Implementação de FSM para Calculadora

Dispositivos Lógicos Progamáveis II

Arthur Cadore Matuella Barcella e Gabriel Luiz Espindola Pedro

08 de Agosto de 2024

Engenharia de Telecomunicações - IFSC-SJ

Sumário

1. Introdução:

O objetivo deste relatório consiste na implementação de uma calculadora utilizando uma Máquina de Estados Finitos (FSM) para controle do datapath. A calculadora deve ser capaz de realizar operações de soma, subtração, adição de 1 e subtração de 1.

2. Implementação ASM da FSM:

A primeira etapa do projeto é em montar o diagrama ASM da FSM que será responsável por controlar a calculadora. A Figura 1 apresenta o diagrama ASM da FSM que será implementada.

Figure 1: Elaborada pelo Autor

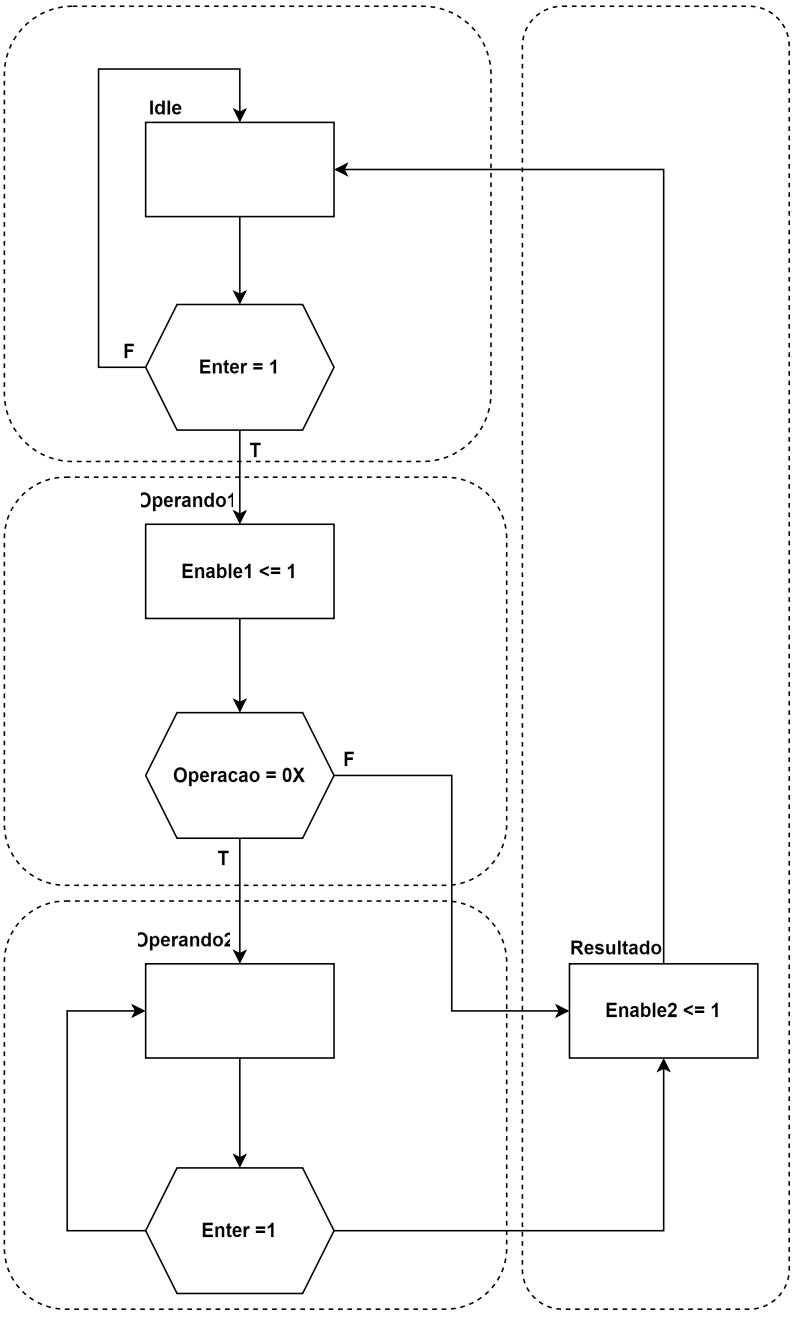


Diagrama ASM da FSM

A FSM é divida em 4 estados, sendos eles:

- **Idle**: Estado inicial da máquina, onda ela fica em loop aguardando um pulso de “Enter” para iniciar a operação.
- **Operando1**: Estado após “idle”, onde a máquina lê habilita o datapath para leitura do primeiro operando utilizando o sinal “Enable1”.

Neste ponto, o estado também valida a operação a se realizada através de um vetor de 2bits, sendo possivel aplicar 00, 01, 10 e 11, oque corresponde respectivamente á soma, subtração, adição+1 e subtração-1.

Nesta verificação, apenas o bit mais significativo 0X é utilizado, pois o bit de maior ordem define se a operação precisará ou não de um segundo operando.

Após a verificação, caso o valor do bit mais significativo seja “0” a máquina irá para o estado “Operando2”, caso seja “1” a máquina irá para o estado “Resultado”.

- **Operando2**: Estado onde a máquina fica em loop aguardando o segundo pulso de “Enter”, para receber o segundo operando.
- **Resultado**: Estado onde a máquina lê habilita o datapath para processamento do resultado utilizando o sinal “Enable2”.

3. Implementação em VHDL:

Uma vez com o diagrama ASM da FSM pronto, é necessário implementar o código VHDL que irá controlar o datapath da calculadora. Além do datapath propriamente. Desta forma, iniciaremos com a implementação da FSM.

Note que uma vez com o código implementado, o diagrama FSM obtido no simulador é análogo ao diagrama FSM obtido no diagrama ASM.

Figure 2: Elaborada pelo Autor

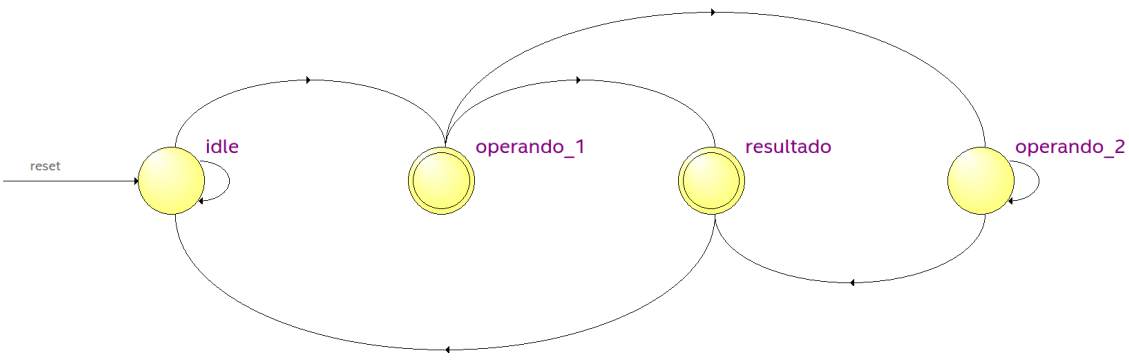


Diagrama ASM da FSM

3.1. Implementação do VHDL da FSM para controle:

A implementação da FSM consiste no recebimento de sinais especificos de entrada, e a partir do estado atual e dos sinais recebidos a máquina decide para qual estado irá, e neste, qual saidas devem ser ativadas.

Abaixo podemos ver o código VHDL da FSM:

Figure 3: Elaborada pelo Autor

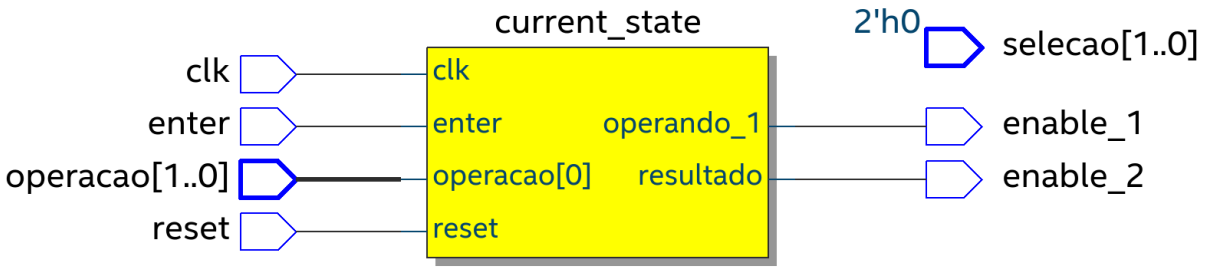


Diagrama ASM da FSM

```
a
```

3.2. Implementação do VHDL datapath da calculadora:

Em seguida, realizamos o desenvolvimento do datapath da calculadora. O datapath da calculadora é responsável por realizar as operações de soma, subtração, adição de 1 e subtração de 1 propriamente ditas.

Esse circuito opera recebendo os sinais de controle da FSM, e a partir destes sinais, realiza as operações de acordo com o que foi solicitado.

Abaixo podemos ver o RTL correspondente ao datapath da calculadora:

Figure 4: Elaborada pelo Autor

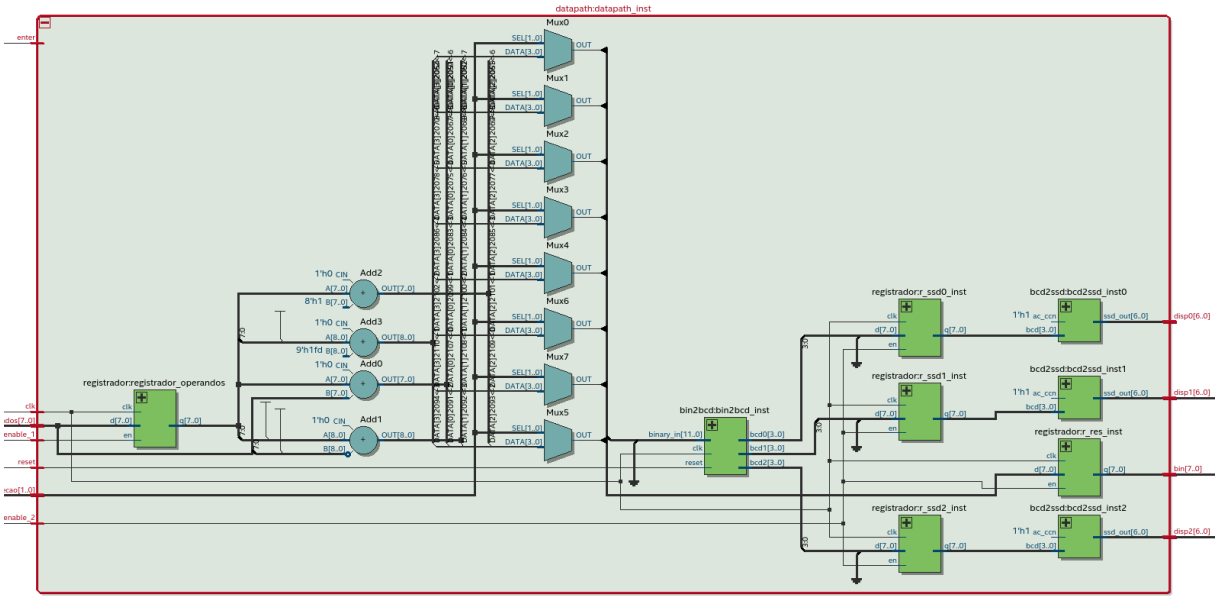


Diagrama ASM da FSM

3.3. Instânciação dos componentes e conexão dos sinais:

Em seguida, com os códigos da FSM e do datapath prontos, é necessário instanciar os componentes e conectar os sinais.

O objetivo da conexão é permitir que a FSM envie os sinais de controle necessários para o datapath operar corretamente, conforme a ilustração abaixo:

Figure 5: Elaborada pelo Autor

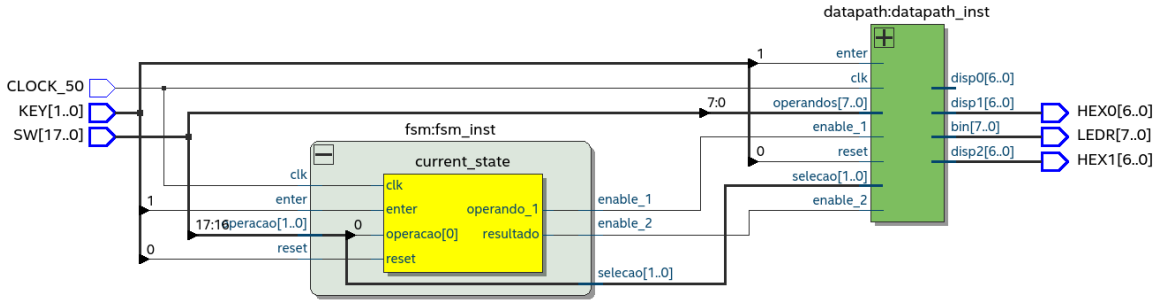


Diagrama ASM da FSM

Note que tanto a FSM quanto o datapath recebem sinais vindo das entradas para a passagem de dados. Porem, os dados voltados para a operação e mudança de estado passam pela FSM, enquanto os dados em si são encaminhados diretamente para o datapath.

Também podemos ver de maneira mais detalhada as conexões da FSM utilizando o diagrama de tecnologia abaixo:

Figure 6: Elaborada pelo Autor

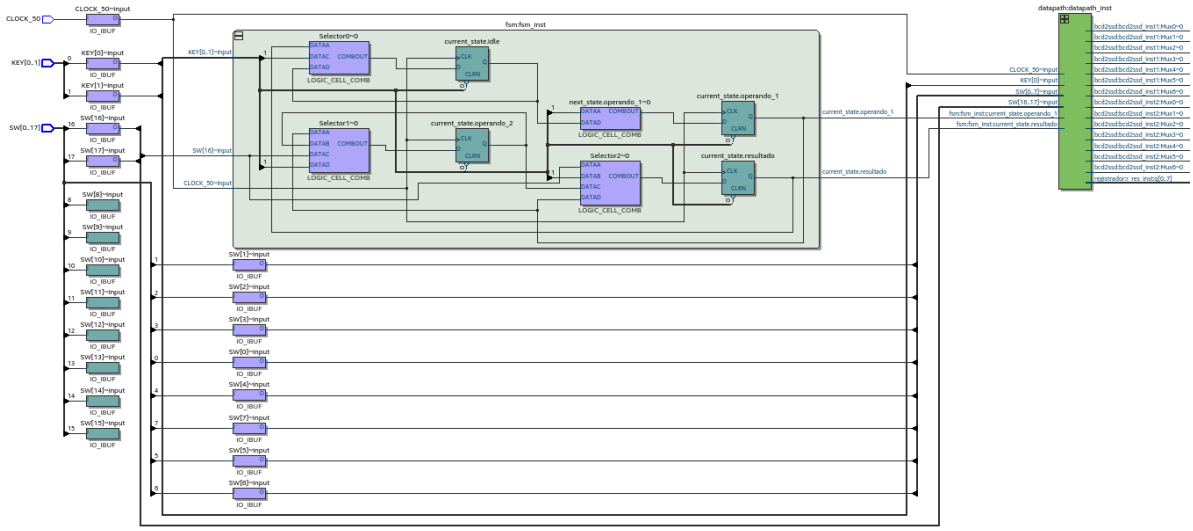


Diagrama ASM da FSM

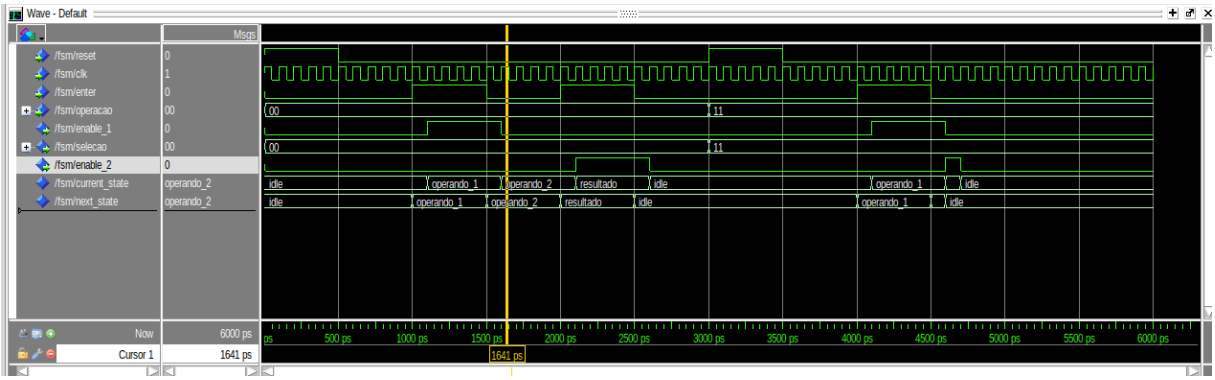
3.4. Teste dos sinais:

Na sequência, realizamos o teste dos sinais de entrada e saída da FSM e do datapath para garantir que a calculadora está operando corretamente.

3.4.1. Teste da FSM:

Inicialmente realizamos o teste dos sinais de entrada e saída da FSM, criando um `.do` para simulação no ModelSim, conforme ilustrado abaixo:

Figure 7: Elaborada pelo Autor



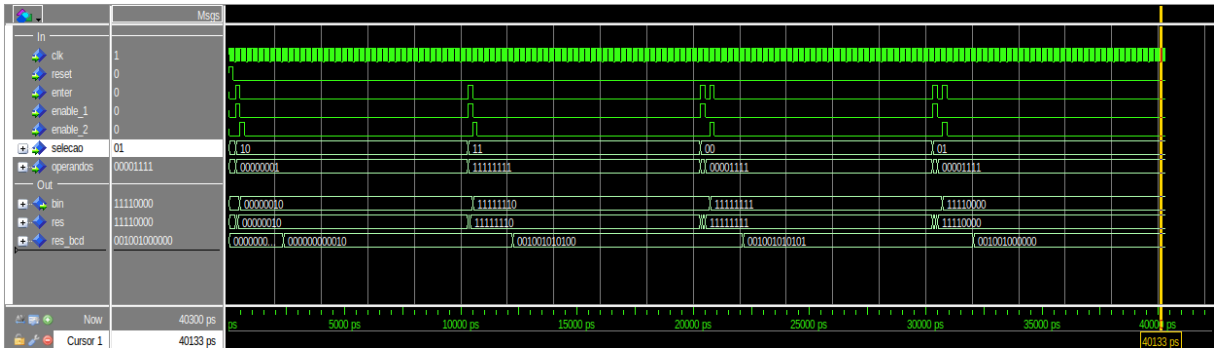
Teste dos sinais de entrada e saída da FSM

Com isso, analisamos as entradas aplicadas e os resultados obtidos, garantindo que a FSM está operando corretamente.

3.4.2. Teste do Datapath:

Na sequência, realizamos o teste dos sinais de entrada e saída do datapath, criando um `.do` para simulação no ModelSim, conforme ilustrado abaixo:

Figure 8: Elaborada pelo Autor



Teste dos sinais de entrada e saída do Datapath

Com isso, analisamos as entradas aplicadas e os resultados obtidos, garantindo que o datapath está operando corretamente.

3.5. Aplicação na placa FPGA:

Por fim, realizamos a aplicação do código na placa FPGA para verificar o funcionamento da calculadora na placa DE-115, para isso, configuramos um arquivo `.qsf` contendo a pinagem a ser aplicada na placa e realizamos o upload para a mesma:

Figure 9: Elaborada pelo Autor

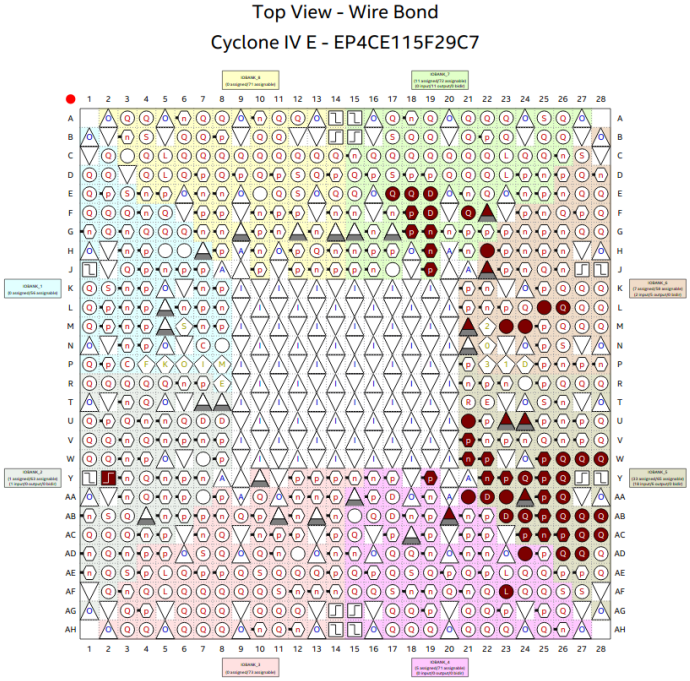


Diagrama ASM da FSM

Sessão do arquivo `.qsf` utilizada, onde é feita a conexão dos pinos da placa com os sinais do datapath e da FSM:

```
set_global_assignment -name FAMILY "Cyclone IV E"
set_global_assignment -name DEVICE EP4CE115F29C7
```

```
set_global_assignment -name TOP_LEVEL_ENTITY topo
set_global_assignment -name ORIGINAL_QUARTUS_VERSION 20.1.1
set_global_assignment -name PROJECT_CREATION_TIME_DATE "08:04:36  JULY 23, 2024"
set_global_assignment -name LAST_QUARTUS_VERSION "20.1.1 Standard Edition"
set_global_assignment -name PROJECT_OUTPUT_DIRECTORY output_files
set_global_assignment -name MIN_CORE_JUNCTION_TEMP 0
set_global_assignment -name MAX_CORE_JUNCTION_TEMP 85
set_global_assignment -name ERROR_CHECK_FREQUENCY_DIVISOR 1
set_global_assignment -name NOMINAL_CORE_SUPPLY_VOLTAGE 1.2V
set_global_assignment -name EDA_SIMULATION_TOOL "ModelSim-Altera (VHDL)"
set_global_assignment    -name    EDA_TIME_SCALE    "1    ps"    -section_id eda_simulation
set_global_assignment   -name   EDA_OUTPUT_DATA_FORMAT   VHDL   -section_id eda_simulation
set_global_assignment -name EDA_GENERATE_FUNCTIONAL_NETLIST OFF -section_id eda_board_design_timing
set_global_assignment -name EDA_GENERATE_FUNCTIONAL_NETLIST OFF -section_id eda_board_design_symbol
set_global_assignment -name EDA_GENERATE_FUNCTIONAL_NETLIST OFF -section_id eda_board_design_signal_integrity
set_global_assignment -name EDA_GENERATE_FUNCTIONAL_NETLIST OFF -section_id eda_board_design_boundary_scan
set_global_assignment -name VHDL_FILE bin2bcd.vhd
set_global_assignment -name VHDL_FILE bcd2ssd.vhd
set_global_assignment -name VHDL_FILE fsm.vhd
set_global_assignment -name VHDL_FILE datapath.vhd
set_global_assignment -name VHDL_FILE topo.vhd
set_global_assignment -name VHDL_FILE registrador.vhd
set_global_assignment -name POWER_PRESET_COOLING_SOLUTION "23 MM HEAT SINK WITH 200 LFPM AIRFLOW"
set_global_assignment -name POWER_BOARD_THERMAL_MODEL "NONE (CONSERVATIVE)"
set_global_assignment -name PARTITION_NETLIST_TYPE SOURCE -section_id Top
set_global_assignment       -name       PARTITION_FITTER_PRESERVATION_LEVEL PLACEMENT_AND_ROUTING -section_id Top
set_global_assignment -name PARTITION_COLOR 16764057 -section_id Top

set_location_assignment PIN_G18 -to HEX0[0]
set_location_assignment PIN_F22 -to HEX0[1]
set_location_assignment PIN_E17 -to HEX0[2]
set_location_assignment PIN_L26 -to HEX0[3]
set_location_assignment PIN_L25 -to HEX0[4]
set_location_assignment PIN_J22 -to HEX0[5]
set_location_assignment PIN_H22 -to HEX0[6]
set_location_assignment PIN_M24 -to HEX1[0]
set_location_assignment PIN_Y22 -to HEX1[1]
set_location_assignment PIN_W21 -to HEX1[2]
set_location_assignment PIN_W22 -to HEX1[3]
set_location_assignment PIN_W25 -to HEX1[4]
set_location_assignment PIN_U23 -to HEX1[5]
set_location_assignment PIN_U24 -to HEX1[6]
set_location_assignment PIN_AA25 -to HEX2[0]
set_location_assignment PIN_AA26 -to HEX2[1]
set_location_assignment PIN_Y25 -to HEX2[2]
set_location_assignment PIN_W26 -to HEX2[3]
set_location_assignment PIN_Y26 -to HEX2[4]
```

```
set_location_assignment PIN_W27 -to HEX2[5]
set_location_assignment PIN_W28 -to HEX2[6]
set_location_assignment PIN_V21 -to HEX3[0]
set_location_assignment PIN_U21 -to HEX3[1]
set_location_assignment PIN_AB20 -to HEX3[2]
set_location_assignment PIN_AA21 -to HEX3[3]
set_location_assignment PIN_AD24 -to HEX3[4]
set_location_assignment PIN_AF23 -to HEX3[5]
set_location_assignment PIN_Y19 -to HEX3[6]

set_location_assignment PIN_M23 -to KEY[0]
set_location_assignment PIN_M21 -to KEY[1]
set_instance_assignment -name IO_STANDARD "2.5 V" -to KEY[0]
set_instance_assignment -name IO_STANDARD "2.5 V" -to KEY[1]

set_location_assignment PIN_Y2 -to CLOCK_50
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to CLOCK_50
```

3.6. Códigos utilizados:

3.6.1. Datapath:

O código do datapath é responsável por realizar as operações de soma, subtração, adição de 1 e subtração de 1, conforme ilustrado abaixo:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY datapath IS
    PORT (
        operandos : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        reset : IN STD_LOGIC;
        clk : IN STD_LOGIC;
        enter : IN STD_LOGIC;
        enable_1 : IN STD_LOGIC;
        selecao : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
        enable_2 : IN STD_LOGIC;

        disp0 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
        disp1 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
        disp2 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
        bin : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
END ENTITY;
ARCHITECTURE datapath_arch OF datapath IS
    COMPONENT registrador IS
        GENERIC (
            N : INTEGER := 7
        );
        PORT (
            d : IN STD_LOGIC_VECTOR(N DOWNTO 0);
            clk : IN STD_LOGIC;
            en : IN STD_LOGIC;
            q : OUT STD_LOGIC_VECTOR(N DOWNTO 0)
```

```
        );
    END COMPONENT;

    COMPONENT bin2bcd IS
        GENERIC (N : POSITIVE := 16);
        PORT (
            clk, reset : IN STD_LOGIC;
            binary_in : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
           bcd0, bcd1, bcd2, bcd3, bcd4 : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
        );
    END COMPONENT;

    COMPONENT bcd2ssd IS
        PORT (
            bcd : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
            ssd_out : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
            ac_ccn : IN STD_LOGIC
        );
    END COMPONENT;

    SIGNAL a, b : STD_LOGIC_VECTOR (7 DOWNTO 0);
    SIGNAL res : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL res_bcd : STD_LOGIC_VECTOR(12 DOWNTO 0);

    SIGNAL q_ssd_0 : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL q_ssd_1 : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL q_ssd_2 : STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN

    registrador_operandos : registrador
    GENERIC MAP(
        N => 7
    )
    PORT MAP(
        d => operandos,
        clk => clk,
        en => enable_1,
        q => a
    );
    b <= operandos;

    WITH selecao SELECT
        res <=
        STD_LOGIC_VECTOR(UNSIGNED(a) + UNSIGNED(b)) WHEN "00",
        STD_LOGIC_VECTOR(UNSIGNED(a) - UNSIGNED(b)) WHEN "01",
        STD_LOGIC_VECTOR(UNSIGNED(a) + 1) WHEN "10",
        STD_LOGIC_VECTOR(UNSIGNED(a) - 1) WHEN OTHERS;

    r_res_inst : registrador
    GENERIC MAP(
        N => 7
    )
    PORT MAP(
        d => res,
        clk => clk,
        en => enable_2,
        q => bin
    );
```

```
89
90      bin2bcd_inst : bin2bcd
91      GENERIC MAP(
92          N => 12
93      )
94      PORT MAP(
95          clk => clk,
96          reset => reset,
97          binary_in => "0000" & res,
98          bcd0 => res_bcd(3 DOWNTO 0),
99          bcd1 => res_bcd(7 DOWNTO 4),
100         bcd2 => res_bcd(11 DOWNTO 8),
101         bcd3 => OPEN,
102         bcd4 => OPEN
103     );
104
105     -- Display unidade
106
107     r_ssd0_inst : registrador
108     GENERIC MAP(
109         N => 7
110     )
111     PORT MAP(
112         d => "0000" & res_bcd(3 DOWNTO 0),
113         clk => clk,
114         en => enable_2,
115         q => q_ssd_0
116     );
117
118     bcd2ssd_inst0 : bcd2ssd
119     PORT MAP(
120         bcd => q_ssd_0(3 DOWNTO 0),
121         ssd_out => disp0,
122         ac_ccn => '1'
123     );
124
125     -- Display dezena
126
127     r_ssd1_inst : registrador
128     GENERIC MAP(
129         N => 7
130     )
131     PORT MAP(
132         d => "0000" & res_bcd(7 DOWNTO 4),
133         clk => clk,
134         en => enable_2,
135         q => q_ssd_1
136     );
137
138     bcd2ssd_inst1 : bcd2ssd
139     PORT MAP(
140         bcd => q_ssd_1(3 DOWNTO 0),
141         ssd_out => disp1,
142         ac_ccn => '1'
143     );
144
145     -- Display centena
146
```

```
147      r_ssd2_inst : registrador
148      GENERIC MAP(
149          N => 7
150      )
151      PORT MAP(
152          d => "0000" & res_bcd(11 DOWNTO 8),
153          clk => clk,
154          en => enable_2,
155          q => q_ssd_2
156      );
157
158      bcd2ssd_inst2 : bcd2ssd
159      PORT MAP(
160          bcd => q_ssd_2(3 DOWNTO 0),
161          ssd_out => disp2,
162          ac_ccn => '1'
163      );
164
165  END ARCHITECTURE;
```

3.6.2. FSM:

O código da FSM é responsável por controlar o datapath da calculadora, conforme ilustrado abaixo:

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  ENTITY fsm IS
5      PORT (
6          reset : IN STD_LOGIC;
7          clk : IN STD_LOGIC;
8          enter : IN STD_LOGIC;
9          operacao : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
10
11         enable_1 : OUT STD_LOGIC;
12         selecao : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
13         enable_2 : OUT STD_LOGIC
14     );
15 END ENTITY;
16
17 ARCHITECTURE fsm_arch OF fsm IS
18     TYPE state IS (idle, operando_1, operando_2, resultado);
19     SIGNAL current_state, next_state : state;
20
21 BEGIN
22     sequential : PROCESS (clk, reset)
23         VARIABLE count : INTEGER := 0;
24     BEGIN
25         IF reset = '1' THEN
26             current_state <= idle;
27         ELSIF rising_edge(clk) THEN
28             current_state <= next_state;
29         END IF;
30     END PROCESS sequential;
31
```

```
    combinational : PROCESS (operacao, enter, current_state)
    BEGIN

        -- Default values
        enable_1 <= '0';
        enable_2 <= '0';

        -- State machine
        next_state <= current_state;

        CASE current_state IS

            WHEN idle =>
                IF enter = '1' THEN
                    next_state <= operando_1;
                ELSE
                    next_state <= idle;
                END IF;

            WHEN operando_1 =>
                enable_1 <= '1';
                IF operacao(0) = '0' THEN
                    next_state <= operando_2;
                ELSE
                    next_state <= resultado;
                END IF;

            WHEN operando_2 =>
                IF enter = '1' THEN
                    next_state <= resultado;
                ELSE
                    next_state <= operando_2;
                END IF;

            WHEN resultado =>
                enable_2 <= '1';
                next_state <= idle;
        END CASE;

    END PROCESS combinational;

    selecao <= operacao;

END ARCHITECTURE;
```

3.6.3. Topo:

Por fim, o código topo é responsável por instanciar os componentes e conectar os sinais, conforme ilustrado abaixo:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY topo IS
    PORT (
        CLOCK_50 : IN STD_LOGIC;
        SW : IN STD_LOGIC_VECTOR (17 DOWNTO 0);
```

```
        KEY : IN STD_LOGIC_VECTOR (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);
        LEDR : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );

END ENTITY;

ARCHITECTURE topo_arch OF topo IS
    COMPONENT datapath IS
        PORT (
            operandos : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
            reset : IN STD_LOGIC;
            clk : IN STD_LOGIC;
            enter : IN STD_LOGIC;
            enable_1 : IN STD_LOGIC;
            selecao : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
            enable_2 : IN STD_LOGIC;

            disp1 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
            disp2 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
            bin : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
        );
    END COMPONENT;

    COMPONENT fsm IS
        PORT (
            reset : IN STD_LOGIC;
            clk : IN STD_LOGIC;
            enter : IN STD_LOGIC;
            operacao : IN STD_LOGIC_VECTOR (1 DOWNTO 0);

            enable_1 : OUT STD_LOGIC;
            selecao : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
            enable_2 : OUT STD_LOGIC
        );
    END COMPONENT;

    SIGNAL operandos : STD_LOGIC_VECTOR (7 DOWNTO 0);
    SIGNAL reset : STD_LOGIC;
    SIGNAL enter : STD_LOGIC;
    SIGNAL operacao : STD_LOGIC_VECTOR (1 DOWNTO 0);
    SIGNAL enable_1 : STD_LOGIC;
    SIGNAL selecao : STD_LOGIC_VECTOR (1 DOWNTO 0);
    SIGNAL enable_2 : STD_LOGIC;

BEGIN
    operandos(7) <= SW(7);
    operandos(6) <= SW(6);
    operandos(5) <= SW(5);
    operandos(4) <= SW(4);
    operandos(3) <= SW(3);
    operandos(2) <= SW(2);
    operandos(1) <= SW(1);
    operandos(0) <= SW(0);

    reset <= KEY(0);
```

```
    enter <= KEY(1);

    operacao(1) <= SW(17);
    operacao(0) <= SW(16);

    datapath_inst : datapath PORT MAP(
        operandos => operandos,
        reset => reset,
        clk => CLOCK_50,
        enter => enter,
        enable_1 => enable_1,
        selecao => selecao,
        enable_2 => enable_2,

        disp1 => HEX0,
        disp2 => HEX1,
        bin => LEDR
    );

    fsm_inst : fsm PORT MAP(
        reset => reset,
        clk => CLOCK_50,
        enter => enter,
        operacao => operacao,

        enable_1 => enable_1,
        selecao => selecao,
        enable_2 => enable_2
    );
END ARCHITECTURE;
```

3.6.4. Registrador:

Além dos códigos apresentados anteriormente, foram utilizados também os códigos abaixo para a implementação da calculadora.

O código registrador é responsável por armazenar um valor de 8bits.

```
-- register

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY registrador IS
    GENERIC (
        N : INTEGER := 7
    );
    PORT (
        d : IN STD_LOGIC_VECTOR(N DOWNTO 0);
        clk : IN STD_LOGIC;
        en : IN STD_LOGIC;
        q : OUT STD_LOGIC_VECTOR(N DOWNTO 0)
    );
END ENTITY;

ARCHITECTURE registrador_arch OF registrador IS
```

```
BEGIN
    PROCESS (clk)
    BEGIN
        IF clk'EVENT AND clk = '1' THEN
            IF en = '1' THEN
                q <= d;
            END IF;
        END IF;
    END PROCESS;
END ARCHITECTURE;
```

3.6.5. bcd2ssd:

O código bcd2ssd é responsável por converter um número BCD de 4bits para um display de 7 segmentos.

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

entity bcd2ssd is
  port (
    BCD : in std_logic_vector (3 downto 0);
    SSD : out std_logic_vector (6 downto 0)
  );

end entity;

architecture arch of bcd2ssd is
begin

  with BCD select
    SSD <= "1000000" when "0000",
    "1111001" when "0001",
    "0100100" when "0010",
    "0110000" when "0011",
    "0011001" when "0100",
    "0010010" when "0101",
    "0000011" when "0110",
    "1111000" when "0111",
    "0000000" when "1000",
    "0011000" when "1001",
    "0111111" when others;
end arch;
```

3.6.6. bin2bcd:

O código bin2bcd é responsável por converter um número binário de 8bits para BCD, onde cada dígito é representado por 4bits.

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

```

```
entity bin2bcd is
    port (
        A      : in  std_logic_vector (7 downto 0);
        sd, su, sc : out std_logic_vector (3 downto 0)
    );
end entity;

architecture ifsc_v1 of bin2bcd is
    signal A_uns          : unsigned (7 downto 0);
    signal sd_uns, su_uns, sc_uns : unsigned (7 downto 0);

begin
    A_uns  <= unsigned(A);
   sc_uns <= A_uns/100;
    sd_uns <= A_uns/10;
    su_uns <= A_uns rem 10;
    sc     <= std_logic_vector(resize(sc_uns, 4));
    sd     <= std_logic_vector(resize(sd_uns, 4));
    su     <= std_logic_vector(resize(su_uns, 4));
end architecture;
```

4. Conclusão:

Com base nos conceitos apresentados e nos resultados obtidos, foi possível implementar uma calculadora utilizando uma Máquina de Estados Finitos (FSM) para controle do datapath. A calculadora é capaz de realizar operações de soma, subtração, adição de 1 e subtração de 1.

Podemos concluir que a implementação da calculadora foi bem sucedida, atendendo aos requisitos propostos e demonstrando o funcionamento correto da FSM e do datapath, abaixo está a tabela de resultados da implementação para consumo de hardware:

Table 1: Elaborada pelo Autor

Implementacao	Área (LE)	Registradores
Parte 1	104	67

Tabela de resultados da implementação