Relatório 8 - Exercício de aplicação

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 **R** (16 bits) da multiplicação de dois números **A** (8 bits) e **B** (8 bits), utilizando a estratégia de somas sucessivas. O circuito tem uma entrada **start** (1 bit) que quando ativada indica o começo da operação e uma saída **busy** (1 bit) que deve ser ativa apenas enquanto o circuito estiver ocupado calculando.

II. VHDL

Inicialmente foi implementado os código VHDL dos componentes necessários, são eles: Comparador, Contador, Registrador de 8 e 16 bits 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 diagrama que resume a organização do circuito está na figura 1.

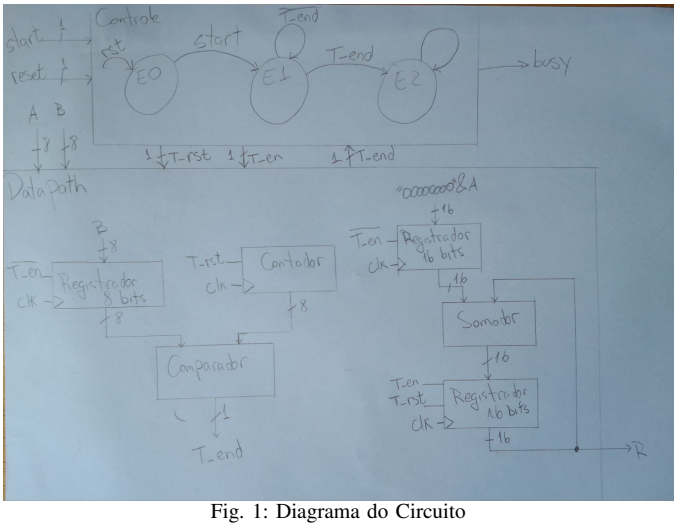

Fig. 1: Diagrama do Circuito

O Controle recebe os sinais de entrada **start** e **reset** e tem como saída dois sinais de controle para o *datapath* e uma saída de controle **busy**, e é composto por uma maquina de estados, que possui 3 possíveis estados: E0, E1 e E2. A tabela de transição de estados abaixo descreve o funcionamento do controle. O circuito possui *reset* que pode ser ativado em qualquer etapa de funcionamento. O circuito fica parado no estado E2 quando termina o calculo até que o **Reset** seja ativado.

Estado	Start	Reset	T_rst	T_en	T_end	Prox
E0	0	X	1	0	X	E0
E0	1	0	1	0	X	E1
E1	X	0	0	1	0	E1
E1	X	0	0	1	1	E2
E2	X	0	0	0	X	E2
E2	X	1	0	0	X	E0

TABLE I: Tabela de Transição de Estados

O *datapath* recebe as entradas **A** e **B** e os sinais vindo do controle, e tem como saída um sinal de *status* **T_end** que indica que a soma foi finalizada e o resultado **R**.

A entradas **A** e **B** são salvas em registradores, não sendo necessário mantê-las na entrada enquanto o calculo está sendo realizado. Uma ressalva para o registrador da entrada **A** tem que ser adequada à entrada do somador, que é de 16 bits, então é feita uma concatenação de oito zeros a esquerda antes de salvar a entrada no registrador. Os registradores recebem o sinal de controle de *enable* negados, para travar a atualização do registrador enquanto o calculo está sendo realizado.

O contador soma um a saída a cada sinal de *clock* e começa a contagem com um (quando reseta, seu valor de saída é atualizado para ”00000001”) por conta do sincronismo entre a soma do contador e a soma do operador, para garantir que o resultado seja correto.

O comparador recebe **B** e a saída do contador e atualiza a saída **T_end** para ’1’ quando o valor do contador é maior ou igual a **B**.

O somador soma o valor da entrada **A** com o valor armazenado no registrador de saída. Quando o calculo acaba, o resultado fica disponível até que o circuito seja resetado.

III. Simulação *Zero-Delay*

Para verificar o funcionamento adequado do circuito foi realizada um simulação com atraso zero (figura 2).

Fig. 2: Simulação Atraso Zero

Como é possível notar, o circuito pode ser resetado no meio do calculo, e o valor das entradas **A** e **B** variar depois que a multiplicação começa a ser feita não importa para o resultado.

IV. Síntese Lógica

Com isso, foi realizada a síntese lógica do circuito, que resultou em 149 *standard cells*, cujo informações de área e

timing estão nas figuras 3 e 4 respectivamente. Destaca-se nesses dados, a área total ocupada de $8470,452\mu m^2$ e atraso critico de $2673ps$.

```
legacy_genus:/> report area
====================================================================
  Generated by:           Genus(TM) Synthesis Solution 21.17-s066_1
  Generated on:           Jun 22 2024  07:15:47 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                   149   6036.173  2434.279     8470.452
legacy_genus:/>
```

Fig. 3: Informações sobre área ocupada

```
g2131__1881/Z                        NAND2_F       3 47.2  122   +79    1940 F
g2127__7098/A                                                +0    1940
g2127__7098/Z                        NOR2_E        2 33.9  169  +108    2047 R
g2123__2802/A                                                +0    2047
g2123__2802/Z                        NAND2_F       3 48.7  125   +81    2128 F
g2122/A                                                      +0    2128
g2122/Z                              INVERT_F      1 21.6   95   +74    2202 R
g2118__5526/A                                                +0    2202
g2118__5526/Z                        NAND2_F       1 20.4   82   +48    2250 F
g2114__6260/B                                                +0    2250
g2114__6260/Z                        XNOR2_C       1 15.1  164   +73    2323 R
g2109__2398/D1                                               +0    2323
g2109__2398/Z                        MUX21_D       1 13.7  123  +119    2442 R
u_Datapath_Regsum_Q_reg[15]/D   <<<  DFFR_E                   +0    2442
u_Datapath_Regsum_Q_reg[15]/CLK      setup                 0  +231    2673 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clk)                          capture                       10000 R
--------------------------------------------------------------------------
Cost Group   : 'clk' (path_group 'clk')
Timing slack :    7327ps
Start-point  : u_Datapath_Regsum_Q_reg[0]/CLK
End-point    : u_Datapath_Regsum_Q_reg[15]/D

legacy_genus:/>
```

Fig. 4: 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 5.

Fig. 5: Simulação com atraso unitário

O circuito funciona como esperado na maior parte da operação, as no final do cálculo, quando ele esta com o resultado na saída a alguns ciclos, sem que nenhum sinal de entrada mude, ele reseta a saída. Não foi identificado a origem do erro. O *bug* está destacado na figura 6

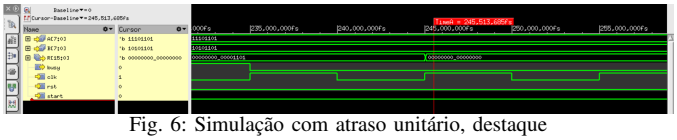

Fig. 6: Simulação com atraso unitário, destaque

VI. Síntese Física

Com a simulação de atraso unitário concluída, foi realizada a síntese física do circuito, que resultou no leiaute da figura 7. O leiaute passou nas verificações DRC e *Process Antenna* como mostram as figura 8 e 9.

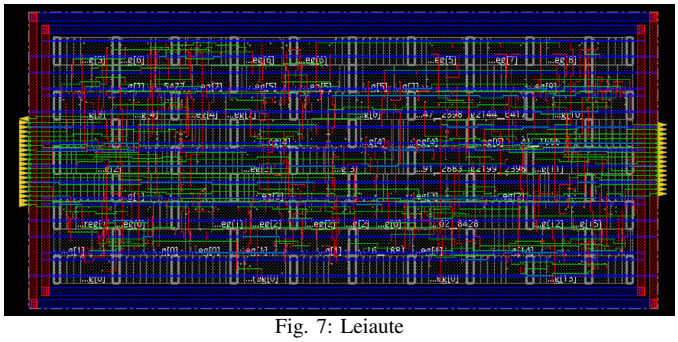

Fig. 7: Leiaute

```
*** Starting Verify DRC (MEM: 1993.8) ***

  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 0.000 155.120 72.800} 1 of 1
  VERIFY DRC ..... Sub-Area : 1 complete 0 Viols.

  Verification Complete : 0 Viols.

 *** End Verify DRC (CPU: 0:00:00.0  ELAPSED TIME: 0.00  MEM: 256.1M) ***

innovus 8>
```

Fig. 8: Verificação DRC

```
innovus 8>
******* START VERIFY ANTENNA ********
Report File: topo.antenna.rpt
LEF Macro File: topo.antenna.lef
Verification Complete: 0 Violations
******* DONE VERIFY ANTENNA ********
(CPU Time: 0:00:00.0  MEM: 0.000M)

innovus 8>
```

Fig. 9: Simulação com atraso unitário

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 10 mostra que o circuito continua funcionando como especificado, tendo um atraso um pouco maior que o unitário apos o sinal de *clock*, e algum ruido nas transições por alguns fentosegundos e depois estabiliza resultado correto.

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

$$FoM = \frac{8470,452\mu m^2}{100MHz} = 84704,52\mu m^2/MHz \quad (1)$$

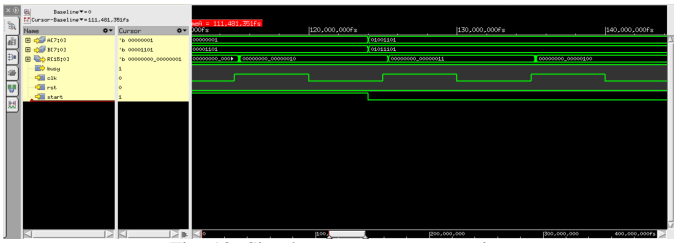

Fig. 10: Simulação com atraso preciso

Topo

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

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


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

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


entity topo is

port( rst, start,clk: in std_logic;
  A,B:    in std_logic_vector(7 downto 0);
  R:    out std_logic_vector(15 downto 0);
  busy:   out std_logic
);
end topo;

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


architecture behavior of topo is
signal S_rst,S_en,S_end: std_logic;

component datapath is
port(   A:    in std_logic_vector(7 downto 0);
  B:    in std_logic_vector(7 downto 0);
  T_rst,T_en,clk: in std_logic;
  T_end:    out std_logic;
  R:    out std_logic_vector(15 downto 0)
);
end component;
component controle is
port( T_rst,T_en,busy:  out std_logic;
  T_end, rst, start,clk:  in std_logic
);
end component;
begin
u_Datapath: datapath port map (A,B,S_rst,S_en,
    clk,S_end,R);
u_Controle: controle port map (S_rst, S_en,
    busy, S_end, rst, start,clk);

end behavior;

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

Controle

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

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


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

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


entity controle is

port( T_rst,T_en,busy:  out std_logic;
  T_end, rst, start,clk:  in std_logic
);
end controle;

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


architecture behavior of controle is

type STATES is (E0,E1,E2);
signal EA, PE: STATES;

begin
P1: process(clk, rst)
  begin
    if rst= '1' then
      EA <= E0;
    elsif clk'event and clk= '1' then
      EA <= PE;
    end if;
  end process;

P2: process(EA,rst,start,T_end)
  begin
    case EA is
      when E0 =>
        if(start = '0') then
          T_rst <= '1';
          T_en <= '0';
          busy <= '0';
          PE <= E0;
        else
          T_rst <= '1';
          T_en <= '0';
          busy <= '0';
          PE <= E1;
        end if;
      when E1 =>
        if(T_end='0') then
          T_rst <= '0';
          T_en <= '1';
          busy <= '1';
```

```
          PE <= E1;
        else
          T_rst <= '0';
          T_en <= '0';
          busy <= '1';
          PE <= E2;
        end if;
      when E2 =>
        T_rst <= '0';
        T_en <= '0';
        busy <= '0';
    end case;
  end process;


end behavior;

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

Datapath

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

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


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

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


entity datapath is

port( A:    in std_logic_vector(7 downto 0);
  B:    in std_logic_vector(7 downto 0);
  T_rst,T_en,clk: in std_logic;
  T_end:    out std_logic;
  R:    out std_logic_vector(15 downto 0)
);
end datapath;

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


architecture behavior of datapath is
signal cont,B1: std_logic_vector(7 downto 0);
signal A1,S1,S2,A_ex: std_logic_vector(15
    downto 0);
component somador is
port(   A:  in std_logic_vector(15 downto 0);
  B:  in std_logic_vector(15 downto 0);
  S:  out std_logic_vector(15 downto 0));
end component;
component reg_16 is
port(   D:  in std_logic_vector(15 downto 0);
  clk,rst,en: in std_logic;
  Q:  out std_logic_vector(15 downto 0)
);
end component;
component reg_8 is
port(   D:  in std_logic_vector(7 downto 0);
  clk,rst,en: in std_logic;
  Q:  out std_logic_vector(7 downto 0)
);
end component;
component comparador is
port(   A:  in std_logic_vector(7 downto 0);
  B:  in std_logic_vector(7 downto 0);
  T_end:  out std_logic);
end component;
component contador is
port(   clk:  in std_logic;
  rst:  in std_logic;
  count:  out std_logic_vector(7 downto 0));
end component;
begin
A_ex <= "00000000" & A;
RegA: reg_16 port map (A_ex, clk, '0', not(
    T_en), A1);
RegB: reg_8 port map (B, clk, '0', not(T_en),
    B1);
ContadorA: contador port map (clk,T_rst,cont);
Comp: comparador port map (cont, B1, T_end);
sum: somador port map (A1, S2, S1);
Regsum: reg_16 port map(S1, clk, T_rst, T_en,
    S2);
R <= S2;


end behavior;

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

Somador

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

-- somador de 16 bits
--
    ------------------------------------------------


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

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


entity somador is

port( A:  in std_logic_vector(15 downto 0);
  B:  in std_logic_vector(15 downto 0);
  S:  out std_logic_vector(15 downto 0)
);
end somador;

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

```
architecture behavior of somador is
begin
S <= A+B;


end behavior;

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

Reg-8bits

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

-- Registrador de 8 bits com reset assincrono
    e enable
--
    ----------------------------------------------


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

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


entity reg_8 is

port( D:  in std_logic_vector(7 downto 0);
  clk,rst,en: in std_logic;
  Q:  out std_logic_vector(7 downto 0)
);
end reg_8;

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


architecture behavior of reg_8 is
begin
P1:process(clk,rst)
  begin
  if rst = '1' then
    Q <= "00000000";
  elsif clk'event and clk = '1' and en = '1'
     then
    Q <= D;
  end if;
end process;


end behavior;

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

Reg-16bits

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

-- Registrador de 16 bits com reset assincrono
    e enable
--
    ----------------------------------------------


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

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


entity reg_16 is

port( D:  in std_logic_vector(15 downto 0);
  clk,rst,en: in std_logic;
  Q:  out std_logic_vector(15 downto 0)
);
end reg_16;

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


architecture behavior of reg_16 is
begin
P1:process(clk,rst)
  begin
  if rst = '1' then
    Q <= "0000000000000000";
  elsif clk'event and clk = '1' and en = '1'
     then
    Q <= D;
  end if;
end process;


end behavior;

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

Contador

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

-- Contador sincrono com reset de modulo 10
--
    ----------------------------------------------


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

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


entity contador is

port( clk:  in std_logic;
  rst:  in std_logic;
  count:  out std_logic_vector(7 downto 0)
```

```
);
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(7 downto 0):= "
    00000000";
begin
P1: process(clk, rst)
  begin
    if rst= '1' then
      count <= "00000001";
      aux <= "00000001";
    elsif clk'event and clk= '1' then
      aux <= aux + "00000001";
      count <= aux;
    end if;
  end process;

end arq_contador;

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

Comparador

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

-- Comparador assincrono
--
    ------------------------------------------------------------


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

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


entity comparador is

port(
  A:  in std_logic_vector(7 downto 0);
  B:  in std_logic_vector(7 downto 0);
  T_end:  out std_logic
);
end comparador;

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


architecture behavior of comparador is

begin
T_end <= '0' when A<B else
   '1';

end behavior;

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