UNIVERSIDADE FEDERAL DE MINAS GERAIS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

INTRODUÇÃO AOS SISTEMAS LÓGICOS

Laboratório de Hardware 3

Fatoração em *Verilog* e Máximo Divisor Comum *(MDC)*

Grupo 11
Christian Vieira
João Pedro
Marcus Oliveira
Paula Jeniffer

Professor: Antônio Otávio Fernandes
Monitor: Omar Vidal Pino

Belo Horizonte
22 de junho de 2016

Sumário

Lista de Figuras

Lista de implementações

Lista de Tabelas

1 Objetivo das atividades:

O *"Laboratório de hardware 3"* teve como objetivo o primeiro contato com *Lógica Programável* utilizando a *linguagem de descrição de hardware Verilog*. O ambiente de programação utilizado foi o *software Quartus II 64-Bit Version 13.0.0 Build 156 04/24/2013 SJ Web Edition* (Altera, 2016). Após a síntese concluída pela ferramenta, a placa de desenvolvimento *Altera DE2 Board* (Terasic, 2016) foi utilizada para testes e depuração.

2 Máximo Divisor Comum (MDC) utilizando o *Algoritmo de Euclides*:

Esta atividade consistiu na implementação do *Algoritmo de Euclides* utilizando a *linguagem de descrição de hardware Verilog*. Uma vez dado duas entradas (através da manipulação das chaves do kit de desenvolvimento), a saída consiste no *MDC* dos dois números. Assim como as entradas, o valor da saída é exibido nos *displays de 7-segmentos* do *kit de desenvolvimento*. As figuras 1 e 2 abaixo, exibem respectivamente a arquitetura utilizada para cálculo do *MCD* e o *hardware alvo*. Diz-se que em matemática, o *Algoritmo de Euclides* consiste em um método simples e eficiente de

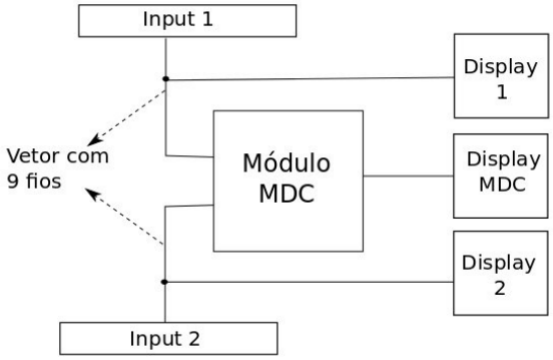

Figura 1 Arquitetura para cálculo do *MDC*

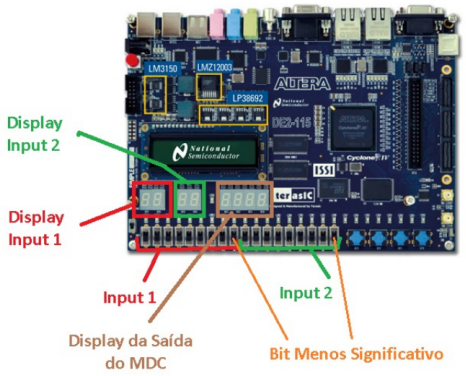

Figura 2 *Hardware* alvo: Altera DE2 Board

encontrar um *Máximo Divisor Comum* entre dois números inteiros diferentes de zero. Considera-se o *Algoritmo de Euclides* um dos *algoritmos* mais antigos conhecidos desde que surgiu nos *Livros VII* e *X* da obra *Elementos de Euclides* aproximadamente em *300 a.C.* A principal vantagem do *Algoritmo de Euclides* consiste no fato de que não é exigido qualquer tipo de fatoração(Wikipedia, 2016a).

O *Algoritmo de Euclides* implementado foi baseado na versão considerada como sendo original, baseada em subtrações repetidas. Abaixo, segue algoritmo o qual baseou-se a implementação:

```
function gcd(a, b)
    while a != b
        if a > b
           a := a - b;
        else
           b := b - a;
    return a;
```

A implementação do *MCD* utilizando o *Algoritmo de Euclides* poderá ser encontrada no apêndice B. Nos apêndices A.1 e A.2, encontram-se as descrições de *Conversão de Binário para BCD* e do *Decodificador Hexadecimal para Display de 7-Segmentos*. A seguir, tem-se o automatizador de execução C.1 seguido da descrição do topo da hierarquia B.2, a descrição da *engine* que efetua o cálculo do *MDC* através de subtrações sucessivas (utilizando uma *máquina de estados finita*) B.3

e o *testbench*B.4 para verificação da descrição e geração das formas de onda quando utilizado o ambiente *Icarus Verilog* e *gtkwave*.

3 Fatoração de inteiro

O problema da *fatoração* (ou ainda *fatoração de inteiros*) consiste em encontrar um divisor não trivial de um número composto. Por exemplo, dado o número 91, o objetivo é encontrar um número tal como 7 que o divida. Se esses inteiros estão restritos a números primos, este processo é chamado de *fatoração por números primos* (ou ainda *fatoração prima*). (Wikipedia, 2016b).

O problema computacional que é a *fatoração de inteiros* para números extremamente grandes tem motivado diversos estudos devido a sua aplicação em sistemas de criptografia. Neste trabalho, a fatoração de inteiros foi realizada através de repetidas extrações de divisão modular, onde os quatro primeiros números primos (caso existam) foram armazenados em um *registro* para posterior exibição nos *displays de 7-segmentos* do *hardware alvo*.

A implementação da *fatoração prima* utilizando divisão modular poderá ser encontrada no apêndice C. Nos apêndices A.1 e A.2, encontram-se as descrições de *Conversão de Binário para BCD* e do *Decodificador Hexadecimal para Display de 7-Segmentos*. A seguir, tem-se o automatizador de execução C.1 seguido da descrição do topo da hierarquia C.2, a descrição da *engine* que efetua o extrai os quatro primeiros fatores primos utilizando uma *máquina de estados finita*) C.3 e o *testbench*C.4 para verificação da descrição e geração das formas de onda quando utilizado o ambiente *Icarus Verilog* e *gtkwave*.

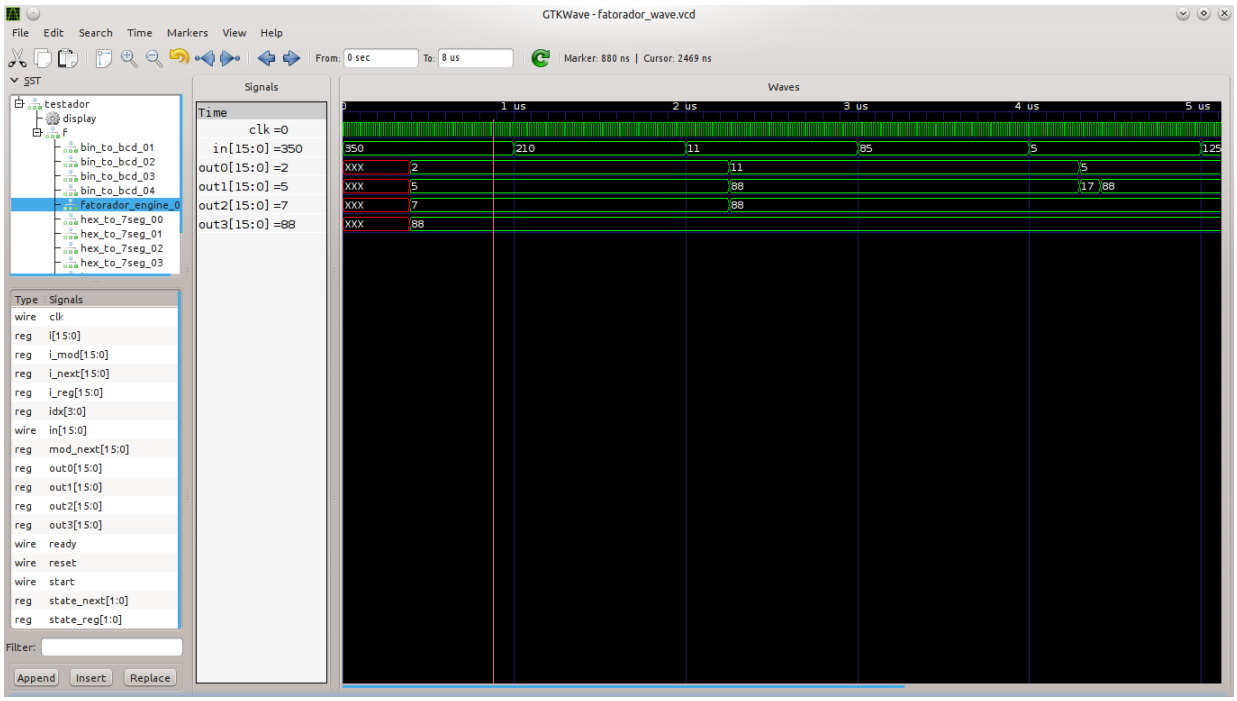

Figura 3 Formas de ondas da implementação da fatoração de inteiros

4 Conclusões

As execuções das atividades previstas (*Cálculo do MDC* e *Fatoração de inteiros*) proporcionaram um primeiro contato com a *descrição e síntese de hardware* utilizando *lógica programável*, sendo o *hardware alvo* a placa de desenvolvimento *Altera DE2 Board* (Terasic, 2016). Desta forma, foi verificado de forma prática os conceitos desenvolvidos ao longo do semestre na disciplina de *Introdução aos Sistemas Lógicos*.

A quebra do *paradigma* habitual de programação como sendo uma seqüência linear de execução inicialmente foi difícil de ser conseguida, uma vez que o objetivo da *descrição de hardware*, como o próprio nome indica, consiste no uso de uma linguagem apropriada (*Verilog, VHDL, SystemVerilog, etc*) que descreve o *comportamento* e/ou *estrutura* de *hardware*. Dessa maneira, foi um desafio transformar um dado algoritmo em uma estrutura de *hardware* que pudesse ser descrita utilizando uma *linguagem de descrição de hardware* (neste trabalho, utilizou-se a linguagem *Verilog* tanto para descrição quanto para a geração do *testbench*).

Opcionalmente foi utilizado os pacotes de *software livre Icarus Verilog*(WILLIAMS, 2016) e *gtkwave*(BYBELL, 2016) respectivamente para verificação/síntese e visualização das formas de ondas do *testbench* gerado. Somente após a confirmação do funcionamento esperado das implementações, as mesmas foram sintetizadas e gravadas no *hardware alvo* utilizando a ferramenta do fabricante *(Altera Quartus II v13.0.0)* do componente de lógica programável.

A instalação do ambiente de desenvolvimento *Altera Quartus II v13.0.0* no *O.S. GNU/Linux Ubuntu 14.04* em uma máquina de *64-bits* ocorreu sem problemas. Entretanto, o gravador não era detectado quando à placa de desenvolvimento era conectada ao computador. A solução encontrada foi a instalação de um *device driver* genérico com suporte ao *JTAG* utilizando como meio a interface *USB*. Após a instalação do *device driver* e a configuração da porta *USB* utilizando o *ID* identificado com o comando no terminal: `lsusb`, foi possível a gravação da placa.

O uso dos *softwares livres Icarus Verilog* e *gtkwave* possibilitaram maior agilidade no processo de desenvolvimento, uma vez que os mesmos eram invocados a partir de um automatizador de execução (utilitário *make*). Apesar de ser possível tal automação com o *software Altera Quartus II*, o mesmo não foi realizado devido a necessidade de se conhecer mais o ambiente bem como os comandos necessários para sere invocados quando na execução do utilitário *make*.

5 Referências Bibliográficas

Altera. *Quartus II v13.0.0 Web Edition*. 2016. [Online; acessado em 21/06/2016]. Disponível em: <https://www.altera.com/downloads/download-center.html>.

BYBELL, T. *gtkwave*. 2016. [Online; acessado em 23/05/2016]. Disponível em: <https://sourceforge.net/projects/gtkwave/>.

Terasic. *Altera DE2 Board*. 2016. [Online; acessado em 21/06/2016]. Disponível em: <https://www.terasic.com.tw/cgi-bin/page/archive.pl?No=30>.

Wikipedia. *Euclidean Algorithm*. 2016. [Online; acessado em 21/06/2016]. Disponível em: <https://pt.wikipedia.org/wiki/Algoritmo_de_Euclides>.

Wikipedia. *Integer factorization*. 2016. [Online; acessado em 21/06/2016]. Disponível em: <https://en.wikipedia.org/wiki/Integer_factorization>.

WILLIAMS, S. *Icarus Verilog*. 2016. [Online; acessado em 23/05/2016]. Disponível em: <http://iverilog.icarus.com/>.

Apêndice A Descrição comum às implementações:

A.1 Conversor Binário para *BCD*

```
1  module bin_to_bcd (
2      input [7:0] binary,
3      output reg [3:0] tens,
4      output reg [3:0] ones
5    );
6
7    integer i;
8    always @(binary) begin
9      tens = 4'd0;
10     ones = 4'd0;
11
12     for(i = 7; i >= 0; i = i - 1) begin
13       if(tens >= 5)
14         tens = tens + 3;
15       if(ones >= 5)
16         ones = ones + 3;
17
18       tens = tens << 1;
19       tens[0] = ones[3];
20       ones = ones << 1;
21       ones[0] = binary[i];
22     end
23   end
24 endmodule
```

Listagem 1 Conversor Binário para *BCD*

A.2 Conversor Hexadecimal para *display de 7-segmentos*

```
1  module hex_to_7seg(hex_digit, seg);
2  input [3:0] hex_digit;
3  output [6:0] seg;
4  reg [6:0] seg;
5  // seg = {g,f,e,d,c,b,a};
6  // 0 is on and 1 is off
7
8  always @(hex_digit)
9    case(hex_digit)
10     4'h0: seg = 7'b1000000;
11     4'h1: seg = 7'b1111001;   // ---a----
12     4'h2: seg = 7'b0100100;   // |      |
13     4'h3: seg = 7'b0110000;   // f      b
14     4'h4: seg = 7'b0011001;   // |      |
15     4'h5: seg = 7'b0010010;   // ---g----
16     4'h6: seg = 7'b0000010;   // |      |
17     4'h7: seg = 7'b1111000;   // e      c
18     4'h8: seg = 7'b0000000;   // |      |
19     4'h9: seg = 7'b0011000;   // ---d----
20     4'ha: seg = 7'b0001000;
21     4'hb: seg = 7'b0000011;
22     4'hc: seg = 7'b1000110;
23     4'hd: seg = 7'b0100001;
24     4'he: seg = 7'b0000110;
25     4'hf: seg = 7'b0001110;
26   endcase
```

```
27 endmodule
```

Listagem 2 Conversor Hexadecimal para *display de 7-segmentos*

Apêndice B *MDC* utilizando o *Algoritmo de Euclides*

B.1 Automatizador de execução: *Makefile*

```
1  TARGET   = gcd
2  SOURCE   = gcd_top.v gcd_engine.v bin_to_bcd.v hex_to_7seg.v gcd_tb.v
3  IVERILOG = iverilog
4  VVP      = vvp
5  WAVE     = gtkwave
6
7  all: $(TARGET).vcd
8
9  $(TARGET).vvp : $(SOURCE)
10  $(IVERILOG) -o $(TARGET).vvp $(SOURCE)
11 #
12 $(TARGET).vcd : $(TARGET).vvp
13  $(VVP) $(TARGET).vvp
14 #
15 view : all
16  $(WAVE) $(TARGET).vcd
17 #
18 clean :
19  rm *.vvp *.vcd
```

Listagem 3 Makefile para automação de execução

B.2 Topo da hierarquia

```
1  module gcd_top(CLOCK_50, SW, done, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6,
       HEX7);
2    input  CLOCK_50;                                          // main clock
3    input  [17:0]SW;                                          // switches
4    output [6:0]HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7; // 7-seg disp
5    output done;
6
7    wire ack;
8    wire [3:0]disp[5:0];
9    wire [6:0]disp7[5:0];
10   wire [6:0]r_old, a_xi, b_xi;
11
12   reg rst, req;
13   reg [6:0]r;
14   reg [9:0]cnt0 = 0;
15
16   always @(posedge CLOCK_50) begin
17     if(cnt0 == 10) begin
18     rst = 1'b0;
19     cnt0 = 10;
20    end else begin
21       rst = 1'b1;
22     cnt0 = cnt0 + 1;
23    end
24
25     if(rst) begin
26       req = 1'b1;
27       r = 0;
```

```
28      end else begin
29        if(req)
30          req = 1'b0;
31        else
32          if(ack) begin
33            req = 1'b1;
34            r = r_old;
35          end
36      end
37    end
38
39    assign done = ack;
40
41    // input a:
42    bin_to_bcd bin_to_bcd_01({1'b0, SW[13:7]}, disp[5], disp[4]);
43    hex_to_7seg hex_to_7seg_01(disp[5], disp7[5]); // tens
44    hex_to_7seg hex_to_7seg_02(disp[4], disp7[4]); // ones
45    assign HEX7 = SW[13:7] == 0 || SW[13:7] > 99 ? 7'b0000110 : disp7[5];
46    assign HEX6 = SW[13:7] == 0 || SW[13:7] > 99 ? 7'b0101111 : disp7[4];
47
48    // input b:
49    bin_to_bcd bin_to_bcd_02({1'b0, SW[6:0]}, disp[3], disp[2]);
50    hex_to_7seg hex_to_7seg_03(disp[3], disp7[3]); // tens
51    hex_to_7seg hex_to_7seg_04(disp[2], disp7[2]); // ones
52    assign HEX5 = SW[6:0] == 0 || SW[6:0] > 99 ? 7'b0000110 : disp7[3];
53    assign HEX4 = SW[6:0] == 0 || SW[6:0] > 99 ? 7'b0101111 : disp7[2];
54
55    // turn off hundred's and thousand's displays:
56    assign HEX3 = 7'b1111111;
57    assign HEX2 = 7'b1111111;
58
59    // output:
60    bin_to_bcd bin_to_bcd_03({1'b0, r}, disp[1], disp[0]);
61    hex_to_7seg hex_to_7seg_05(disp[1], disp7[1]); // tens
62    hex_to_7seg hex_to_7seg_06(disp[0], disp7[0]); // ones
63    assign HEX1 = SW[6:0]  == 0 || SW[6:0] > 99 ||
64                  SW[13:7] == 0 || SW[13:7] > 99 ? 7'b0000110 : disp7[1];
65    assign HEX0 = SW[6:0]  == 0 || SW[6:0] > 99 ||
66                  SW[13:7] == 0 || SW[13:7] > 99 ? 7'b0101111 : disp7[0];
67
68    // gcd engine:
69    assign a_xi = SW[13:7] == 0 ? 1 : SW[13:7];
70    assign b_xi = SW[6:0]  == 0 ? 1 : SW[6:0];
71    gcd_engine gcd_engine_01(CLOCK_50, rst, req, a_xi, b_xi, ack, r_old);
72  endmodule
```

Listagem 4 Topo da hierarquia

B.3 Cálculo do *MDC*

```
1  module gcd_engine (
2      input wire clk, reset,
3      input wire start,
4      input wire [6:0]a_in, b_in,
5      output wire ready,
6      output reg [6:0]r
7    );
8
9    // symbolic state declaration:
```

```
10    localparam[1:0]
11      idle = 2'b01,
12      op   = 2'b10;
13
14    // Signal declaration:
15    reg [1:0]state_reg, state_next;
16    reg [6:0]swap, a_reg, a_next, b_reg, b_next;
17
18    // FSMD state and data registers:
19    always @(posedge clk) begin
20      if(reset) begin
21        state_reg = idle;
22        a_reg = 0;
23        b_reg = 0;
24      end else begin
25        state_reg = state_next;
26        a_reg = a_next;
27        b_reg = b_next;
28      end
29    end
30
31    // Next-state logic and data path functional unit:
32    always @(a_reg or b_reg or state_reg) begin
33      a_next = a_reg;
34      b_next = b_reg;
35      state_next = state_reg;
36      case(state_reg)
37        idle: begin
38          if(start) begin
39            a_next = a_in;
40            b_next = b_in;
41            state_next = op;
42          end
43        end
44        op: begin
45          if(a_next < b_next) begin
46            swap   = a_next;
47            a_next = b_next;
48            b_next = swap;
49          end else if(b_next != 0) begin
50            a_next = a_next - b_next;
51          end else begin
52            r = a_next;
53            state_next = idle;
54          end
55      end
56      endcase
57    end
58
59    assign ready = (state_reg == idle);
60  endmodule
```

Listagem 5 Cálculo do *MDC*

B.4 *Testbench* da implementação

```
1  `timescale 1ns/100ps
2
3  module gcd_tb();
```

```
4     reg clk;
5     wire done;
6     reg  [6:0]i, j;
7     wire [6:0]disp[7:0];
8     //wire [6:0]y;
9
10    initial begin
11      clk = 1'b0;
12      i = 1;
13      j = 1;
14    end
15
16    always begin
17      #1 clk = ~clk;
18    end
19
20    always @(posedge clk) begin
21      j = j + 1;
22      if(j > 15) begin
23        i = i + 1;
24        j = 1;
25        if(i > 15) begin
26          i = 1;
27          $finish;
28        end
29      end
30      $display("%d %d", i, j);
31    wait(done);
32    //wait(!done);
33    end
34
35    gcd_top gcd_top_01(clk, {i, j}, done, disp[0], disp[1], disp[2], disp[3],
          disp[4], disp[5], disp[6], disp[7]);
36
37    initial begin
38      $dumpfile("gcd_tb.vcd");
39      $dumpvars;
40    end
41  endmodule
```

Listagem 6 *Testbench* da implementação

Apêndice C Fatoração de inteiro usando divisão sucessiva

C.1 Automatizador de execução: *Makefile*

```
1  TARGET   = fatorador
2  #SOURCE   = pfact_top.v pfact_engine.v bin_to_bcd.v hex_to_7seg.v
3  SOURCE   = fatorador_top.v fatorador_engine.v hex_to_7seg.v bin_to_bcd.v
       testador.v
4  IVERILOG = iverilog
5  VVP      = vvp
6  WAVE     = gtkwave
7
8  all: $(TARGET).vcd
9
10 $(TARGET).vvp : $(SOURCE)
11  $(IVERILOG) -o $(TARGET).vvp $(SOURCE)
12 #
```

```
13 $(TARGET).vcd : $(TARGET).vvp
14  $(VVP) $(TARGET).vvp
15 #
16 view : all
17  $(WAVE) $(TARGET).vcd
18 #
19 clean :
20  rm *.vvp *.vcd
```

Listagem 7 Makefile para automação de execução

C.2 Topo da hierarquia

```
1  module fatorador_top (CLOCK_50, SW, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6,
       HEX7);
2    input CLOCK_50;                                                    // clock
       signal
3    input  [15:0]SW;                                                   // value to
       factorize
4    output [6:0]HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7;  // outputs
       display's
5
6    wire ack;
7    wire [3:0]dd0, dd1, dd2, dd3, dd4, dd5, dd6, dd7;
8    reg  rst, req;
9    integer cnt = 0;
10   wire [15:0]disp0, disp1, disp2, disp3;
11
12   always @(posedge CLOCK_50) begin
13     if(cnt == 10) begin
14       rst = 0;
15       cnt = 10;
16     end else begin
17       rst = 1;
18       cnt = cnt + 1;
19     end
20 /*
21     if(rst) begin
22       req = 1;
23     end else begin
24       if(req)
25         req = 0;
26       else
27         if(ack) begin
28           req = 1;
29         end
30     end
31 */
32   end
33
34   //assign ready = ack;
35   fatorador_engine fatorador_engine_00(CLOCK_50, rst, req, SW, ack, disp0,
       disp1, disp2, disp3);
36
37   bin_to_bcd bin_to_bcd_01(disp0, dd1, dd0);
38   hex_to_7seg hex_to_7seg_00(dd0, HEX0);     // tens
39   hex_to_7seg hex_to_7seg_01(dd1, HEX1);     // ones
40
41   bin_to_bcd bin_to_bcd_02(disp1, dd3, dd2);
```

```
42    hex_to_7seg hex_to_7seg_02(dd2, HEX2);     // tens
43    hex_to_7seg hex_to_7seg_03(dd3, HEX3);     // ones
44
45    bin_to_bcd bin_to_bcd_03(disp2, dd5, dd4);
46    hex_to_7seg hex_to_7seg_04(dd4, HEX4);     // tens
47    hex_to_7seg hex_to_7seg_05(dd5, HEX5);     // ones
48
49    bin_to_bcd bin_to_bcd_04(disp3, dd7, dd6);
50    hex_to_7seg hex_to_7seg_06(dd6, HEX6);     // tens
51    hex_to_7seg hex_to_7seg_07(dd7, HEX7);     // ones
52  endmodule
```

Listagem 8 Topo da hierarquia

C.3 Cálculo da fatoracao de inteiro usando divisao sucessiva

```
1   module fatorador_engine (
2       input clk, reset,
3       input start,
4       input [15:0]in,
5       output wire ready,
6       output reg [15:0]out0,
7       output reg [15:0]out1,
8       output reg [15:0]out2,
9       output reg [15:0]out3
10    );
11
12    // symbolic state declaration:
13    localparam[2:0]
14      idle = 2'b01,
15      op   = 2'b10;
16
17    // signal declaration:
18    reg  [1:0]state_reg, state_next = idle;
19    reg  [3:0]idx;
20    reg [15:0]i, i_reg, i_next, i_mod, mod_next, primes_buff[3:0];
21    // FSMD state and data registers:
22    always @(posedge clk) begin
23      if(reset) begin
24        state_reg = idle;
25        i_reg = 0;
26      end else begin
27        state_reg = state_next;
28        i_reg = i_next;
29        i_mod = mod_next;
30      end
31    end
32
33    always @(i_reg or i) begin
34      mod_next = i_reg % i;
35    end
36
37    // Next-state logic and data path functional unit:
38    always @(in or i_reg or i_mod or state_reg) begin
39      i_next = i_reg;
40      state_next = state_reg;
41      mod_next = i_mod;
42      case(state_reg)
43        idle: begin
```

```
44          i = 2;
45          i_next = in;
46          state_next = op;
47          for(idx = 0; idx < 4; idx = idx + 1) begin
48            primes_buff[idx] = 88;
49          end
50          idx = 0;
51        end
52
53        op: begin
54          if(i_next > 1) begin
55            if(i_next%i == 0) begin
56              i_next = i_next / i;
57              if(idx > 0) begin
58                if(primes_buff[idx - 1] !=  i) begin
59                  primes_buff[idx] = i;
60                  idx = idx + 1;
61                end
62              end else begin
63                primes_buff[idx] = i;
64                idx = idx + 1;
65              end
66            end else begin
67              i = i + 1;
68            end
69          end else begin
70            out0 = primes_buff[0];
71            out1 = primes_buff[1];
72            out2 = primes_buff[2];
73            out3 = primes_buff[3];
74            state_next = idle;
75          end
76        end
77      endcase
78    end
79  endmodule
```

Listagem 9 Cálculo da fatoracao de inteiro usando divisao sucessiva

C.4 *Testbench* da implementação

```
1  `timescale 1ns/100ps
2
3  /* Module to test your module fatorador */
4  module testador();
5
6    // clock signal
7    reg clock;
8    // number value tu be factorize
9    reg [15:0] value;
10
11   wire[6:0] hex0, hex1, hex2, hex3, hex4, hex5, hex6, hex7;
12
13   // temp. values to print output
14   reg[6:0] ohex0, ohex1, ohex2, ohex3, ohex4, ohex5, ohex6, ohex7;
15
16   //
17   always
18   begin
```

```
19      # 10 clock = ~clock; // clock frecuency 10
20    end
21
22    always @(hex0, hex1, hex2, hex3, hex4, hex5, hex6, hex7) begin
23      // task display used to convert to digit
24      display(hex0, ohex0);
25      display(hex1, ohex1);
26      display(hex2, ohex2);
27      display(hex3, ohex3);
28      display(hex4, ohex4);
29      display(hex5, ohex5);
30      display(hex6, ohex6);
31      display(hex7, ohex7);
32      /*
33        $display:
34        This command prints a message on the screen when executed.
35        You can add a list of variables.The message must be declared in
              quotation marks (""),
36        followed by the list of variables to be printed and the format.
37         At the end of the message a carriage return is entered.
38        Ex :       if ohex1 =0,ohex0=5
39                 $display("fator 1:",ohex1,ohex0);
40        output:
41                       factor 1 : 0 5
42      */
43      $display("fator 1:",ohex1,ohex0);
44      $display("fator 2:",ohex3,ohex2);
45      $display("fator 3:",ohex5,ohex4);
46      $display("fator 4:",ohex7,ohex6);
47    end
48
49  // ------------------------------------     display
50  /*
51   Task to convert temp_in to number in ten base (temp_out)
52  */
53    task display;
54    input [7:0] temp_in;     // input
55    output [7:0] temp_out;  // output (digit)
56    begin
57     case(temp_in)
58      7'b1000000: temp_out = 0;
59      7'b1111001: temp_out = 1;
60      7'b0100100: temp_out = 2;
61      7'b0110000: temp_out = 3;
62      7'b0011001: temp_out = 4;
63      7'b0010010: temp_out = 5;
64      7'b0000010: temp_out = 6;
65      7'b1111000: temp_out = 7;
66      7'b0000000: temp_out = 8;
67      7'b0010000: temp_out = 9;
68      default: temp_out = -1;
69     endcase
70    end
71
72    endtask
73  //------------------------------------------------------ end task
74    initial begin
75      clock = 0;
```

```
76        value = 350;     // number to factorize
77        #1000;
78        value = 210;     // number to factorize
79        #1000;
80        value =  11;     // number to factorize
81        #1000;
82        value =  85;     // number to factorize
83        #1000
84        value =  5;      // number to factorize
85        #1000
86        value =  125;      // number to factorize
87        #1000
88        value =  115;      // number to factorize
89        #1000
90        value =  3;      // number to factorize
91        #1000
92
93        $finish;     // number to finish simulation
94      end
95      // module test
96
97      fatorador_top f(clock, value, hex0, hex1, hex2, hex3, hex4, hex5, hex6,
            hex7);
98
99      initial begin
100       $dumpfile("fatorador_wave.vcd");
101       $dumpvars;
102     end
103
104   /*---------------------------------
105   value = 350
106   350 = 2 * 5^2 * 7
107
108   VSIM 11 >run
109   # fator 1:  0  2
110   # fator 2:  0  5
111   # fator 3:  0  7
112   # fator 4:  8  8
113
114   OUTPUT EXAMPLES
115
116   value = 210
117   210 = 2 * 3* 5* 7
118
119   VSIM 11 >run
120   # fator 1:  0  2
121   # fator 2:  0  3
122   # fator 3:  0  5
123   # fator 4:  0  7
124
125   //---------------------------------
126   value = 85
127   85 = 5 * 17
128
129   VSIM 11 >run
130   # fator 1:  0  5
131   # fator 2:  1  7
132   # fator 3:  8  8
```

```
133 # fator 4:  8  8
134 ----------------------------------*/
135 endmodule
```

Listagem 10 *Testbench* da implementação