Universidade Federal do Rio Grande do Norte

ELE1717 - SISTEMAS DIGITAIS

Máquina de Vendas

Projeto e Documentação de uma máquina de vendas

Discente:

Kaike Castro Carvalho

Docente:
Samaherni Morais Dias

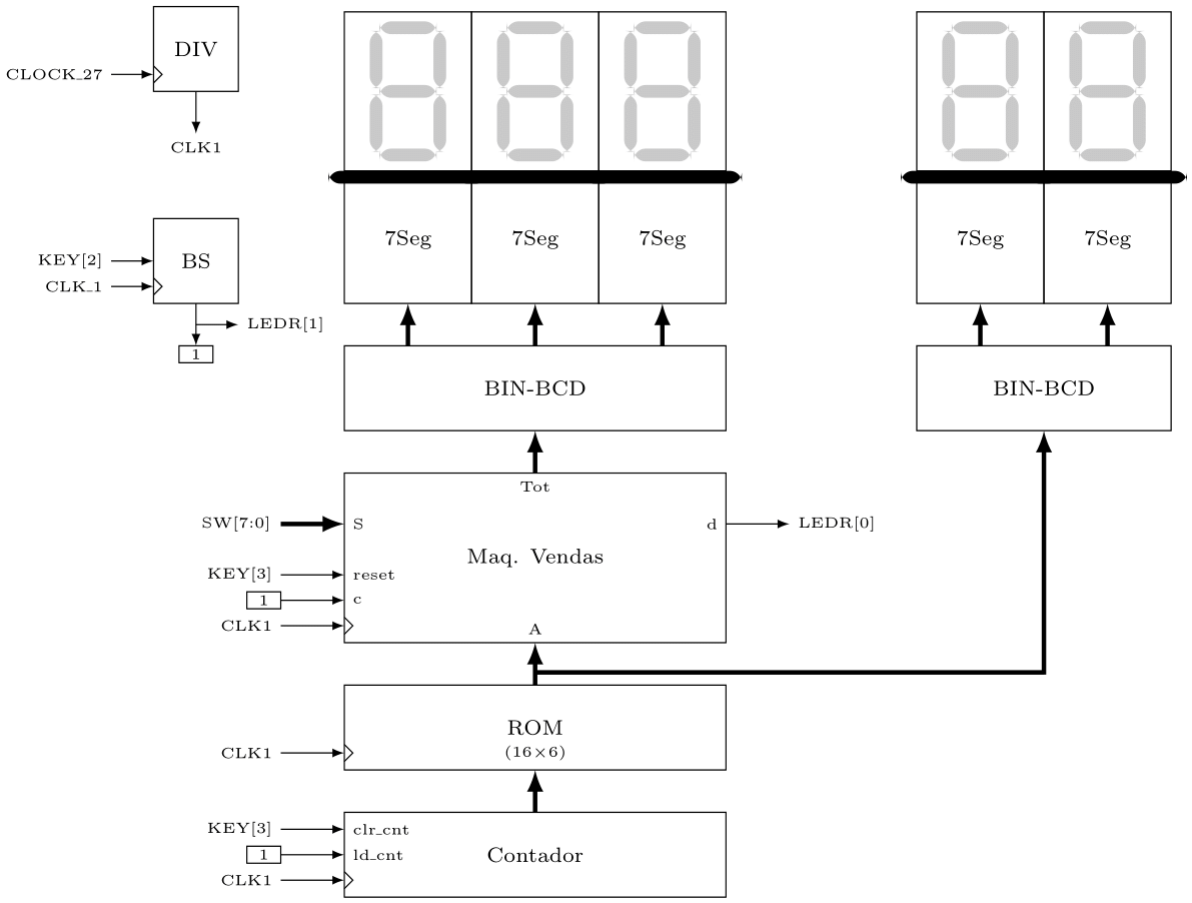
10 de março de 2020

1 Introdução

Esse documento descreve e documenta o projeto de uma máquina de vendas para a disciplina de Sistemas Digitais (ELE1717). O trabalho foi implementado em VHDL, simulado com o auxílio do Quartus II e também executado no kit DE2 FPGA Cyclone II.

A máquina possui duas entradas de dados, sendo uma para informar o valor da moeda inserida (A) e uma outra para entrar com o valor do produto (S). Também conta com uma entrada para indicar quando uma moeda foi depositada (c=1) e uma saída (d=1) para liberar o produto.

Figura 1: Visão geral

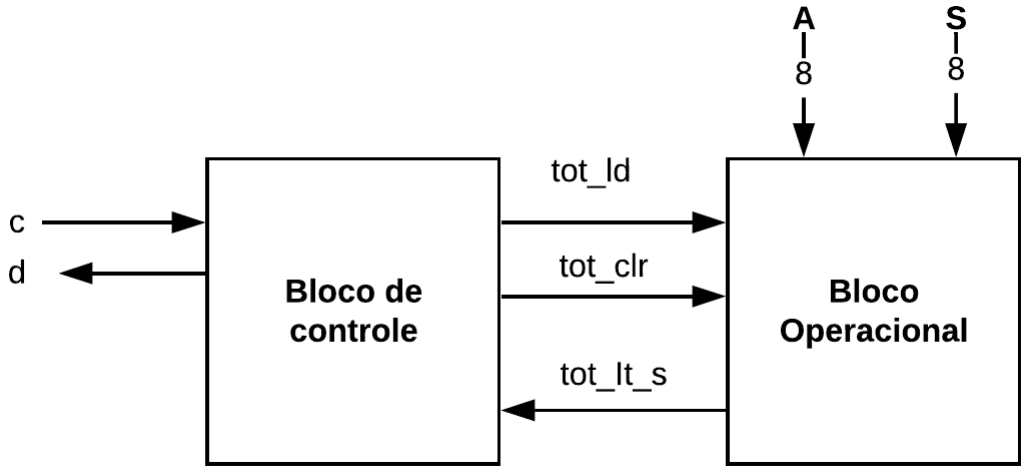


Depois que o processador detecta um total de moedas e que o valor é igual ou maior do que o custo do valor do produto, ele deve atribuir ao bit d o valor 1 durante um ciclo de relógio.

A máquina de vendas não fornece troco e qualquer execesso da inserção de moeda é retido. A Figura 1 acima exemplifica o comportamento geral do processo.

O projeto foi implementado o metódo de projeto em Nível de Transferência entre Registradores (RTL) o qual se divide em BLOCO DE CONTROLE e o BLOCO OPERACIONAL em que a combinação desses blocos é chamada de processaor.

Figura 2: Modelo RTL



2 Objetivo

Desenvolver uma máquina de vendas que se insere o valor do produto e a mémoria ROM que possui os valores da moeda e de acordo com isso o produto é liberado ou não.

3 Materiais e Métodos

A construção do projeto foi baseado no modelo RTL e com isso seguiu cinco passos para a elaboração da solução final. A seguir estão listados os itens que descreve o projeto RTL o qual foi seguido passo a passo.

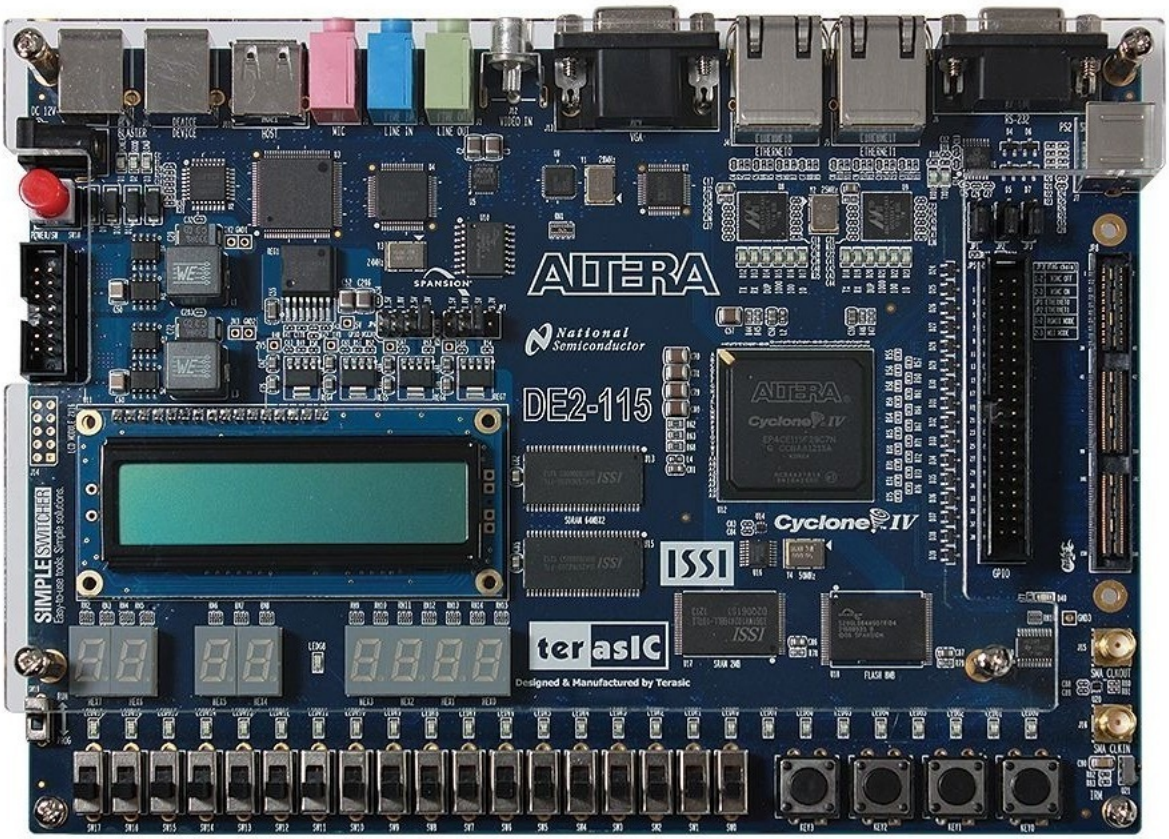
1. Obtenha uma máquina de estados de alto nível
2. Crie um bloco operacional
3. Conecte o bloco operacional a um bloco de controle
4. Obetenha a FSM do bloco de controle

O desenvolvimento da solução do problema foi baseado em blocos que facilita revisar os erros e concatenar as operações até formar uma extrutura maior.

No passo número 1 foi criado o bloco de controle em que operará sinais boleanos e para isso a implementação da Máquina de Estados Finitos do inglês - Finite State Machine (FSM) que procura capcturar o comportamento desejado do sistema. No segundo passo, o bloco operacional visa trabalhar com dados onde são inseridos entradas com vários bits a serem operadas de acordo com o bloco de controle.

Após a simulação o código em vhdl é passado para o software Quartus II em que há um ambiente de simulação e configuração para o circuito integrado Field Programmable Gate Array (FPGA) Ciclone II, a Figura 3 abaixo exibe o embarcado do FPGA.

Figura 3: Kit DE2



3.1 Bloco de Controle

O bloco de controle possui as entradas **clr controller** (bit) e **c** (bit) que inicializa a máquina de estados e adiciona moeda, respecitivamente. A máquina de estados possui quatro estados sendos eles: **início**, **esperar**, **somar** e **fornecer**.

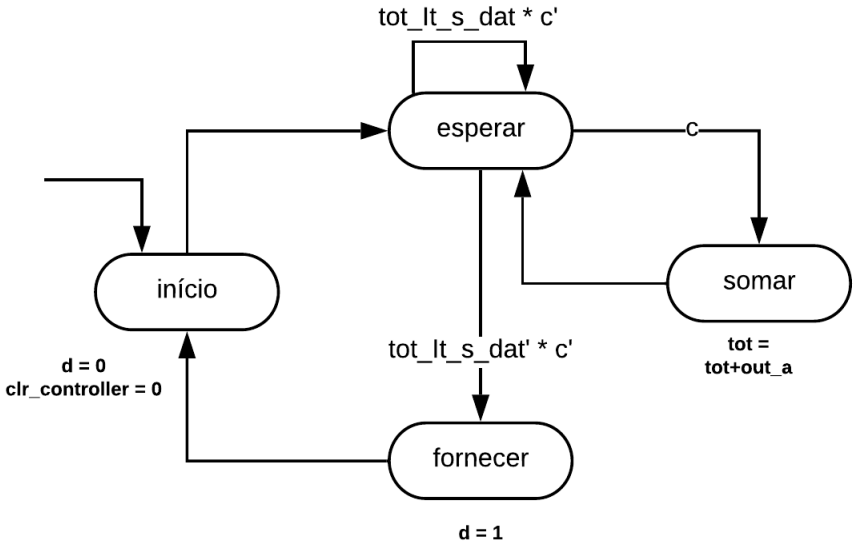
O estado de início configura as variáveis d e clr controller iguais a zero, com isso a saída d tem nível lógico 0 e os registradores de soma são inicializados com zero para que a soma possa acontecer sem problema.

Após o início, o estado próximo é esperar e nele as variáveis c que é a inserção de uma moeda e (tot It s dat) que retorna se o valor das moedas inseridas é igual ou superior ao acumulado pelo registrador de soma. Quando o bit c é igual a 1 o próximo estado é somar, caso seja 0 pode continuar no em esperar se o valor da moeda não foi ultrapassada e se passada o estado próximo é fornecer.

O estado de somar é o que constitui todo o bloco operacional nele o valor da moeda é acumulada até igual e passar do valor do produto em tot é nome do registrador. Após efetuar as somas o estado de esperar é retornado.

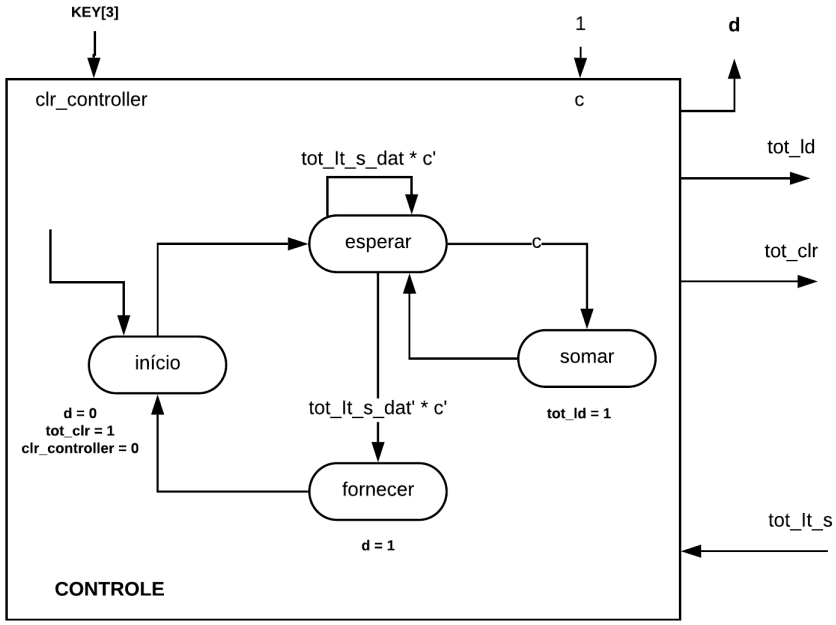
Uma vez acumulado o valor das moedas, o último estado do processo é fornecer que libera o produto e o bit d recebe 1, em seguida retorna para o ínício para que seja reconfigurado os bits iniciais.

Figura 4: Máquina de estados de alto nível



A Figura 5, mostra o bloco completo de controle que as entradas estão com setas apontado para dentro do bloco e as saídas para fora.

Figura 5: Bloco de Controle



3.2 Bloco Operacional

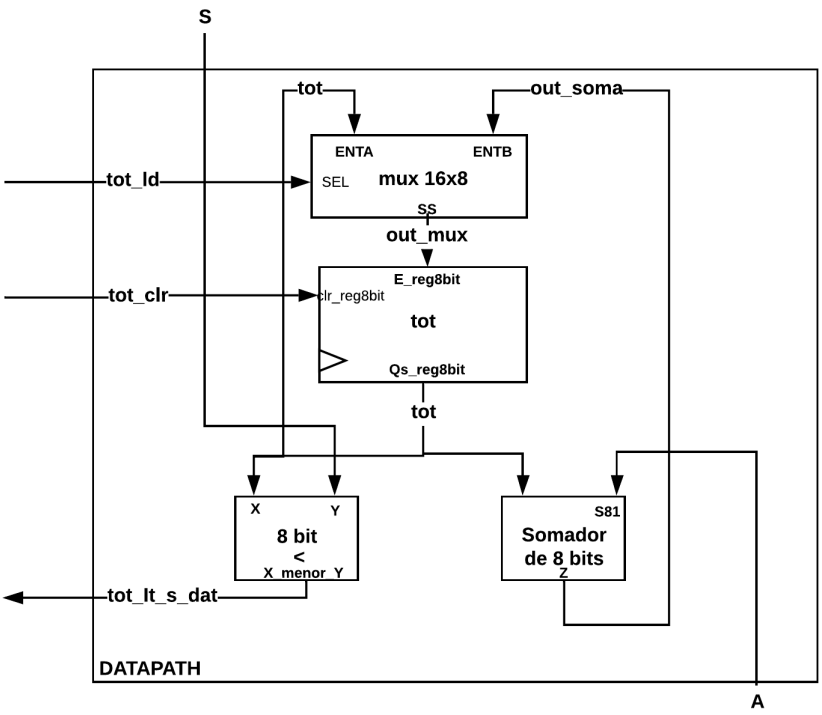
Esse processador tem predominância no trabalho de dados, pois é no datapath que ocorre toda a operação do processador. O bloco operacional é composto por um multiplexador de 16x8, um registrador de 8 bits, um somador de 8 bits e um comparador de magnitude de 8 bits. As entradas A (8 bits) corresponde o valor da moeda e S (8 bits) o valor do produto são essas variáveis que serão operadas para mandar o sinal para o bloco de controle.

A junção do multiplexador com o registrador permite o que os bits não sofram alteração com clock, pois o multiplexador se encarregará de fazer isso. No multiplexador recebe o resultado da soma e o outro canal recebe a saída do registrador.

O bloco de soma recebe o valor da moeda A e soma a saída do registrador de 8 bits e com há um incremento a cada valor da moeda até esse resultado chegar no valor do produto.

Por fim, o comparador de magnitude menor recebe o valor do produto e compara com saída do registrador e se o valor do produto for menor que o valor da saída do registrador a variável (tot It s dat) (bit) é igual a 1, caso contrário é 0 e significa que o valor da moeda foi ultrapassada que estará produto para fornecer o produto.

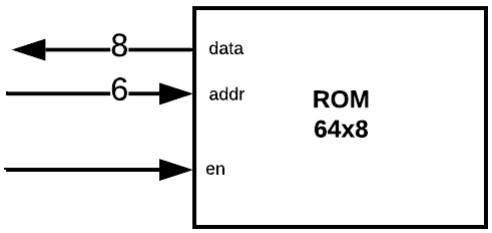
Figura 6: Modelo RTL



3.3 Mémoria ROM

A memória somente de leitura ou ROM (acrônimo em inglês de read-only memory) é um tipo de memória que permite apenas a leitura, ou seja, as suas informações são gravadas pelo fabricante uma única vez e após isso não podem ser alteradas ou apagadas, somente acessadas.

Figura 7: Extrutura da mémoria ROM



O uso da mémoria ROM é utilizada para salvar os valores da moedas que será requisitada pelo bloco operacional A (8 bits). Nela, há 64 posições de memória com 8 bits o tamanho de cada item e o en é o bit de habilitação de acesso a mémoria. O acesso do endereço de mémoria é feito por um contador de 6 bits, pois corresponde a quantidade de palavras da ROM.

4 Resultados e Discurssões

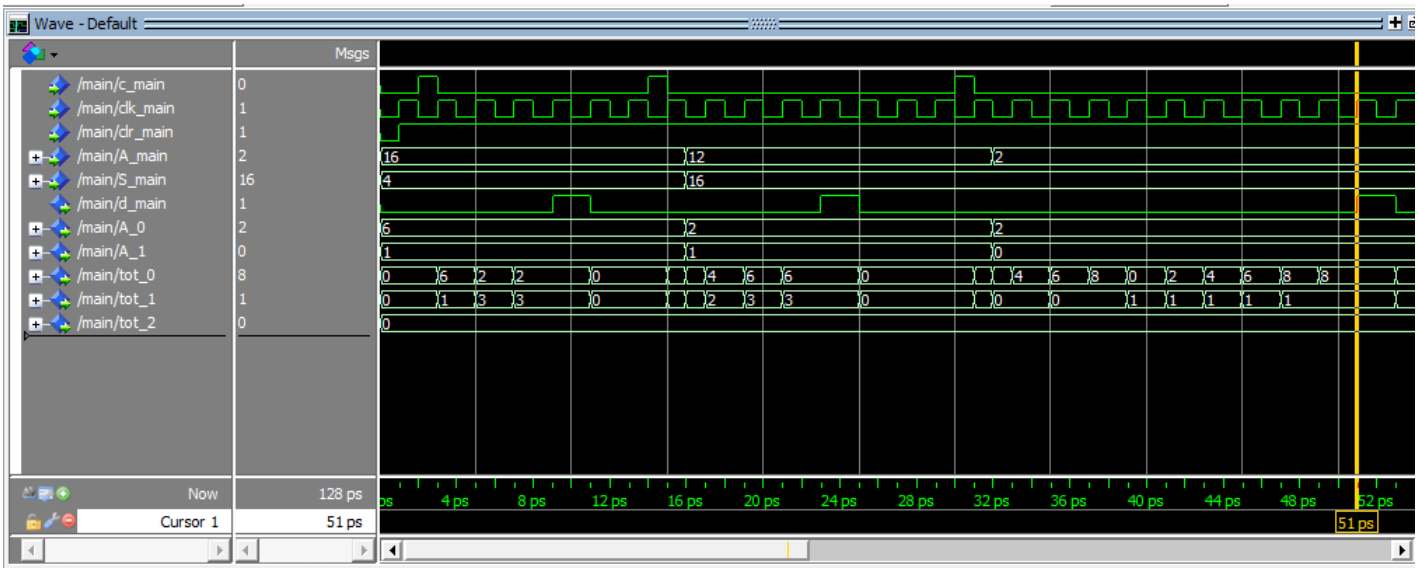
O projeto foi criado no ModelSim que permitiu gerar as formas de ondas com mais detalhes e a Figura 8 mostra um exemplo em que foi inserido três moedas de valores diferentes.

A primeira moeda de valor 16 (A main) e o produto de valor 4 (S main), neste caso foi observado que o sistema se comportou como esperado com a transições entre os estados e logo pecebido que o valor da moeda é superior ao produto em quatro pulsos de clock o produto é liberado conforme visto na Figura 8 com a variável (d main).

O segundo teste foi feito com valor de moeda igual 12 e o produto 16 e em cinco pulsos de clock o produto foi liberado, neste caso o valor da moeda teve que ser acumulado para que ficasse superior ao valor do produto. O primeiro pulso de clock corresponde ao estado início, o segundo ao esperar, o terceiro é somar onde também verificará o valor do produto, quarto é a soma de 12 mais 12 e assim superior ao valor do produto e por fim o quinto pulso é a liberação do produto.

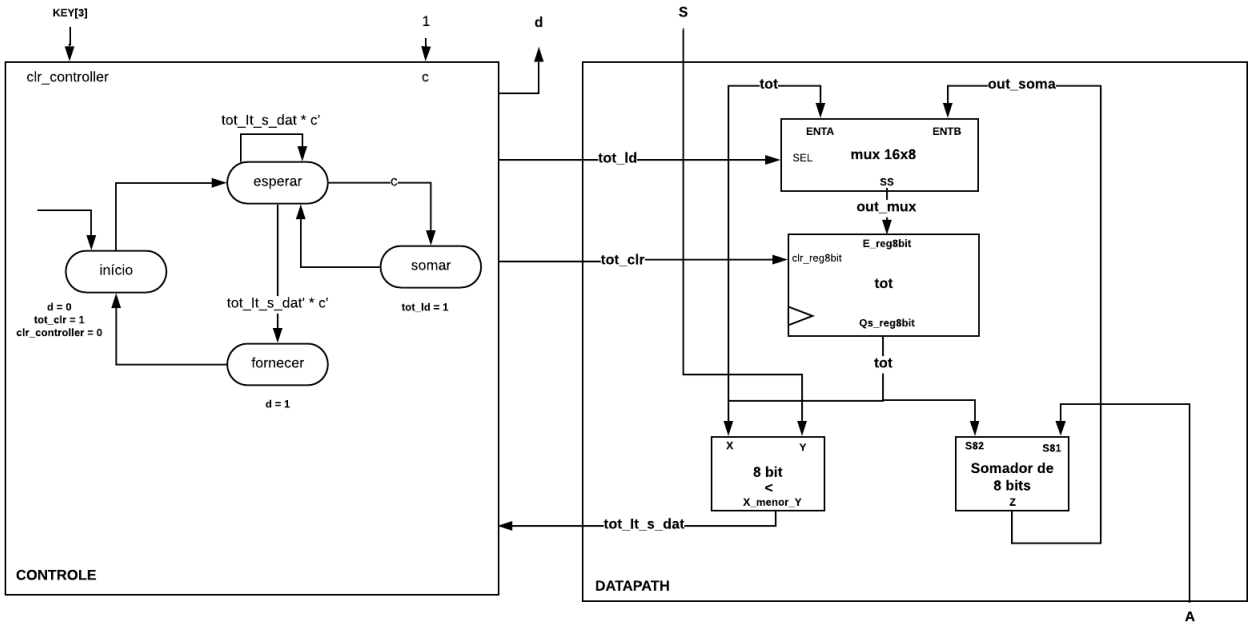
O terceiro teste o valor da moeda é igual a 2 e o produto igual a 16 e levou 11 pulsos de clock para liberar o produto e oito pulsos foram só do registrador de soma.

Figura 8: Simulação no ModelSim



A máquina de vendas é composta pela união do bloco de controle e o operacional conforme a Figura 9.

Figura 9: União dos blocos



5 Conclusão

O desenvolvimento do projeto teve uma base de simulação antes de levar a placa do kit DE2, com isso a máquina de vendas foi construída para ser apresentada nos displays de 7 segmentos e as etradas utilizando os botões push-bottom.

6 Anexo

- Código em VHD do bloco de Controle

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

entity mde_b is
port ( clk , r , c, tot_It_s: in std_logic ;
d_mde, tot_ld, tot_clr : out std_logic );
end mde_b;

architecture ckt of mde_b is

type state_type is (inicio, esperar, somar, fornecer);
signal y_present , y_next : state_type;
signal d: std_logic;

begin
process (c, tot_It_s, y_present)
begin

case y_present is

when inicio =>
y_next <= esperar;

when esperar =>
if (c = '0' and tot_It_s = '1') then y_next <= esperar;
elsif (c = '0' and tot_It_s = '0') then y_next <= fornecer;
elsif (c = '1' and tot_It_s = '1') then y_next <= somar;
elsif (c = '1' and tot_It_s = '0') then y_next <= somar; end if;

when somar =>
 if (tot_It_s = '1') then y_next <= somar;
 else y_next <= esperar; end if;

when fornecer =>
  y_next <= inicio;

end case;
end process ;

process ( clk , r)
begin
if r = '0' then
y_present <= inicio ;
elsif ( clk ' event and clk = '1') then
```

```
y_present <= y_next ;
end if ;
end process ;

d <= '1' when y_present = fornecer else '0';
tot_clr <= '0' when y_present = inicio else '1';
tot_ld <= '1' when y_present = somar else '0';

d_mde <= d;

end ckt;
```

- Código em VHD do bloco de Operacioal

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

entity datapath is

  port(a_dat,s_dat: in std_logic_vector(7 downto 0);
  tot_ld_dat, tot_clr_dat, clk_dat: in std_logic;
  tot_It_s_dat: out std_logic;
  out_reg8bit: out std_logic_vector(7 downto 0));

end;

architecture ckt_dat of datapath is

signal  out_mux, out_reg, out_somar: std_logic_vector(7 downto 0);
signal s0: std_logic;

component reg8bit is
  port(
  clk_reg8bit: in std_logic;
  clr_reg8bit: in std_logic;
  E_reg8bit: in std_logic_vector(7 downto 0);
  Qs_reg8bit: out std_logic_vector(7 downto 0));

end component;

component mux16x8 is

  port( ENTA, ENTB: in std_logic_vector(7 downto 0);
        SELECIONAR: in std_logic;
        SS : out std_logic_vector(7 downto 0));
end component;

component comparador8bit is
```

```
  port(X, Y : in std_logic_vector(7 downto 0);
      X_igual_Y, X_maior_Y, X_menor_Y: out std_logic);

end component;

component somador8bit is

  port( S81, S82: in std_logic_vector(7 downto 0);
        Ce: in std_logic;
        Z: out std_logic_vector( 7 downto 0);
        Cs: out std_logic);
end component;

begin

MUX: mux16x8 port map(out_reg, out_somar, tot_ld_dat, out_mux);
REG8: reg8bit port map(clk_dat, tot_clr_dat, out_mux, out_reg);
COMP: comparador8bit port map(out_reg, s_dat, s0);
ADDER: somador8bit port map(a_dat, out_reg, '0', out_somar);

tot_It_s_dat <= s0;
out_reg8bit <= out_mux;

end ckt_dat;
```