Implementação de FSM para Calculadora

Dispositivos Lógicos Progamáveis II

Arthur Cadore Matuella Barcella e Gabriel Luiz Espindola Pedro

23 de Abril de 2024

Engenharia de Telecomunicações - IFSC-SJ

Sumário

1. Introdução: 3
2. Implementação ASM da FSM: 3
3. Implementação em VHDL: 4
3.1. Implementação do VHDL da FSM para controle: 4
3.2. Implementação do VHDL datapath da calculadora: 4
3.3. Instânciação dos componentes e conexão dos sinais: 4
3.4. Aplicação na placa FPGA: 4
3.5. Demais códigos utilizados: 4
3.5.1. bcd2ssd: 5
3.5.2. bin2bcd: 5
4. Conclusão: 6
5. Referências Bibliográficas: 6

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

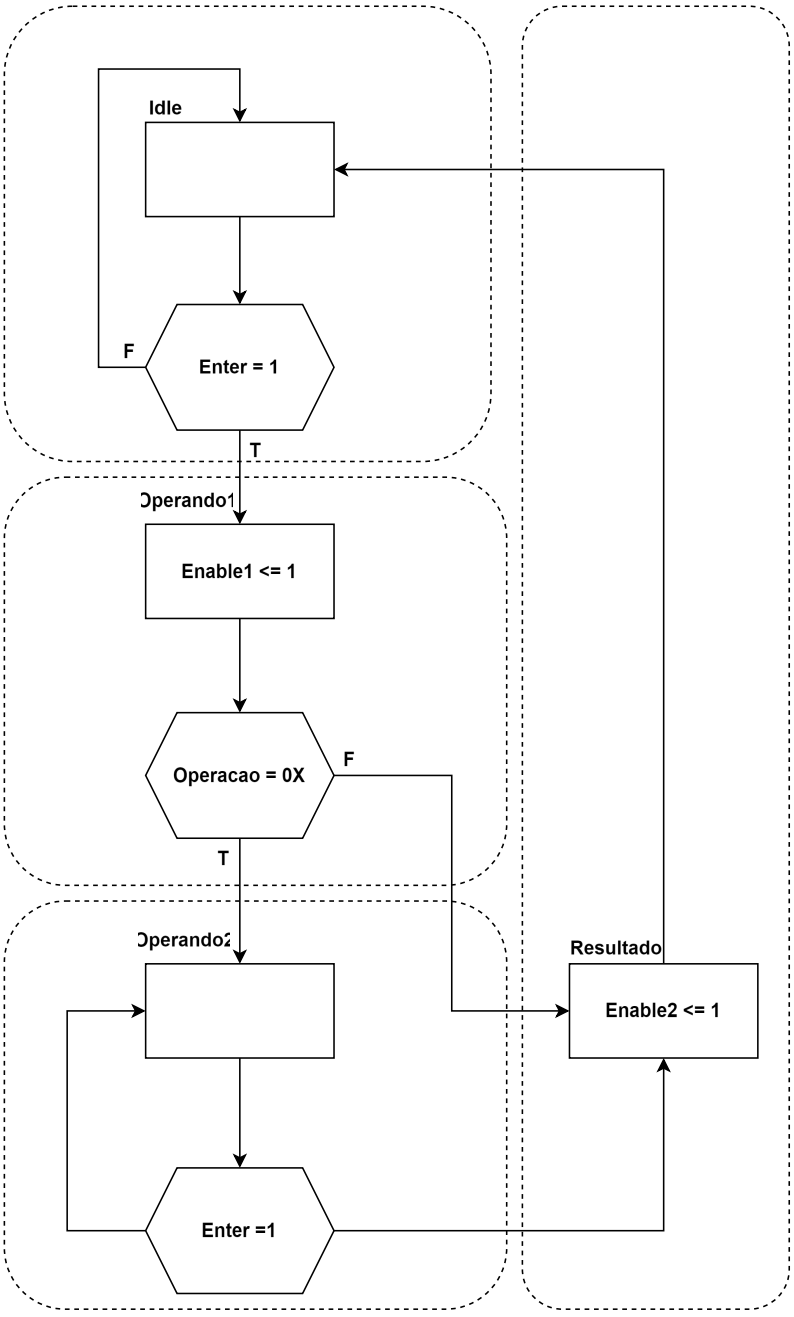

Diagrama ASM da FSM

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.

3.1. Implementação do VHDL da FSM para controle:

A implementação da FSM consiste na

3.2. Implementação do VHDL datapath da calculadora:

3.3. Instânciação dos componentes e conexão dos sinais:

3.4. Aplicação na placa FPGA:

3.5. Demais códigos utilizados:

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

3.5.1. 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;
```

3.5.2. 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);
```

```
18     sc_uns <= A_uns/100;
19      sd_uns <= A_uns/10;
20      su_uns <= A_uns rem 10;
21      sc     <= std_logic_vector(resize(sc_uns, 4));
22      sd     <= std_logic_vector(resize(sd_uns, 4));
23      su     <= std_logic_vector(resize(su_uns, 4));
24  end architecture;
```

4. Conclusão:

5. Referências Bibliográficas: