UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA - CT
DEPARTAMENTO DE ENGENHARIA ELÉTRICA - DEE
BACHARELADO ENGENHARIA MECATRÔNICA

RELATÓRIO SISTEMAS DIGITAIS – MÁQUINA DE VENDAS - PROJETO 01

NATAL
2020

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA - CT
DEPARTAMENTO DE ENGENHARIA ELÉTRICA - DEE
BACHARELADO ENGENHARIA MECATRÔNICA
SISTEMAS DIGITAIS

ATYSON JAIME DE SOUSA MARTINS

RELATÓRIO SISTEMAS DIGITAIS – MÁQUINA DE VENDAS - PROJETO 01

NATAL
2020

Sumário

1. Introdução **03**

2. Desenvolvimento **04**

2.1 Divisor de Clock **04**

2.2 Botão Síncrono **04**

2.3 Maquina de Vendas **06**

2.4 BINBCD – Display7Segmentos **08**

2.5 ROM **08**

2.6 Contador **09**

3. Resultados Obtidos **10**

4. Conclusão **14**

5. Referências Bibliográficas **15**

6. Anexos **16**

1. Introdução

Neste relatório estará presente todo o procedimento para escolha e construção da máquina RTL, Máquina de Vendas em VHDL para ser usado em uma FPGA em que seguiremos a arquitetura desejada pelo professor, apresentada na figura abaixo (imagem 1).

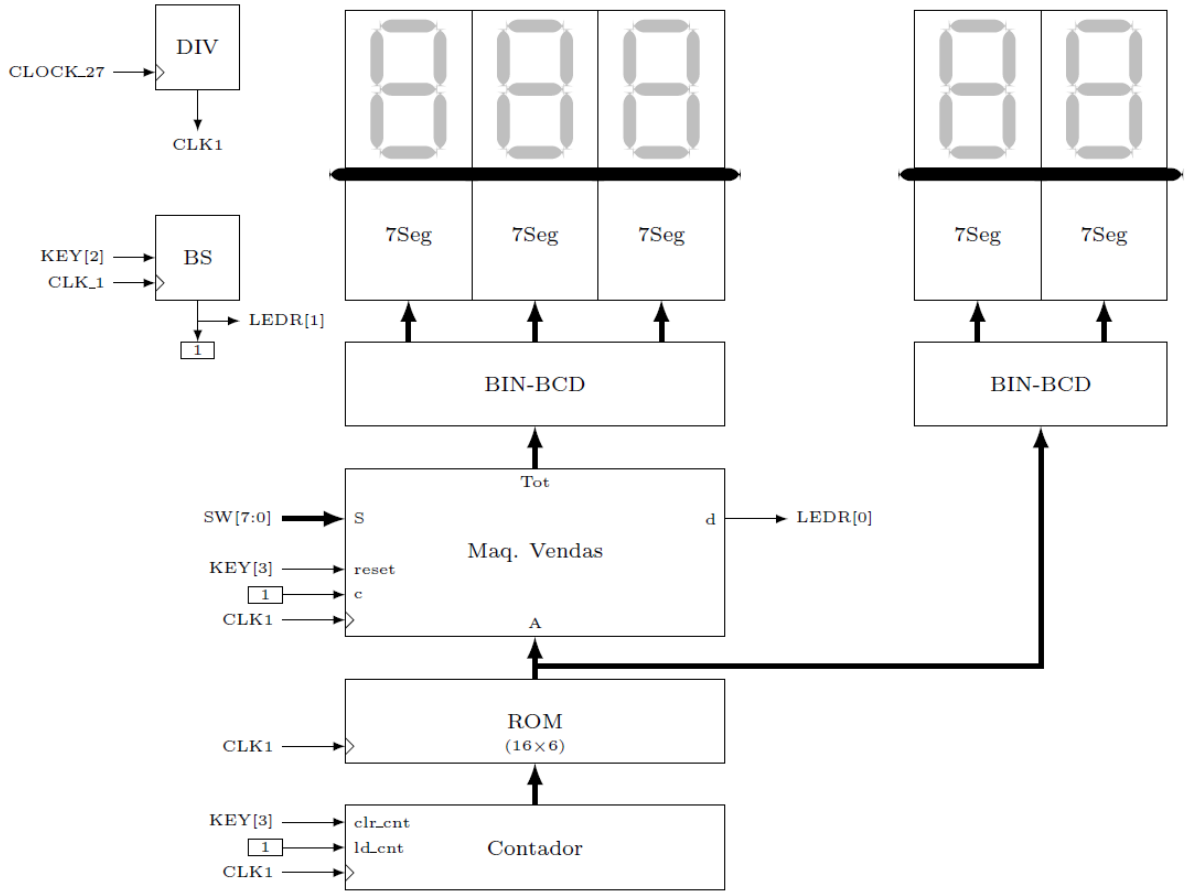

Imagem 1 - Arquitetura desejada para ser implementada

Todavia, antes de apresentar o procedimento, iremos falar sobre o que é uma Máquina RTL (Register Transfer Level); essas máquinas são métodos usados para criar processadores que são a junção de um bloco de controle (onde fica toda a parte que controla o processador) com um bloco operacional (onde fica toda a parte que mexe com dados do processador). Para se projetar uma máquina dessa é preciso seguir alguns passos: Obter a máquina de estados de nível alto; criar o bloco operacional; obter a máquina de estados finito do bloco de controle (FSM).

No seguinte estudo, foi necessária a utilização de outros componentes juntos a máquina RTL para a perfeito funcionamento desejado e adequado da estrutura referida na imagem 1.

2. Desenvolvimento

Para a realização do projeto proposto decidi fazer de forma estruturada, ou seja, destrinchar o projeto em partes e ao final juntar todas elas em um único arquivo. Desse modo, o código fica mais enxuto e mais fácil de ser testado. Assim, seguindo como roteiro a imagem 1, dividi os blocos da seguinte maneira: divisor de clock, botão síncrono, máquina de vendas (máquina de estado), BIN-BCD com display 7 segmentos, ROM e contador.

2.1. Divisor de Clock

Considerando que a FPGA utilizada para os testes possui em sua circuitaria interna um clock de 27MHz, foi necessário a criação de um bloco no qual reduza esse clock de operação da placa para um clock de operação de 10Hz.

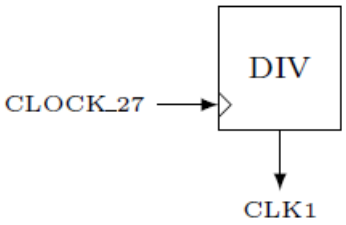

Imagem 2 – Bloco Divisor de Clock

Para a confecção desse, utilizou-se o código disponibilizado pelo professor, o qual a posteriori foi implementado em VHDL e simulado na plataforma.

2.2 – Botão Síncrono

Umas das exigências apresentadas era que o botão de acionamento utilizado no projeto, como a colocação de uma moeda na máquina, funcionasse síncrono com o clock do projeto. Dessa forma, quando o botão é apertado, o resultado ficará em nível alto por exatamente um ciclo de clock. Isso servirá para que a máquina evite o erro de considerar um aperto contínuo como múltiplos abertos no botão.

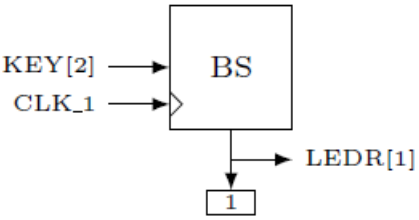

Imagem 3 – Bloco Botão Síncrono

Para a sua construção, busquei usar a máquina de estado presente no livro Sistemas Digitais – Projeto, Otimização e HDL's pelo autor por Frank Vahid.

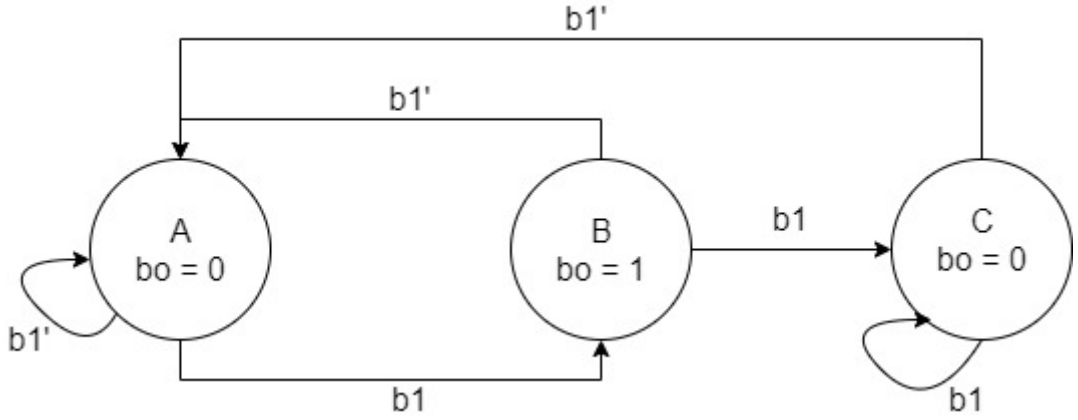

Imagem 4 – Maquina de Estados Botão Síncrono

Seu funcionamento se baseia da seguinte forma, quando a máquina está no estado A, ela espera o acionamento do botão b1. Enquanto esse botão não for acionado, ela continuará no estado A. Após o botão b1 ser pressionado, a máquina irá se dirigir ao estado B, cuja função é mandar a saída b0 para nível logico alto. Caso b1 ainda esteja acionado, a máquina vai para o estado C, caso não esteja, volta para o Estado A. Em ambos os casos, a máquina deixará a saída b0 em nível lógico baixo. No estado C a máquina continuará nesse estado caso o botão fica acionado continuamente, se não, voltará para o estado A. Desse modo, a saída b0 só estará em nível logico alto por um ciclo de clock apenas.

Por sua vez, observando essa máquina de estado, percebemos que ela é do tipo Moore, pois sua saída depende apenas do estado que a máquina se encontra. Sendo assim, realizamos a modificação do código base para máquina de estado do tipo Moore disponibilizado pelo professor para ficar de acordo com a lógica requerida, e assim implementar em VHDL e simular na plataforma.

2.3 – Maquina de Vendas

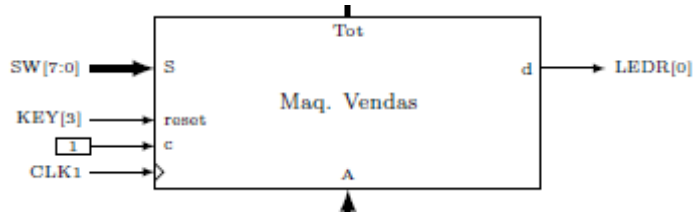

Imagem 5 – Bloco Maquina de Vendas

Utilizando também como base o projeto RTL para a máquina de vendas presente no livro Sistemas Digitais – Projeto, Otimização e HDL's pelo autor por Frank Vahid, com uma adicional de um botão de reset na máquina de estados. Em vista disso, ficamos com a seguinte formatação para a máquina de estado de alto nível.

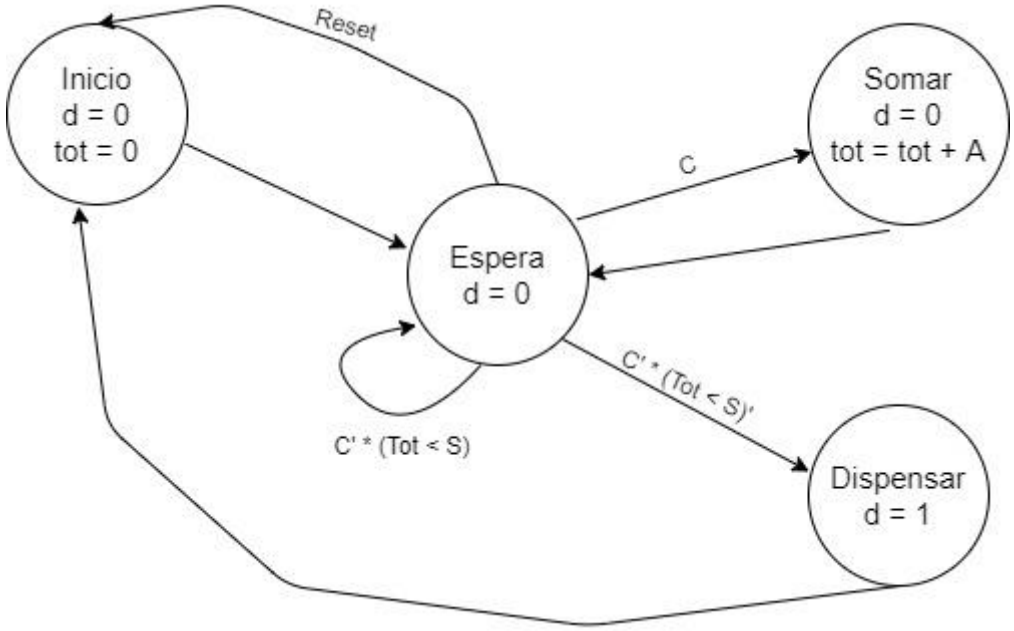

Imagem 6 – Máquina de estado de alto nível

Funcionalmente da maquina se dar da seguinte maneira:

Início: estado no qual a máquina inicia;

Espera: após o pulso de clock a maquina vai para esse estado, no qual, fica esperando o depósito de alguma moeda, enquanto o total acumulado for menor que o valor final do produto, a máquina continuará nesse estado.

Somar: caso o usuário insira alguma moeda, ele irá para esse estado, no qual, basicamente ocorrerá a soma, do valor da moeda inserida com o total acumulado na máquina. Voltando para o estado de Espera após o pulso de clock.

Dispensar: quando o total se igualar ou for maior que o valor final do produto, ele é dispensado pela máquina para o usuário. Voltando para o estado de início após o pulso de clock.

Com esse esquemático pronto, dividimos o projeto RTL em datapath (local onde fica a parte que manipula dados) e bloco de controle (local onde fica a parte que controla a máquina). Para o datapath, foi necessário criar um Somador no qual somara o total já acumulado com a entrada A (valor referente a moeda depositada), como também, a necessidade de um registrador para guarda o valor do total acumulador, além disso, criou-se um comparador para comparar o total acumulado com a entrada S (valor final do produto desejado). Como as entradas são de 8bits, o registrador, somador e acumulador criados são de 8Bits também. Uma exigência para a arquitetura final desejada, era que o total acumulado fosse mostrado em alguns displays de 7 segmentos, por isso teremos uma saída do datapath equivalente ao valor guardado no registrado. Para o bloco de controle, os sinais antes literários foram trocados por sinal de 1bit. Portanto, ficando assim, com a seguinte estrutura para o bloco de controle é o datapath.

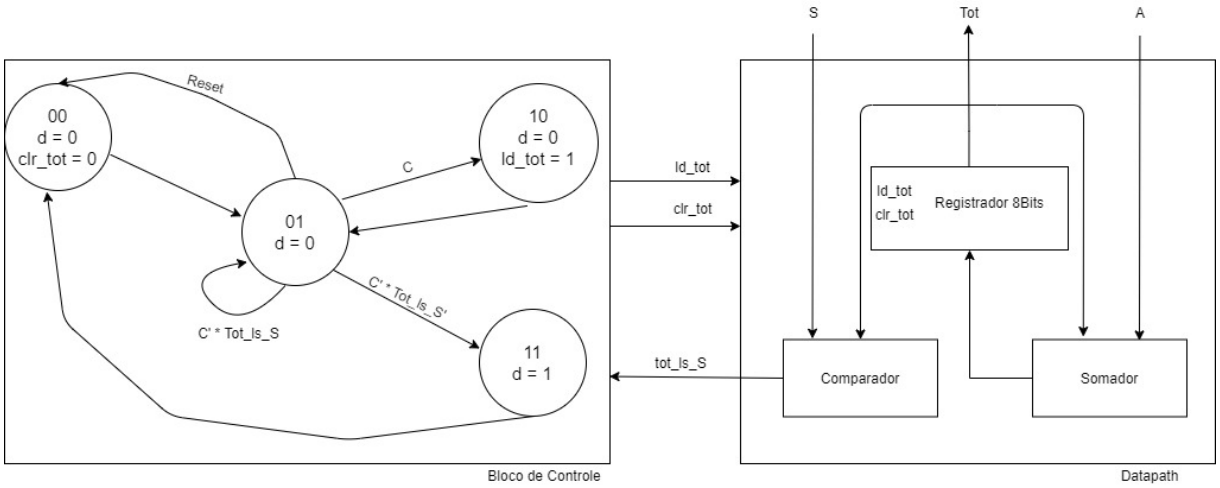

Imagem 7– Bloco de controle e datapath da máquina de vendas

Após implementação em VHDL, o mesmo foi sumulado na plataforma.

2.4 – BIN-BCD com display7Segmentos

Se olharmos para a imagem 1 novamente vemos que a exigência da arquitetura é que, a saída tanto da ROM, como do registrador localizada dentro da maquina de vendas, tenha seu resultado mostrado em um display 7 segmentos. Para isso ocorrer, precisou-se primeiramente fazer uma conversão de binário para BCD.

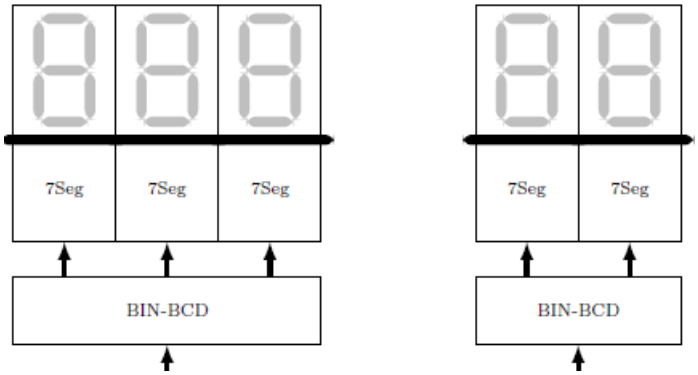

Imagem 8 – Bloco BINBCD – Display7Segmentos

Os códigos utilizados para fazer todas essas partes, tanto para a conversão de binário para BCD, quanto BCD para display 7 segmentos, foram retirados das atividades antes realizadas na matéria de Circuitos Digitais, ah qual, é requisito obrigatório para se pagar Sistemas Digitais.

2.5 – ROM

O bloco de memoria vista na arquitetura da imagem 1, funcionará basicamente como um guardado dos valores das moedas possíveis para a maquina de vendas, por isso que sua saída vai tanto para entrada da maquina de vendas (A) vista na imagem 5, quando para o display de 7 segmentos (mostrando o valor da moeda) visto na imagem 8. Como no projeto não precisamos gravar nada em alguma memoria, só ler o que está salvo nela, a ROM é a melhor escolha, já que, pela sua definição a ROM (*Read-Only Memory)* é uma memória apenas de leitura.

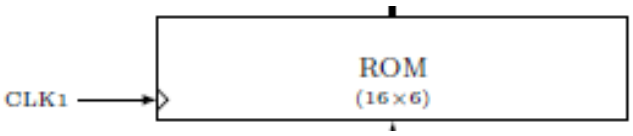

Imagem 9 – Bloco da ROM

Para a implementação da mesma em VHDL na plataforma, utilizou-se do passo a passo em pdf disponibilizado pelo professor por meio da turma virtual.

2.6 – Contador

No projeto, o contador foi empregado para acessar a posição da ROM e assim, pegar o valor que será usado como moeda na máquina de vendas.

Imagem 10 – Bloco Contador

Dado que o tamanho da ROM é possível ter 64 posições diferentes, foi necessário criar um contador de 6Bits, pois quando fazemos $2^6 = 64$. Internamente, seu funcionamento se dar por flip flop JK juntos, com pequena diferença empregada devido a imposição quista na arquitetura final, no qual, foi a colocação de um load e um clear no contador. Desse modo, o contador só funcionará quando o botão for apertado, ou seja, quando o usuário inserir uma moeda. Quando isso ocorrer, o valor que está no contador ditará a posição da ROM no qual terá a quantia X para aquela moeda.

3. Resultados Obtidos

Após testes exaustivos em simulador o código foi posteriormente testado em FPGA, utilizando interfaces de saída de dados destinados aos displays da placa, aliado a um código divisor de clock utilizado para reduzir o clock de operação da placa, 27 MHz, para um clock de operação de 10Hz, previamente citado no projeto. Desse modo, começamos com a seguinte interface no FPGA:

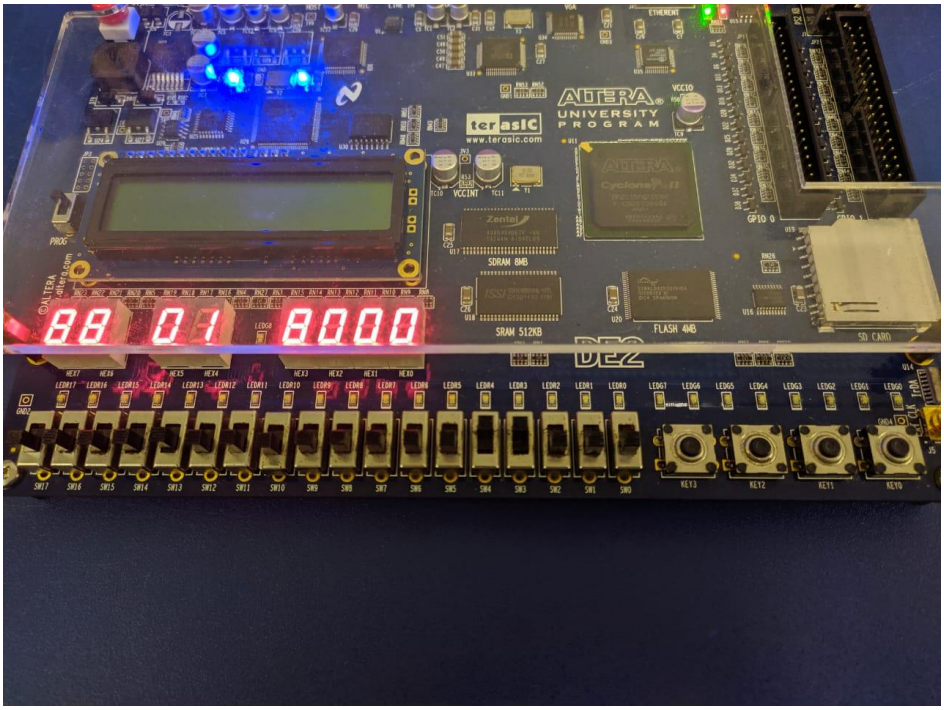

Imagem 11 – momento inicial do teste em FPGA

Como podemos ver na imagem 11, os 2 displays que se encontram no meio (o qual possuem valor 01) mostram o valor de entrada da moeda, ou seja, valor guardado na ROM. Já os 3 displays da ponta (o qual possuem valor 000 inicialmente) mostra o valor do total guardado na máquina. Para fiz didáticos e de teste, colocamos a entrada SW igual a ‘00011000’ o que gera um produto com um valor de 24.

Quando apertamos o botão KEY2 aciona-se o bloco do botão síncrono fazendo o load do projeto funcionar apenas por um ciclo de clock, quando isso ocorre o led LEDR1 da FPGA acende.

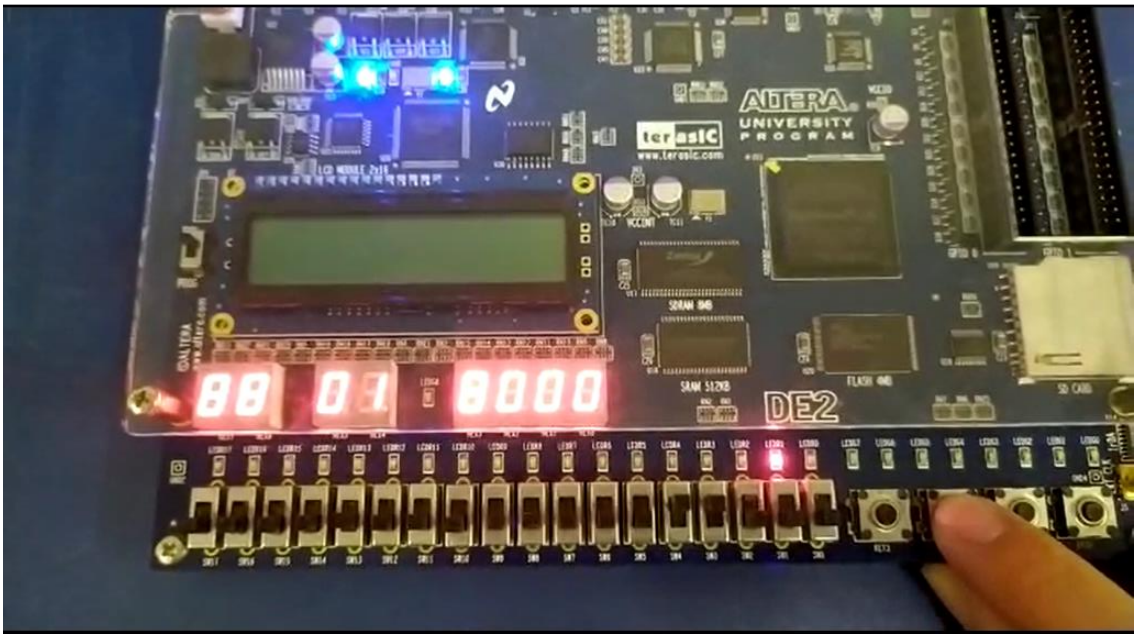

Imagem 12 – Funcionalmente do botão síncrono

Após o led se apagar, os 3 displays da ponta mostraram o resultado da entrada do valor da moeda presente nos 2 displays do meio somado com o total presente na máquina guardado anteriormente nos 3 displays da ponta. Outrossim, os 2 displays do meio agora apresentaram o próximo valor de moeda, pois a ROM está mostrando outra posição internamente devido ao contador ter sofrido load com o acionamento do botão.

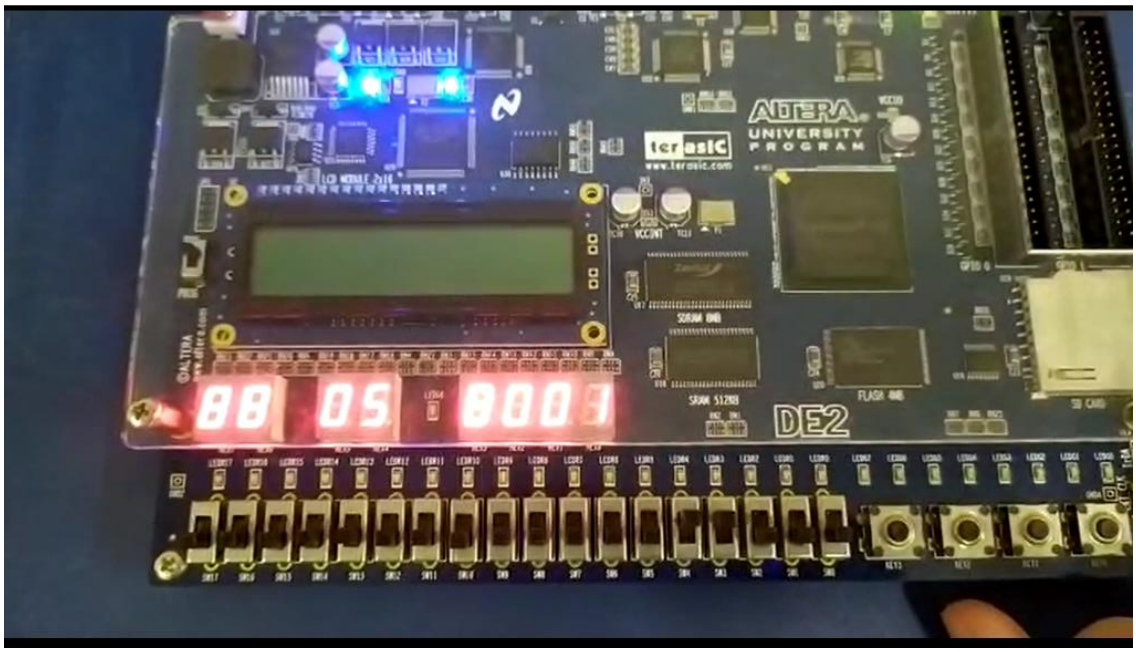

Imagem 13 – Troca dos valores dos displays 7 segmentos

Continuando esse procedimento do acionamento do botão, chegaremos ao momento onde o total presente na maquina é maior ou igual ao valor anteriormente colocado para o preço do produto. Assim, quando isso ocorrer, o led LEDR0 da FPGA é acesso, informando ao usuário que o produto foi dispensado pela máquina.

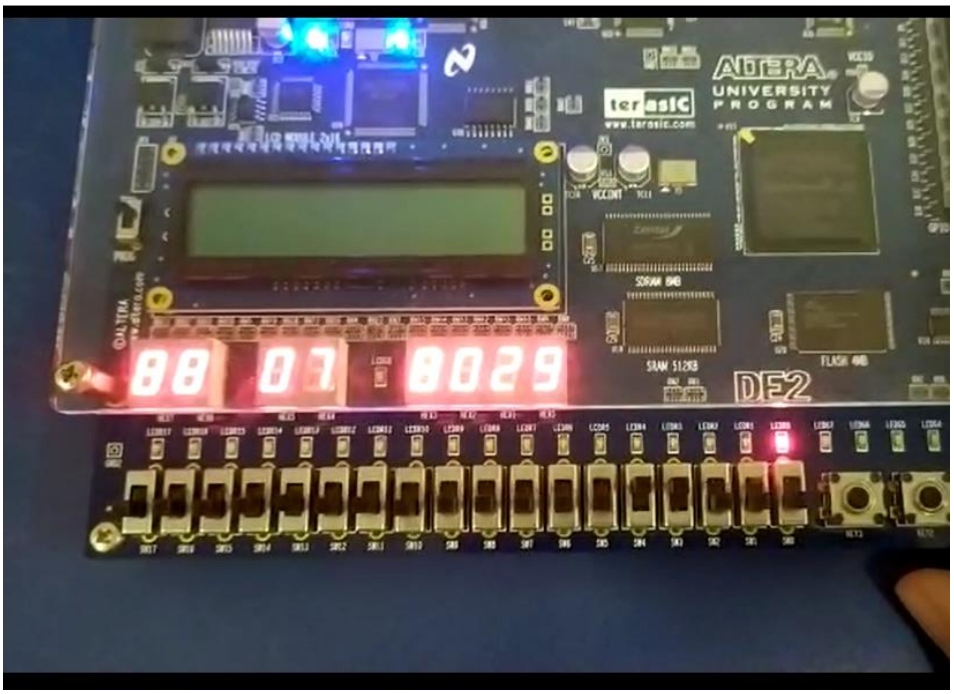

Imagem 14 – produto dispensado pela maquina

Quando o led LEDR0 se apaga, os 3 displays voltaram ao seu estado inicial mostrando zero, pois dado que o produto foi dispensado não tem mais valores guardado na máquina e o total igualou-se a 0 novamente.

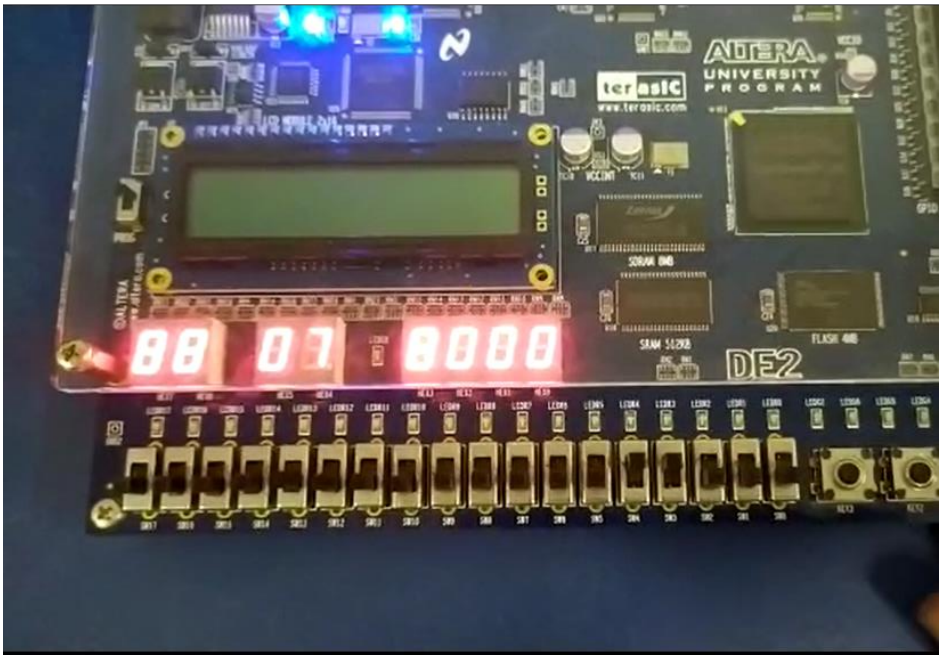

Imagem 15 – Total zerando após dispensa do produto

Adicionalmente, ao apertamos no botão KEY3 da FPGA independente do local que a maquina de vendas esteja, ou a posição que a ROM se encontra, todo o projeto é reset. Voltando assim, as suas configurações iniciais, presente na imagem 11 para esse respectivo teste.

Portanto, todos os testes físicos retornaram os resultados obtidos em simulador, constituindo um projeto implementável em FPGA.

4. Conclusão

Nos dias atuais, o projeto circuitos aumentaram significativamente. Como também, a utilização de memória seja RAM ou ROM junto das mesmas. Para tal, a utilização de maquinas RTL para a projeção do circuito tornou sua construção e codificação mais fáceis de serem entendidas e realizadas.

Nesse documento, foi exposto o passo a passo de como projetar uma máquina RTL para uma máquina de vendas, com uma memória ROM para salvar os tipos de moeda. Ademais, foi mostrado os resultados obtidos quando simulado o projeto utilizando a FPGA Ciclone II.

Disto podemos retirar as seguintes conclusões, dado a quantidade de componentes exigidos para a máquina de vendas foi uma importante revisão dos conceitos visto anteriormente na matéria de Circuitos Digitais que agora serão de extrema importância para o que iremos enfrentar pela frente em Sistemas Digitais. Como também, o fato de utilizamos a FPGA para manipular e olhar nosso código funcionando como se fosse uma situação real, nos mostra a importância da matéria para os desafios que iremos encontrar no futuro da nossa profissão.

5. Referências Bibliográficas

D’AMORE, R.; VHDL: descrição e síntese de circuitos digitais. 2.ed. Rio de Janeiro: LTC, 2015. 292p.

STALLINGS, William. Arquitetura e organização de computadores. 8. ed. São Paulo: Pearson Prentice Hall, 2010.

VAHID, F.; Sistemas Digitais: projeto, otimização e HSLs; tradução Anatólio Laschuk. – Porto Alegre: Artmed, 2008. 560p.

```
1   library ieee ;
2   use ieee.std_logic_1164.all;
3
4   entity projeto01_MV is
5      port (clock_27, KEY2, KEY3: in std_logic;
6            SW: in std_logic_vector(7 downto 0);
7            LEDR0,LEDR1: out std_logic;
8            LHEX2,LHEX1,LHEX0,LHEX5,LHEX4: out std_logic_vector(6 downto 0));
9   end projeto01_MV;
10
11  architecture ckt of projeto01_MV is
12     component maquinaVendas is
13        port (CLK_mv, C, reset: in std_logic;
14              S, A: in std_logic_vector(7 downto 0);
15              D: out std_logic;
16              saida_mv: out std_logic_vector(1 downto 0);
17              Tot_mv: out std_logic_vector(7 downto 0));
18     end component;
19     component romMV IS
20        PORT(address: IN STD_LOGIC_VECTOR (5 DOWNTO 0);
21             clock: IN STD_LOGIC  := '1';
22             q: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
23     END component;
24     component divisorClock is
25        port ( clk_in : in std_logic ;
26               clk_out : out std_logic );
27     end component ;
28     component contador6Bits is
29        port ( clk_c6B,ld_c6B,clr_c6B: in std_logic;
30               out_c6B: out std_logic_vector(5 downto 0));
31     end component;
32     component botaoSincrono is
33        port (clk, b_in: in std_logic ;
34              b_out: out std_logic );
35     end component;
36     component bitToBcdToD7Seg is
37        port (SW_in: in std_logic_vector(7 downto 0);
38              HEX0,HEX1,HEX2: out std_logic_vector(6 downto 0));
39     end component;
40
41     signal CLK_1, saida_botao, saida_MV, ld_reverso, clr_reverso: std_logic;
42     signal saida_SS: std_logic_vector(1 downto 0);
43     signal saida_contador: std_logic_vector(5 downto 0);
44     signal entrada_A_mv, total_sum_mv: std_logic_vector(7 downto 0);
45     signal saida_hex2, saida_hex1, saida_hex0, saida_hex5, saida_hex4,lixo:
   std_logic_vector(6 downto 0);
46
47     begin
48
49        ld_reverso <= not KEY2;
50        clr_reverso <= not KEY3;
51
52        DivClock: divisorClock port map(
53           clk_in => clock_27,
54           clk_out => CLK_1);
55
56        BS: botaoSincrono port map(
57           clk => CLK_1,
58           b_in => ld_reverso,
59           b_out => saida_botao);
60
61        ROM: romMV port map (
62           clock => CLK_1,
63           address => saida_contador,
64           q => entrada_A_mv);
65
```

```
66          Cont: contador6Bits port map(
67             clk_c6B => CLK_1,
68             ld_c6B => saida_botao,
69             clr_c6B => clr_reverso,
70             out_c6B => saida_contador);
71
72          MV: maquinaVendas port map(
73             CLK_mv => CLK_1,
74             C => saida_botao,
75             reset => clr_reverso,
76             S => SW,
77             A => entrada_A_mv,
78             D => saida_MV,
79             saida_mv => saida_SS,
80             Tot_mv => total_sum_mv);
81
82          LedTotalMV: bitToBcdToD7Seg port map(
83             SW_in => total_sum_mv,
84             HEX0 => saida_hex0,
85             HEX1 => saida_hex1,
86             HEX2 => saida_hex2);
87
88          LedROM: bitToBcdToD7Seg port map(
89             SW_in => entrada_A_mv,
90             HEX0 => saida_hex4,
91             HEX1 => saida_hex5,
92             HEX2 => lixo);
93
94          LEDR0 <= saida_MV;
95          LEDR1 <= saida_botao;
96          LHEX2 <= saida_hex2;
97          LHEX1 <= saida_hex1;
98          LHEX0 <= saida_hex0;
99          LHEX4 <= saida_hex4;
100         LHEX5 <= saida_hex5;
101   end ckt ;
```

```
1   library ieee ;
2   use ieee.std_logic_1164.all;
3   entity divisorClock is
4      port ( clk_in : in std_logic ;
5             clk_out : out std_logic );
6   end divisorClock ;
7   architecture ckt of divisorClock is
8      signal ax : std_logic ;
9      begin
10        process ( clk_in )
11           variable cnt: integer range 0 to 13500000 := 0;
12           begin
13              if ( rising_edge ( clk_in )) then
14                 if (cnt =13500000) then
15                    cnt := 0;
16                    ax <= not ax;
17                 else
18                    cnt := cnt +1;
19                 end if;
20              end if;
21        end process ;
22        clk_out <= ax;
23  end ckt;
```

```
1   library ieee ;
2   use ieee.std_logic_1164.all;
3   entity botaoSincrono is
4      port (clk, b_in: in std_logic ;
5            b_out: out std_logic );
6   end botaoSincrono ;
7
8   architecture ckt of botaoSincrono is
9      type state_type is (E1, E2, E3);
10     signal y_present , y_next : state_type ;
11     begin
12        process (b_in, y_present )
13        begin
14           case y_present is
15              when E1 =>
16                 if b_in = '0' then y_next <= E1;
17                 else y_next <= E2; end if;
18              when E2 =>
19                 if b_in = '0' then y_next <= E1;
20                 else y_next <= E3; end if;
21              when E3 =>
22                 if b_in = '0' then y_next <= E1;
23                 else y_next <= E3; end if;
24           end case ;
25        end process ;
26        process (clk)
27        begin
28           if (clk'event and clk = '1') then
29              y_present <= y_next;
30           end if;
31        end process;
32        b_out <= '1' when y_present = E2 else '0';
33  end ckt;
```

```
1  -- megafunction wizard: %ROM: 1-PORT%
2  -- GENERATION: STANDARD
3  -- VERSION: WM1.0
4  -- MODULE: altsyncram
5
6  -- ============================================================
7  -- File Name: romMV.vhd
8  -- Megafunction Name(s):
9  --          altsyncram
10 --
11 -- Simulation Library Files(s):
12 --          altera_mf
13 -- ============================================================
14 -- ************************************************************
15 -- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
16 --
17 -- 13.0.0 Build 156 04/24/2013 SJ Web Edition
18 -- ************************************************************
19
20
21 --Copyright (C) 1991-2013 Altera Corporation
22 --Your use of Altera Corporation's design tools, logic functions
23 --and other software and tools, and its AMPP partner logic
24 --functions, and any output files from any of the foregoing
25 --(including device programming or simulation files), and any
26 --associated documentation or information are expressly subject
27 --to the terms and conditions of the Altera Program License
28 --Subscription Agreement, Altera MegaCore Function License
29 --Agreement, or other applicable license agreement, including,
30 --without limitation, that your use is for the sole purpose of
31 --programming logic devices manufactured by Altera and sold by
32 --Altera or its authorized distributors.  Please refer to the
33 --applicable agreement for further details.
34
35
36 LIBRARY ieee;
37 USE ieee.std_logic_1164.all;
38
39 LIBRARY altera_mf;
40 USE altera_mf.all;
41
42 ENTITY romMV IS
43    PORT
44    (
45       address     : IN STD_LOGIC_VECTOR (5 DOWNTO 0);
46       clock    : IN STD_LOGIC  := '1';
47       q     : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
48    );
49 END romMV;
50
51
52 ARCHITECTURE SYN OF rommv IS
53
54    SIGNAL sub_wire0  : STD_LOGIC_VECTOR (7 DOWNTO 0);
55
56
57
58    COMPONENT altsyncram
59    GENERIC (
60       clock_enable_input_a    : STRING;
61       clock_enable_output_a      : STRING;
62       init_file      : STRING;
63       intended_device_family     : STRING;
64       lpm_hint    : STRING;
65       lpm_type    : STRING;
66       numwords_a     : NATURAL;
```

```
67          operation_mode    : STRING;
68          outdata_aclr_a    : STRING;
69          outdata_reg_a     : STRING;
70          widthad_a       : NATURAL;
71          width_a     : NATURAL;
72          width_byteena_a       : NATURAL
73       );
74       PORT (
75             address_a   : IN STD_LOGIC_VECTOR (5 DOWNTO 0);
76             clock0   : IN STD_LOGIC ;
77             q_a   : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
78       );
79       END COMPONENT;
80
81    BEGIN
82       q    <= sub_wire0(7 DOWNTO 0);
83
84       altsyncram_component : altsyncram
85       GENERIC MAP (
86          clock_enable_input_a => "BYPASS",
87          clock_enable_output_a => "BYPASS",
88          init_file => "romMV.mif",
89          intended_device_family => "Cyclone II",
90          lpm_hint => "ENABLE_RUNTIME_MOD=NO",
91          lpm_type => "altsyncram",
92          numwords_a => 64,
93          operation_mode => "ROM",
94          outdata_aclr_a => "NONE",
95          outdata_reg_a => "UNREGISTERED",
96          widthad_a => 6,
97          width_a => 8,
98          width_byteena_a => 1
99       )
100      PORT MAP (
101         address_a => address,
102         clock0 => clock,
103         q_a => sub_wire0
104      );
105
106
107
108   END SYN;
109
110   -- ============================================================
111   -- CNX file retrieval info
112   -- ============================================================
113   -- Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
114   -- Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
115   -- Retrieval info: PRIVATE: AclrByte NUMERIC "0"
116   -- Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
117   -- Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
118   -- Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
119   -- Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
120   -- Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
121   -- Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
122   -- Retrieval info: PRIVATE: Clken NUMERIC "0"
123   -- Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
124   -- Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
125   -- Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
126   -- Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone II"
127   -- Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
128   -- Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
129   -- Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
130   -- Retrieval info: PRIVATE: MIFfilename STRING "romMV.mif"
131   -- Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "64"
132   -- Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
```

```
133   -- Retrieval info: PRIVATE: RegAddr NUMERIC "1"
134   -- Retrieval info: PRIVATE: RegOutput NUMERIC "0"
135   -- Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
136   -- Retrieval info: PRIVATE: SingleClock NUMERIC "1"
137   -- Retrieval info: PRIVATE: UseDQRAM NUMERIC "0"
138   -- Retrieval info: PRIVATE: WidthAddr NUMERIC "6"
139   -- Retrieval info: PRIVATE: WidthData NUMERIC "8"
140   -- Retrieval info: PRIVATE: rden NUMERIC "0"
141   -- Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
142   -- Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
143   -- Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
144   -- Retrieval info: CONSTANT: INIT_FILE STRING "romMV.mif"
145   -- Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone II"
146   -- Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
147   -- Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
148   -- Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "64"
149   -- Retrieval info: CONSTANT: OPERATION_MODE STRING "ROM"
150   -- Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
151   -- Retrieval info: CONSTANT: OUTDATA_REG_A STRING "UNREGISTERED"
152   -- Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "6"
153   -- Retrieval info: CONSTANT: WIDTH_A NUMERIC "8"
154   -- Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
155   -- Retrieval info: USED_PORT: address 0 0 6 0 INPUT NODEFVAL "address[5..0]"
156   -- Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
157   -- Retrieval info: USED_PORT: q 0 0 8 0 OUTPUT NODEFVAL "q[7..0]"
158   -- Retrieval info: CONNECT: @address_a 0 0 6 0 address 0 0 6 0
159   -- Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
160   -- Retrieval info: CONNECT: q 0 0 8 0 @q_a 0 0 8 0
161   -- Retrieval info: GEN_FILE: TYPE_NORMAL romMV.vhd TRUE
162   -- Retrieval info: GEN_FILE: TYPE_NORMAL romMV.inc FALSE
163   -- Retrieval info: GEN_FILE: TYPE_NORMAL romMV.cmp FALSE
164   -- Retrieval info: GEN_FILE: TYPE_NORMAL romMV.bsf FALSE
165   -- Retrieval info: GEN_FILE: TYPE_NORMAL romMV_inst.vhd FALSE
166   -- Retrieval info: LIB_FILE: altera_mf
167
```

```
1   library ieee ;
2   use ieee.std_logic_1164.all;
3
4   entity maquinaVendas is
5      port (CLK_mv, C, reset: in std_logic;
6            S, A: in std_logic_vector(7 downto 0);
7            D: out std_logic;
8            saida_mv: out std_logic_vector(1 downto 0);
9            Tot_mv: out std_logic_vector(7 downto 0));
10  end maquinaVendas;
11
12  architecture ckt of maquinaVendas is
13     component blocoDeControleMV is
14        port (clk_mv , rst_mv , c_mv, tot_ld: in std_logic ;
15              d_mv, load_tot, clr_tot: out std_logic;
16              saida: out std_logic_vector(1 downto 0));
17     end component;
18     component datapathMV is
19        port (clk_dmv , clr_total , ld_total: in std_logic;
20              S_mv, A_mv: in std_logic_vector(7 downto 0);
21              total_ld: out std_logic;
22              total_mv: out std_logic_vector(7 downto 0));
23     end component;
24
25     signal result_total: std_logic_vector(7 downto 0);
26     signal result_comp_lt, result_despacho, sinal_somar_tot, clear_tot: std_logic;
27     signal saida_mc: std_logic_vector(1 downto 0);
28     begin
29        BlocodeControle: blocoDeControleMV port map(
30           clk_mv => CLK_mv,
31           rst_mv => reset,
32           c_mv => C,
33           tot_ld => result_comp_lt,
34           d_mv => result_despacho,
35           load_tot => sinal_somar_tot,
36           clr_tot => clear_tot,
37           saida => saida_mc);
38
39        Datapath: datapathMV port map(
40           clk_dmv => CLK_mv,
41           clr_total => clear_tot,
42           ld_total => sinal_somar_tot,
43           S_mv => S,
44           A_mv => A,
45           total_ld => result_comp_lt,
46           total_mv => result_total);
47
48        D <= result_despacho;
49        Tot_mv <= result_total;
50        saida_mv <= saida_mc;
51  end ckt ;
```

```
1   library ieee ;
2   use ieee.std_logic_1164.all;
3
4   entity datapathMV is
5      port (clk_dmv , clr_total , ld_total: in std_logic;
6            S_mv, A_mv: in std_logic_vector(7 downto 0);
7            total_ld: out std_logic;
8            total_mv: out std_logic_vector(7 downto 0));
9   end datapathMV;
10
11  architecture ckt of datapathMV is
12     component Comparador_8Bits is
13       port (eA8,eB8: in std_logic_vector(7 downto 0);
14             AeqB8,AltB8,AgtB8: out std_logic);
15     end component;
16     component reg8Bits is
17        port (clk,preSet,clr,load: in std_logic;
18              d: in std_logic_vector(7 downto 0);
19              q : out std_logic_vector(7 downto 0));
20     end component;
21     component SUM_8Bits is
22          port (A8_in,B8_in: in std_logic_vector(7 downto 0);
23                C8_in: in std_logic;
24                S8_out: out std_logic_vector(7 downto 0);
25                C8_out: out std_logic);
26     end component;
27
28     signal result_somador, result_tot: std_logic_vector(7 downto 0);
29     signal lixo_sum, result_comp_lt: std_logic;
30     signal lixo_comp: std_logic_vector(1 downto 0);
31     begin
32        Reg: reg8Bits port map(
33           clk => clk_dmv,
34           preSet => '1',
35           clr => not clr_total,
36           load => ld_total,
37           d => result_somador,
38           q => result_tot);
39
40        SUM: SUM_8Bits port map(
41           A8_in => result_tot,
42           B8_in => A_mv,
43           C8_in => '0',
44           S8_out => result_somador,
45           C8_out => lixo_sum);
46
47        Comp: Comparador_8Bits port map(
48           eA8 => result_tot,
49           eB8 => S_mv,
50           AeqB8 => lixo_comp(1),
51           AltB8 => result_comp_lt,
52           AgtB8 =>lixo_comp(0));
53
54        total_mv <= result_tot;
55        total_ld <= result_comp_lt;
56  end ckt ;
```

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

entity blocoDeControleMV is
   port (clk_mv , rst_mv , c_mv, tot_ld: in std_logic ;
         d_mv, load_tot, clr_tot: out std_logic;
         saida: out std_logic_vector(1 downto 0));
end blocoDeControleMV;

architecture ckt of blocoDeControleMV is
   type st is (E1, E2, E3, E4);
   signal estado : st;
   begin
      process (clk_mv , rst_mv)
      begin
         if rst_mv = '1' then
            estado <= E1 ;
         elsif (clk_mv'event and clk_mv ='1') then
            case estado is
               when E1 =>
                  estado <= E2;
               when E2 =>
                  if c_mv='1' then estado <= E3;
                  elsif tot_ld='0' then estado <= E4;
                  else estado <= E2;
                  end if;
               when E3 =>
                  estado <= E2;
               when E4 =>
                  estado <= E1;
            end case ;
         end if;
      end process;
      d_mv <= '1' when estado = E4 else '0';
      load_tot <= '1' when estado = E3 else '0';
      clr_tot <= '1' when estado = E1 else '0';
      with estado select
         saida <= "00" when E1 ,
         "01" when E2 ,
         "11" when E4 ,
         "10" when E3 ;
end ckt ;
```

```
1   library ieee ;
2   use ieee.std_logic_1164.all;
3
4   entity Comparador is
5     port (in_gt, in_eq, in_lt,a,b: in std_logic;
6           out_eq,out_lt,out_gt: out std_logic);
7   end Comparador;
8
9   architecture ckt of Comparador is
10    begin
11      out_gt <= in_gt OR (in_eq AND a AND (NOT b));
12      out_lt <= in_lt OR (in_eq AND (NOT a) AND b);
13      out_eq <= in_eq AND (a XNOR b);
14  end ckt;
```

```
1   library ieee ;
2   use ieee.std_logic_1164.all;
3
4   entity Comparador_4Bits is
5     port (eA,eB: in std_logic_vector(3 downto 0);
6           gt,lt,eq: in std_logic;
7           AeqB,AltB,AgtB: out std_logic);
8   end Comparador_4Bits;
9
10  architecture ckt of Comparador_4Bits is
11
12    component Comparador is
13      port (in_gt, in_eq, in_lt,a,b: in std_logic;
14            out_eq,out_lt,out_gt: out std_logic);
15     end component;
16
17    signal saida_gt,saida_lt,saida_eq:std_logic_vector(3 downto 0);
18    begin
19    Comp1:Comparador port map(
20          in_gt => gt,
21          in_lt => lt,
22          in_eq => eq,
23          a => eA(3),
24          b => eB(3),
25          out_eq => saida_eq(3),
26          out_gt => saida_gt(3),
27          out_lt => saida_lt(3));
28
29    Comp2:Comparador port map(
30          in_gt => saida_gt(3),
31          in_lt => saida_lt(3),
32          in_eq => saida_eq(3),
33          a => eA(2),
34          b => eB(2),
35          out_eq => saida_eq(2),
36          out_gt => saida_gt(2),
37          out_lt => saida_lt(2));
38
39    Comp3:Comparador port map(
40          in_gt => saida_gt(2),
41          in_lt => saida_lt(2),
42          in_eq => saida_eq(2),
43          a => eA(1),
44          b => eB(1),
45          out_eq => saida_eq(1),
46          out_gt => saida_gt(1),
47          out_lt => saida_lt(1));
48
49    Comp4:Comparador port map(
50          in_gt => saida_gt(1),
51          in_lt => saida_lt(1),
52          in_eq => saida_eq(1),
53          a => eA(0),
54          b => eB(0),
55          out_eq => saida_eq(0),
56          out_gt => saida_gt(0),
57          out_lt => saida_lt(0));
58
59    AeqB <= saida_eq(0);
60    AltB <= saida_lt(0);
61    AgtB <= saida_gt(0);
62  end ckt;
63
64
65
```

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

entity Comparador_8Bits is
  port (eA8,eB8: in std_logic_vector(7 downto 0);
        AeqB8,AltB8,AgtB8: out std_logic);
end Comparador_8Bits;

architecture ckt of Comparador_8Bits is

  component Comparador_4Bits is
    port (eA,eB: in std_logic_vector(3 downto 0);
          gt,lt,eq: in std_logic;
          AeqB,AltB,AgtB: out std_logic);
   end component;

  signal saida_gt,saida_lt,saida_eq: std_logic_vector(1 downto 0);
  begin
      Comp1:Comparador_4Bits port map(
           gt => '0',
           lt => '0',
           eq => '1',
           eA(3 downto 0) => eA8(7 downto 4),
           eB(3 downto 0) => eB8(7 downto 4),
           AeqB => saida_eq(1),
           AgtB => saida_gt(1),
           AltB => saida_lt(1));

      Comp2: Comparador_4Bits port map(
           gt => saida_gt(1),
           lt => saida_lt(1),
           eq => saida_eq(1),
           eA(3 downto 0) => eA8(3 downto 0),
           eB(3 downto 0) => eB8(3 downto 0),
           AeqB => saida_eq(0),
           AgtB => saida_gt(0),
           AltB => saida_lt(0));

      AeqB8 <= saida_eq(0);
      AltB8 <= saida_lt(0);
      AgtB8 <= saida_gt(0);

end ckt;
```

```
1   library ieee ;
2   use ieee.std_logic_1164.all;
3
4   entity reg8Bits is
5      port (clk,preSet,clr,load: in std_logic;
6      d: in std_logic_vector(7 downto 0);
7      q : out std_logic_vector(7 downto 0));
8   end reg8Bits;
9   architecture ckt of reg8Bits is
10
11     signal qs: std_logic_vector(7 downto 0);
12
13     begin
14        process (clk ,preSet,clr)
15        begin
16           if preSet = '0' then qs <= "11111111";
17           elsif clr = '0' then qs <= "00000000";
18           elsif clk ='1' and clk ' event then
19              if load = '1' then
20                 qs <= d;
21              end if;
22           end if;
23        end process ;
24        q <= qs;
25  end ckt;
```

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity SUM_1Bit is
5     port (A_in,B_in,C_in: in std_logic;
6           S_out, C_out: out std_logic);
7   end SUM_1Bit;
8
9   Architecture ckt of SUM_1Bit is
10
11  Begin
12    S_out <= ((B_in and ((C_in nor A_in) or (C_in and A_in))) or ((not B_in) and (((not C_in
    ) and A_in) or (C_in and (not A_in)))));
13    C_out <= ((C_in and (A_in or B_in)) or (A_in and B_in));
14  end ckt;
```

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity SUM_2Bits is
5     port (A2_in,B2_in: in std_logic_vector(1 downto 0);
6           C2_in: in std_logic;
7           S2_out: out std_logic_vector(1 downto 0);
8           C2_out: out std_logic);
9   end SUM_2Bits;
10
11  architecture ckt of SUM_2Bits is
12    component SUM_1Bit is
13      port (A_in,B_in,C_in: in std_logic;
14            S_out, C_out: out std_logic);
15    end component;
16
17    signal Sum_01_out : std_logic;
18
19    begin
20      SUM01: SUM_1Bit port map(
21            A_in => A2_in(0),
22            B_in => B2_in(0),
23            C_in => C2_in,
24            S_out => S2_out(0),
25            C_out => Sum_01_out);
26
27      SUM02: SUM_1Bit port map(
28            A_in => A2_in(1),
29            B_in => B2_in(1),
30            C_in => Sum_01_out,
31            S_out => S2_out(1),
32            C_out => C2_out);
33
34  end ckt;
```

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity SUM_4Bits is
5     port (A4_in,B4_in: in std_logic_vector(3 downto 0);
6           C4_in: in std_logic;
7           S4_out: out std_logic_vector(3 downto 0);
8           C4_out: out std_logic);
9   end SUM_4Bits;
10
11  architecture ckt of SUM_4Bits is
12    component SUM_2Bits is
13      port (A2_in,B2_in: in std_logic_vector(1 downto 0);
14            C2_in: in std_logic;
15            S2_out: out std_logic_vector(1 downto 0);
16            C2_out: out std_logic);
17    end component;
18
19    signal Sum_01_out : std_logic;
20
21    begin
22      SUM01: SUM_2Bits port map(
23            A2_in => A4_in(1 downto 0),
24            B2_in => B4_in(1 downto 0),
25            C2_in => C4_in,
26            S2_out => S4_out(1 downto 0),
27            C2_out => Sum_01_out);
28
29      SUM02: SUM_2Bits port map(
30            A2_in => A4_in(3 downto 2),
31            B2_in => B4_in(3 downto 2),
32            C2_in => Sum_01_out,
33            S2_out => S4_out(3 downto 2),
34            C2_out => C4_out);
35
36  end ckt;
```

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity SUM_8Bits is
5     port (A8_in,B8_in: in std_logic_vector(7 downto 0);
6           C8_in: in std_logic;
7           S8_out: out std_logic_vector(7 downto 0);
8           C8_out: out std_logic);
9   end SUM_8Bits;
10
11  architecture ckt of SUM_8Bits is
12    component SUM_4Bits is
13      port (A4_in,B4_in: in std_logic_vector(3 downto 0);
14            C4_in: in std_logic;
15            S4_out: out std_logic_vector(3 downto 0);
16            C4_out: out std_logic);
17    end component;
18
19    signal Sum_01_out : std_logic;
20
21    begin
22      SUM01: SUM_4Bits port map(
23            A4_in => A8_in(3 downto 0),
24            B4_in => B8_in(3 downto 0),
25            C4_in => C8_in,
26            S4_out => S8_out(3 downto 0),
27            C4_out => Sum_01_out);
28
29      SUM02: SUM_4Bits port map(
30            A4_in => A8_in(7 downto 4),
31            B4_in => B8_in(7 downto 4),
32            C4_in => Sum_01_out,
33            S4_out => S8_out(7 downto 4),
34            C4_out => C8_out);
35
36  end ckt;
```

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity ciBitToBcd is
5     port (BtB_in: in bit_vector(3 downto 0);
6           BtB_out: out bit_vector(3 downto 0));
7   end ciBitToBcd;
8
9   architecture ckt of ciBitToBcd is
10  begin
11    BtB_out(3) <= (BtB_in(3) or (BtB_in(2) and (BtB_in(1) or BtB_in(0))));
12    BtB_out(2)<= ((BtB_in(2) and (not BtB_in(1)) and (not BtB_in(0))) or (BtB_in(3) and
    BtB_in(0)));
13    BtB_out(1)<= ((BtB_in(3) and (not BtB_in(0))) or (BtB_in(1) and ((not BtB_in(2)) or
    BtB_in(0))));
14    BtB_out(0)<= (((not BtB_in(3)) and (not BtB_in(2)) and BtB_in(0)) or ((not BtB_in(0))
    and (BtB_in(3) or (BtB_in(2) and BtB_in(1)))));
15  end ckt;
16
```

```
 1  library ieee;
 2  use ieee.std_logic_1164.all;
 3
 4  entity bitToBcd is
 5    port (bit_in: in std_logic_vector(7 downto 0);
 6          bcd_out: out std_logic_vector(11 downto 0));
 7  end bitToBcd ;
 8
 9  architecture ckt of bitToBcd is
10    component ciBitToBcd is
11      port (BtB_in: in std_logic_vector(3 downto 0);
12            BtB_out: out std_logic_vector(3 downto 0));
13    end component;
14
15    signal ciBtB_01_out, ciBtB_02_out, ciBtB_03_out, ciBtB_04_out, ciBtB_05_out,
    ciBtB_06_out, ciBtB_07_out: std_logic_vector(3 downto 0);
16
17    begin
18        ciBtB01: ciBitToBcd port map(
19             BtB_in(3) => '0',
20             BtB_in(2 downto 0) => bit_in(7 downto 5),
21             BtB_out => ciBtB_01_out);
22
23        ciBtB02: ciBitToBcd port map(
24             BtB_in(3 downto 1) => ciBtB_01_out(2 downto 0),
25             BtB_in(0) => bit_in(4),
26             BtB_out => ciBtB_02_out);
27
28        ciBtB03: ciBitToBcd port map(
29             BtB_in(3 downto 1) => ciBtB_02_out(2 downto 0),
30             BtB_in(0) => bit_in(3),
31             BtB_out => ciBtB_03_out);
32
33        ciBtB04: ciBitToBcd port map(
34             BtB_in(3) => '0',
35             BtB_in(2) => ciBtB_01_out(3),
36             BtB_in(1) => ciBtB_02_out(3),
37             BtB_in(0) => ciBtB_03_out(3),
38             BtB_out => ciBtB_04_out);
39
40        ciBtB05: ciBitToBcd port map(
41             BtB_in(3 downto 1)=> ciBtB_03_out(2 downto 0),
42             BtB_in(0) => bit_in(2),
43             BtB_out => ciBtB_05_out);
44
45        ciBtB06: ciBitToBcd port map(
46             BtB_in(3 downto 1)=> ciBtB_04_out(2 downto 0),
47             BtB_in(0) => ciBtB_05_out(3),
48             BtB_out => ciBtB_06_out);
49
50        ciBtB07: ciBitToBcd port map(
51             BtB_in(3 downto 1)=> ciBtB_05_out(2 downto 0),
52             BtB_in(0) => bit_in(1),
53             BtB_out => ciBtB_07_out);
54
55        bcd_out(11) <= '0';
56        bcd_out(10) <= '0';
57        bcd_out(9) <= ciBtB_04_out(3);
58        bcd_out(8 downto 5) <= ciBtB_06_out;
59        bcd_out(4 downto 1) <= ciBtB_07_out;
60        bcd_out(0) <= bit_in(0);
61
62  end ckt;
```

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity d7Seg is
5     port(S_in: in std_logic_vector(3 downto 0);
6          D_out: out std_logic_vector(6 downto 0));
7   end d7Seg;
8
9   architecture display of d7Seg is
10    begin
11      D_out(0) <= not ((S_in(3) or S_in(1) or (S_in(2) and S_in(0)) or (S_in(2) nor S_in(0
    ))));
12
13      D_out(1) <= not (((not S_in(2)) or (S_in(1) nor S_in(0)) or (S_in(1) and S_in(0))));
14
15      D_out(2) <= not (((not(S_in(3)) and ((S_in(1) and S_in(0)) or S_in(2))) or (not(S_in(2
    ) or S_in(1)))));
16
17      D_out(3) <= not((S_in(3) or (S_in(1) and (S_in(2) nand S_in(0))) or (S_in(2) nor S_in(
    0)) or (S_in(2) and (not S_in(1)) and S_in(0))));
18
19      D_out(4) <= not (((S_in(1) and (not S_in(0))) or (S_in(2) nor S_in(0))));
20
21      D_out(5) <= not ((S_in(3) or (S_in(1) nor S_in(0)) or (S_in(2) and (S_in(1) nand S_in(
    0)))));
22
23      D_out(6) <= not ((S_in(3) or (S_in(1) and (not S_in(2))) or (S_in(2) and (S_in(1) nand
     S_in(0)))));
24
25  end display;
```

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity bitToBcdToD7Seg is
5     port (SW_in: in std_logic_vector(7 downto 0);
6           HEX0,HEX1,HEX2: out std_logic_vector(6 downto 0));
7   end bitToBcdToD7Seg ;
8
9   architecture ckt of bitToBcdToD7Seg is
10    component bitToBcd is
11      port (bit_in: in std_logic_vector(7 downto 0);
12            bcd_out: out std_logic_vector(11 downto 0));
13    end component;
14    component d7Seg is
15       port(S_in: in std_logic_vector(3 downto 0);
16            D_out: out std_logic_vector(6 downto 0));
17    end component;
18
19    signal bcd_ax_out: std_logic_vector(11 downto 0);
20    signal hex0_ax_out,hex1_ax_out,hex2_ax_out : std_logic_vector(6 downto 0);
21
22    begin
23        BtB: BitToBcd port map(
24             bit_in => SW_in,
25             bcd_out => bcd_ax_out);
26
27        D7S2: d7Seg port map(
28              S_in => bcd_ax_out(11 downto 8),
29              D_out => hex2_ax_out);
30
31        D7S1: d7Seg port map(
32              S_in => bcd_ax_out(7 downto 4),
33              D_out => hex1_ax_out);
34
35        D7S0: d7Seg port map(
36              S_in => bcd_ax_out(3 downto 0),
37              D_out => hex0_ax_out);
38        HEX0 <= hex0_ax_out;
39        HEX1 <= hex1_ax_out;
40        HEX2 <= hex2_ax_out;
41  end ckt;
```

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   ENTITY ffjk IS
5     port ( clk ,J ,K ,P ,C : in std_logic;
6              q : out std_logic );
7   END ffjk ;
8   ARCHITECTURE ckt OF ffjk IS
9   SIGNAL qS : std_logic;
10  BEGIN
11    PROCESS ( clk ,P ,C )
12    BEGIN
13      IF P = '0' THEN qS <= '1';
14      ELSIF C ='0' THEN qS <= '0';
15      ELSIF clk = '1' AND clk ' EVENT THEN
16        IF J = '1' AND K = '1' THEN qS <= NOT qS ;
17        ELSIF J = '1' AND K = '0' THEN qS <= '1';
18        ELSIF J = '0' AND K = '1' THEN qS <= '0';
19        END IF;
20      END IF;
21    END PROCESS ;
22  q <= qS ;
23  END ckt ;
```

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity contador6Bits is
5     port ( clk_c6B,ld_c6B,clr_c6B: in std_logic;
6            out_c6B: out std_logic_vector(5 downto 0));
7   end contador6Bits ;
8   architecture ckt of contador6Bits is
9      component ffjk is
10        port (clk ,J ,K ,P ,C: in std_logic;
11              q: out std_logic);
12     end component;
13
14     signal q0_out,q1_out,q2_out,q3_out,q4_out,q5_out, clr_reverso: std_logic;
15     signal resp_and: std_logic_vector(4 downto 0);
16
17     begin
18
19        clr_reverso <= not clr_c6B;
20        Q0: ffjk port map(
21             clk => clk_c6B,
22             J => ld_c6B,
23             K => ld_c6B,
24             P => '1',
25             C => clr_reverso,
26             q => q0_out);
27
28        resp_and(0) <= q0_out and ld_c6B;
29
30        Q1: ffjk port map(
31             clk => clk_c6B,
32             J => resp_and(0),
33             K => resp_and(0),
34             P => '1',
35             C => clr_reverso,
36             q => q1_out);
37
38        resp_and(1) <= q1_out and resp_and(0);
39
40        Q2: ffjk port map(
41             clk => clk_c6B,
42             J => resp_and(1),
43             K => resp_and(1),
44             P => '1',
45             C => clr_reverso,
46             q => q2_out);
47
48        resp_and(2) <= q2_out and resp_and(1);
49
50        Q3: ffjk port map(
51             clk => clk_c6B,
52             J => resp_and(2),
53             K => resp_and(2),
54             P => '1',
55             C => clr_reverso,
56             q => q3_out);
57
58        resp_and(3) <= q3_out and resp_and(2);
59
60        Q4: ffjk port map(
61             clk => clk_c6B,
62             J => resp_and(3),
63             K => resp_and(3),
64             P => '1',
65             C => clr_reverso,
66             q => q4_out);
```

```
67
68          resp_and(4) <= q4_out and resp_and(3);
69
70          Q5: ffjk port map(
71               clk => clk_c6B,
72               J => resp_and(4),
73               K => resp_and(4),
74               P => '1',
75               C => clr_reverso,
76               q => q5_out);
77
78          out_c6B(0) <= q0_out;
79          out_c6B(1) <= q1_out;
80          out_c6B(2) <= q2_out;
81          out_c6B(3) <= q3_out;
82          out_c6B(4) <= q4_out;
83          out_c6B(5) <= q5_out;
84
85
86    end ckt ;
```