logic_vector(63
      downto 0);
  T_clearA,T_shift,T_loadA,T_c0,T_clearQ,
      T_loadQ,T_clearC,T_en,clk:  in std_logic
      ;
  Q0, Z:                  out std_logic;
  M:                  out std_logic_vector(127
       downto 0)
);
end component;
component controle is
port( T_clearA,T_shift, T_loadA,T_clearQ,
    T_loadQ, T_c0, T_clearC, T_en,T_Idle: out
    std_logic;
  go, Q0, Z,rst,clk:              in
      std_logic
);
end component;
begin
u_Datapath: datapath port map (A,B,T_clearA,
    T_shift,T_loadA,T_c0,T_clearQ,T_loadQ,
    T_clearC,T_en,clk,Q0,Z,M);
u_Controle: controle port map (T_clearA,
    T_shift,T_loadA,T_clearQ,T_loadQ,T_c0,
    T_clearC,T_en,Idle,go,Q0,Z,reset,clk);

end behavior;
```

```
--
    ---------------------------------------------
```

Controle

```
--
    ---------------------------------------------

-- Controle
--
    ---------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use work.all;

--
    ---------------------------------------------


entity controle is

port( T_clearA,T_shift, T_loadA,T_clearQ,
    T_loadQ, T_c0, T_clearC, T_en,T_Idle: out
    std_logic;
  go, Q0, Z,rst,clk:                in
      std_logic
);
end controle;

--
    ---------------------------------------------


architecture behavior of controle is

type STATES is (start, M0, MQ0, MQ10, MQ11,
    Idle);
signal EA, PE: STATES;

begin
P1: process(clk, rst)
  begin
    if rst= '1' then
      EA <= start;
    elsif clk'event and clk= '1' then
      EA <= PE;
    end if;
  end process;

P2: process(EA,go,Z,Q0)
  begin
    case EA is
      when start =>
        T_clearA <= '1';
        T_shift <= '0';
        T_loadA <= '0';
        T_clearQ <= '0';
        T_loadQ <= '1';
        T_c0 <= '1';
        T_clearC <= '1';
        T_en <= '0';
        T_idle <= '0';
        if(go = '0') then
          PE <= start;
        else
          PE <= M0;
        end if;
      when M0 =>
        T_clearA <= '0';
        T_shift <= '0';
        T_loadA <= '0';
        T_clearQ <= '0';
        T_loadQ <= '0';
        T_c0 <= '1';
        T_clearC <= '0';
        T_en <= '0';
        T_idle <= '0';
        if(Q0 ='0') then
          PE <= MQ0;
        else
          PE <= MQ10;
        end if;
      when MQ0 =>
        T_clearA <= '0';
        T_shift <= '1';
        T_loadA <= '0';
        T_clearQ <= '0';
        T_loadQ <= '0';
        T_c0 <= '1';
        T_clearC <= '0';
        T_en <= '1';
        T_idle <= '0';
        if(Z ='0') then
          PE <= M0;
        else
          PE <= Idle;
        end if;
      when MQ10 =>
        T_clearA <= '0';
        T_shift <= '0';
        T_loadA <= '1';
        T_clearQ <= '0';
        T_loadQ <= '0';
        T_c0 <= '0';
        T_clearC <= '0';
        T_en <= '0';
        T_idle <= '0';
        PE <= MQ11;
      when MQ11 =>
        T_clearA <= '0';
        T_shift <= '1';
        T_loadA <= '0';
        T_clearQ <= '0';
        T_loadQ <= '0';
        T_c0 <= '0';
        T_clearC <= '0';
        T_en <= '1';
        T_idle <= '0';
        if(Z ='0') then
          PE <= M0;
        else
          PE <= Idle;
        end if;
      when Idle =>
        T_clearA <= '0';
        T_shift <= '0';
        T_loadA <= '0';
        T_clearQ <= '0';
        T_loadQ <= '0';
        T_c0 <= '1';
        T_clearC <= '0';
```

```
        T_en <= '0';
        T_idle <= '1';
        PE <= Idle;

    end case;
  end process;


end behavior;

--
    ----------------------------------------------
```

Datapath

```
--
    ----------------------------------------------

-- Datapath
--
    ----------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use work.all;

--
    ----------------------------------------------


entity datapath is

port( A:                  in std_logic_vector
    (63 downto 0);
  B:                  in std_logic_vector(63
      downto 0);
  T_clearA,T_shift,T_loadA,T_c0,T_clearQ,
      T_loadQ,T_clearC,T_en,clk:  in std_logic
      ;
  Q0, Z:                  out std_logic;
  M:                  out std_logic_vector(127
       downto 0)
);
end datapath;

--
    ----------------------------------------------


architecture behavior of datapath is
signal LSBA,LSBQ,cout: std_logic;
signal P_M,P_L,sum: std_logic_vector(63 downto
     0);
--signal RA,RQ: std_logic_vector(64 downto 0);
component somador is
port(   A:  in std_logic_vector(63 downto 0);
  B:  in std_logic_vector(63 downto 0);
  cin:  in std_logic;
  S:  out std_logic_vector(63 downto 0);
  cout: out std_logic);
end component;
component reg_64 is
port(   D:      in std_logic_vector(63 downto
    0);
  LSI:      in std_logic;
  clk,clear,load,shift: in std_logic;
  Q:      out std_logic_vector(63 downto 0)
);
end component;
component contador is
port(   clk:    in std_logic;
  clear,enable: in std_logic;
  A:    in std_logic_vector(5 downto 0);
  Z:    out std_logic);
end component;
component FFD is
port( clk,clear:  in std_logic;
  D:    in std_logic;
  Q:    out std_logic);
end component;
begin

RegA: reg_64 port map (sum,LSBA, clk, T_clearA
    , T_loadA,T_shift,P_M);
RegQ: reg_64 port map (B, LSBQ, clk, T_clearQ,
    T_loadQ,T_shift, P_L);
Counter: contador port map (clk,T_clearC,T_en,
    "111111",Z);
sum1: somador port map (A, P_M, '0',sum,cout);
flipflop: FFD port map (clk, T_c0, cout, LSBA)
    ;
M <= P_M & P_L;
Q0 <= P_L(0);
LSBQ <= P_M(0);



end behavior;

--
    ----------------------------------------------
```

Somador

```
--
    ----------------------------------------------

-- somador de 64 bits com carry in/out
--
    ----------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use work.all;

--
    ----------------------------------------------


entity somador is

port( A:  in std_logic_vector(63 downto 0);
  B:  in std_logic_vector(63 downto 0);
  cin:  in std_logic;
  S:  out std_logic_vector(63 downto 0);
  cout: out std_logic
);
end somador;
```

```
--
    ----------------------------------------------


architecture behavior of somador is
signal aux,cin_extended: std_logic_vector(64
    downto 0);
begin
cin_extended <= (others => '0');
cin_extended(0) <= cin;
aux <= unsigned(A)+unsigned(B)+ unsigned(
    cin_extended);
S <= std_logic_vector(aux(63 downto 0));
cout <= aux(64);

end behavior;

--
    ----------------------------------------------
```

Reg-64bits com shift

```
--
    ----------------------------------------------

-- Registrador de 8 bits com reset assincrono,
     enable e shift right
--
    ----------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use work.all;

--
    ----------------------------------------------


entity reg_64 is

port( D:      in std_logic_vector(63 downto 0)
    ;
  LSI:      in std_logic;
  clk,clear,load,shift: in std_logic;
  Q:      out std_logic_vector(63 downto 0)
);
end reg_64;

--
    ----------------------------------------------


architecture behavior of reg_64 is
signal aux: std_logic_vector(63 downto 0);
begin
P1:process(clk,clear,load,shift)
  begin
  if clk'event and clk = '1' then
    if clear = '1' then
      aux <= (others => '0');
    elsif load = '1' then
      aux <= D ;
    elsif shift = '1' then
      aux <= LSI & aux(63 downto 1);
    end if;
  end if;
end process;
Q <= aux;

end behavior;

--
    ----------------------------------------------
```

Contador

```
--
    ----------------------------------------------

-- Contador sincrono com clear e enable
--
    ----------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use work.all;

--
    ----------------------------------------------


entity contador is

port( clk:    in std_logic;
  clear,enable: in std_logic;
  A:    in std_logic_vector(5 downto 0);
  Z:    out std_logic
);
end contador;

--
    ----------------------------------------------


architecture arq_contador of contador is

--type STATES is (E0,E1,E2,E3,E4,E5,E6,E7,E8,
    E9);
--signal EA, PE: STATES;
--signal aux: std_logic_vector(5 downto 0):=
    "000000";
signal aux_unsigned: unsigned(5 downto 0);
begin
P1: process(clk, clear,enable,A)
  begin
    if clk'event and clk= '1' then
      if clear= '1' then
      aux_unsigned <= unsigned(A);

      elsif enable = '1'then
        --aux_unsigned <= unsigned(aux) - 1;
        --aux <= std_logic_vector(aux_unsigned
            );
        aux_unsigned <= aux_unsigned -1;
      end if;
    end if;
  end process;

Z <=  '1' when std_logic_vector(aux_unsigned)
    = "000000" else
```

```
  '0';

--P2: process(aux_unsigned)
--  begin
--    if std_logic_vector(aux_unsigned) =
    "000000" then
--      Z <= '1';
--    else
--      Z <= '0';
--    end if;
--  end process;
end arq_contador;

--
    ------------------------------------------------------------
```

Comparador Flip-Flip D

```
--
    ------------------------------------------------------------

-- Flip-Flop D
--
    ------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use work.all;

--
    ------------------------------------------------------------


entity FFD is

port( clk,clear:  in std_logic;
  D:    in std_logic;
  Q:    out std_logic
);
end FFD;

--
    ------------------------------------------------------------


architecture behavior of FFD is
begin
P1: process(clk, clear,D)
  begin
    if clear= '1' then
      Q<= '0';
    elsif clk'event and clk= '1' then
      Q <= D;
    end if;
  end process;
end behavior;
```