Atividade 6: Detector de Valor Máximo e Mínimo

Maria Eduarda Soares Romana & Tiago Fernandes Soucek

9 de junho de 2025

1 Resumo

O presente relatório descreve a atividade 6 da disciplina de Lógica Reconfigurável, a qual teve como objetivo implementar um **detector genérico de valor mínimo e máximo** utilizando VHDL, com foco em **tipos e larguras de dados parametrizáveis**, por meio de estruturas como *generic*, *package*, *type* e *procedure/function*. O projeto compara duas entradas de mesmo tamanho, armazenadas em um vetor do tipo customizado *data_ array*, e utiliza funções *find_ min* e *find_ max* para determinar o resultado da comparação. A entrada e o resultado são exibidos em **displays de sete segmentos**, com controle por botões físicos presentes na FPGA. A tarefa foi simulada e sintetizada utilizando o software *Quartus*.

2 Requisitos do Projeto

Os seguintes itens foram requisitos do projeto:

- Criar um detector de valor mínimo e máximo entre valores *unsigned*.
- Utilizar *generic* para permitir quantidade genérica de bits e entradas, e *package* com tipo *(type)* e funções para cálculo de mínimo e máximo.
- Criar um tipo definido pelo usuário em um *package* separado.

3 Métodos e Técnicas

A implementação foi realizada no software Quartus Prime, e os Listings 5 e 1 apresentam os códigos VHDL desenvolvidos para resolver o problema proposto.

3.1 Pacote reutilizável: min_max_pkg.vhd

```
1  -- ----------------------------------------
2  -- ATV 06 - MINIMO E MAXIMO
3  -- MARIA EDUARDA SOARES ROMANA SILVA
4  -- TIAGO FERNANDES SOUCEK
5  -- ----------------------------------------
6
7
8  library ieee;
9  use ieee.std_logic_1164.all;
10 use ieee.numeric_std.all;
11
12 package min_max_pkg is
13
```

```
14      constant DATA_WIDTH : integer := 4; -- tamanho fixo do std_logic_vector
15
16      -- Tipo de vetor de vetores com largura definida
17      type data_array is array (natural range <>) of std_logic_vector(
            DATA_WIDTH-1 downto 0);
18
19      function find_min(data : data_array) return std_logic_vector;
20      function find_max(data : data_array) return std_logic_vector;
21
22  end package min_max_pkg;
23
24  package body min_max_pkg is
25
26      function find_min(data : data_array) return std_logic_vector is
27          variable result : std_logic_vector(data(data'low)'range) := data(
                data'low);
28      begin
29          for i in data'low + 1 to data'high loop
30              if unsigned(data(i)) < unsigned(result) then
31                  result := data(i);
32              end if;
33          end loop;
34          return result;
35      end function;
36
37      function find_max(data : data_array) return std_logic_vector is
38          variable result : std_logic_vector(data(data'low)'range) := data(
                data'low);
39      begin
40          for i in data'low + 1 to data'high loop
41              if unsigned(data(i)) > unsigned(result) then
42                  result := data(i);
43              end if;
44          end loop;
45          return result;
46      end function;
47
48  end package body min_max_pkg;
```

Listing 1: Código VHDL: ATV 06 - Mínimo e Máximo

Esse código (Listing 1) define um **package** reutilizável chamado *min_ max_ pkg*, com foco em generalidade e modularidade.

3.1.1 Constante de configuração

```
1  constant DATA_WIDTH : integer := 4;
```

Listing 2: Código VHDL: Constante de configuração

Define o número de bits dos dados que serão manipulados.

3.1.2 Tipo definido pelo usuário

```
1  type data_array is array (natural range <>) of std_logic_vector(DATA_WIDTH-1
       downto 0);
```

Listing 3: Código VHDL: Tipo definido pelo usuário

Cria um vetor genérico *(data_ array)* contendo elementos do tipo *std_ logic_ vector*, onde o número de elementos é definido externamente, exemplo: 2 entradas, 4 entradas, etc.

3.1.3 Funções find_min e find_max

```
if unsigned(data(i)) > unsigned(result) then
    result := data(i);
end if;
```

Listing 4: Código VHDL: Funções $\text{find}_m ine find_m ax$

Ambas as funções *find_ min* e *find_ max*, percorrem o vetor *data_ array* comparando os valores convertidos para *unsigned* e retornam o menor ou maior elemento. Essas funções encapsulam a lógica de comparações, evitando duplicação de código na entidade principal, o que torna o projeto mais "limpo"e organizado.

3.2 Código principal: atv06.vhd

```
-- ----------------------------------------
-- ATV 06 - MINIMO E MAXIMO
-- MARIA EDUARDA SOARES ROMANA SILVA
-- TIAGO FERNANDES SOUCEK
-- ----------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.min_max_pkg.all; -- importa o pacote com find_max e find_min

entity atv06 is
    generic (
        NUM_BITS   : integer := 4; -- quantidade de bits por numero
        NUM_INPUTS : integer := 2 -- quantidade de entradas para comparacao
    );
    port (
        CLOCK_50 : in  std_logic;
        KEY      : in  std_logic_vector(1 downto 0);
        SW       : in  std_logic_vector(NUM_BITS-1 downto 0);

        HEX0     : out std_logic_vector(6 downto 0);
        HEX1     : out std_logic_vector(6 downto 0);
        HEX2     : out std_logic_vector(6 downto 0);
        HEX3     : out std_logic_vector(6 downto 0);
        HEX4     : out std_logic_vector(6 downto 0);
        HEX5     : out std_logic_vector(6 downto 0)
    );
end entity atv06;

architecture rtl of atv06 is

    -- Sinais internos
    signal num1_salvo : std_logic_vector(NUM_BITS-1 downto 0) := (others =>
        '0');
    signal max_val    : std_logic_vector(NUM_BITS-1 downto 0) := (others =>
        '0');

```

```
37      signal key_pressed : std_logic_vector(1 downto 0);
38      signal key_reg1    : std_logic_vector(1 downto 0) := "11";
39      signal key_reg2    : std_logic_vector(1 downto 0) := "11";
40      signal key_reg3    : std_logic_vector(1 downto 0) := "11";
41
42      -- Vetor de dados generico com 2 entradas
43      signal entradas : data_array(0 to NUM_INPUTS-1);
44
45      -- Conversor para display
46      function hex_to_7seg(hex_digit : std_logic_vector(3 downto 0)) return
           std_logic_vector is
47      begin
48          case hex_digit is
49              when "0000" => return "1000000"; when "0001" => return "1111001"
                   ;
50              when "0010" => return "0100100"; when "0011" => return "0110000"
                   ;
51              when "0100" => return "0011001"; when "0101" => return "0010010"
                   ;
52              when "0110" => return "0000010"; when "0111" => return "1111000"
                   ;
53              when "1000" => return "0000000"; when "1001" => return "0010000"
                   ;
54              when "1010" => return "0001000"; when "1011" => return "0000011"
                   ;
55              when "1100" => return "1000110"; when "1101" => return "0100001"
                   ;
56              when "1110" => return "0000110"; when "1111" => return "0001110"
                   ;
57              when others => return "1111111";
58          end case;
59      end function;
60
61  begin
62
63      -- Debounce dos botoes
64      process(CLOCK_50)
65      begin
66          if rising_edge(CLOCK_50) then
67              key_reg1 <= not KEY;
68              key_reg2 <= key_reg1;
69              key_reg3 <= key_reg2;
70              key_pressed <= key_reg2 and (not key_reg3);
71          end if;
72      end process;
73
74      -- Logica principal
75      process(CLOCK_50)
76      begin
77          if rising_edge(CLOCK_50) then
78              if key_pressed(1) = '1' then
79                  num1_salvo <= SW;
80              end if;
81
82              if key_pressed(0) = '1' then
83                  entradas(0) <= num1_salvo;
84                  entradas(1) <= SW;
85                  max_val <= find_max(entradas);
```

```
86              end if;
87          end if;
88      end process;
89
90      -- Displays
91      HEX0 <= hex_to_7seg(SW); -- entrada ao vivo
92      HEX5 <= hex_to_7seg(max_val); -- resultado
93
94      -- Apaga os displays nao usados
95      HEX1 <= "1111111";
96      HEX2 <= "1111111";
97      HEX3 <= "1111111";
98      HEX4 <= "1111111";
99
100 end architecture rtl;
```

Listing 5: Código VHDL: ATV 06

Esta é a parte principal do projeto (Listing 5), responsável por ler os dados das chaves (SW), detectar os comandos via botões (KEY), armazenar os valores, comparar e exibir os resultados.

3.2.1 Generic

```
1 generic (
2     NUM_BITS   : integer := 4;
3     NUM_INPUTS : integer := 2
4 );
```

Listing 6: Código VHDL: generic

Define dois parâmetros configuráveis:

- **NUM_BITS:** número de bits de cada valor
- **NUM_INPUTS:** número de entradas no vetor de dados.

3.2.2 Importação de pacotes

```
1 use work.min_max_pkg.all;
```

Listing 7: Código VHDL: Importação de pacotes

Torna acessível o tipo *data_array* e as funções *find_min* e *find_max*.

3.2.3 Vetores interno

```
1 signal num1_salvo : std_logic_vector(NUM_BITS-1 downto 0);
2 signal entradas : data_array(0 to NUM_INPUTS-1);
```

Listing 8: Código VHDL: Vetores internos

- **num1_salvo:** armazena o primeiro valor inserido.
- **entradas:** Vetor usado como entrada para a função *find_max*.

3.2.4 Lógica principal

```
if key_pressed(1) = '1' then
    num1_salvo <= SW;
end if;

if key_pressed(0) = '1' then
    entradas(0) <= num1_salvo;
    entradas(1) <= SW;
    max_val <= find_max(entradas);
end if;
```

Listing 9: Código VHDL: Lógica principal

- Ao pressionar **KEY(1)**, o valor atual das chaves é salvo.
- Ao pressionar **KEY(0)**, o valor salvo e o valor atual são inseridos no vetor *entradas*, e o maior valore entre eles é obtido usando a função *find_max*.

3.2.5 Exibição nos displays

```
HEX0 <= hex_to_7seg(SW);
HEX5 <= hex_to_7seg(max_val);
```

Listing 10: Código VHDL: Exibição em displays

- **HEX0** mostra o valor atual das chaves, em tempo real.
- **HEX5** mostra o maior valor.

A função *hex_to_7seg* faz a conversão de 4 bits para o padrão dos displays de sete segmentos, permitindo a exibição numérica direta.

4 Resultados e Considerações

A implementação foi concluída com sucesso. O comparador exibe corretamente o valor atual e o maior valor entre dois inseridos sequencialmente. A RTL Figura 1 confirma o fluxo de dados e uso das funções do *package*.

Apesar de a atividade funcionar com apenas duas entradas ao invés de um conjunto maior (ex: 4 ou 5), como foi pedido na descrição da tarefa, a lógica foi corretamente generalizada com o uso de *generic*, *type*, *package* e funções, e atende todos os requisitos centrais da proposta.

4.0.1 RTL do circuito

A Figura 1 apresenta o diagrama RTL gerado automaticamente pelo software *Quartus* após a compilação do projeto. É possível observar a separação clara entre os blocos de controle, registradores intermédiarios e blocos de exibição. As funções de comparação do pacote integradas corretamente à arquitetura principal do projeto.

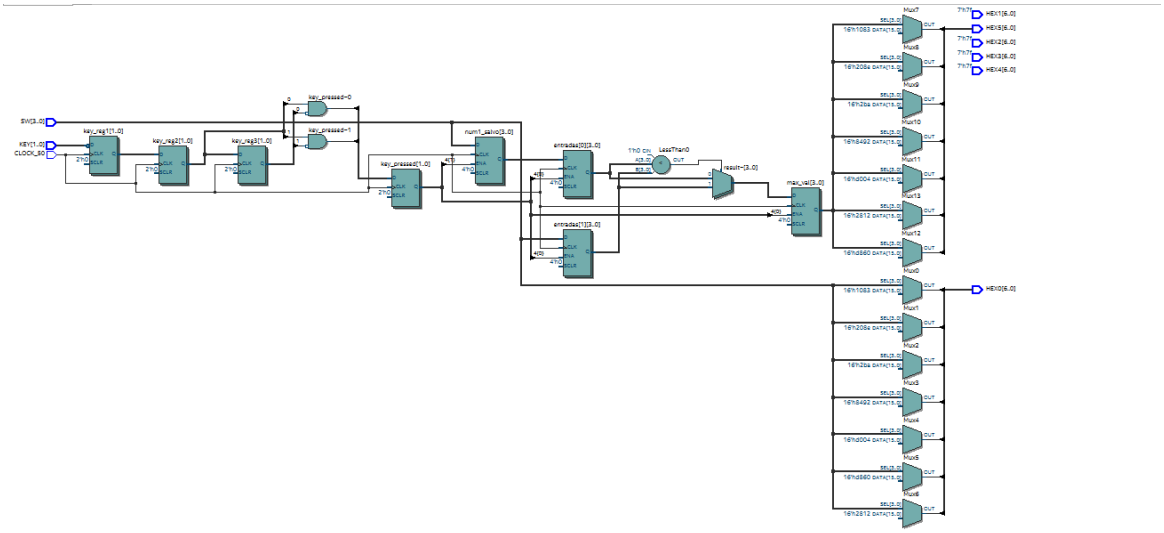

Figura 1: RTL do circuito

4.0.2 Pin Planner

As Figuras 2 e 3 apresentam o mapeamento dos pinos da FPGA para os sinais utilizados no projeto.

CLOCK_50	Input	PIN_P11	3	B3_N0	PIN_P11	2.5 V		12mA (default)			
HEX0[6]	Output	PIN_C17	7	B7_N0	PIN_C17	2.5 V		12mA (default)	2 (default)		
HEX0[5]	Output	PIN_D17	7	B7_N0	PIN_D17	2.5 V		12mA (default)	2 (default)		
HEX0[4]	Output	PIN_E16	7	B7_N0	PIN_E16	2.5 V		12mA (default)	2 (default)		
HEX0[3]	Output	PIN_C16	7	B7_N0	PIN_C16	2.5 V		12mA (default)	2 (default)		
HEX0[2]	Output	PIN_C15	7	B7_N0	PIN_C15	2.5 V		12mA (default)	2 (default)		
HEX0[1]	Output	PIN_E15	7	B7_N0	PIN_E15	2.5 V		12mA (default)	2 (default)		
HEX0[0]	Output	PIN_C14	7	B7_N0	PIN_C14	2.5 V		12mA (default)	2 (default)		

Figura 2: Pin Planner - clock e HEX0

HEX5[6]	Output	PIN_B22	6	B6_N0	PIN_B22	2.5 V		12mA (default)	2 (default)		
HEX5[5]	Output	PIN_C22	6	B6_N0	PIN_C22	2.5 V		12mA (default)	2 (default)		
HEX5[4]	Output	PIN_B21	6	B6_N0	PIN_B21	2.5 V		12mA (default)	2 (default)		
HEX5[3]	Output	PIN_A21	6	B6_N0	PIN_A21	2.5 V		12mA (default)	2 (default)		
HEX5[2]	Output	PIN_B19	7	B7_N0	PIN_B19	2.5 V		12mA (default)	2 (default)		
HEX5[1]	Output	PIN_A20	7	B7_N0	PIN_A20	2.5 V		12mA (default)	2 (default)		
HEX5[0]	Output	PIN_B20	6	B6_N0	PIN_B20	2.5 V		12mA (default)	2 (default)		
KEY[1]	Input	PIN_A7	7	B7_N0	PIN_A7	2.5 V		12mA (default)			
KEY[0]	Input	PIN_B8	7	B7_N0	PIN_B8	2.5 V		12mA (default)			
SW[3]	Input	PIN_C12	7	B7_N0	PIN_C12	2.5 V		12mA (default)			
SW[2]	Input	PIN_D12	7	B7_N0	PIN_D12	2.5 V		12mA (default)			
SW[1]	Input	PIN_C11	7	B7_N0	PIN_C11	2.5 V		12mA (default)			
SW[0]	Input	PIN_C10	7	B7_N0	PIN_C10	2.5 V		12mA (default)			

Figura 3: Pin Planner - HEX5, KEYs e CHAVES

4.0.3 Funcionamento da placa

Neste `LINK` (clique aqui) há o vídeo demonstrando o funcionamento do projeto na placa.