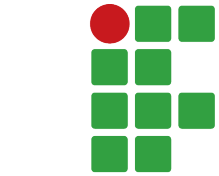

INSTITUTO FEDERAL
Santa Catarina

Câmpus
São José

Programação do kit Mercurio IV

Dispositivos Lógicos Progamáveis I

Arthur Cadore Matuella Barcella

29 de Agosto de 2023

Sumário

1 Orientações

1.1 Objetivo

- Revisar o processo de programação do FPGA usando um kit de desenvolvimento
- Fazer as adaptações necessárias para o circuito funcionar no kit
- Verificar se o contador proposto funciona, tanto carregando o valor inicial como na contagem progressiva.
- Analisar o que ocorre em um contador quando atinge o seu valor máximo.
- Verificar e corrigir o problema do repique (bouncing) da chave usada no CLK

1.2 Passo 1 - Escrever o código do projeto counter:

- Escrever o código do projeto counter (já simulado em aula anterior), incluindo as adaptações necessárias para o uso dos LEDs da matriz de leds do kit Mercurio IV.
- Fazer a análise e síntese e corrigir eventuais erros.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4
5  entity counter is
6    generic (WIDTH : in natural := 4);
7    port (
8      RST   : in std_logic;
9      CLK   : in std_logic;
10     LOAD  : in std_logic;
11     DATA  : in std_logic_vector(WIDTH-1 downto 0);
12     RO          : out std_logic;
13     Q     : out std_logic_vector(WIDTH-1 downto 0));
14 end entity;
15
16 architecture ifsc_v1 of counter is
17         signal Q_aux : std_logic_vector(WIDTH-1 downto 0);
18 begin
19   process(RST,CLK) is
20   begin
21     if RST = '1' then
22       Q_aux <= (others => '0');
23     elsif rising_edge(CLK) then
24       if LOAD= '1' then
25         Q_aux <= DATA;
26       else
27         Q_aux <= std_logic_vector(unsigned(Q_aux) + 1);
28       end if;
29     end if;
30   end process;
31   -- Adaptacao feita devido a matriz de leds acender com ZERO
32   Q <= not Q_aux;
33   -- Para acender um led eh necessario colocar ZERO na linha correspondente da matriz.
34   RO <= '0';
```

```
35 end architecture;
36
```

1.3 Passo 2 - Preparando para gravar o circuito lógico no FPGA:

- Escolher a FAMILY: **Cyclone® IV E**
- Escolher o DEVICE: **EP4CE30F23C7**
- Configurar como entrada e saída do FPGA os seguintes pinos:

```
1  CLK:     PIN_Y17 ou PIN_V21
2  DATA[3]: PIN_H18
3  DATA[2]: PIN_H20
4  DATA[1]: PIN_K21
5  DATA[0]: PIN_J21
6  LOAD:    PIN_Y22
7  Q[3]:    PIN_J6
8  Q[2]:    PIN_K8
9  Q[1]:    PIN_J8
10 Q[0]:    PIN_L8
11 RST:     PIN_W21
12 RO:      PIN_F10
```

1.4 Passo 3 - Programando o FPGA através da USB-Blaster:

- Realizar os seguintes testes, acionando as chaves e observando o resultado nos LEDs:
- Carregar um valor nas chaves DATA[3..0], mudar LOAD para ALTO e acionar a chave CLK. Verificar e anotar o comportamento. Repetir com valores diferentes nas DATA[3..0].
- Mudar RST para ALTO, e verificar e anotar o comportamento.
- Manter LOAD em BAIXO e acionar a chave CLK várias vezes (no mínimo 16 vezes). Verificar e anotar o comportamento. O comportamento é o esperado para o número de mudanças da chave CLK?
- Dica para configuração das interfaces:

```
1  -- insira na declaração das portas da entity a linha
2  LCD_BACKLIGHT:        out std_logic;
3
4  -- insira na architecture a linha
5  LCD_BACKLIGHT <= '0';
```

- Após fazer a Análise e Síntese, defina o pino v10 para essa porta.

```
1  LCD_BACKLIGHT: PIN_V10
```

1.5 Passo 4 - Eliminar o repique da chave CLK:

- Eliminar o repique da chave CLK, inserindo no código um circuito anti-repique, com um tempo de anti-repique de 10ms:

```
entity COUNTER_db is
...
    CLK50MHz : in std_logic;
...
end entity

architecture ifsc_v2 of COUNTER_db is
...
        signal CLK_db:        std_logic := '0';
...
begin
        -- debouncer de 10ms
        process (CLK50MHz, CLK, RST, CLK_db) is
                constant max_cnt: natural := 500000; -- 500000 10ms para clk 20ns
                variable cnt_db : integer range 0 to max_cnt-1;
        begin
                        if (RST = '1') then
                                cnt_db := 0;
                                CLK_db <= '0';
                        elsif ((CLK = '0') and (CLK_db = '0')) or
                              ((CLK = '1') and (CLK_db = '1')) then
                                cnt_db := 0;
                        elsif (rising_edge(CLK50MHz)) then
                                if (cnt_db = max_cnt - 1) then
                                        CLK_db <= not CLK_db;
                                else
                                        cnt_db := cnt_db + 1;
                                end if;
                        end if;
         end process;
...
-- Troque no process(RST,CLK) a entrada '''CLK''' do circuito anterior pela entrada '''CLK_db'''
```

- Acrescentar o pinos de entrada CLK 50MHz:

```
CLK50MHz:     PIN_T1
```

- Acrescente um arquivo para restringir a análise temporal (Timing Analysis) a 50MHz para a entrada de clock CLK50MHz. Restringir a frequencia máxima de clock no Quartus II:

```
create_clock -name CLK50MHz -period 50MHz [get_ports -no_case {clk*}]
```

1.6 Passo 5 - Adicionando as chaves e observando o resultado nos LEDs:

- Repita os teste feitos no Passo 3, acionando as chaves e observando o resultado nos LEDs:

- Carregar um valor nas chaves DATA[3..0], mudar LOAD para ALTO e acionar a chave CLK. Verificar e anotar o comportamento. Repetir com valores diferentes nas DATA[3..0].
- Mudar RST para ALTO, e verificar e anotar o comportamento.
- Manter LOAD em BAIXO e acionar a chave CLK várias vezes (no mínimo 16 vezes). Verificar e anotar o comportamento. O comportamento é o esperado para o número de mudanças da chave CLK?
- O que ocorre quando o contador chega ao seu valor máximo? Quais seriam as alternativas "teóricas"para evitar que isso ocorra? Proponha soluções, sem se preocupar com um código de descrição do hardware (HDL).
- Reduza o tempo do circuito anti-repique para 1us (microsegundo) **maxcnt = 50**, e verifique o funcionamento da chave CLK.

2 Desenvolvimento

2.1 Passo 1: Criando o projeto e escrevendo o código base

Inicialmente, configurei um novo projeto utilizando a familia e modelo de FPGA requisitado no laboratório, para que os resultados resultassem no esperado pelo avaliador.

FONTE: Elaborado pelo autor

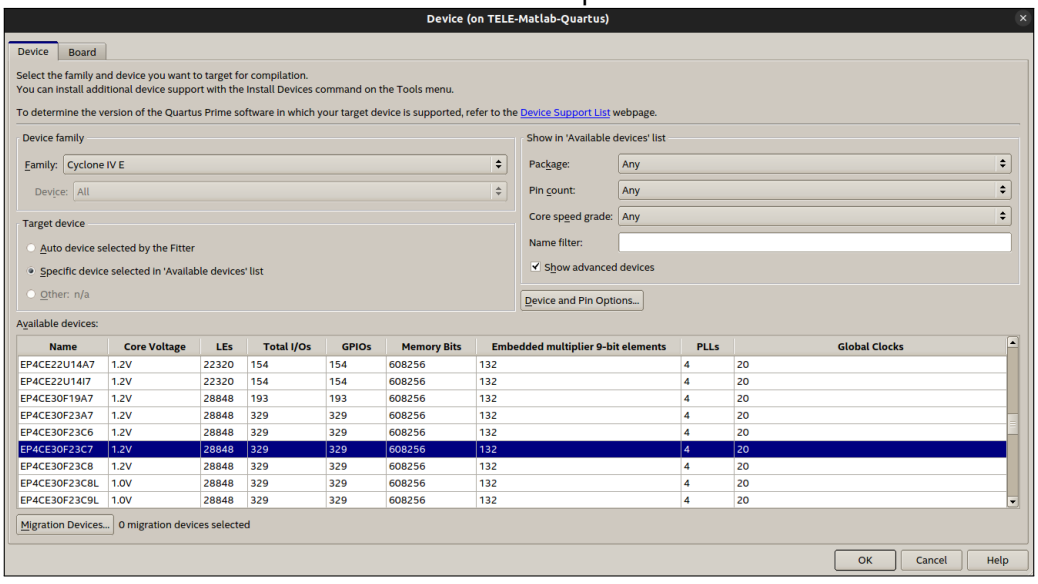

Figura 1: Configurações iniciais do projeto AE3

Uma vez com o projeto configurado, adicionei o código VHDL orientado pelo professor durante a aula, fazendo as modificações no software para torna-lo o código principal do projeto para compilação.

FONTE: Elaborado pelo autor

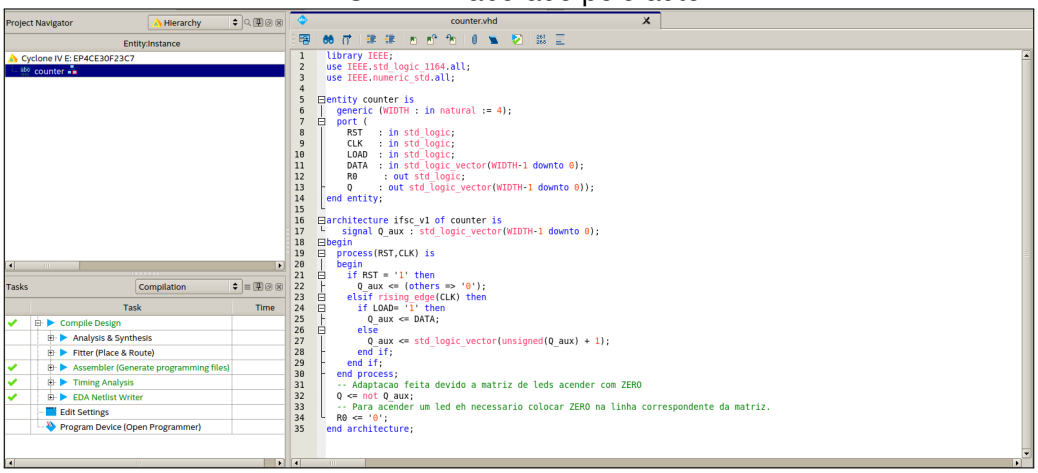

Figura 2: Adicionado código VHDL ao projeto

Em seguida, fiz a análise e síntese do código e observei os logs de compilação registrados pelo quartus, apenas para garantir que nenhum erro havia sido apresentado durante a compilação.

FONTE: Elaborado pelo autor

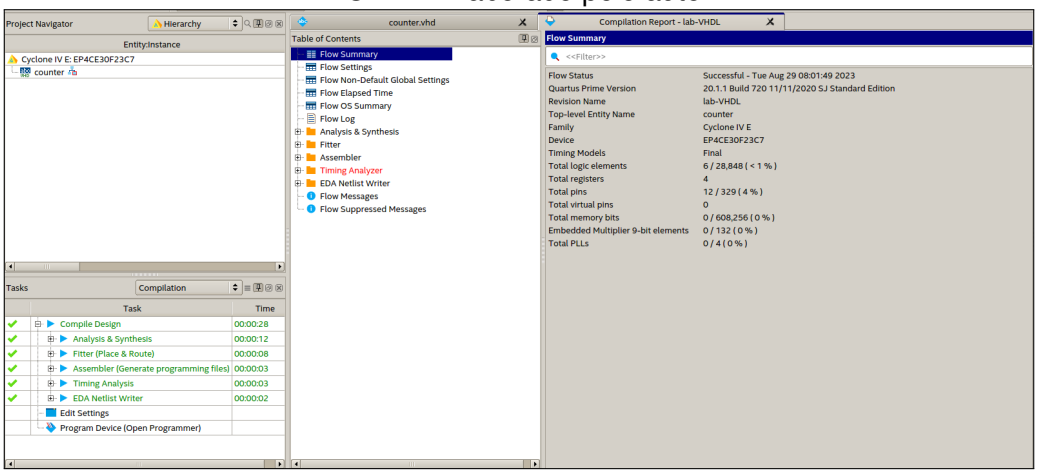

Figura 3: Resumo da compilação no software

A partir da figura 3, nota-se os recursos de hardware foram de fatos consumidos pelo código configurado no quartus, indicando que o circuito lógico foi montado na FPGA.

Apenas para verificar se a semântica foi mantida durante a compilação, verifiquei o diagrama RTL do circuito configurado pelo quartus para determinar se está de acordo com o código inserido, abaixo está o diagrama completo:

FONTE: Elaborado pelo autor

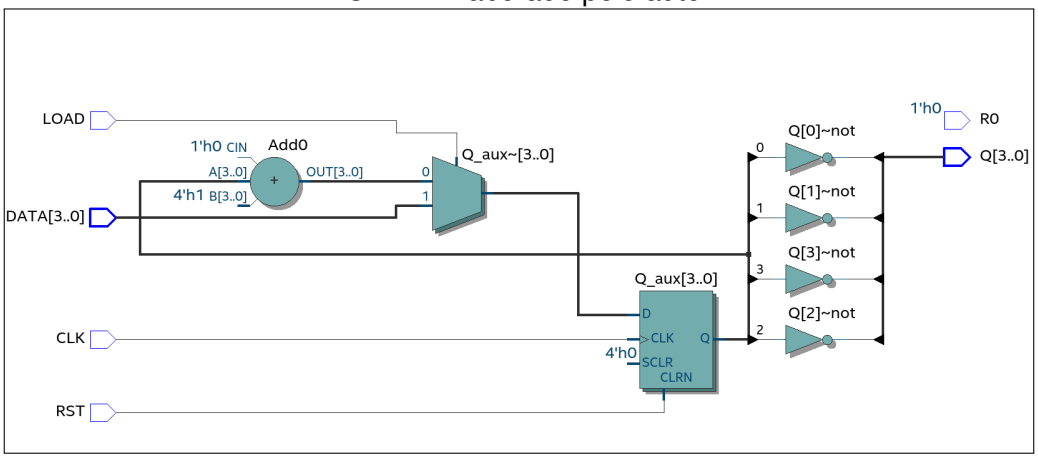

Figura 4: Diagrama RTL do circuito configurado a partir do código VHDL

Também verifiquei o estado do circuito através do diagrama "Technology Map", a vantagem deste diagrama é mostrar não necessáriamente blocos que compõe parte do circuito, mas sim componentes lógicos utilizados para estruturar todo o hardware que monta o circuito proposto.

FONTE: Elaborado pelo autor

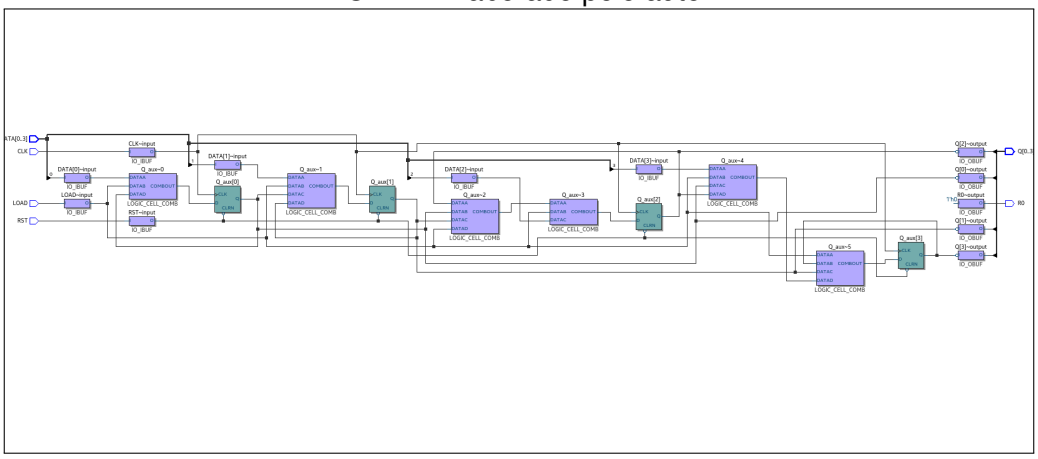

Figura 5: Technology Map do circuito montado através de VHDL.

2.2 Passo 2: Gravando o circuito lógico no FPGA

Em seguida, antes de encaminhar o código para gravação na FPGA, verifiquei o diagrama de pin-planner para garantir que todos os pinos haviam sido atribuidos a suas respectivas portas no circuito eletrônico.

FONTE: Elaborado pelo autor

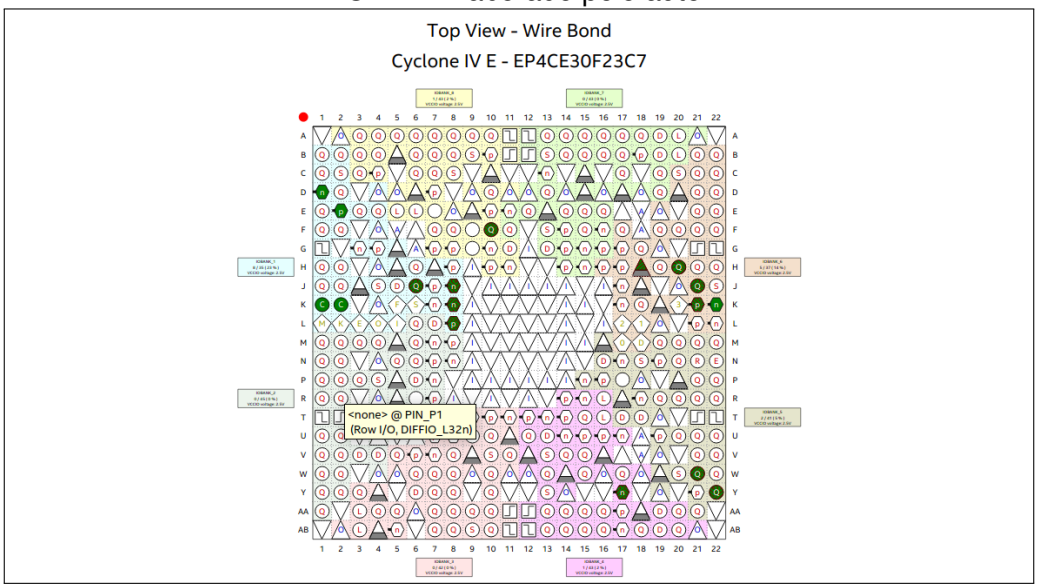

Figura 6: Representação da pinagem do chip VHDL

A partir da visualização, confirmei que todos os pinos haviam sido associados a um pino de I/O para interface com usuário ou com outros circuitos eletrônicos.

FONTE: Elaborado pelo autor

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strengtl	Slew Rate	er I/O Rule Resu	Differential Pair	trict Preservatic
CLK	Input	PIN_Y17	4	B4_N0	PIN_Y17	2.5 V		8mA (default)		Pass		
DATA[3]	Input	PIN_H18	6	B6_N1	PIN_H18	2.5 V		8mA (default)		Pass		
DATA[2]	Input	PIN_H20	6	B6_N1	PIN_H20	2.5 V		8mA (default)		Pass		
DATA[1]	Input	PIN_K21	6	B6_N3	PIN_K21	2.5 V		8mA (default)		Pass		
DATA[0]	Input	PIN_J21	6	B6_N3	PIN_J21	2.5 V		8mA (default)		Pass		
LOAD	Input	PIN_Y22	5	B5_N3	PIN_Y22	2.5 V		8mA (default)		Pass		
Q[3]	Output	PIN_J6	1	B1_N1	PIN_J6	2.5 V		8mA (default)	2 (default)	Pass		
Q[2]	Output	PIN_K8	1	B1_N2	PIN_K8	2.5 V		8mA (default)	2 (default)	Pass		
Q[1]	Output	PIN_J8	1	B1_N1	PIN_J8	2.5 V		8mA (default)	2 (default)	Pass		
Q[0]	Output	PIN_L8	1	B1_N2	PIN_L8	2.5 V		8mA (default)	2 (default)	Pass		
RO	Output	PIN_F10	8	B8_N2	PIN_F10	2.5 V		8mA (default)	2 (default)	Pass		
RST	Input	PIN_W21	5	B5_N2	PIN_W21	2.5 V		8mA (default)		Pass		

Figura 7: Technology Map do circuito montado através de VHDL.

Com a configuração ja finalizada e o arquivo compilado, verifiquei no diretório raiz do projeto a localização do arquivo .sof, que contém o binário a ser enviado á FPGA, encontrei o arquivo na pasta **output-files**:

FONTE: Elaborado pelo autor

```
arthur.cmb@TELE-Matlab-Quartus:~/projects/lab-VHDL/output_files$ ls -l
total 1524
-rw-r--r-- 1 arthur.cmb aluno    3841 ago 29 08:01 lab-VHDL.asm.rpt
-rw-r--r-- 1 arthur.cmb aluno      25 ago 29 08:03 lab-VHDL.done
-rw-r--r-- 1 arthur.cmb aluno    5889 ago 29 08:01 lab-VHDL.eda.rpt
-rw-r--r-- 1 arthur.cmb aluno  175447 ago 29 08:01 lab-VHDL.fit.rpt
-rw-r--r-- 1 arthur.cmb aluno     695 ago 25 09:10 lab-VHDL.fit.smsg
-rw-r--r-- 1 arthur.cmb aluno     596 ago 29 08:01 lab-VHDL.fit.summary
-rw-r--r-- 1 arthur.cmb aluno    9525 ago 29 08:01 lab-VHDL.flow.rpt
-rw-r--r-- 1 arthur.cmb aluno     219 ago 29 08:01 lab-VHDL.jdi
-rw-r--r-- 1 arthur.cmb aluno   22504 ago 29 08:01 lab-VHDL.map.rpt
-rw-r--r-- 1 arthur.cmb aluno     451 ago 29 08:01 lab-VHDL.map.summary
-rw-r--r-- 1 arthur.cmb aluno   58627 ago 29 08:01 lab-VHDL.pin
-rw-r--r-- 1 arthur.cmb aluno      20 ago 29 08:01 lab-VHDL.sld
-rw-r--r-- 1 arthur.cmb aluno 1171670 ago 29 08:01 lab-VHDL.sof
-rw-r--r-- 1 arthur.cmb aluno   68725 ago 29 08:01 lab-VHDL.sta.rpt
-rw-r--r-- 1 arthur.cmb aluno     890 ago 29 08:01 lab-VHDL.sta.summary
arthur.cmb@TELE-Matlab-Quartus:~/projects/lab-VHDL/output_files$
```

Figura 8: Technology Map do circuito montado através de VHDL.

Em seguida, transferi o arquivo diretamente para minha máquina onde fiz o envio para a placa FPGA utilizando a própria interface do quartus.

FONTE: Elaborado pelo autor

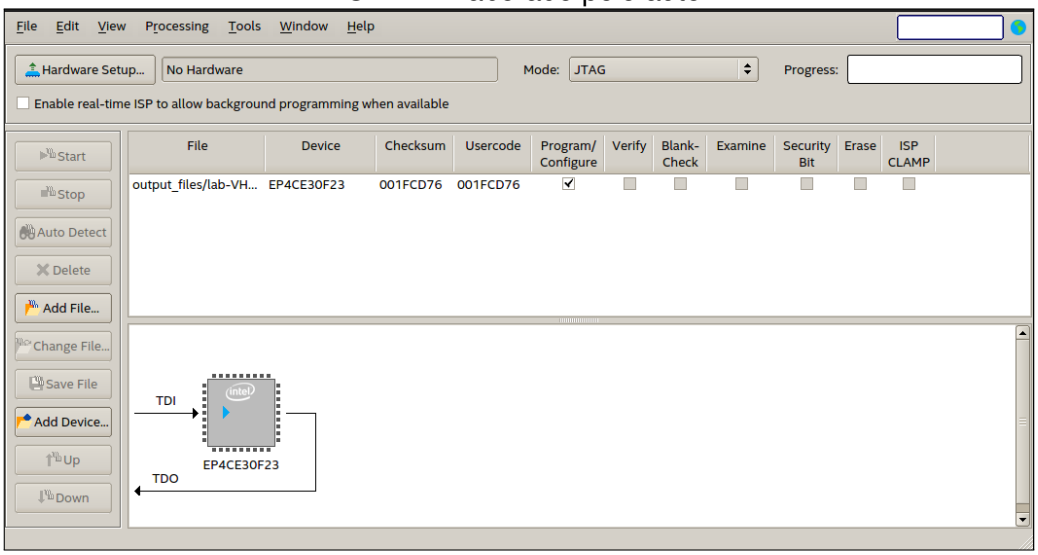

Figura 9: Envio do arquivo utilizando a ferramenta programmer do quartus

2.3 Passo 3: Realizar os testes de repique da chave

Com arquivo ja na placa de teste (FPGA Físico), fiz a alternância das chaves de contato conforme descrito nas orientações do item 3.

Nota-se que ao alterar as chaves/botões da placa de teste, so primeiros LEDs da matriz de LED apresentada nas figures 9, 10 11 e 12, sofrem variações.

FONTE: Elaborado pelo autor

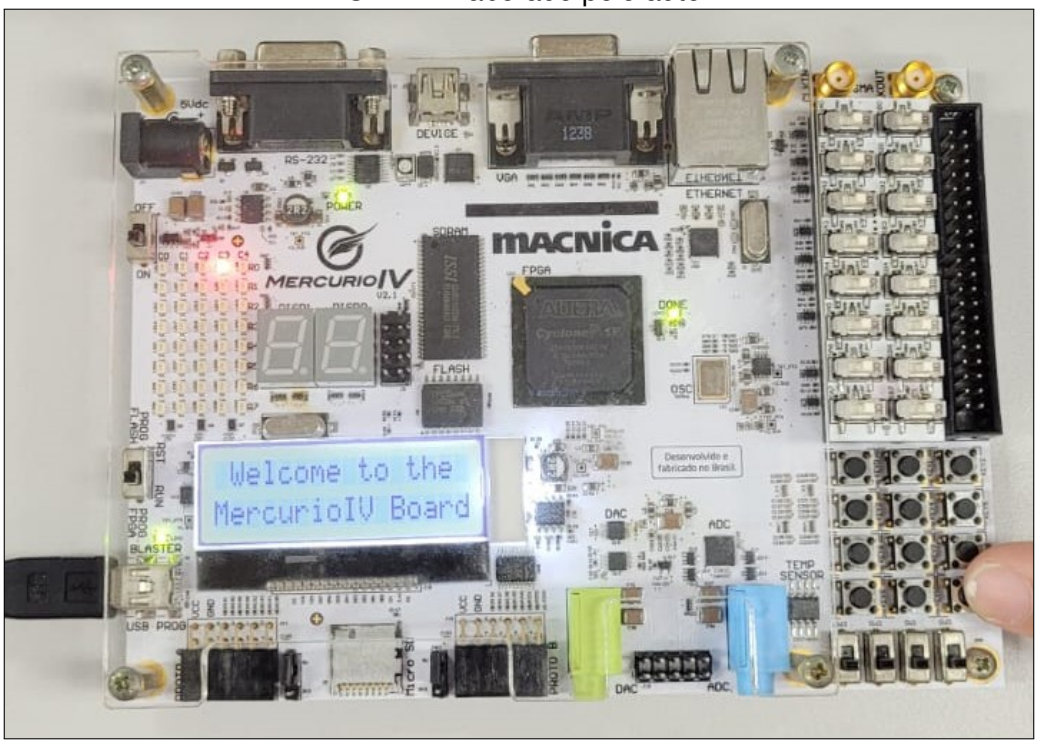

Figura 10: Captura da operação dos LEDs na FPGA

Idealmente, como um contador está sendo utilizado a representação dos LEDs deveria ser sequencia, como em uma tabela verdade de 4Bits, variando em apenas 1Bit a cada ciclo de execução.

FONTE: Elaborado pelo autor

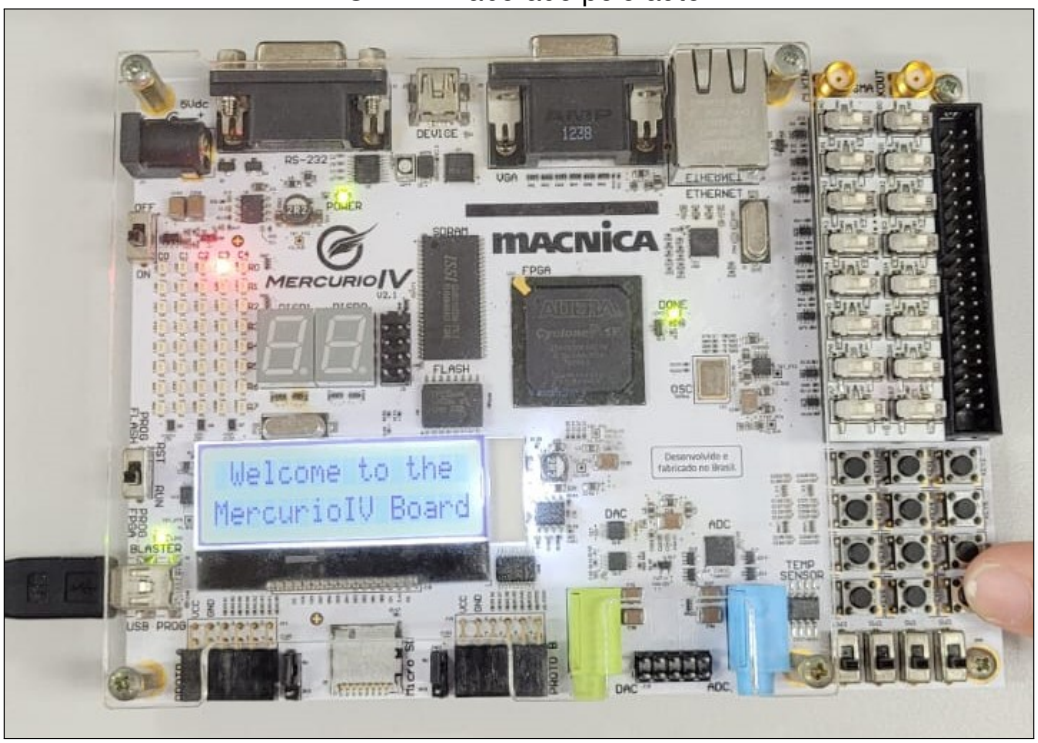

Figura 11: Captura da operação dos LEDs na FPGA

Entretanto, como não há um circuito de deboucing integrado nativamente no dispositivo, a chave durante o processo de abertura/fechamento sofre depridações (vibração). Essas vibrações fazem com que em um curto periodo de tempo a chave abra e feche diversas vezes, e para nossa percepção, a chave está pulando a contagem.

FONTE: Elaborado pelo autor

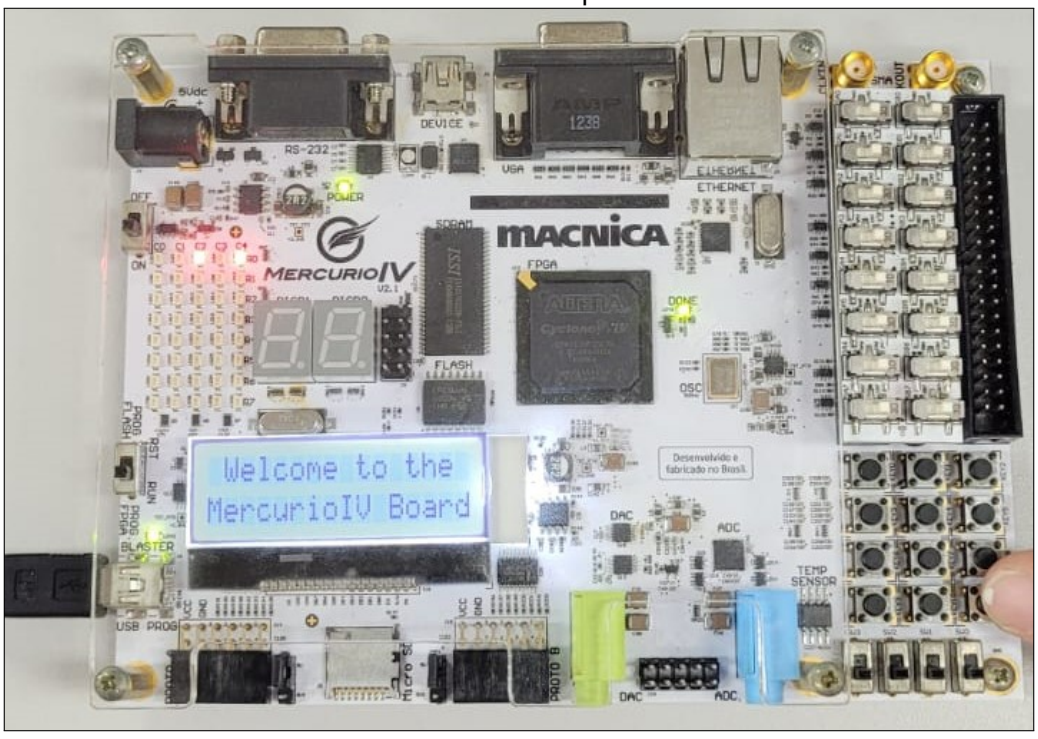

Figura 12: Captura da operação dos LEDs na FPGA

2.4 Passo 4: Eliminar o repique da chave

Em seguida, para resolver o sintoma descrito acima, adicionei a seguinte sessão de código ao projeto (abaixo da arquitetura já existente). O circuito abaixo é um debouncer, utilizado justamente para evitar que este tipo de trepidação mecânica entre os contatos secos atrapalhe o funcionamento do circuito.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4
5  entity counter is
6    generic (WIDTH : in natural := 4);
7    port (
8           CLK50MHz : in std_logic;
9      RST    : in std_logic;
10     CLK    : in std_logic;
11     LOAD   : in std_logic;
12     DATA   : in std_logic_vector(WIDTH-1 downto 0);
13     RO           : out std_logic;
14     Q      : out std_logic_vector(WIDTH-1 downto 0));
15 end entity;
16
17 architecture ifsc_v1 of counter is
18         signal Q_aux : std_logic_vector(WIDTH-1 downto 0);
19         signal CLK_db:         std_logic := '0';
20 begin
21   process(RST,CLK) is
22   begin
23     if RST = '1' then
24       Q_aux <= (others => '0');
25     elsif rising_edge(CLK) then
26       if LOAD= '1' then
27         Q_aux <= DATA;
28       else
29         Q_aux <= std_logic_vector(unsigned(Q_aux) + 1);
30       end if;
31     end if;
32   end process;
33   -- Adaptacao feita devido a matriz de leds acender com ZERO
34   Q <= not Q_aux;
35   -- Para acender um led eh necessario colocar ZERO na linha correspondente da matriz.
36   RO <= '0';
37
38   -- debouncer de 10ms
39         process (CLK50MHz, CLK, RST, CLK_db) is
40                 constant max_cnt: natural := 500000; -- 500000 10ms para clk 20ns
41                 variable cnt_db : integer range 0 to max_cnt-1;
42         begin
43                         if (RST = '1') then
44                                 cnt_db := 0;
45                                 CLK_db <= '0';
46                         elsif ((CLK = '0') and (CLK_db = '0')) or
47                               ((CLK = '1') and (CLK_db = '1')) then
48                                 cnt_db := 0;
49                         elsif (rising_edge(CLK50MHz)) then
```

```
50                                  if (cnt_db = max_cnt - 1) then
51                                          CLK_db <= not CLK_db;
52                                  else
53                                          cnt_db := cnt_db + 1;
54                                  end if;
55                          end if;
56          end process;
57  end architecture;
```

Após adicionar o código ao VHDL, refiz a compilação e repeti o passo 2, não descrevi passo a passo o ocorrido, pois todos os passos até a escrita na FPGA estão descritos no passo 2.

FONTE: Elaborado pelo autor

```
 5  entity counter is
 6    generic (WIDTH : in natural := 4);
 7    port (
 8      CLK50MHz : in std_logic;
 9      RST   : in std_logic;
10      CLK   : in std_logic;
11      LOAD  : in std_logic;
12      DATA  : in std_logic_vector(WIDTH-1 downto 0);
13      R0    : out std_logic;
14      Q     : out std_logic_vector(WIDTH-1 downto 0));
15  end entity;
16
17  architecture ifsc_v1 of counter is
18     signal Q_aux : std_logic_vector(WIDTH-1 downto 0);
19     signal CLK_db: std_logic := '0';
20  begin
21    process(RST,CLK) is
22    begin
23      if RST = '1' then
24        Q_aux <= (others => '0');
25      elsif rising_edge(CLK) then
26        if LOAD= '1' then
27          Q_aux <= DATA;
28        else
29          Q_aux <= std_logic_vector(unsigned(Q_aux) + 1);
30        end if;
31      end if;
32    end process;
33    -- Adaptacao feita devido a matriz de leds acender com ZERO
34    Q <= not Q_aux;
35    -- Para acender um led eh necessario colocar ZERO na linha correspondente da matriz.
36    R0 <= '0';
```

Figura 13: Código VHDL alterado adicionando o Clock de 50MHz

FONTE: Elaborado pelo autor

```
38    -- debouncer de 10ms
39    process (CLK50MHz, CLK, RST, CLK_db) is
40       constant max_cnt: natural := 500000; -- 500000 10ms para clk 20ns
41       variable cnt_db : integer range 0 to max_cnt-1;
42    begin
43          if (RST = '1') then
44             cnt_db := 0;
45             CLK_db <= '0';
46          elsif ((CLK = '0') and (CLK_db = '0')) or
47                ((CLK = '1') and (CLK_db = '1')) then
48             cnt_db := 0;
49          elsif (rising_edge(CLK50MHz)) then
50             if (cnt_db = max_cnt - 1) then
51                CLK_db <= not CLK_db;
52             else
53                cnt_db := cnt_db + 1;
54             end if;
55          end if;
56    end process;
57  end architecture;
```

Figura 14: Technology Map do circuto com debouncer

Após a alteração do código, refiz a compilação e observei o diagrama RTL do circuito e também o TechnologyMap. Com a adição do código de deboucer, nota-se que um novo bloco foi introduzido nas figuras 15 e 16, na entrada dos dados.

FONTE: Elaborado pelo autor

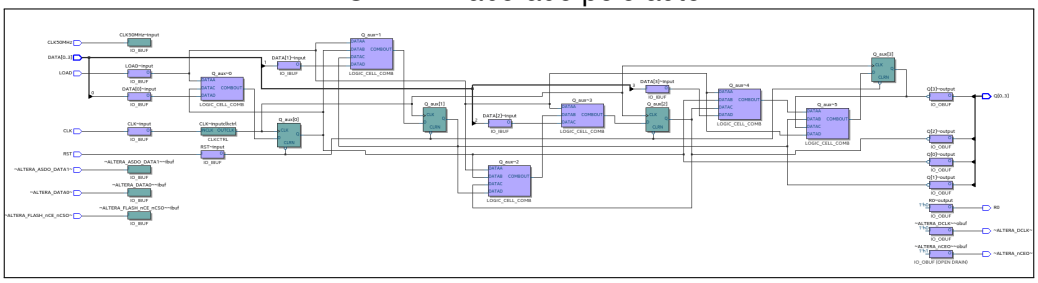

Figura 15: Diagrama RTL do circuto com debouncer

Este bloco é utilizado para evitar que o repique da chave seja propagado para o resto do circuto e afete na contagem.

FONTE: Elaborado pelo autor

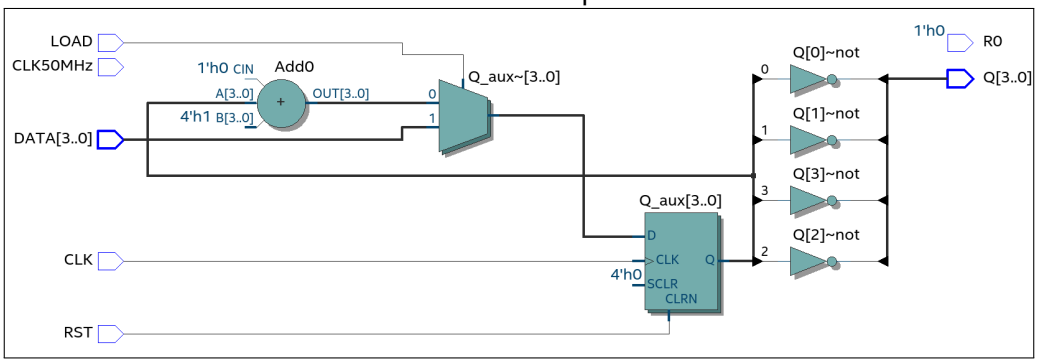

Figura 16: Código VHDL alterado adicionando o bloco de função de debouncer

2.5 Passo 5: Observando circuito com debouncer

Após re-enviar o código, agora com o circuito de debouncer adicionado, nota-se uma melhora no processo de avanço da chave de seleção, agora ao avaçar a posição com a chave seletora, o circuito evita que posições de contagem sejam puladas, e assim a contagem de alteração dos LEDs é apresentada corretamente.

Abaixo estão duas capturas sequencias, apresentado o led 0011 e 0101, correspondendo a duas variações na chave de seleção, o primeiro valor equivale a 3 em decimal (figura 10) e o segundo valor equivale á 5 em decimal (figura 11).

FONTE: Elaborado pelo autor

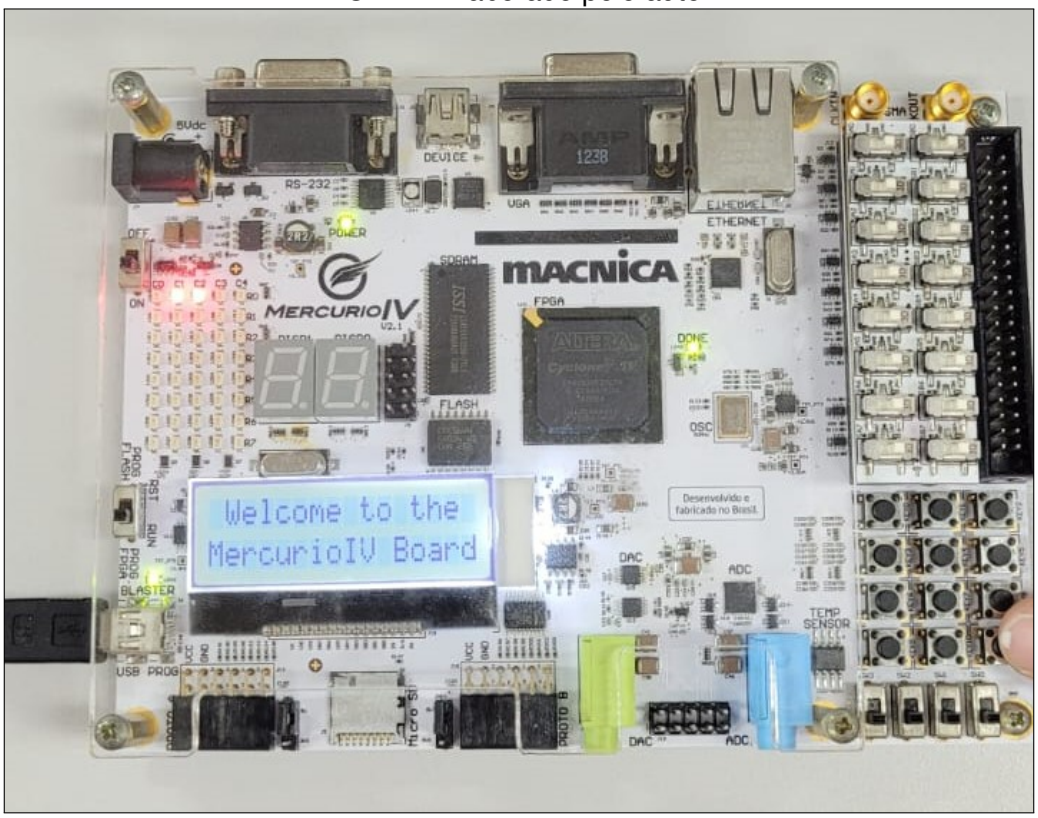

Figura 17: Technology Map do circuito montado através de VHDL.

FONTE: Elaborado pelo autor

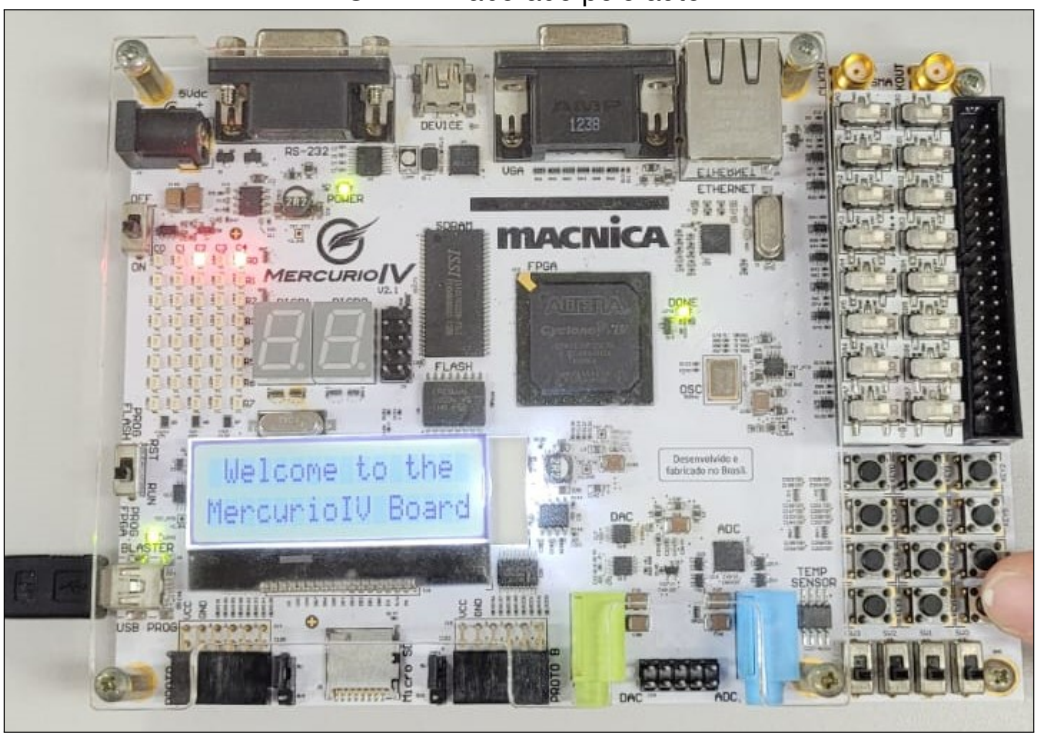

Figura 18: Captura da operação dos LEDs na FPGA

3 Referências bibliográficas

Wiki diária da disciplina de DLP - IFSC SJ
Tempo de propagação em circuitos combinacionais
Conhecendo os dispositivos lógicos programáveis
Acesso a Nuvem do IFSC (Para realização da atividade)