

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

RELATÓRIO SISTEMAS DIGITAIS – FIFO – PROJETO 02

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 – FIFO – PROJETO 02

NATAL

2020

Sumário

1. Introdução	03
2. Desenvolvimento	04
2.1 Divisor de Clock	04
2.2 FIFO	04
2.4 BINBCD – Display7Segmentos	07
2.5 ROM	08
2.6 Contador	08
3. Resultados Obtidos	09
4. Conclusão	13
5. Referências Bibliográficas	14
6. Anexos	15

1. Introdução

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

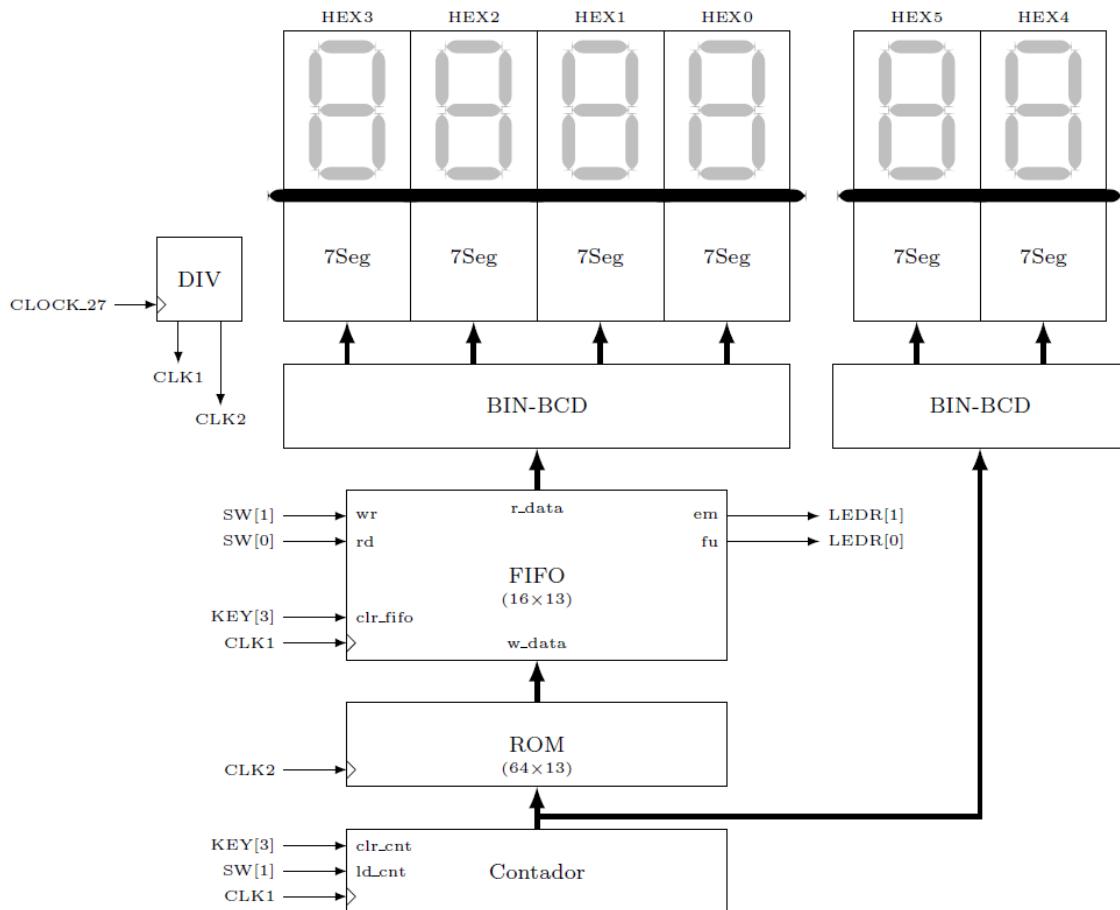


Imagen 1 - Arquitetura desejada para ser implementada

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, projeto FIFO (funcionalmente, bloco de controle e datapath), 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.

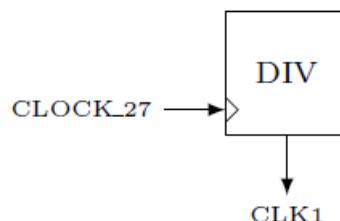


Imagen 2 – Bloco Divisor de Clock

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 – FIFO

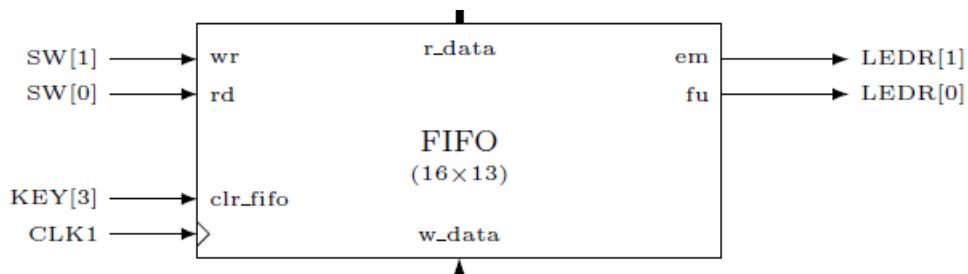


Imagen 3 – Bloco FIFO

A FIFO, do inglês *First-in First-out*, é um tipo de memória que tem como definição o seguinte dilema: o primeiro dado que entra na memória é sempre o primeiro a sair. No projeto proposto, tínhamos que projetar uma FIFO com 16 posições que suportasse uma leitura de 13Bits.

Para tal, tiramos como base o projeto RTL 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.

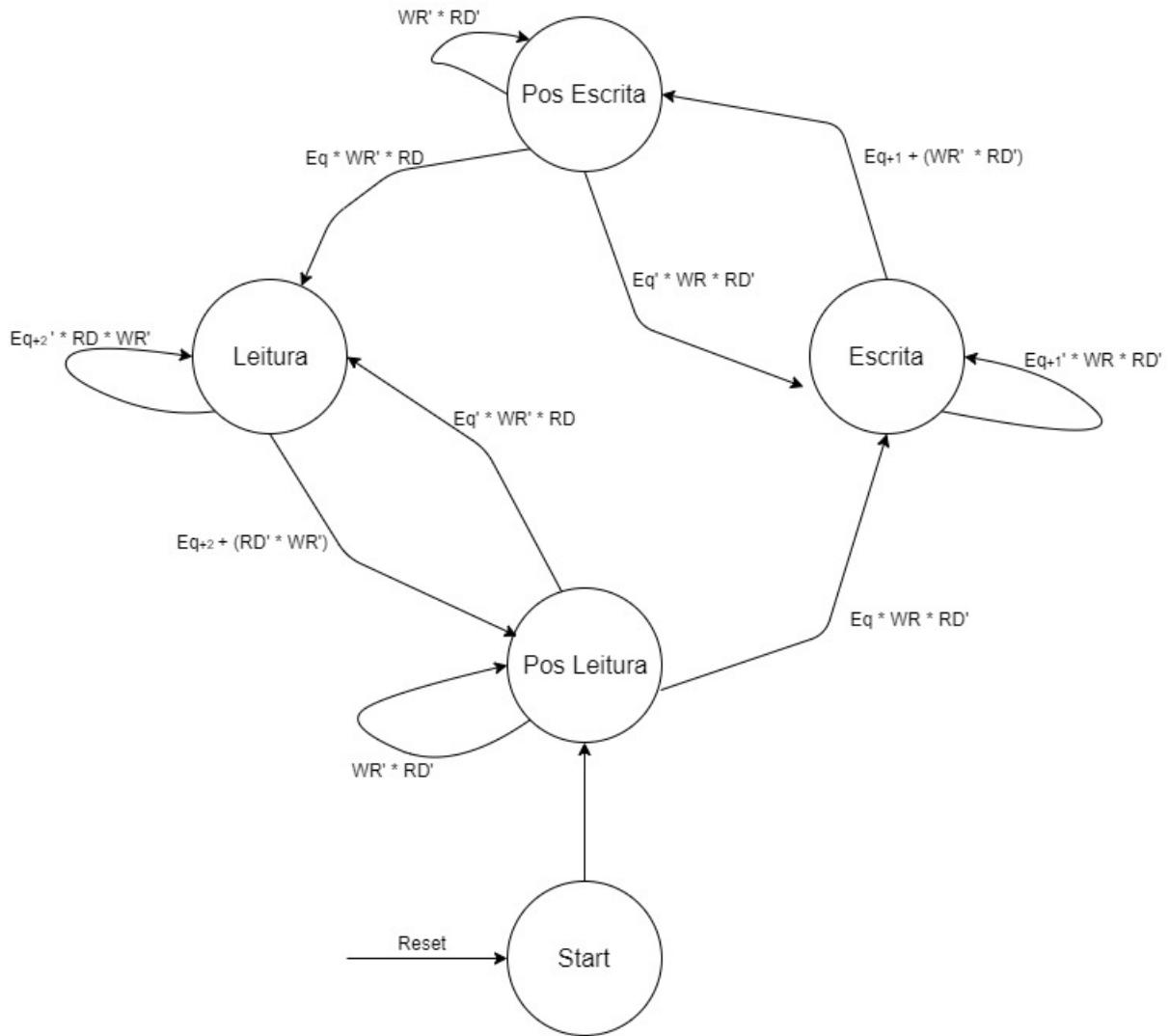


Imagen 4 – Máquina de estado de alto nível

Funcionalmente da máquina se dar da seguinte maneira:

Start: Estado no qual a máquina inicia;

Pós Leitura: Nesse estado, a máquina espera o acionamento do botão de escrita na memória (WR), caso acionado, vai para o estado de Escrita. Como também, se a comparação dos contadores de posição de escrita e leitura forem iguais, acionará um LED avisando que a fila se encontra vazia. Caso esse LED não esteja acionado, e for apertado o botão de leitura (RD) a máquina irá para o estado de leitura.

Escrita: Nesse estado, a máquina mandará um sinal para gravar o dado que está sendo enviado para a memória. Se o botão WR continuar acionado, EQ₊₁ (comparação dos contadores de posições de escrita mais um e leitura. Basicamente, serve para dizer se a próxima posição do contador de escrita encherá a memória FIFO) e RD estiverem barrados a máquina continuará no estado de escrita. Caso EQ₊₁ vinher a se tornar *true* ou WR parar de ser acionado, a máquina irá para o estado de pós escrita.

Pós Escrita: Nesse estado, a máquina espera o acionamento do botão de leitura na memória (RD), caso acionado, vai para o estado de Leitura. Como também, se a comparação dos contadores de posição de escrita e leitura forem iguais, acionará um LED avisando que a fila se encontra cheia. Caso esse LED não esteja acionado, e for apertado o botão de escrita (WR) a máquina irá para o estado de leitura.

Leitura: Nesse estado, a máquina mandará um sinal para ler os dados que estão gravados na memória. Se o botão RD continuar acionado, EQ₊₂ (comparação dos contadores de posições de escrita e leitura mais um. Basicamente, serve para dizer se a próxima posição do contador de leitura esvaziará a memória FIFO) e RD estiverem barrados a máquina continuará no estado de escrita. Caso EQ₊₂ vinher a se tornar *true* ou WR parar de ser acionado, a máquina irá para o estado de pós escrita.

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 Banco de Registradores no qual, em sua estrutura interna se encontra um decodificador e um multiplexador ambos de 16bits, como também, dezesseis registradores de 13Bits feitos com flip-flop D; Dois contadores de 4bits, que servem para informar ao bloco operacional qual as posições que iram ser escritas ou lidas pelo bloco de controle; Dois somadores de 4bits, que exercem a função de somar a saída dos contadores mais um para ser usado nos comparadores; Três comparadores de 4bits, onde desempenham a função de comparar as posições dadas pelos contadores. Portanto, ficando com a seguinte estrutura para o bloco de controle e datapath.

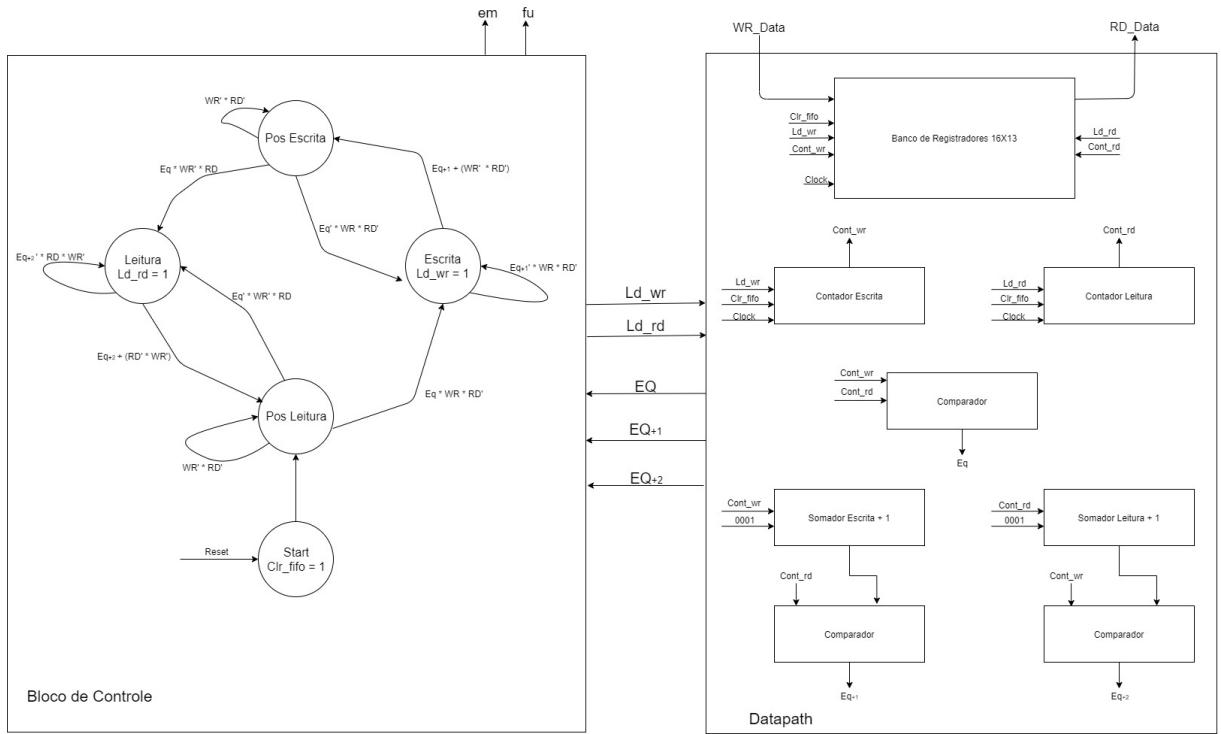


Imagen 5 – Bloco de controle e datapath da máquina de vendas

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

2.3 – BIN-BCD com display7Segmentos

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

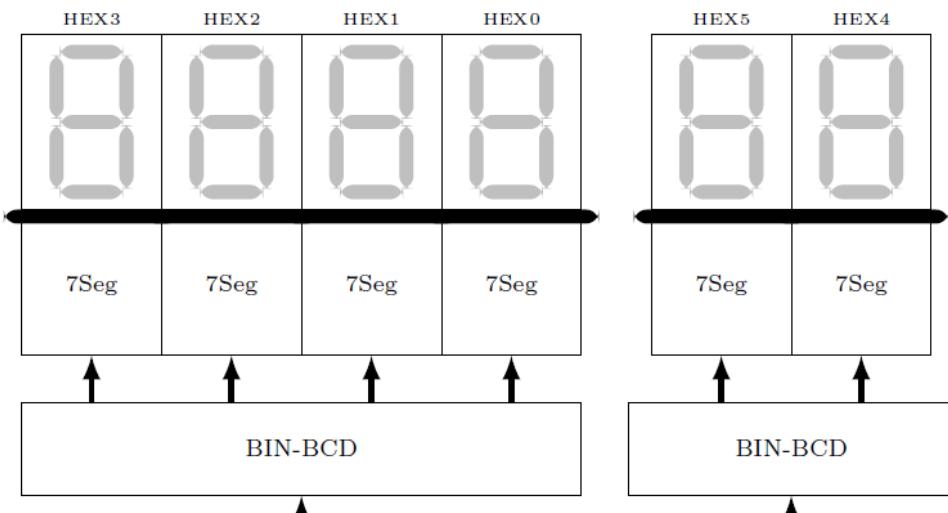


Imagen 6 – Bloco BINBCD – Display7Segmentos

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, a qual, é requisito obrigatório para se pagar Sistemas Digitais, com alterações no código de binário para BCD a fim de que ele consiga converter até a casa do milhar.

2.4 – ROM

O bloco de memória vista na arquitetura da imagem 1, funcionará basicamente como um guarda-dado dos valores que serão enviados para a FIFO, retirando a necessidade de o usuário ficar colocando valores. Por isso, que sua saída vai direto para a entrada da FIFO. Visto que, como no projeto só queremos ler os valores que estão salvos nela, a ROM é a melhor escolha. Já que, pela sua definição a ROM (*Read-Only Memory*) é uma memória apenas de leitura.

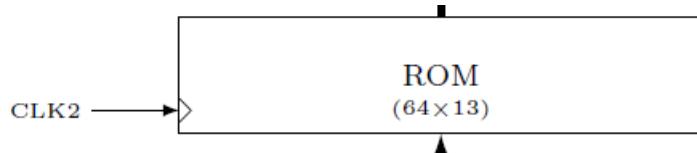


Imagen 7 – Bloco da ROM

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.

2.5 – Contador

No projeto, o contador foi empregado para acessar a posição da ROM e assim, pegar o valor que serão usados como valores para a FIFO.



Imagen 8 – 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 dará por flip flop JK juntos, com pequena diferença empregada devido a imposição existente 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 quiser gravar um novo valor na FIFO. Quando isso ocorrer, o valor que está no contador ditará a posição da ROM no qual terá a quantia X guardada na FIFO. Para sabermos em qual posição a ROM se encontra a saída do contador também vai para dois displays.

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:

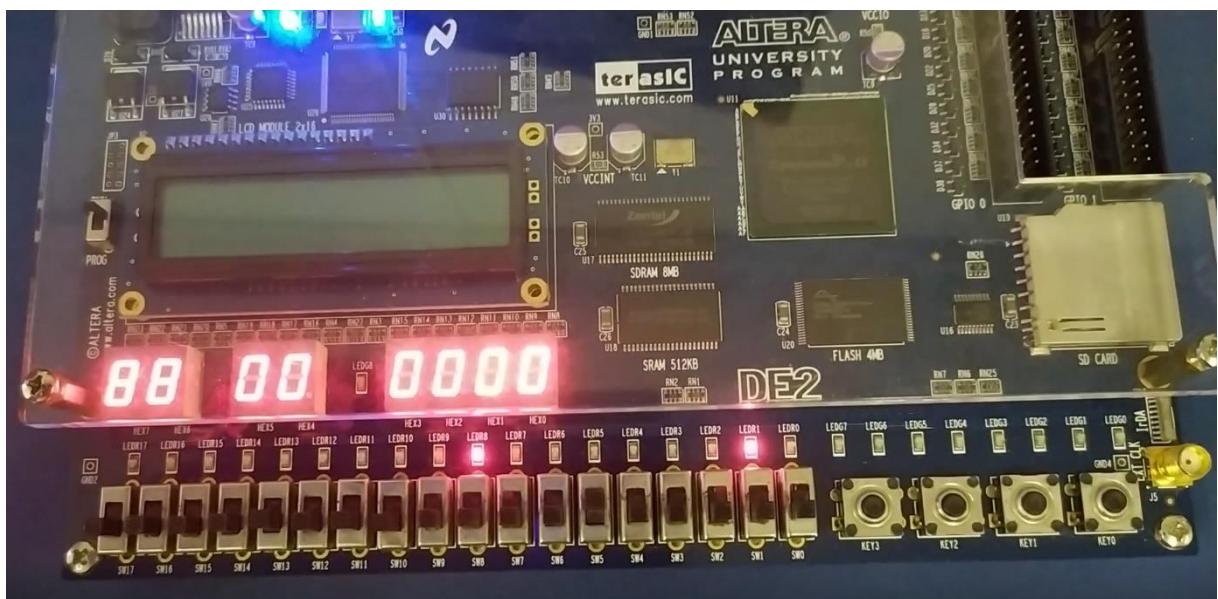


Imagen 9 – momento inicial do teste em FPGA

Como podemos ver na imagem 11, os 2 displays que se encontram no meio (o qual possuem valor 00) mostra a posição da ROM. Já os 4 displays da ponta (o qual possuem valor 0000 inicialmente) mostra o valor da saída da FIFO, ou seja, a ordem dos valores inseridos será mostrada em ordem nesses displays. Os LEDs LEDR10, LEDR9 e LEDR8 foram usados a fins didáticos para se visualizar em qual estado a máquina se encontra.

Nesse primeiro momento, vemos a FIFO completamente vazia, pois o LEDR1 está aceso. Quando levantamos a chave SW1, mandamos o sinal para a máquina para escrever o valor da primeira posição da ROM na fila. Quando isso ocorre, o LEDR1 se apaga, informando que a fila não está mais vazia, como também, os displays do meio mostram a nova posição que o contador da ROM se encontra.

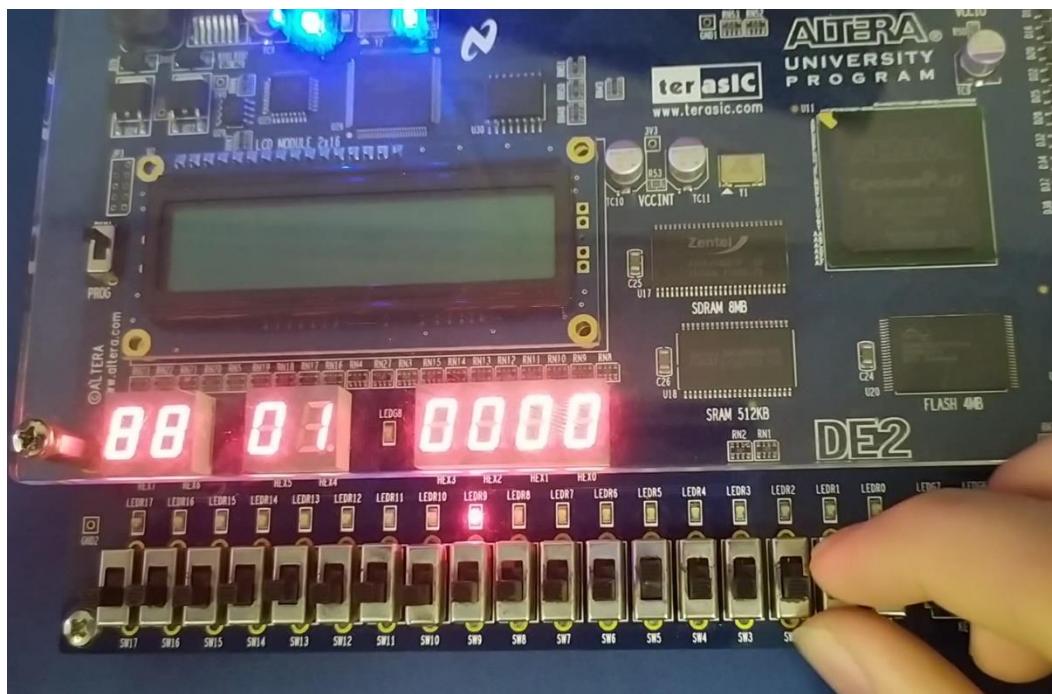


Imagen 10 – Funcionalmente da escrita na FIFO

Para fins de mostrar seu funcionalmente quando a mesma se encontra cheia, continuemos a dar o sinal de escrita para a máquina, até que chegamos à posição de 16 do contador da ROM. Porquê, como a exigência da arquitetura vista na imagem 1 desse relatório, a FIFO tinha como tamanho 16x13, ou seja, 16 posições sendo cada uma delas de 13Bits. Assim, ao chegar nesse ponto a FIFO se encontrou cheia, visto que, o LEDR0 se acendeu.

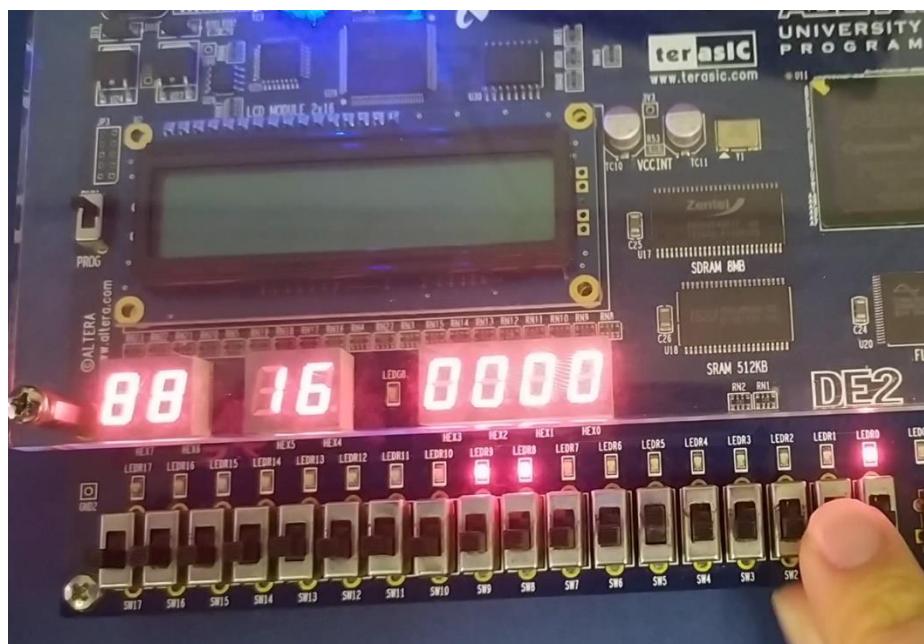
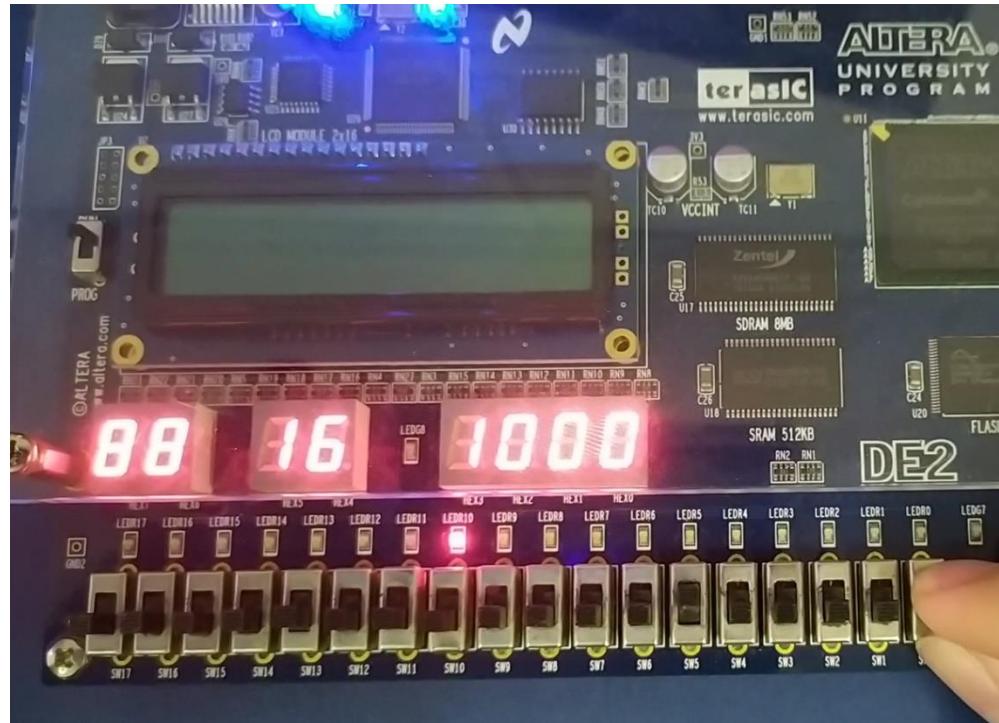


Imagen 11 – Funcionamento da FIFO quando cheia

Já partir daqui, começamos a acionar a leitura da fila. Ao levantar a chave SW0 mandamos um sinal a máquina informando-a que desejamos fazer a leitura dos valores salvos na FIFO. Quando isso ocorre, o led LEDR0 se apaga, comunicando que a fila não se encontra mais cheia e assim o primeiro valor inserido na fila é mostrado nos 4 displays da ponta. Em meu caso, a primeira posição da ROM tinha valor 1000 guardado nela.



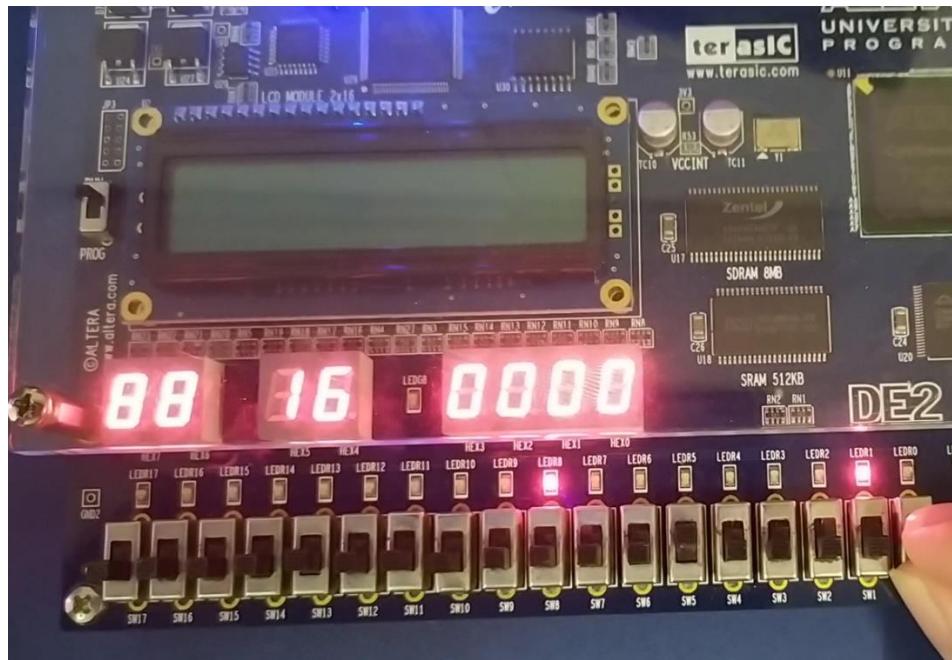


Imagen 13 – FIFO novamente vazia, após leitura de todas os valores guardados.

Os testes realizaram também mostraram o perfeito funcionalmente, caso se deseja-se fazer uma leitura sem antes ter enchido a fila, ou fazer uma escrita sem antes ter esvaziado a fila. Adicionalmente, ao apertamos no botão KEY3 da FPGA independentemente das posições que a FIFO ou o contador da ROM se encontram, todo o projeto sofre reset. Voltando assim, as suas configurações iniciais, presente na imagem 9 para esse respectivo teste.

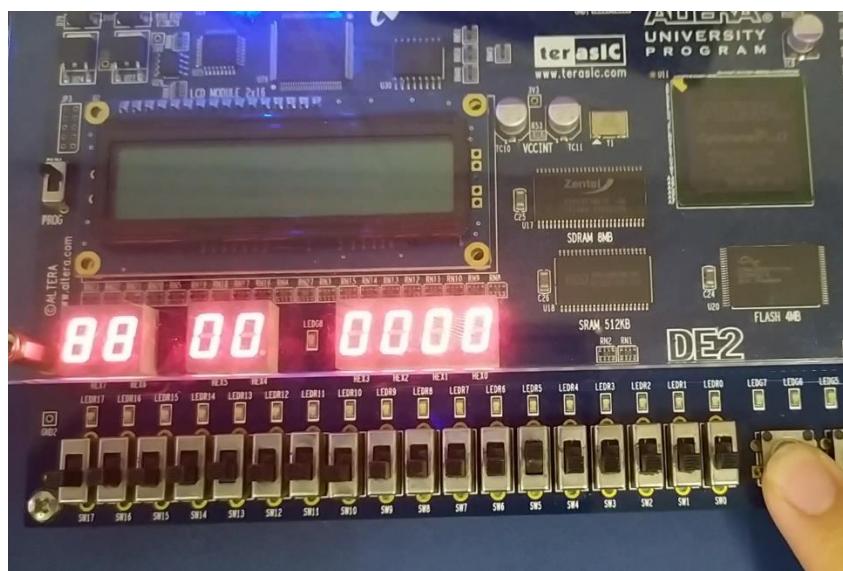


Imagen 14 – FIFO após botão reset ser pressionado.

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

4. Conclusão

Nos dias atuais, os projetos de circuitos aumentaram significativamente. Como também, a utilização de memória seja RAM ou ROM junto das mesmas. Para tal, a utilização de máquinas RTL para a projeção de circuitos tornou-se 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 FIFO, com uma memória ROM para salvar os valores que serão inseridos na fila. Ademais, foi mostrado os resultados obtidos quando simulado o projeto utilizando a FPGA Cyclone II.

Disto podemos retirar as seguintes conclusões, dado a quantidade de componentes exigidos para a FIFO 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 projeto02 is
5      port ( clock_27,SW1,SW0,KEY3: in std_logic ;
6             LEDR1,LEDR0: out std_logic;
7             P02HEX5,P02HEX4, P02HEX3, P02HEX2, P02HEX1, P02HEX0: out std_logic_vector(6
8 downto 0);
9             est_maq_fifo: out std_logic_vector(2 downto 0));
10 end projeto02;
11
12 architecture ckt of projeto02 is
13     -- Divisor de Clock
14     component divisorClock is
15         port (clk_in : in std_logic ;
16                clk_out : out std_logic );
17     end component;
18
19     -- ROM
20     component romFIFO is
21         PORT (address: IN STD_LOGIC_VECTOR (5 DOWNTO 0);
22                clock: IN STD_LOGIC  := '1';
23                q: OUT STD_LOGIC_VECTOR (12 DOWNTO 0));
24     end component;
25
26     --FIFO
27     component fifo is
28         port (CLK_fifo, WR, RD, reset: in std_logic;
29                W_data: in std_logic_vector(12 downto 0);
30                em,fu: out std_logic;
31                R_data: out std_logic_vector(12 downto 0);
32                Estados_maquina: out std_logic_vector(2 downto 0));
33     end component;
34
35     --Contador
36     component contador6Bits is
37         port ( clk_c6B,ld_c6B,clr_c6B: in std_logic;
38                out_c6B: out std_logic_vector(5 downto 0));
39     end component;
40
41     --BINBCDtoD7SEG
42     component bitToBcd13bitsToD7Seg is
43         port (SW_in: in std_logic_vector(12 downto 0);
44                HEX0,HEX1,HEX2,HEX3: out std_logic_vector(6 downto 0));
45     end component;
46
47     signal CLK_1, led_full, led_empty, clr_reverso: std_logic;
48     signal saida_rom, saida_fifo: std_logic_vector(12 downto 0);
49     signal saida_contador: std_logic_vector(5 downto 0);
50     signal liox1,lixo0, hx00,hx01,hx02,hx03,hx04,hx05: std_logic_vector(6 downto 0);
51     signal estados: std_logic_vector(2 downto 0);
52
53 begin
54
55     clr_reverso <= not KEY3;
56
57     DClock: divisorClock port map(
58         clk_in => clock_27,
59         clk_out => CLK_1);
60
61     FIFO16x13: fifo port map(
62         CLK_fifo => CLK_1,
63         WR => SW1,
64         RD => SW0,
65         reset => clr_reverso,
66         W_data => saida_rom,

```

```
66          em => led_empty,
67          fu => led_full,
68          R_data => saida_fifo,
69          Estados_maquina => estados);
70
71      ROM: romFIFO port map(
72          address => saida_contador,
73          clock => CLK_1,
74          q => saida_rom);
75
76      Contador: contador6Bits port map(
77          clk_c6B => CLK_1,
78          ld_c6B => SW1,
79          clr_c6B => clr_reverso,
80          out_c6B => saida_contador);
81
82      DisplayRead: bitToBcd13bitsToD7Seg port map(
83          SW_in => saida_fifo,
84          HEX0 => hx00,
85          HEX1 => hx01,
86          HEX2 => hx02,
87          HEX3 => hx03);
88
89      DisplayContador: bitToBcd13bitsToD7Seg port map(
90          SW_in(12 downto 6) =>"0000000",
91          SW_in(5 downto 0) => saida_contador,
92          HEX0 => hx04,
93          HEX1 => hx05,
94          HEX2 => lixo0,
95          HEX3 => lixo1);
96
97      -- HEX
98      P02HEX0 <= hx00;
99      P02HEX1 <= hx01;
100     P02HEX2 <= hx02;
101     P02HEX3 <= hx03;
102     P02HEX4 <= hx04;
103     P02HEX5 <= hx05;
104
105     -- Estados da Maquina FIFO
106     est_mag_fifo <= estados;
107
108     -- Luzes de aviso
109     LEDR0 <= led_full;
110     LEDR1 <= led_empty;
111 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  -- megafunction wizard: %ROM: 1-PORT%
2  -- GENERATION: STANDARD
3  -- VERSION: WM1.0
4  -- MODULE: altsyncram
5
6  -- =====
7  -- File Name: romFIFO.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 romFIFO 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 (12 DOWNTO 0)
48     );
49 END romFIFO;
50
51
52 ARCHITECTURE SYN OF romfifo IS
53
54     SIGNAL sub_wire0  : STD_LOGIC_VECTOR (12 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 (12 DOWNTO 0)
78  );
79 END COMPONENT;
80
81 BEGIN
82     q    <= sub_wire0(12 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 => "romFIFO.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 => 13,
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 "romFIFO.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 "13"
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 "romFIFO.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 "13"
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 13 0 OUTPUT NODEFVAL "q[12..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 13 0 @q_a 0 0 13 0
161  -- Retrieval info: GEN_FILE: TYPE_NORMAL romFIFO.vhd TRUE
162  -- Retrieval info: GEN_FILE: TYPE_NORMAL romFIFO.inc FALSE
163  -- Retrieval info: GEN_FILE: TYPE_NORMAL romFIFO.cmp FALSE
164  -- Retrieval info: GEN_FILE: TYPE_NORMAL romFIFO.bsf FALSE
165  -- Retrieval info: GEN_FILE: TYPE_NORMAL romFIFO_inst.vhd FALSE
166  -- Retrieval info: LIB_FILE: altera_mf
167
```

```
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 ;
```

```
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 bitToBcd13bitsToD7Seg is
5      port (SW_in: in std_logic_vector(12 downto 0);
6             HEX0,HEX1,HEX2,HEX3: out std_logic_vector(6 downto 0));
7  end bitToBcd13bitsToD7Seg ;
8
9  architecture ckt of bitToBcd13bitsToD7Seg is
10     component bitToBcd13bits is
11         port (bit_in: in std_logic_vector(12 downto 0);
12                bcd_out: out std_logic_vector(15 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(15 downto 0);
20     signal hex0_ax_out,hex1_ax_out,hex2_ax_out,hex3_ax_out : std_logic_vector(6 downto 0);
21
22 begin
23     BTB: BitToBcd13bits port map(
24         bit_in => SW_in,
25         bcd_out => bcd_ax_out);
26
27     D7S3: d7Seg port map(
28         S_in => bcd_ax_out(15 downto 12),
29         D_out => hex3_ax_out);
30
31     D7S2: d7Seg port map(
32         S_in => bcd_ax_out(11 downto 8),
33         D_out => hex2_ax_out);
34
35     D7S1: d7Seg port map(
36         S_in => bcd_ax_out(7 downto 4),
37         D_out => hex1_ax_out);
38
39     D7S0: d7Seg port map(
40         S_in => bcd_ax_out(3 downto 0),
41         D_out => hex0_ax_out);
42     HEX0 <= hex0_ax_out;
43     HEX1 <= hex1_ax_out;
44     HEX2 <= hex2_ax_out;
45     HEX3 <= hex3_ax_out;
46 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 bitToBcd13bits is
5      port (bit_in: in std_logic_vector(12 downto 0);
6             bcd_out: out std_logic_vector(15 downto 0));
7  end bitToBcd13bits ;
8
9  architecture ckt of bitToBcd13bits 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,
16    ciBtB_06_out, ciBtB_07_out, ciBtB_08_out, ciBtB_09_out: std_logic_vector(3 downto 0);
17    signal ciBtB_10_out, ciBtB_11_out, ciBtB_12_out, ciBtB_13_out, ciBtB_14_out,
18    ciBtB_15_out, ciBtB_16_out, ciBtB_17_out, ciBtB_18_out: std_logic_vector(3 downto 0);
19    signal ciBtB_19_out, ciBtB_20_out, ciBtB_21_out: std_logic_vector(3 downto 0);
20
21 begin
22     ciBtB01: ciBitToBcd port map(
23         BtB_in(3) => '0',
24         BtB_in(2 downto 0) => bit_in(12 downto 10),
25         BtB_out => ciBtB_01_out);
26
27     ciBtB02: ciBitToBcd port map(
28         BtB_in(3 downto 1) => ciBtB_01_out(2 downto 0),
29         BtB_in(0) => bit_in(9),
30         BtB_out => ciBtB_02_out);
31
32     ciBtB03: ciBitToBcd port map(
33         BtB_in(3 downto 1) => ciBtB_02_out(2 downto 0),
34         BtB_in(0) => bit_in(8),
35         BtB_out => ciBtB_03_out);
36
37     ciBtB04: ciBitToBcd port map(
38         BtB_in(3) => '0',
39         BtB_in(2) => ciBtB_01_out(3),
40         BtB_in(1) => ciBtB_02_out(3),
41         BtB_in(0) => ciBtB_03_out(3),
42         BtB_out => ciBtB_04_out);
43
44     ciBtB05: ciBitToBcd port map(
45         BtB_in(3 downto 1) => ciBtB_03_out(2 downto 0),
46         BtB_in(0) => bit_in(7),
47         BtB_out => ciBtB_05_out);
48
49     ciBtB06: ciBitToBcd port map(
50         BtB_in(3 downto 1) => ciBtB_04_out(2 downto 0),
51         BtB_in(0) => ciBtB_05_out(3),
52         BtB_out => ciBtB_06_out);
53
54     ciBtB07: ciBitToBcd port map(
55         BtB_in(3 downto 1) => ciBtB_05_out(2 downto 0),
56         BtB_in(0) => bit_in(6),
57         BtB_out => ciBtB_07_out);
58
59     ciBtB08: ciBitToBcd port map(
60         BtB_in(3 downto 1) => ciBtB_06_out(2 downto 0),
61         BtB_in(0) => ciBtB_07_out(3),
62         BtB_out => ciBtB_08_out);
63
64     ciBtB09: ciBitToBcd port map(
65         BtB_in(3 downto 1) => ciBtB_07_out(2 downto 0),
66         BtB_in(0) => bit_in(5),
```

```
65          BtB_out => ciBtB_09_out);
66
67  ciBtB10: ciBitToBcd port map(
68      BtB_in(3) => '0',
69      BtB_in(2) => ciBtB_04_out(3),
70      BtB_in(1) => ciBtB_06_out(3),
71      BtB_in(0) => ciBtB_08_out(3),
72      BtB_out => ciBtB_10_out);
73
74  ciBtB11: ciBitToBcd port map(
75      BtB_in(3 downto 1)=> ciBtB_08_out(2 downto 0),
76      BtB_in(0) => ciBtB_09_out(3),
77      BtB_out => ciBtB_11_out);
78
79  ciBtB12: ciBitToBcd port map(
80      BtB_in(3 downto 1)=> ciBtB_09_out(2 downto 0),
81      BtB_in(0) => bit_in(4),
82      BtB_out => ciBtB_12_out);
83
84  ciBtB13: ciBitToBcd port map(
85      BtB_in(3 downto 1)=> ciBtB_10_out(2 downto 0),
86      BtB_in(0) => ciBtB_11_out(3),
87      BtB_out => ciBtB_13_out);
88
89  ciBtB14: ciBitToBcd port map(
90      BtB_in(3 downto 1)=> ciBtB_11_out(2 downto 0),
91      BtB_in(0) => ciBtB_12_out(3),
92      BtB_out => ciBtB_14_out);
93
94  ciBtB15: ciBitToBcd port map(
95      BtB_in(3 downto 1)=> ciBtB_12_out(2 downto 0),
96      BtB_in(0) => bit_in(3),
97      BtB_out => ciBtB_15_out);
98
99  ciBtB16: ciBitToBcd port map(
100     BtB_in(3 downto 1)=> ciBtB_13_out(2 downto 0),
101     BtB_in(0) => ciBtB_14_out(3),
102     BtB_out => ciBtB_16_out);
103
104 ciBtB17: ciBitToBcd port map(
105     BtB_in(3 downto 1)=> ciBtB_14_out(2 downto 0),
106     BtB_in(0) => ciBtB_15_out(3),
107     BtB_out => ciBtB_17_out);
108
109 ciBtB18: ciBitToBcd port map(
110     BtB_in(3 downto 1)=> ciBtB_15_out(2 downto 0),
111     BtB_in(0) => bit_in(2),
112     BtB_out => ciBtB_18_out);
113
114 ciBtB19: ciBitToBcd port map(
115     BtB_in(3 downto 1)=> ciBtB_16_out(2 downto 0),
116     BtB_in(0) => ciBtB_17_out(3),
117     BtB_out => ciBtB_19_out);
118
119 ciBtB20: ciBitToBcd port map(
120     BtB_in(3 downto 1)=> ciBtB_17_out(2 downto 0),
121     BtB_in(0) => ciBtB_18_out(3),
122     BtB_out => ciBtB_20_out);
123
124 ciBtB21: ciBitToBcd port map(
125     BtB_in(3 downto 1)=> ciBtB_18_out(2 downto 0),
126     BtB_in(0) => bit_in(1),
127     BtB_out => ciBtB_21_out);
128
129 bcd_out(15) <= ciBtB_10_out(3);
130 bcd_out(14) <= ciBtB_13_out(3);
```

```
131      bcd_out(13) <= ciBtB_16_out(3);
132      bcd_out(12 downto 9) <= ciBtB_19_out;
133      bcd_out(8 downto 5) <= ciBtB_20_out;
134      bcd_out(4 downto 1) <= ciBtB_21_out;
135      bcd_out(0) <= bit_in(0);
136
137  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 fifo is
5      port (CLK_fifo, WR, RD, reset: in std_logic;
6             W_data: in std_logic_vector(12 downto 0);
7             em,fu: out std_logic;
8             R_data: out std_logic_vector(12 downto 0);
9             Estados_maquina: out std_logic_vector(2 downto 0));
10 end fifo;
11
12 architecture ckt of fifo is
13
14     component blocoDeControleFIFO is
15         port (clk_ff, rst_ff, rd_ff, wr_ff: in std_logic;
16                equal_cont_wd_e_rd, equal_cont_wd_1_e_rd, equal_cont_wd_e_rd_1: in std_logic;
17                led_vazio_ff, led_cheio_ff,load_wr,load_rd,clr_ff: out std_logic;
18                saida_mv_fifo: out std_logic_vector(2 downto 0));
19     end component;
20
21     component datapathFIFO is
22         port (wr_data_dp: in std_logic_vector(12 downto 0);
23                ld_wr_dp, ld_rd_dp, clr_fifo_dp, clk_fifo_dp: in std_logic;
24                eq_comp_wr_rd,eq_comp_wrl_rd,eq_comp_wr_rdl: out std_logic;
25                rd_data_dp: out std_logic_vector(12 downto 0));
26     end component;
27
28
29     signal leitura_banco_registradores: std_logic_vector(12 downto 0);
30     signal l_vazio, l_cheio, saida_load_wr, saida_load_rd, clr_dt : std_logic;
31     signal saida_eq_cont_wr_rd, saida_eq_cont_wrl_rd, saida_eq_cont_wr_rdl: std_logic;
32     signal saida_fifo: std_logic_vector(2 downto 0);
33
34 begin
35     BlocodeControle: blocoDeControleFIFO port map(
36         clk_ff => CLK_fifo,
37         rst_ff => reset,
38         rd_ff => RD,
39         wr_ff => WR,
40         equal_cont_wd_e_rd => saida_eq_cont_wr_rd,
41         equal_cont_wd_1_e_rd => saida_eq_cont_wrl_rd,
42         equal_cont_wd_e_rd_1 => saida_eq_cont_wr_rdl,
43         led_vazio_ff => l_vazio,
44         led_cheio_ff => l_cheio,
45         load_wr => saida_load_wr,
46         load_rd => saida_load_rd,
47         clr_ff => clr_dt,
48         saida_mv_fifo => saida_fifo);
49
50     Datapath: datapathFIFO port map(
51         clk_fifo_dp => CLK_fifo,
52         wr_data_dp => W_data,
53         ld_wr_dp => saida_load_wr,
54         ld_rd_dp => saida_load_rd,
55         clr_fifo_dp => clr_dt,
56         eq_comp_wr_rd => saida_eq_cont_wr_rd,
57         eq_comp_wrl_rd => saida_eq_cont_wrl_rd,
58         eq_comp_wr_rdl => saida_eq_cont_wr_rdl,
59         rd_data_dp => leitura_banco_registradores);
60
61
62
63     R_data <= leitura_banco_registradores;
64     em <= l_vazio;
65     fu <= l_cheio;
66     Estados_maquina <= saida_fifo;
```

```
67  
68     end ckt ;
```

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity blocoDeControleFIFO is
5      port (clk_ff , rst_ff, rd_ff, wr_ff: in std_logic;
6              equal_cont_wd_e_rd, equal_cont_wd_1_e_rd, equal_cont_wd_e_rd_1: in std_logic;
7              led_vazio_ff, led_cheio_ff,load_wr,load_rd, clr_ff: out std_logic;
8              saida_mv_fifo: out std_logic_vector(2 downto 0));
9  end blocoDeControleFIFO;
10
11 architecture ckt of blocoDeControleFIFO is
12     type st is (start, p_leitura, escrita, p_escrita, leitura);
13     signal estado : st;
14     signal p_l,p_e: std_logic;
15 begin
16     process (clk_ff , rst_ff)
17     begin
18         if rst_ff = '1' then
19             estado <= start ;
20         elsif (clk_ff'event and clk_ff ='1') then
21             case estado is
22                 when start =>
23                     estado <= p_leitura;
24                 when p_leitura =>
25                     if (equal_cont_wd_e_rd='0') and (rd_ff ='1') and (wr_ff ='0') then
26                         estado <= leitura;
27                     elsif (wr_ff='1') then estado <= escrita;
28                     else estado <= p_leitura;
29                     end if;
30                 when escrita =>
31                     if equal_cont_wd_1_e_rd='1' then estado <= p_escrita;
32                     elsif (wr_ff='1') and (rd_ff='0') then estado <= escrita;
33                     else estado <= p_escrita;
34                     end if;
35                 when p_escrita =>
36                     if (equal_cont_wd_e_rd='0') and (wr_ff ='1') and (rd_ff='0') then estado
37                         <= escrita;
38                     elsif (rd_ff='1') then estado <= leitura;
39                     else estado <= p_escrita;
40                     end if;
41                 when leitura =>
42                     if equal_cont_wd_e_rd_1='1' then estado <= p_leitura;
43                     elsif (wr_ff='0') and (rd_ff='1') then estado <= leitura;
44                     else estado <= p_leitura;
45                     end if;
46             end case ;
47         end if;
48     end process;
49     clr_ff <= '1' when estado = start else '0';
50     load_wr <= '1' when estado = escrita else '0';
51     load_rd <= '1' when estado = leitura else '0';
52     p_e <= '1' when estado = p_escrita else '0';
53     p_l <= '1' when estado = p_leitura else '0';
54
55     led_vazio_ff <= p_l and equal_cont_wd_e_rd;
56     led_cheio_ff <= p_e and equal_cont_wd_e_rd;
57
58     with estado select
59         saida_mv_fifo <= "000" when start ,
60         "001" when p_leitura,
61         "010" when escrita,
62         "011" when p_escrita,
63         "100" when leitura;
64 end ckt ;
```

```

1  library ieee ;
2  use ieee.std_logic_1164.all;
3
4  entity datapathFIFO is
5      port (wr_data_dp: in std_logic_vector(12 downto 0);
6             ld_wr_dp, ld_rd_dp, clk_fifo_dp: in std_logic;
7             eq_comp_wr_rd,eq_comp_wrl_rd,eq_comp_wr_rdl: out std_logic;
8             rd_data_dp: out std_logic_vector(12 downto 0));
9  end datapathFIFO;
10
11 architecture ckt of datapathFIFO is
12     component bancoDeRegistrador16X13 is
13         port (wr_data: in std_logic_vector(12 downto 0);
14                cont_wr,cont_rd: in std_logic_vector(3 downto 0);
15                clk_br, ld_wr, ld_rd, clr_reg: in std_logic;
16                rd_data: out std_logic_vector(12 downto 0));
17     end component;
18
19     component Comparador_4Bits is
20         port (eA,eB: in std_logic_vector(3 downto 0);
21                gt,lt,eq: in std_logic;
22                AeqB,AltB,AgtB: out std_logic);
23     end component;
24
25     component contador4Bits is
26         port ( clk_c4B,ld_c4B,clr_c4B: in std_logic;
27                out_c4B: out std_logic_vector(3 downto 0));
28     end component;
29
30     component SUM_4Bits is
31         port (A4_in,B4_in: in std_logic_vector(3 downto 0);
32                C4_in: in std_logic;
33                S4_out: out std_logic_vector(3 downto 0);
34                C4_out: out std_logic);
35     end component;
36
37     signal saida_cont_wr, saida_cont_rd: std_logic_vector(3 downto 0);
38     signal saida_sum_wr, saida_sum_rd: std_logic_vector(3 downto 0);
39     signal saida_bancoRegistros: std_logic_vector(12 downto 0);
40     signal clr_reverso_reg: std_logic;
41     signal lixo: std_logic_vector(1 downto 0);
42     signal saida_eq_comparador_wd_com_rd, saida_ld_comparador_wd_com_rd,
43     saida_gt_comparador_wd_com_rd: std_logic;
44     signal saida_eq_comparador_wd1_com_rd, saida_ld_comparador_wd1_com_rd,
45     saida_gt_comparador_wd1_com_rd: std_logic;
46     signal saida_eq_comparador_wd_com_rdl, saida_ld_comparador_wd_com_rdl,
47     saida_gt_comparador_wd_com_rdl: std_logic;
48
49 begin
50
51     clr_reverso_reg <= not clr_fifo_dp;
52
53     BancoDeRegistradores: bancoDeRegistrador16X13 port map(
54         wr_data => wr_data_dp,
55         cont_wr => saida_cont_wr,
56         cont_rd => saida_cont_rd,
57         clk_br => clk_fifo_dp,
58         ld_wr => ld_wr_dp,
59         ld_rd => ld_rd_dp,
60         clr_reg => clr_reverso_reg,
61         rd_data => saida_bancoRegistros);
62
63     Contador_WR: contador4Bits port map(
64         clk_c4B => clk_fifo_dp,
65         ld_c4B => ld_wr_dp,
66         clr_c4B => clr_fifo_dp,

```

```
64          out_c4B => saida_cont_wr);
65
66 Contador_RD: contador4Bits port map(
67     clk_c4B => clk_fifo_dp,
68     ld_c4B => ld_rd_dp,
69     clr_c4B => clr_fifo_dp,
70     out_c4B => saida_cont_rd);
71
72 comparador_wr_com_rd: Comparador_4Bits port map(
73     eA => saida_cont_wr,
74     eB => saida_cont_rd,
75     gt => '0',
76     lt => '0',
77     eq => '1',
78     AeqB => saida_eq_comparador_wd_com_rd,
79     AltB => saida_ld_comparador_wd_com_rd,
80     AgtB => saida_gt_comparador_wd_com_rd;
81
82 comparador_wrl_com_rd: Comparador_4Bits port map(
83     eA => saida_sum_wr,
84     eB => saida_cont_rd,
85     gt => '0',
86     lt => '0',
87     eq => '1',
88     AeqB => saida_eq_comparador_wd1_com_rd,
89     AltB => saida_ld_comparador_wd1_com_rd,
90     AgtB => saida_gt_comparador_wd1_com_rd;
91
92 comparador_wr_com_rdl: Comparador_4Bits port map(
93     eA => saida_cont_wr,
94     eB => saida_sum_rd,
95     gt => '0',
96     lt => '0',
97     eq => '1',
98     AeqB => saida_eq_comparador_wd_com_rdl,
99     AltB => saida_ld_comparador_wd_com_rdl,
100    AgtB => saida_gt_comparador_wd_com_rdl);
101
102 somador_wrl: SUM_4Bits port map(
103     A4_in => saida_cont_wr,
104     B4_in => "0001",
105     C4_in => '0',
106     S4_out => saida_sum_wr,
107     C4_out => lixo(0));
108
109 somador_rdl: SUM_4Bits port map(
110     A4_in => saida_cont_rd,
111     B4_in => "0001",
112     C4_in => '0',
113     S4_out => saida_sum_rd,
114     C4_out => lixo(1));
115
116
117 rd_data_dp <= saida_bancoRegistros;
118 eq_comp_wr_rd <= saida_eq_comparador_wd_com_rd;
119 eq_comp_wrl_rd <= saida_eq_comparador_wd1_com_rd;
120 eq_comp_wr_rdl <= saida_eq_comparador_wd_com_rdl;
121
122 end ckt;
```

```
1  library ieee ;
2  use ieee.std_logic_1164.all;
3
4  entity bancoDeRegistrador16X13 is
5      port (wr_data: in std_logic_vector(12 downto 0);
6             cont_wr,cont_rd: in std_logic_vector(3 downto 0);
7             clk_br, ld_wr, ld_rd, clr_reg: in std_logic;
8             rd_data: out std_logic_vector(12 downto 0));
9  end bancoDeRegistrador16X13;
10
11 architecture ckt of bancoDeRegistrador16X13 is
12     component MUX16_1_13Bits is
13         port (I15_16, I14_16, I13_16, I12_16, I11_16, I10_16, I9_16, I8_16, I7_16:
14             std_logic_vector(12 downto 0);
15             I6_16, I5_16, I4_16, I3_16, I2_16, I1_16, I0_16: std_logic_vector(12 downto 0);
16             S_16: in std_logic_vector(3 downto 0);
17             ld_mux_16: in std_logic;
18             d_16: out std_logic_vector(12 downto 0));
19     end component;
20
21     component decodificador1X16 is
22         port (ld_dec: in std_logic;
23                i_in: in std_logic_vector(3 downto 0);
24                d_out: out std_logic_vector(15 downto 0));
25     end component;
26
27     component reg13Bits is
28         port (clk,preSet,clr,load: in std_logic;
29                d: in std_logic_vector(12 downto 0);
30                q: out std_logic_vector(12 downto 0));
31     end component;
32
33     signal saida_reg_00, saida_reg_01, saida_reg_02, saida_reg_03, saida_reg_04,
34     saida_reg_05, saida_reg_06, saida_reg_07: std_logic_vector(12 downto 0);
35     signal saida_reg_08, saida_reg_09, saida_reg_10, saida_reg_11, saida_reg_12,
36     saida_reg_13, saida_reg_14, saida_reg_15, saida_mux: std_logic_vector(12 downto 0);
37     signal saida_decod: std_logic_vector(15 downto 0);
38
39 begin
40     Dec_wr: decodificador1X16 port map(
41         ld_dec => ld_wr,
42         i_in => cont_wr,
43         d_out => saida_decod);
44
45     Reg00: reg13Bits port map(
46         clk => clk_br,
47         preSet => '1',
48         clr => clr_reg,
49         load => saida_decod(0),
50         d => wr_data,
51         q => saida_reg_00);
52
53     Reg01: reg13Bits port map(
54         clk => clk_br,
55         preSet => '1',
56         clr => clr_reg,
57         load => saida_decod(1),
58         d => wr_data,
59         q => saida_reg_01);
60
61     Reg02: reg13Bits port map(
62         clk => clk_br,
63         preSet => '1',
64         clr => clr_reg,
65         load => saida_decod(2),
```

```
64          d => wr_data,
65          q => saida_reg_02);
66
67  Reg03: reg13Bits port map(
68      clk => clk_br,
69      preSet => '1',
70      clr => clr_reg,
71      load => saida_decod(3),
72      d => wr_data,
73      q => saida_reg_03);
74
75  Reg04: reg13Bits port map(
76      clk => clk_br,
77      preSet => '1',
78      clr => clr_reg,
79      load => saida_decod(4),
80      d => wr_data,
81      q => saida_reg_04);
82
83  Reg05: reg13Bits port map(
84      clk => clk_br,
85      preSet => '1',
86      clr => clr_reg,
87      load => saida_decod(5),
88      d => wr_data,
89      q => saida_reg_05);
90
91  Reg06: reg13Bits port map(
92      clk => clk_br,
93      preSet => '1',
94      clr => clr_reg,
95      load => saida_decod(6),
96      d => wr_data,
97      q => saida_reg_06);
98
99  Reg07: reg13Bits port map(
100     clk => clk_br,
101     preSet => '1',
102     clr => clr_reg,
103     load => saida_decod(7),
104     d => wr_data,
105     q => saida_reg_07);
106
107 Reg08: reg13Bits port map(
108     clk => clk_br,
109     preSet => '1',
110     clr => clr_reg,
111     load => saida_decod(8),
112     d => wr_data,
113     q => saida_reg_08);
114
115 Reg09: reg13Bits port map(
116     clk => clk_br,
117     preSet => '1',
118     clr => clr_reg,
119     load => saida_decod(9),
120     d => wr_data,
121     q => saida_reg_09);
122
123 Reg10: reg13Bits port map(
124     clk => clk_br,
125     preSet => '1',
126     clr => clr_reg,
127     load => saida_decod(10),
128     d => wr_data,
129     q => saida_reg_10);
```

```
130
131     Reg11: reg13Bits port map(
132         clk => clk_br,
133         preSet => '1',
134         clr => clr_reg,
135         load => saida_decod(11),
136         d => wr_data,
137         q => saida_reg_11);
138
139     Reg12: reg13Bits port map(
140         clk => clk_br,
141         preSet => '1',
142         clr => clr_reg,
143         load => saida_decod(12),
144         d => wr_data,
145         q => saida_reg_12);
146
147     Reg13: reg13Bits port map(
148         clk => clk_br,
149         preSet => '1',
150         clr => clr_reg,
151         load => saida_decod(13),
152         d => wr_data,
153         q => saida_reg_13);
154
155     Reg14: reg13Bits port map(
156         clk => clk_br,
157         preSet => '1',
158         clr => clr_reg,
159         load => saida_decod(14),
160         d => wr_data,
161         q => saida_reg_14);
162
163     Reg15: reg13Bits port map(
164         clk => clk_br,
165         preSet => '1',
166         clr => clr_reg,
167         load => saida_decod(15),
168         d => wr_data,
169         q => saida_reg_15);
170
171     MUX_rd: mux16_1_13Bits port map(
172         I15_16 => saida_reg_15,
173         I14_16 => saida_reg_14,
174         I13_16 => saida_reg_13,
175         I12_16 => saida_reg_12,
176         I11_16 => saida_reg_11,
177         I10_16 => saida_reg_10,
178         I9_16 => saida_reg_09,
179         I8_16 => saida_reg_08,
180         I7_16 => saida_reg_07,
181         I6_16 => saida_reg_06,
182         I5_16 => saida_reg_05,
183         I4_16 => saida_reg_04,
184         I3_16 => saida_reg_03,
185         I2_16 => saida_reg_02,
186         I1_16 => saida_reg_01,
187         I0_16 => saida_reg_00,
188         S_16 => cont_rd,
189         ld_mux_16 => ld_rd,
190         d_16 => saida_mux);
191
192     rd_data <= saida_mux;
193 end ckt;
```

```
1  library ieee ;
2  use ieee.std_logic_1164.all;
3
4  entity decodificador1X16 is
5      port (ld_dec: in std_logic;
6             i_in: in std_logic_vector(3 downto 0);
7             d_out: out std_logic_vector(15 downto 0));
8  end decodificador1X16;
9
10 architecture ckt of decodificador1X16 is
11 begin
12     d_out(0) <= (not (i_in(3))) and (not(i_in(2))) and (not (i_in(1))) and (not(i_in(0)))
13     and ld_dec;
14     d_out(1) <= (not (i_in(3))) and (not(i_in(2))) and (not (i_in(1))) and      (i_in(0))
15     and ld_dec;
16     d_out(2) <= (not (i_in(3))) and (not(i_in(2))) and      (i_in(1)) and (not(i_in(0)))
17     and ld_dec;
18     d_out(3) <= (not (i_in(3))) and (not(i_in(2))) and      (i_in(1)) and      (i_in(0))
19     and ld_dec;
20     d_out(4) <= (not (i_in(3))) and      (i_in(2)) and (not (i_in(1))) and (not(i_in(0)))
21     and ld_dec;
22     d_out(5) <= (not (i_in(3))) and      (i_in(2)) and (not (i_in(1))) and      (i_in(0))
23     and ld_dec;
24     d_out(6) <= (not (i_in(3))) and      (i_in(2)) and      (i_in(1)) and (not(i_in(0)))
25     and ld_dec;
26     d_out(7) <= (not (i_in(3))) and      (i_in(2)) and      (i_in(1)) and      (i_in(0))
27     and ld_dec;
28     d_out(8) <=      (i_in(3)) and (not(i_in(2))) and (not (i_in(1))) and (not(i_in(0)))
29     and ld_dec;
30     d_out(9) <=      (i_in(3)) and (not(i_in(2))) and (not (i_in(1))) and      (i_in(0))
31     and ld_dec;
32     d_out(10) <=      (i_in(3)) and (not(i_in(2))) and      (i_in(1)) and (not(i_in(0)))
33     and ld_dec;
34     d_out(11) <=      (i_in(3)) and (not(i_in(2))) and      (i_in(1)) and      (i_in(0))
35     and ld_dec;
36     d_out(12) <=      (i_in(3)) and      (i_in(2)) and (not (i_in(1))) and (not(i_in(0)))
37     and ld_dec;
38     d_out(13) <=      (i_in(3)) and      (i_in(2)) and (not (i_in(1))) and      (i_in(0))
39     and ld_dec;
40     d_out(14) <=      (i_in(3)) and      (i_in(2)) and      (i_in(1)) and (not(i_in(0)))
41     and ld_dec;
42     d_out(15) <=      (i_in(3)) and      (i_in(2)) and      (i_in(1)) and      (i_in(0))
43     and ld_dec;
44
45 end ckt;
```

```
1  library ieee ;
2  use ieee.std_logic_1164.all;
3
4  entity reg13Bits is
5      port (clk,preSet,clr,load: in std_logic;
6             d: in std_logic_vector(12 downto 0);
7             q : out std_logic_vector(12 downto 0));
8  end reg13Bits;
9  architecture ckt of reg13Bits is
10
11     signal qs: std_logic_vector(12 downto 0);
12
13 begin
14     process (clk ,preSet,clr)
15     begin
16         if preSet = '0' then qs <= "111111111111";
17         elsif clr = '0' then qs <= "000000000000";
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 MUX16_1_13Bits is
5      port (I15_16, I14_16, I13_16, I12_16, I11_16, I10_16, I9_16, I8_16, I7_16:
6          std_logic_vector(12 downto 0);
7          I6_16, I5_16, I4_16, I3_16, I2_16, I1_16, I0_16: std_logic_vector(12 downto 0);
8          S_16: in std_logic_vector(3 downto 0);
9          ld_mux_16: in std_logic;
10         d_16: out std_logic_vector(12 downto 0));
11 end MUX16_1_13Bits;
12
13 Architecture ckt of MUX16_1_13Bits is
14     component MUX2_1_13Bits is
15         port (I_1,I_0: in std_logic_vector(12 downto 0);
16                S,ld_mux: in std_logic;
17                d: out std_logic_vector(12 downto 0));
18     end component;
19
20     signal saida_i0i1, saida_i2i3, saida_i4i5, saida_i6i7, saida_i8i9, saida_i10i11,
21     saida_i12i13, saida_i14i15: std_logic_vector(12 downto 0);
22     signal saida_S0102, saida_S0304, saida_S0506, saida_S0708 : std_logic_vector(12 downto 0)
23 );
24     signal saida_SS0102, saida_SS0304, saida_final : std_logic_vector(12 downto 0);
25
26 begin
27     muxI0I1: MUX2_1_13Bits port map(
28         I_1 => I1_16,
29         I_0 => I0_16,
30         S => S_16(0),
31         ld_mux => ld_mux_16,
32         d => saida_i0i1);
33
34     muxI2I3: MUX2_1_13Bits port map(
35         I_1 => I3_16,
36         I_0 => I2_16,
37         S => S_16(0),
38         ld_mux => ld_mux_16,
39         d => saida_i2i3);
40
41     muxI4I5: MUX2_1_13Bits port map(
42         I_1 => I5_16,
43         I_0 => I4_16,
44         S => S_16(0),
45         ld_mux => ld_mux_16,
46         d => saida_i4i5);
47
48     muxI6I7: MUX2_1_13Bits port map(
49         I_1 => I7_16,
50         I_0 => I6_16,
51         S => S_16(0),
52         ld_mux => ld_mux_16,
53         d => saida_i6i7);
54
55     muxI8I9: MUX2_1_13Bits port map(
56         I_1 => I9_16,
57         I_0 => I8_16,
58         S => S_16(0),
59         ld_mux => ld_mux_16,
60         d => saida_i8i9);
61
62     muxI10I11: MUX2_1_13Bits port map(
63         I_1 => I11_16,
64         I_0 => I10_16,
65         S => S_16(0),
66         ld_mux => ld_mux_16,
```

```
64          d => saida_i10i11);  
65  
66      muxI12I13: MUX2_1_13Bits port map(  
67          I_1 => I13_16,  
68          I_0 => I12_16,  
69          S => S_16(0),  
70          ld_mux => ld_mux_16,  
71          d => saida_i12i13);  
72  
73      muxI14I15: MUX2_1_13Bits port map(  
74          I_1 => I15_16,  
75          I_0 => I14_16,  
76          S => S_16(0),  
77          ld_mux => ld_mux_16,  
78          d => saida_i14i15);  
79  
80      muxSaida0102: MUX2_1_13Bits port map(  
81          I_1 => saida_i2i3,  
82          I_0 => saida_i0i1,  
83          S => S_16(1),  
84          ld_mux => ld_mux_16,  
85          d => saida_S0102);  
86  
87      muxSaida0304: MUX2_1_13Bits port map(  
88          I_1 => saida_i6i7,  
89          I_0 => saida_i4i5,  
90          S => S_16(1),  
91          ld_mux => ld_mux_16,  
92          d => saida_S0304);  
93  
94      muxSaida0506: MUX2_1_13Bits port map(  
95          I_1 => saida_i10i11,  
96          I_0 => saida_i8i9,  
97          S => S_16(1),  
98          ld_mux => ld_mux_16,  
99          d => saida_S0506);  
100  
101     muxSaida0708: MUX2_1_13Bits port map(  
102          I_1 => saida_i14i15,  
103          I_0 => saida_i12i13,  
104          S => S_16(1),  
105          ld_mux => ld_mux_16,  
106          d => saida_S0708);  
107  
108     muxSSaida0102: MUX2_1_13Bits port map(  
109          I_1 => saida_S0304,  
110          I_0 => saida_S0102,  
111          S => S_16(2),  
112          ld_mux => ld_mux_16,  
113          d => saida_SS0102);  
114  
115     muxSSaida0304: MUX2_1_13Bits port map(  
116          I_1 => saida_S0708,  
117          I_0 => saida_S0506,  
118          S => S_16(2),  
119          ld_mux => ld_mux_16,  
120          d => saida_SS0304);  
121     muxFinal: MUX2_1_13Bits port map(  
122          I_1 => saida_SS0304,  
123          I_0 => saida_SS0102,  
124          S => S_16(3),  
125          ld_mux => ld_mux_16,  
126          d => saida_final);  
127  
128  
129     d_16 <= saida_final;
```

```
130      end ckt;
```

```
1  library ieee ;
2  use ieee.std_logic_1164.all;
3
4  entity MUX2_1_13Bits is
5      port (I_1,I_0: std_logic_vector(12 downto 0);
6             S, ld_mux: in std_logic;
7             d: out std_logic_vector(12 downto 0));
8  end MUX2_1_13Bits;
9
10 Architecture ckt of MUX2_1_13Bits is
11
12 Begin
13     d(0) <= (((not S) and I_0(0)) or (S and I_1(0))) and ld_mux;
14     d(1) <= (((not S) and I_0(1)) or (S and I_1(1))) and ld_mux;
15     d(2) <= (((not S) and I_0(2)) or (S and I_1(2))) and ld_mux;
16     d(3) <= (((not S) and I_0(3)) or (S and I_1(3))) and ld_mux;
17     d(4) <= (((not S) and I_0(4)) or (S and I_1(4))) and ld_mux;
18     d(5) <= (((not S) and I_0(5)) or (S and I_1(5))) and ld_mux;
19     d(6) <= (((not S) and I_0(6)) or (S and I_1(6))) and ld_mux;
20     d(7) <= (((not S) and I_0(7)) or (S and I_1(7))) and ld_mux;
21     d(8) <= (((not S) and I_0(8)) or (S and I_1(8))) and ld_mux;
22     d(9) <= (((not S) and I_0(9)) or (S and I_1(9))) and ld_mux;
23     d(10) <= (((not S) and I_0(10)) or (S and I_1(10))) and ld_mux;
24     d(11) <= (((not S) and I_0(11)) or (S and I_1(11))) and ld_mux;
25     d(12) <= (((not S) and I_0(12)) or (S and I_1(12))) and ld_mux;
26
27 end ckt;
```

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity contador4Bits is
5      port ( clk_c4B,ld_c4B,clr_c4B: in std_logic;
6             out_c4B: out std_logic_vector(3 downto 0));
7  end contador4Bits ;
8  architecture ckt of contador4Bits 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, clr_reverso: std_logic;
15     signal resp_and: std_logic_vector(2 downto 0);
16
17 begin
18
19     clr_reverso <= not clr_c4B;
20     Q0: ffjk port map(
21         clk => clk_c4B,
22         J => ld_c4B,
23         K => ld_c4B,
24         P => '1',
25         C => clr_reverso,
26         q => q0_out);
27
28     resp_and(0) <= q0_out and ld_c4B;
29
30     Q1: ffjk port map(
31         clk => clk_c4B,
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_c4B,
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_c4B,
52         J => resp_and(2),
53         K => resp_and(2),
54         P => '1',
55         C => clr_reverso,
56         q => q3_out);
57
58     out_c4B(0) <= q0_out;
59     out_c4B(1) <= q1_out;
60     out_c4B(2) <= q2_out;
61     out_c4B(3) <= q3_out;
62
63 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 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;
```

```
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 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_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_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)
13 ) and A_in) or (C_in and (not A_in))));  
14     C_out <= ((C_in and (A_in or B_in)) or (A_in and B_in));
15 End ckt;
```