Instituto Tecnológico de Aeronáutica

Apostila de EA-21: uma introdução a VHDL

Prof. Duarte Oliveira

Autor: Lucas Moura Santana

lmsantana129@gmail.com

2018

Sumário

1 Introdução

As partes de um código VHDL são conforme vistos na figura a seguir.

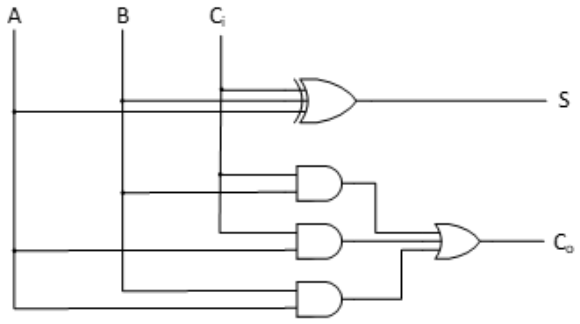
```
------------------------------------------------------------------
-- região com inclusão de bibliotecas e pacotes
library ieee;
use ieee.std_logic_1164.all
------------------------------------------------------------------
-- região de inclusão da entidade com as interfaces
entity circuito is
port (entrada1, entrada2: in bit;
      saida1, saida2: out bit);
end entity circuito;
------------------------------------------------------------------
-- região da descrição da arquitetura
architecture comportamento of circuito is

    -- região para declaração de componente, funções, sinais...
signal auxiliar: bit;

begin
    -- região para inclusão de código concorrente
saida1 <= entrada1 and entrada2;
auxiliar <= not entrada2;
saida2 <= auxiliar;

end architecure comportamento;
------------------------------------------------------------------
```

As características principais da linguagem que devemos ter em mente para este tutorial são

- Não é uma linguagem *case-sensitive*;
- Não é possível misturar tipos de variáveis por *default*;
- O comportamento padrão do código é concorrente;
- A linguagem não necessita de indentação;

2 Célula do somador completo

Para descrever o somador completo, levamos em conta o seguinte esquemático, equações booleanas e tabela verdade.



$$S = A \oplus B \oplus S$$
$$C_o = AB + AC_i + BC_i$$

A	B	C_i	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

2.1 Implementado das equações booleanas

A região de entidade desta implementação será a seguinte

```
entity somador_boolean is
port (A, B, CI: in bit;
      S, CO: out bit);
end entity somador_boolean;
```

A arquitetura do somador será como se segue

```
architecture boolean_equations of somador_boolean is

begin
S  <= A xor B xor CI;
CO <= (A and B) or (A and CI) or (B and CI);

end architecture boolean_equations;
```

2.2 Implementando da tabela verdade

A implementação da tabela verdade leva em conta o conjunto de valores de saída dado as combinações da entrada. Isso pode ser feito usando-se do comando *with select* e o comando *when else.*

A região da entidade será como segue

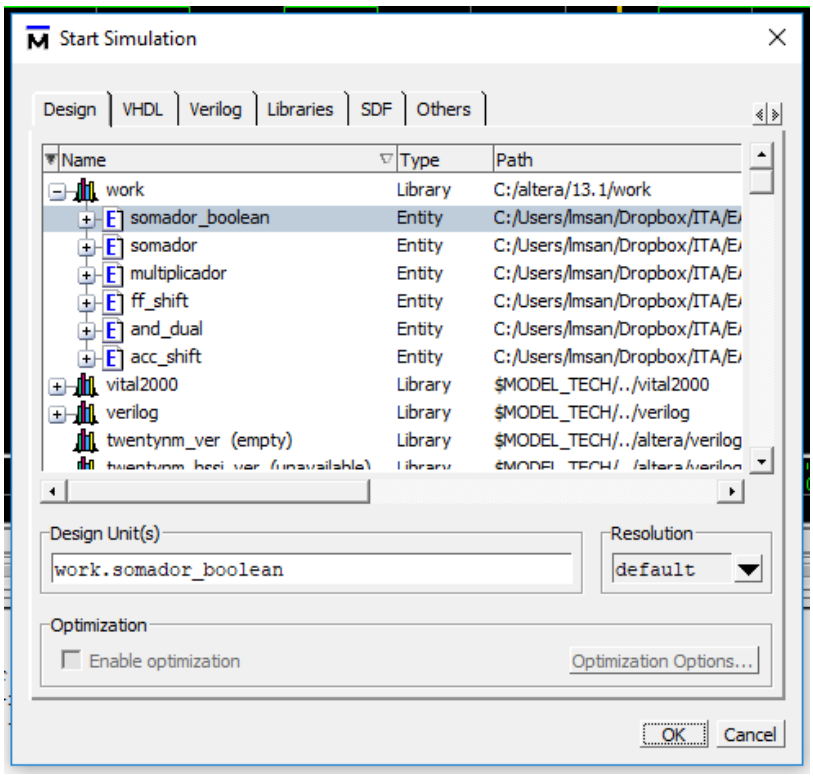
```
entity somador_table is
port (A, B, CI: in bit;
      S, CO: out bit);
end entity somador_table;
```

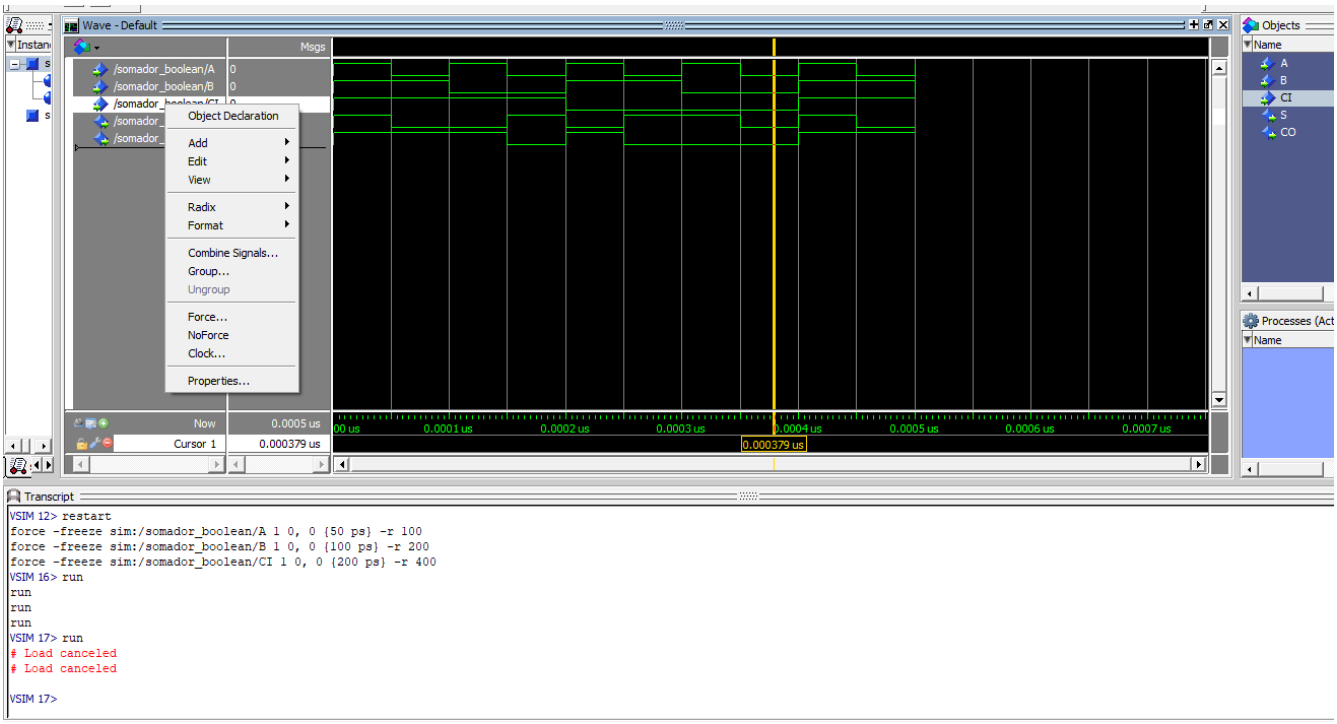
A arquitetura do somador será como segue

```
architecture truth_table of somador_table is
signal cubo : bit_vector (2 downto 0);

begin
cubo <= A & B & CI; -- concatenando sinais

-- comando concorrente when else
S <= '1' when (cubo = "001") or (cubo = "010") or (cubo = "100")
               or (cubo = "111") else
     '0';

-- comando concorrente with select
with cubo select
    CO <= '1' when "011" | "101" | "110" | "111",
          '0' when others;

end architecture truth_table;
```

3 Simulando no MODELSIM

1. Aba **Compile → Compile** e selecionar o arquivo .vhd
2. Aba **Simulate → Start Simulation** e selecionar a entidade compilada dentro da biblioteca **Work**
3. Arrastar os sinais para a janela **Wave** para ver seus valores e comportamento
4. Comandos importantes na janela de simulação
    a. **Force "Signal" 1**: faz com que o "signal" mude para 1 (tem que ser de entrada)
    b. **Force "Signal" 0**: faz com que o "signal" mude para 0 (tem que ser de entrada)
    c. **Run**: roda a simulação e calcula o valor dos sinais pelo tempo pré-definido (100ps)
    d. Clicando com o botão direito sobre o sinal é possível encontrar outras opções

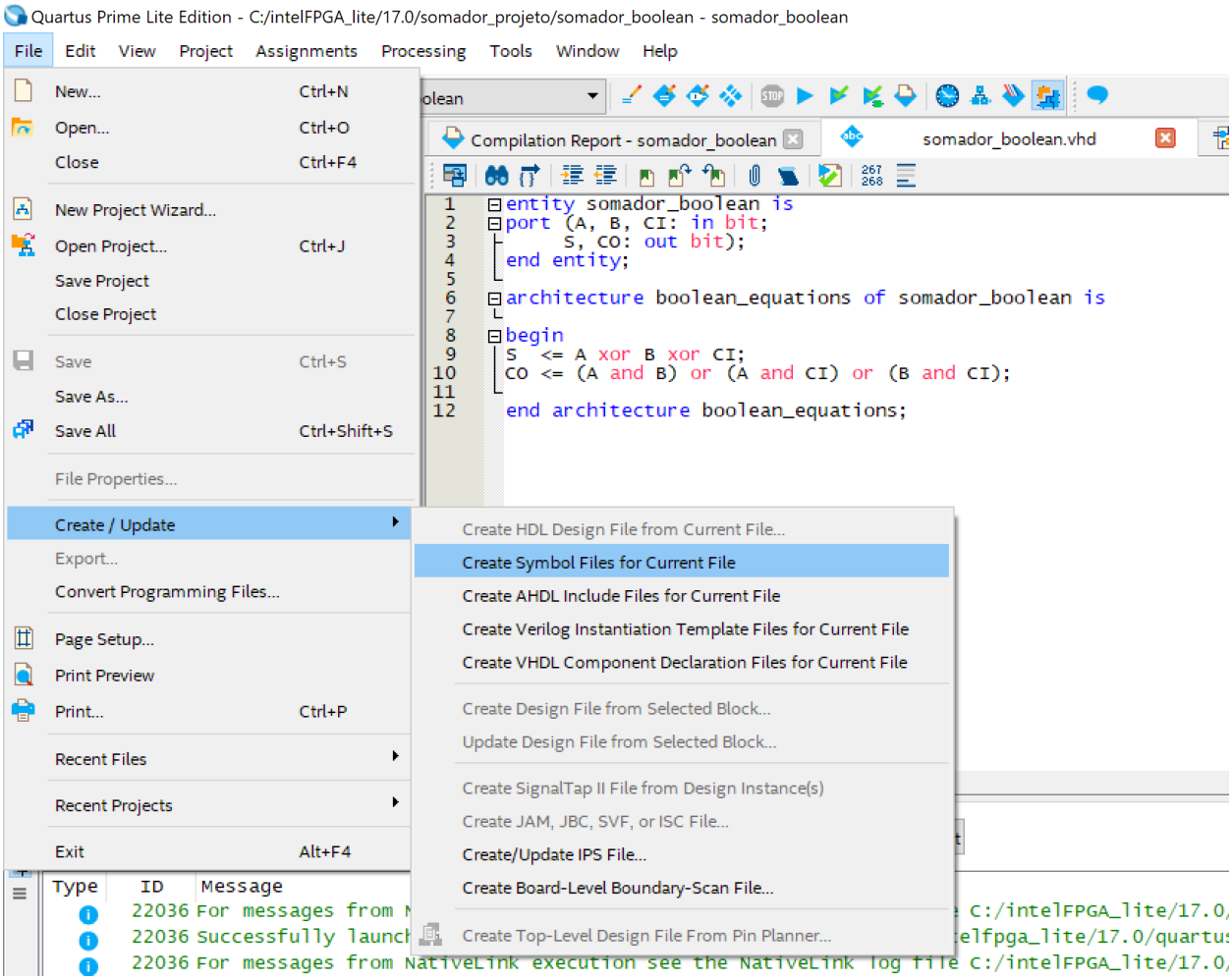

Start Simulation
Design
VHDL
Verilog
Libraries
SDF
Others
Name
Type
Path
work
Library
C:/altera/13.1/work
somador_boolean
Entity
somador
Entity
multiplicador
Entity
ff_shift
Entity
and_dual
Entity
acc_shift
Entity
vital2000
Library
$MODEL_TECH/../vital2000
verilog
Library
$MODEL_TECH/../verilog
twentynm_ver (empty)
Library
$MODEL_TECH/../altera/verilog
Design Unit(s)
work.somador_boolean
Resolution
default
Optimization
Enable optimization
Optimization Options...
OK
Cancel


Wave - Default
Msgs
/somador_boolean/A
/somador_boolean/B
Object Declaration
Add
Edit
View
Radix
Format
Combine Signals...
Group...
Ungroup
Force...
NoForce
Clock...
Properties...
Now
0.0005 us
Cursor 1
0.000379 us
Objects
Name
A
B
CI
S
CO
Processes (Act
Transcript
VSIM 12> restart
force -freeze sim:/somador_boolean/A 1 0, 0 {50 ps} -r 100
force -freeze sim:/somador_boolean/B 1 0, 0 {100 ps} -r 200
force -freeze sim:/somador_boolean/CI 1 0, 0 {200 ps} -r 400
VSIM 16> run
run
run
run
VSIM 17> run
Load canceled
Load canceled
VSIM 17>

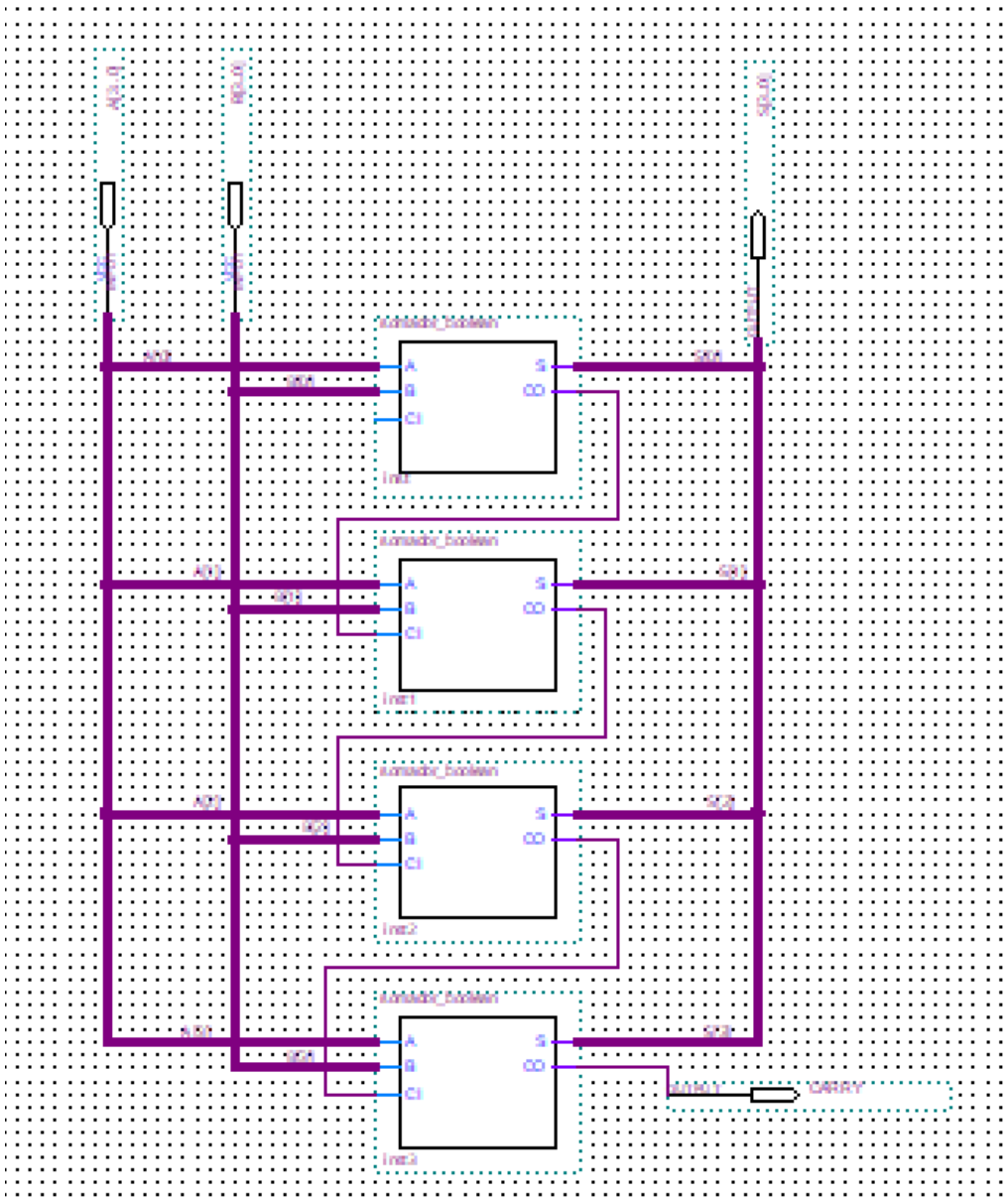
4 Importando o somador de 1 bit para o Quartus

Para esta parte foi utilizada a ferramenta Quartus da Altera, versão 17.0.

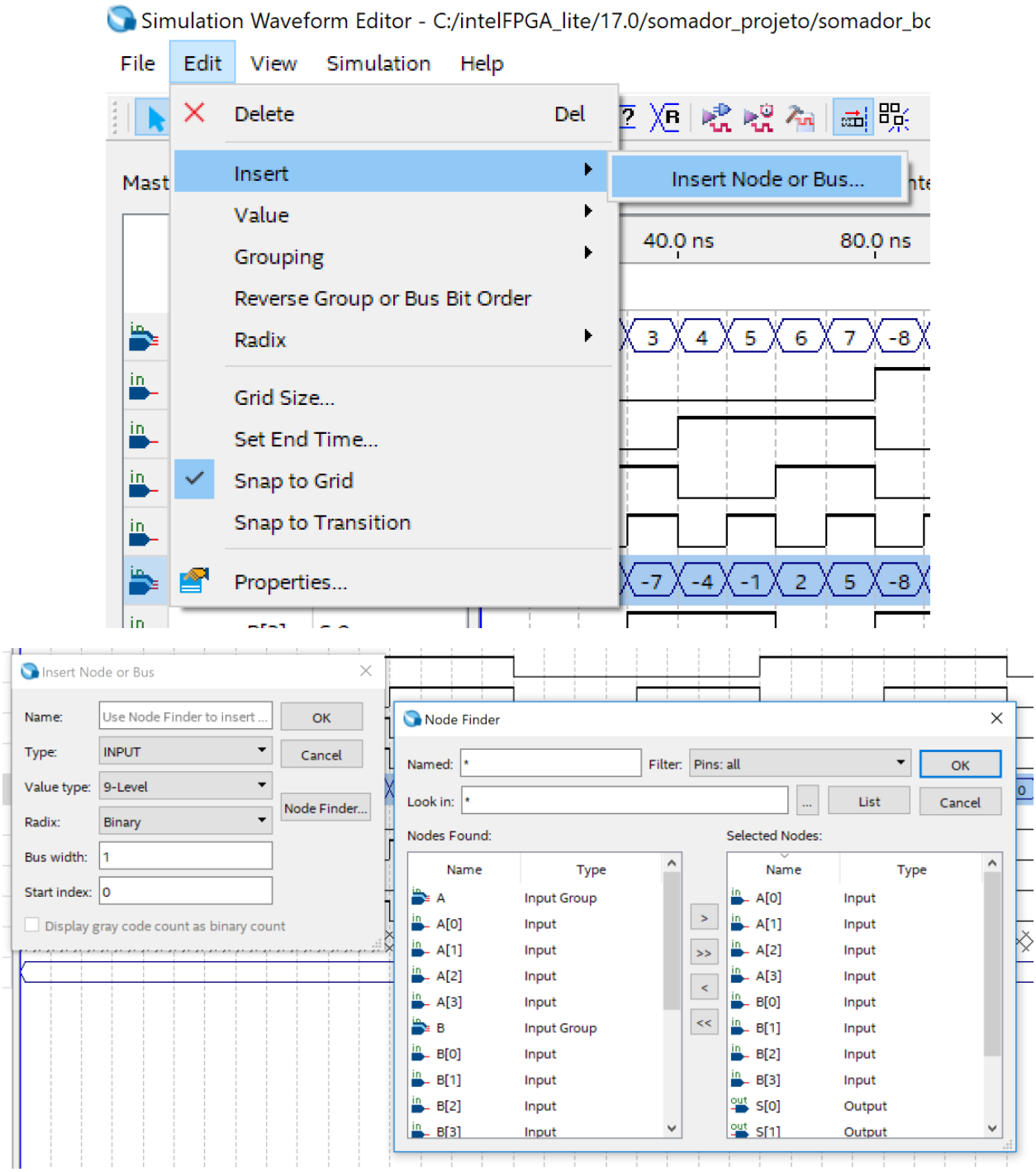
1. Começar usando a ferramenta **New Project Wizard** conforme e adicionar o arquivo VHDL que iremos utilizar, no caso “somador_boolean”.
2. Abrir o arquivo VHDL e selecionar **File → Create/Update → Create Symbol Files for Current File**



3. Criar um arquivo **File → New→ Block diagram/Schematic File**
4. Fazer o desenho do somador *ripple carry* com as células criadas anteriormente. Vetores são nomeados com o padrão “A[3..0]” e cada linha que representa um bit deve ser nomeada com “A[3]”. Usar “*Bus tool*” para desenhar linhas que levam vetores.



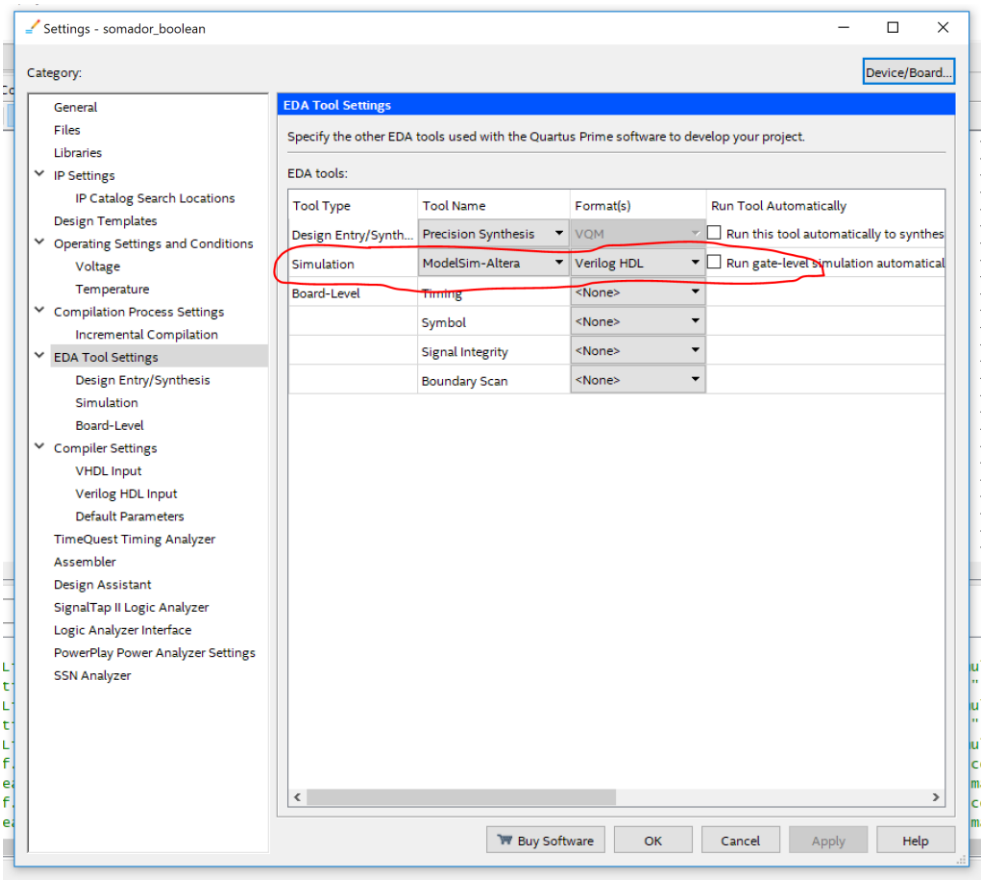
1. Clicar com o botão direto sobre o arquivo “.bdf” esquemático de mudar para entidade de top-level. Salvar o arquivo esquemático com um nome diferente da entidade somador de 1 bit. Compilar o projeto novamente
2. Adicionar um arquivo do tipo **University Program VWF** em **File → New**
3. Adicionar os pinos de interesse em **Edit → Insert → Insert Node or Bus** e usando o **Node Finder**



4. Alterar os valores dos pinos e rodar **Simulation → Run Functional Simulation** ou **Simulation → Run Timing Simulation** para simulação real do circuito

Observação: talvez seja necessário configurar manualmente a ferramenta de simulação

1. Selecionar simulação “Modelsim-Altera” em **Assignments → Settings → EDA Tool Setting → Simulation**



2. Adicionar o caminho da instalação do ModelSim em **Tool → Options → EDA Tool Options** para algo parecido com **C:\intelFPGA\17.0\modelsim_ase\win32aloem** (buscar no seu próprio computador onde a pasta win32aloem está)

5 Projeto de ULA de 8 bits usando tipo *Signed*

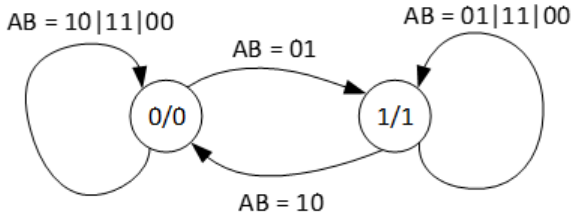
Um exemplo de projeto usando tipos pronto da biblioteca IEEE que suporta operações aritméticas com vetores de bit.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity ula8bits is
port(A, B: in signed(7 downto 0);
     sel: in std_logic;
     S: out signed (7 downto 0));
end entity ula8bits;

architecture comportamento of ula8bits is

begin

sequencial: process(A, B, sel)
begin
    if sel = '1' then
        S <= A + B;
    elsif sel = '0' then
        S <= A - B;
    else
        S <= "00000000";
    end if;
end process sequencial;

end architecture comportamento;
```

6 Descrevendo controlador síncrono em VHDL

6.1 Descrevendo a partir das equações de próximo estado

Considere o seguinte grafo de transição de estados resultando em um Flip-flop D um pouco modificado



Q \ AB	00	01	11	10
0	0	1	0	0
1	1	1	1	0

$$Q_{proximo} = \bar{A}B + \bar{A}Q + BQ$$
$$Saida = Q$$

Para descrever o mesmo em VHDL será necessário utilizar de um componente FFD. Já existem estes componentes prontos na biblioteca da Altera e agora aprenderemos a invocá-lo.

Usaremos as seguintes bibliotecas

```
library ieee;
use ieee.std_logic_1164.all;

library altera;
use altera.altera_primitives_components.all;
```

A seguir está descrição de entidade. Note que a saída é igual a Q, portanto suprimimos o sinal S e usamos somente Q.

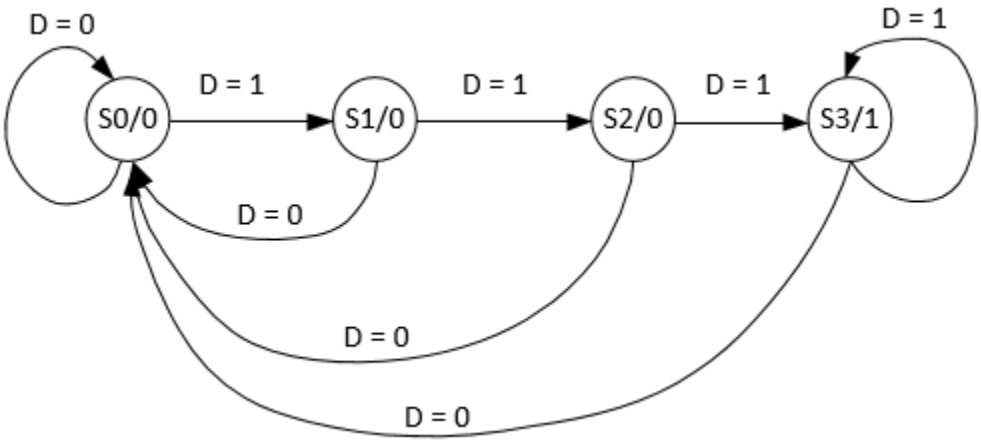
```
entity control_ffd is
port (A, B, clk, rst: in std_logic;
      Q: buffer std_logic);
end entity control_ffd;
```

Por fim a arquitetura será a seguinte

```
architecture equacoes of control_ffd is
signal q_prox : std_logic;

-- Declaração do componente FFD descrito na biblioteca de
componentes
component dff
port(d, clk, clrn, prn :  in  std_logic;
     q                 :  out std_logic);
end component;

begin

q_prox <= ((not A) and B) or ((not A) and Q) or (B and Q);

-- Invocando o FFD como um componente chamado "d1"
d1: dff port map (d => q_prox,
                  clk => clk,
                  clrn => rst,
                  prn => '1',
                  q => Q);

end architecture equacoes;
```

6.2 Descrevendo a partir das transições – Modelo Moore

A seguinte máquina trata de contar quantos “1´s” ocorrem a cada subida de sinal de relógio e mudar a saída para 1 quando três “1´s” ou mais seguidos acontecerem.



As entidades e bibliotecas usadas serão como segue

```
library ieee;
use ieee.std_logic_1164.all;

entity contador_moore is
port (D, clk, rst: in std_logic;
      S: out std_logic);
end entity contador_moore;
```

Será necessário alguns sinais internos conforme visto aqui

```
architecture maquina of contador_moore is

type estado is (S0, S1, S2, S3);
signal est: estado;
signal prox_est: estado;
```

E por fim, a descrição do grafo conterá 3 processos, a saber: cálculo do próximo estado, registro de estado e processamento da saída.

Cálculo do próximo estado

```
atualizacao_estado: process (est, D)
begin
    case est is
        when S0 => if (D = '1') then prox_est <= S1;
                   else prox_est <= S0;
                   end if;
        when S1 => if (D = '1') then prox_est <= S2;
                   else prox_est <= S0;
                   end if;
        when S2 => if (D = '1') then prox_est <= S3;
                   else prox_est <= S0;
                   end if;
        when S3 => if (D = '1') then prox_est <= S3;
                   else prox_est <= S0;
                   end if;
    end case;
end process atualizacao_estado;
```

Registro de próximo estado

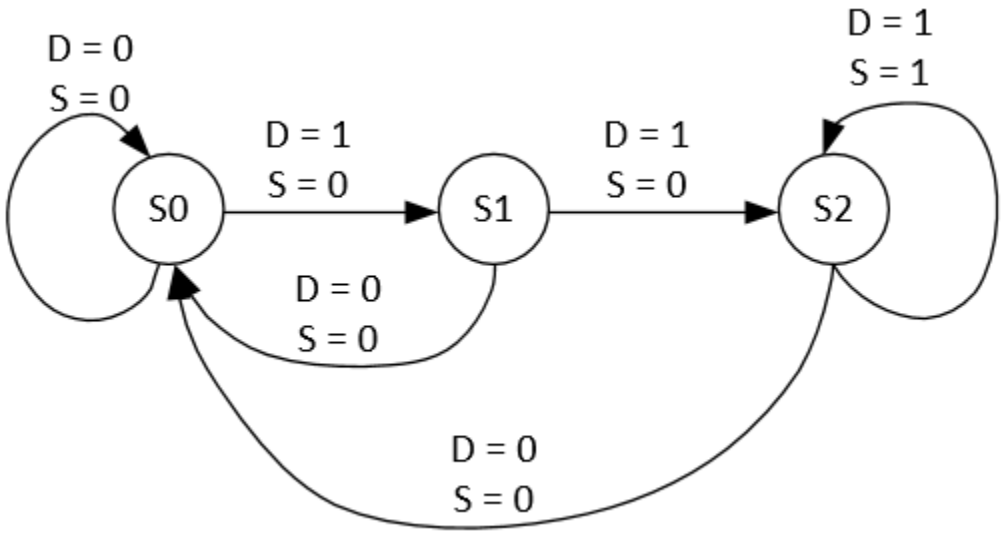
```
registro_estado: process (clk, rst)
begin
    if (rst = '0') then
        est <= S0;
    elsif (rising_edge(clk)) then
        est <= prox_est;
    end if;
end process registro_estado;
```

Processamento da saída

```
processamento_saida: process (est)
begin
    case est is
        when S0 => S <= '0';
        when S1 => S <= '0';
        when S2 => S <= '0';
        when S3 => S <= '1';
    end case;
end process processamento_saida;
```

6.3 Descrevendo a partir das transições – Modelo Mealey

Um problema de contar um´s descrito pelo modelo Mealey pode ser visto a seguir. Note que como a entrada atua diretamente na saída neste modelo, a quantidade de tempo que D =1 está sendo contada muda sensivelmente (pode ser visto através da simulação).



Também para esse modelo, o processamento de saída é incorporado na atualização de estado num mesmo processo. O código final é como se segue.

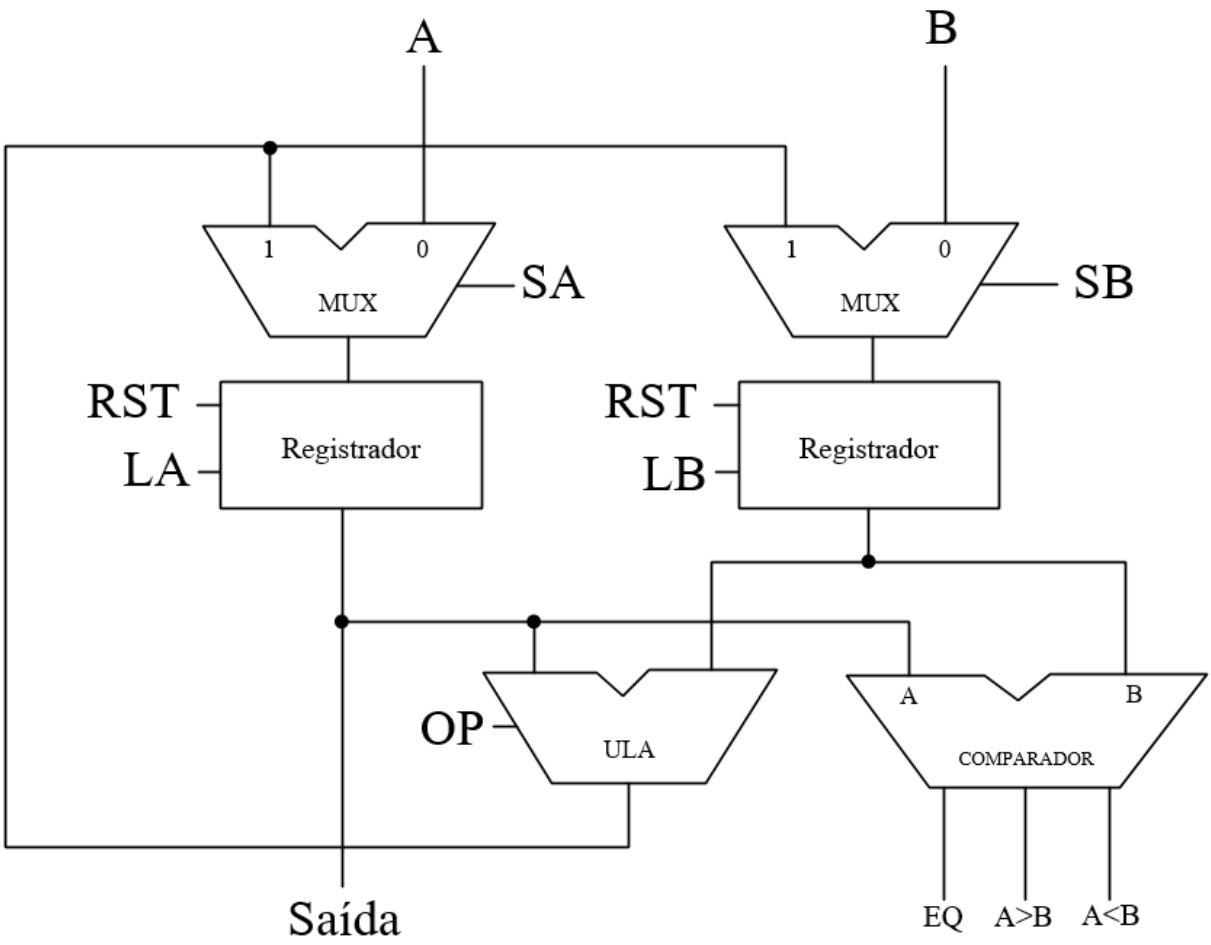
```
architecture maquina of contador_mealey is

type estado is (S0, S1, S2);
signal est: estado;
signal prox_est: estado;

begin

atualizacao_estado: process (est, D)
begin
    case est is
        when S0 => if (D = '1') then prox_est <= S1; S <= '0';
                   else             prox_est <= S0; S <= '0';
                   end if;
        when S1 => if (D = '1') then prox_est <= S2; S <= '0';
                   else             prox_est <= S0; S <= '0';
                   end if;
        when S2 => if (D = '1') then prox_est <= S2; S <= '1';
                   else             prox_est <= S0; S <= '0';
                   end if;
    end case;
end process atualizacao_estado;

registro_estado: process (clk, rst)
begin
    if (rst = '0') then
        est <= S0;
    elsif (rising_edge(clk)) then
        est <= prox_est;
    end if;
end process registro_estado;

end architecture maquina;
```

7 Descrevendo e sintetizando um algoritmo

Para o estudo de caso desta seção, usaremos o algoritmo de MDC (Máximo Divisor Comum) e a arquitetura de implementação será de uma MEF síncrona Moore controlando um *Datapath*. O pseudocódigo que descreve o algoritmo é como se segue

```
1    Inicializar;
2    Carregar RegA e RegB;
3
4    Começar loop
5        Se A>B então A <= A - B;   continuar loop;
6        Se A<B então B <= B - A;   continuar loop;
7        Se A=B então Solução é A;  terminar loop;
8    Fim loop
```

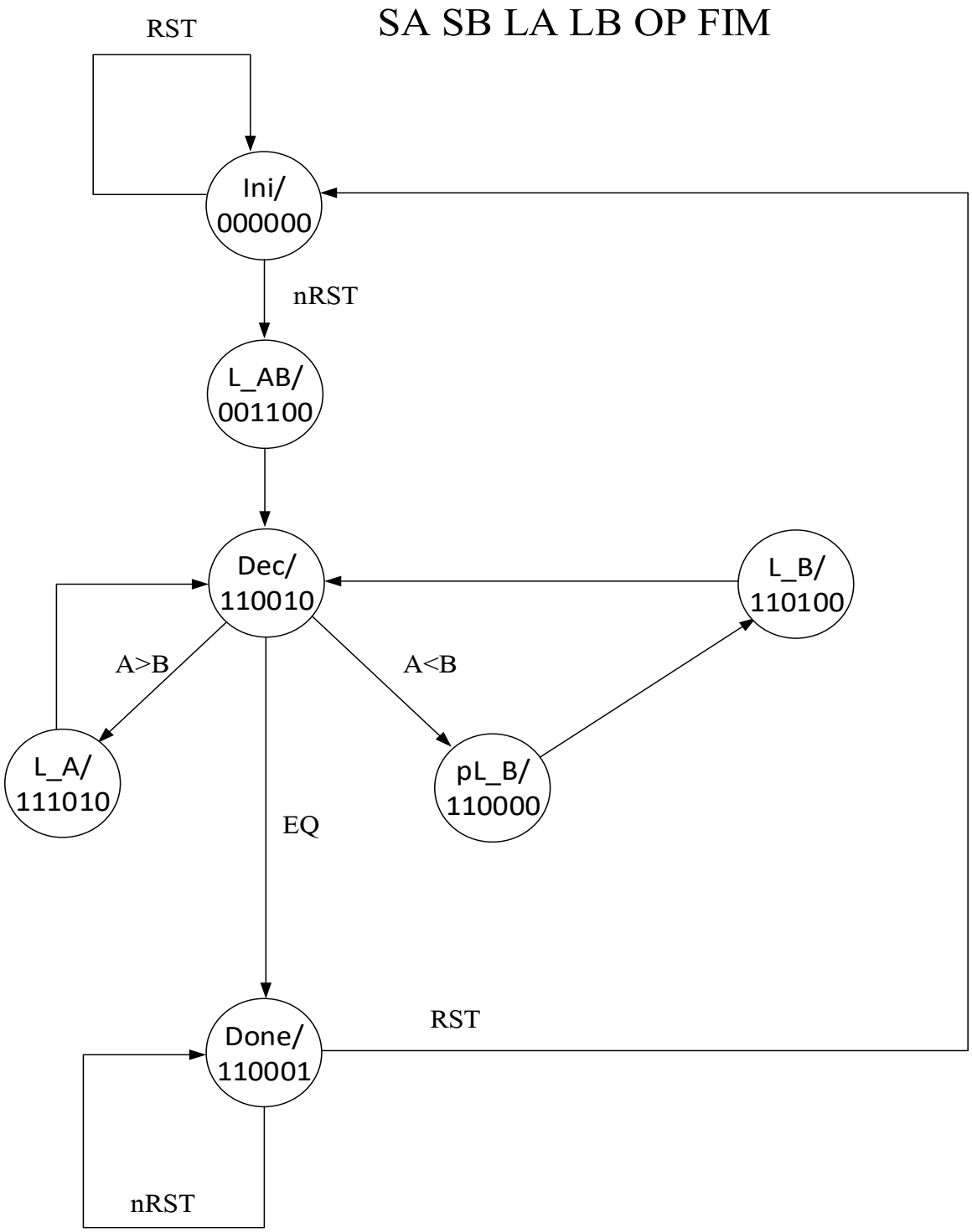
Para criar o caminho de dados (*Datapath*) deste algoritmo, usaremos de uma ULA, dois MUXs, dois registradores e um comparador conforme o esquema abaixo



Os sinais SA, SB, LA, LB e OP são entradas para o *Datapath*, os sinais EQ, A>B, A<B são saídas do *Datapath* para o controlador, o sinal RST é comum ao controlador e datapath. A ULA realiza as operações

(Operando1 – Operando2) e (Operando2 – Operando1) dependendo do sinal de OP. Os outros sinais são auto explicativos.

Um possível controlador para o *Datapath* descrito é conforme se segue



A descrição da arquiteutura do *datapth* pode ser feita como se segue

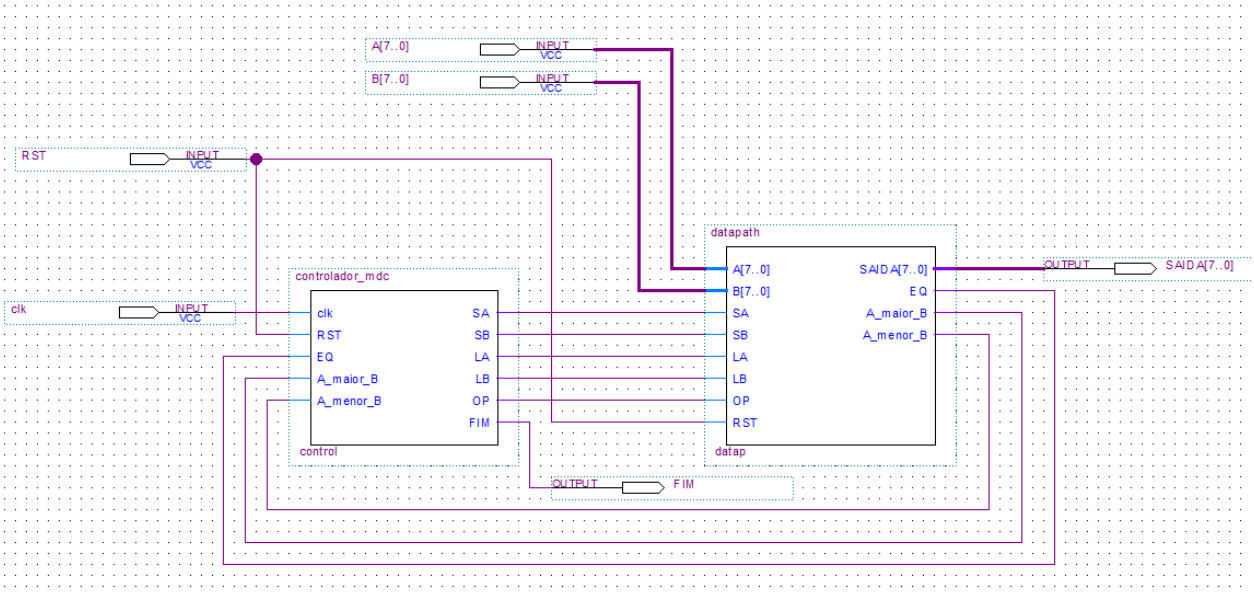
```
library ieee;
use ieee.numeric_bit.all;

entity datapath is
port (A, B: in unsigned(7 downto 0) := "00000000";
      SAIDA: out unsigned(7 downto 0) := "00000000";
      SA, SB, LA, LB, OP, RST: in bit := '0';
      EQ, A_maior_B, A_menor_B: out bit := '0');
end datapath;

architecture comportamento of datapath is
signal auxA: unsigned (7 downto 0) := "00000000";
signal auxB: unsigned (7 downto 0) := "00000000";
signal out_regA: unsigned (7 downto 0);
signal out_regB: unsigned (7 downto 0);
signal out_ula: unsigned (7 downto 0) := "00000000";
begin
-- Multiplexer A
auxA <= A when SA = '0' else out_ula;
-- Multiplexer B
auxB <= B when SA = '0' else out_ula;
-- Registrador A
regA: process (LA)
begin
    if (RST = '1') then
        out_regA <= "00000000";
    elsif (rising_edge(LA)) then
         out_regA <= auxA;
    else out_regA <= out_regA;
    end if;
end process regA;
-- Registrador B
regB: process (LB)
begin
    if (RST = '1') then
        out_regB <= "00000000";
    elsif (rising_edge(LB)) then
         out_regB <= auxB;
    else out_regB <= out_regB;
    end if;
end process regB;
-- ULA
out_ula <= (out_regA - out_regB) when OP = '1' else (out_regB - out_regA);
-- Comparador
EQ        <= '1' when (out_regA = out_regB) else '0';
A_maior_B <= '1' when (out_regA > out_regB) else '0';
A_menor_B <= '1' when (out_regA < out_regB) else '0';
-- Saida do sistema
SAIDA <= out_regA;
end architecture comportamento;
```

A descrição do controlador pode ser feita como se segue

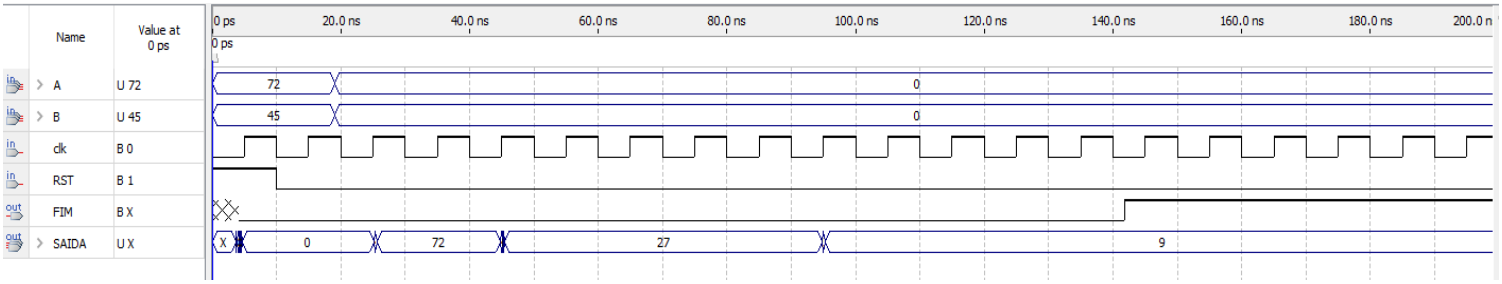
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_bit.all;

entity controlador_mdc is
port (clk, RST, EQ, A_maior_B, A_menor_B: in bit;
      SA, SB, LA, LB, OP, FIM: out bit := '0');
end controlador_mdc;

architecture mef of controlador_mdc is
type estado is (INI, L_AB, DEC, L_A, pL_B, L_B, DONE);
signal est: estado := INI;
signal prox_est: estado := INI;
begin

mudanca_estado: process (clk, RST)
begin
    if (rising_edge(clk)) then
        if (RST = '1') then est <= INI;
        else est <= prox_est;
        end if;
    end if;
end process mudanca_estado;

atualizacao_estado: process (est, RST, EQ, A_maior_B, A_menor_B)
begin
    case est is
        when INI    =>  if (RST = '0') then     prox_est <= L_AB;
                        else                    prox_est <= INI;
                        end if;
        when L_AB   =>  prox_est <= DEC;
        when DEC    =>  if (EQ = '1') then              prox_est <= DONE;
                        elsif (A_maior_B = '1') then    prox_est <= L_A;
                        elsif (A_menor_B = '1') then    prox_est <= pL_B;
                        else                            prox_est <= DEC;
                        end if;
        when L_A    =>  prox_est <= DEC;
        when pL_B   =>  prox_est <= L_B;
        when L_B    =>  prox_est <= DEC;
        when DONE   =>  if (RST = '1') then     prox_est <= INI;
                        else                    prox_est <= DONE;
                        end if;
    end case;
end process atualizacao_estado;

processamento_saida: process (est)
begin
    case est is
        when INI    =>  SA   <= '0';
                        SB   <= '0';
                        LA   <= '0';
                        LB   <= '0';
                        OP   <= '1';
                        FIM  <= '0';
```

```
        when L_AB    =>  SA   <= '0';
                         SB   <= '0';
                         LA   <= '1';
                         LB   <= '1';
                         OP   <= '1';
                         FIM  <= '0';

        when DEC     =>  SA   <= '1';
                         SB   <= '1';
                         LA   <= '0';
                         LB   <= '0';
                         OP   <= '1';
                         FIM  <= '0';

        when L_A     =>  SA   <= '1';
                         SB   <= '1';
                         LA   <= '1';
                         LB   <= '0';
                         OP   <= '1';
                         FIM  <= '0';

        when pL_B    =>  SA   <= '1';
                         SB   <= '1';
                         LA   <= '0';
                         LB   <= '0';
                         OP   <= '0';
                         FIM  <= '0';

        when L_B     =>  SA   <= '1';
                         SB   <= '1';
                         LA   <= '0';
                         LB   <= '1';
                         OP   <= '0';
                         FIM  <= '0';

        when DONE    =>  SA   <= '1';
                         SB   <= '1';
                         LA   <= '0';
                         LB   <= '0';
                         OP   <= '0';
                         FIM  <= '1';

    end case;
end process processamento_saida;

end architecture mef;
```

Ambos VHDLs podem ser exportados para o Quartus como um *Symbol File* e podem ser ligados da seguinte forma.



Por fim, a simulação real do circuito retorna o seguinte comportamento



Note que a velocidade de relógio deve estar lenta o suficiente para acomodar o tempo de atraso decorrente dos cálculos numéricos do *datapath.*

8 Lista de comandos essenciais em VHDL

8.1 Para código concorrente

- When… else
- With… select
- …. <= …. After

8.2 Para código sequencial (Process)

- If… elsif… else… end if
- Case… is when…

8.3 Operadores

- AND, OR, NAND, NOR, NOT, XOR, XNOR
- "<=" assinalamento de sinal
- ":=" assinalamento de variáveis (dentro de Process)
- "&" concatenação
- "/=" operador diferença
- "=" operador igualdade