Centro de Informática
Universidade Federal da Paraíba

Relatório Final

Controle Automático de Travessia de Ponte

Aluno(a) Fábio Trajano de Souza
Aluno(a) Joan Vitor Delfino Medeiros
Aluno(a) Luiz Henrique Coutinho Mariano

João Pessoa, 2018

Centro de Informática
Universidade Federal da Paraíba

Controle Automático de Travessia de Ponte

Relatório elaborado para o projeto final da disciplina Circuito Lógicos II, ministrada pelo Professor Eudisley Gomes dos Anjos do Centro de Informática da Universidade Federal da Paraíba.

João Pessoa, 2018

Resumo

Este trabalho apresenta um projeto que gerencia o controle automático de travessia de uma ponte com o uso de FPGAs “Field Programable Gate Array” (Matriz de Portas Lógicas Programáveis no Campo) e a linguagem de descrição de hardware (HDL) Verilog, com implementação no Ambiente de desenvolvimento integrado (IDE) Quartus II da altera. O objetivo do projeto é controlar o fluxo de carros que atravessa uma ponte levadiça e o fluxo de navios que precisa passar por baixo da ponte, esse controle é feito dando prioridade a passagens de navios, durante este controle de trafego será feita a contagem de carros e navios que passam pela ponte.

Palavras-chave: FPGA, Verilog.

Lista de siglas

FPGA - Field Programmable Gate Array (Arranjo de portas programáveis em campo)

IDE - Integrated Development Environment (Ambiente de desenvolvimento integrado)

HDL - Hardware Description Language (linguagem de Descrição de Hardware)

Sumário

1. Introdução.

A linguagem de descrição de hardware (HDL) possibilita várias aplicações, como no teste de circuitos e na síntese do circuito descrito. No projeto de programação para o controle de travessia de navios e carros em uma ponte feito com FPGA, usamos para implementação do circuito a linguagem de descrição de hardware Verilog, pois a mesma pode suportar projetos com múltiplos níveis de hierarquias e o uso de ferramentas computacionais que auxiliam na implementação do projeto.

Para desenvolvimento do projeto usamos Kits de desenvolvimento FPGA da Altera Cyclone II (Altera DE2), fabricado pela Altera, juntamente com o programa de simulação e implementação Quartus II, fornecido pela Altera. Nas figuras a seguir temos a plataforma usada no projeto e o Ambiente do Software do Quartus II utilizado para as simulações.

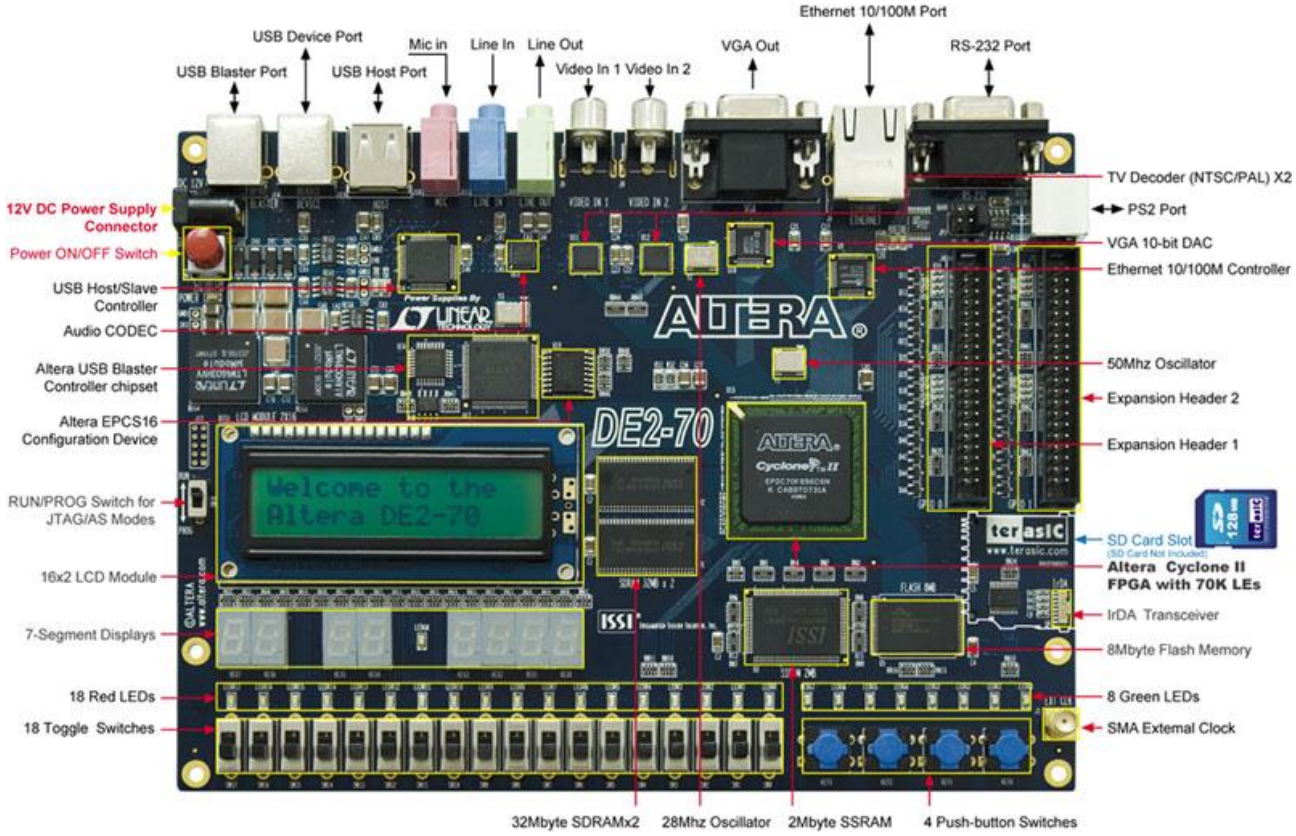

Figura 1-1 Placa Altera DE2.

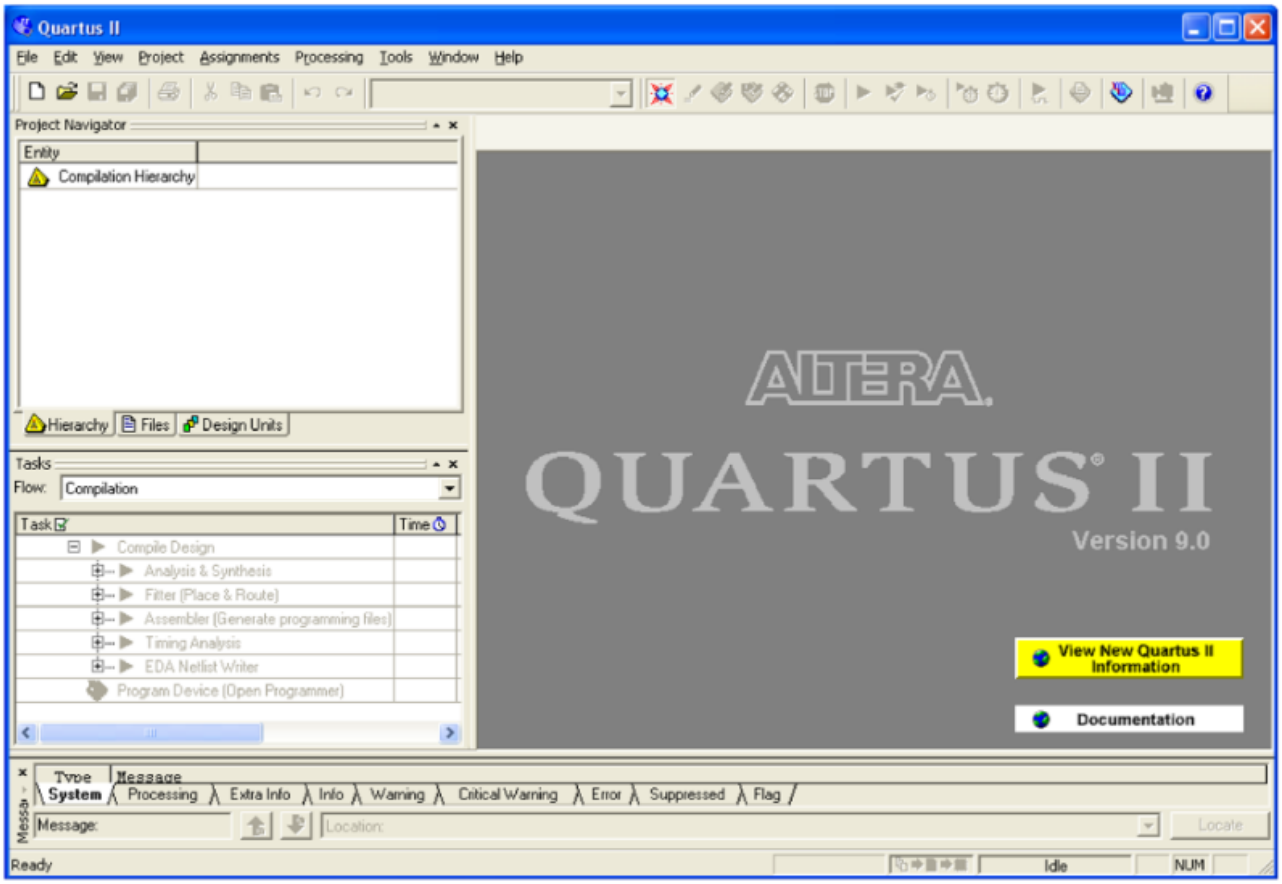

Figura 1-2 Ambiente do Software utilizado para as simulações

Para implementação do projeto e criação do código em Verilog devemos considerar que a prioridade de travessia é sempre do navio. Quando um navio chegar, o seu semáforo estará em vermelho, nesse momento o semáforo do carro ficará vermelho, e cancela irá baixar para impedir o fluxo de carros a ponte esperará 3 segundos (tempo de segurança para os carros terminarem a travessia) para começar a se elevar para permitir a travessia do navio. A elevação da ponte dura 3 segundos. Quando finalmente estiver elevada, o navio poderá atravessar. O semáforo do navio ficará vermelho caso não haja mais nenhum navio. Cada navio leva 4 segundos para fazer a travessia. Sem nenhum navio aguardando, a ponte voltará a seu estado normal, o que leva 3 segundos, a cancela sobe e o semáforo do carro fica em verde liberando o fluxo de carros.

2. Metodologia.

O desenvolvimento do projeto para o controle automático de travessia de ponte possui as seguintes etapas:

2.1 - Análise e estudo do problema proposto para a interpretação da descrição do sistema, pois o mesmo possui uma sequencia de comandos e prioridades que devem ser seguidas.

2.2 - Desenvolvimento de um código na linguagem de descrição de hardware Verilog, usando o Ambiente do Software do Quartus II. Este código usa os comandos básicos usados na descrição de algoritmos, como comandos em sequência, comandos de decisão, comparações e estruturas em laço.

2.3 - Elaboração de um diagramas de fluxos que modela o comportamento do circuito mostrando as etapas de execução do problema proposto;

2.4 – Implementação e testes do projeto no FPGA.

3. Descrição do Projeto.

Para representar o funcionamento do controle da ponte, passagem de navios e carros usamos do FPGA:

- Monitor de LCD para indicar o movimento da ponte e dos navios;
- Dois displays de sete segmentos(HEX7 e HEX6) para contagem dos carros parados;
- Um displays de sete segmentos(HEX4) para mostrar quando um navio chegar;
- Dois LEDs (LEDR) vermelhos para representar o sinal de parada;
- Dois LEDs verdes (LEDG) para representar o sinal de passagem livre;

Quando o FPGA for inicializado apresentara a seguinte configuração:

- Monitor de LCD: Navio: Nenhum;
  Ponte: Normal;

- Sinal dos carros: LEDG[0] (ligado) sinal verde para os carros;
  LEDR[0] (desligado) sinal vermelho para os carros;

- Sinal dos navios: LEDG[2] (desligado) sinal verde para os navios;
  LEDR[2] (ligado) sinal vermelho para os navios;

- Display para contagem dos carros: HEX7 = 0 e HEX6 = 0;
- Display para contagem dos navios: HEX4 = 0;

No projeto a prioridade de travessia é sempre do navio. Quando um navio chegar:

- Monitor de LCD: Navio: Esperando;
  Ponte: Elevando;

- Sinal dos carros: LEDG[0] (desligado) sinal verde para os carros;
  LEDR[0] (ligado) sinal vermelho para os carros;

- Sinal dos navios: LEDG[2] (desligado) sinal verde para os navios;
  LEDR[2] (ligado) sinal vermelho para os navios;

- Display para contagem dos carros: HEX7 = 0 e HEX6 = 5;
- Display para contagem dos navios: HEX4 = 1;

Ponte elevada:

- Monitor de LCD: Navio: Passando;
  Ponte: Elevada;

- Sinal dos carros: LEDG[0] (desligado) sinal verde para os carros;
  LEDR[0] (ligado) sinal vermelho para os carros;

- Sinal dos navios: LEDG[2] (ligado) sinal verde para os navios;
  LEDR[2] (desligado) sinal vermelho para os navios;
- Display para contagem dos carros: HEX7 =1 e HEX6 = 0;
- Display para contagem dos navios: HEX4 = 1;

Ponte descendo:

- Monitor de LCD: Navio: Nenhum;
  Ponte: Descendo;

- Sinal dos carros: LEDG[0] (desligado) sinal verde para os carros;
  LEDR[0] (ligado) sinal vermelho para os carros;

- Sinal dos navios: LEDG[2] (desligado) sinal verde para os navios;
  LEDR[2] (ligado) sinal vermelho para os navios;

- Display para contagem dos carros: HEX7 =1 e HEX6 = 5;
- Display para contagem dos navios: HEX4 = 0;

Aguardando a chegada do próximo navio:

- Monitor de LCD: Navio: Nenhum;
  Ponte: Normal;

- Sinal dos carros: LEDG[0] (ligado) sinal verde para os carros;
  LEDR[0] (desligado) sinal vermelho para os carros;

- Sinal dos navios: LEDG[2] (desligado) sinal verde para os navios;
  LEDR[2] (ligado) sinal vermelho para os navios;

- Display para contagem dos carros: HEX7 = 0 e HEX6 = 0;
- Display para contagem dos navios: HEX4 = 0;

A figura a seguir mostra o diagrama de fluxo com a sequencia e etapas de execução dos comando:

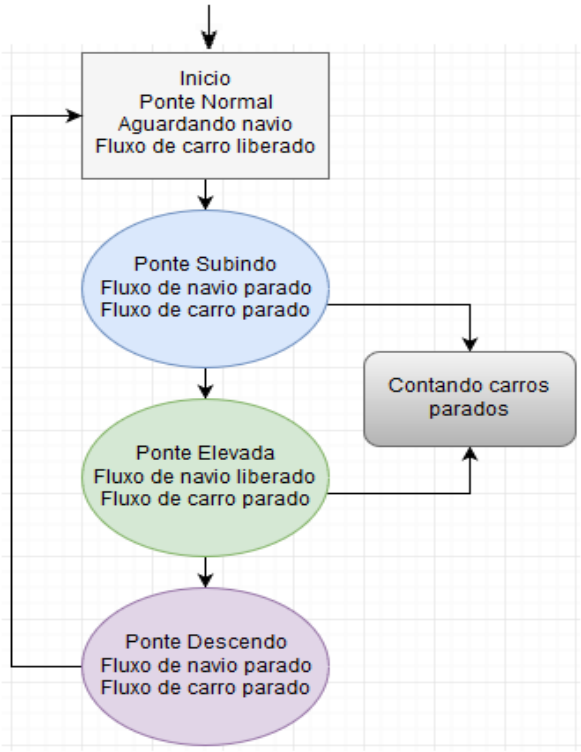

Figura 3-1 Diagrama de Fluxo

4. Execução do Projeto, Testes e Resultados.

As imagens a baixo mostram a execução no FPGA e os resultados obtidos.

4.1 - Estado inicial do projeto: Ponte Baixa.

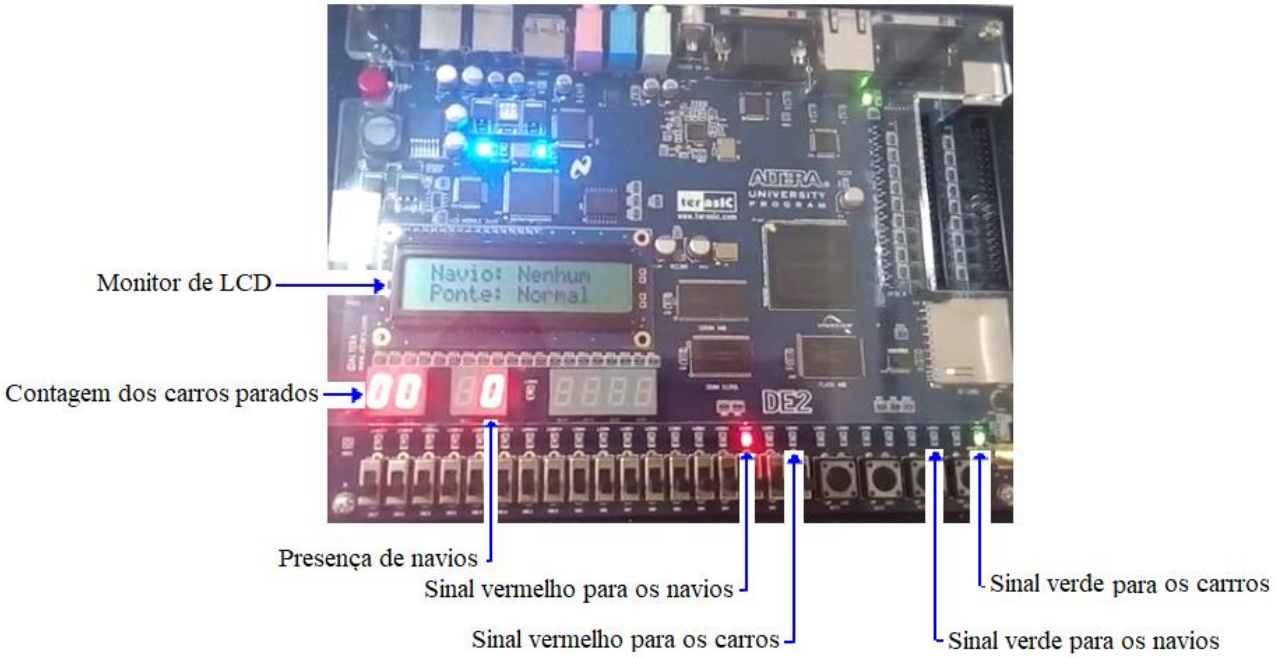

Figura 4-1 – Estado Inicial

4.2 – Segundo estado: Ponte Levantando.

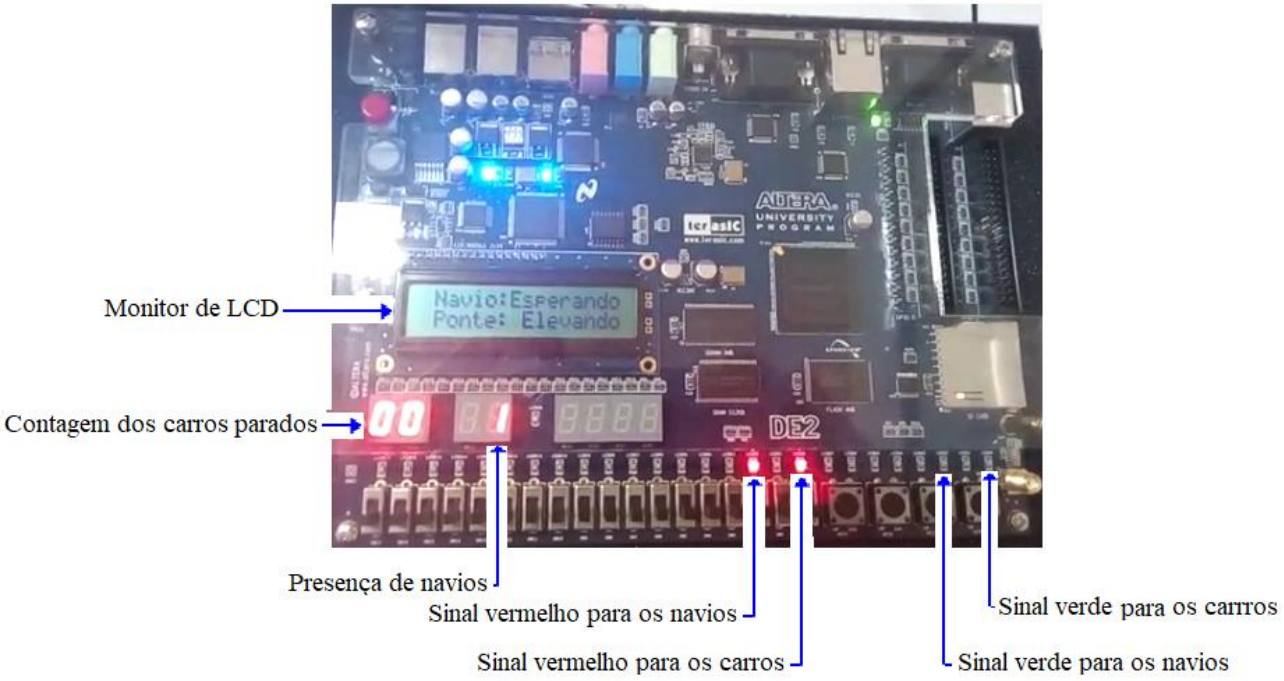

Figura 4-2 Segundo Estado

4.3 – Terceiro estado: Ponte Elevada.

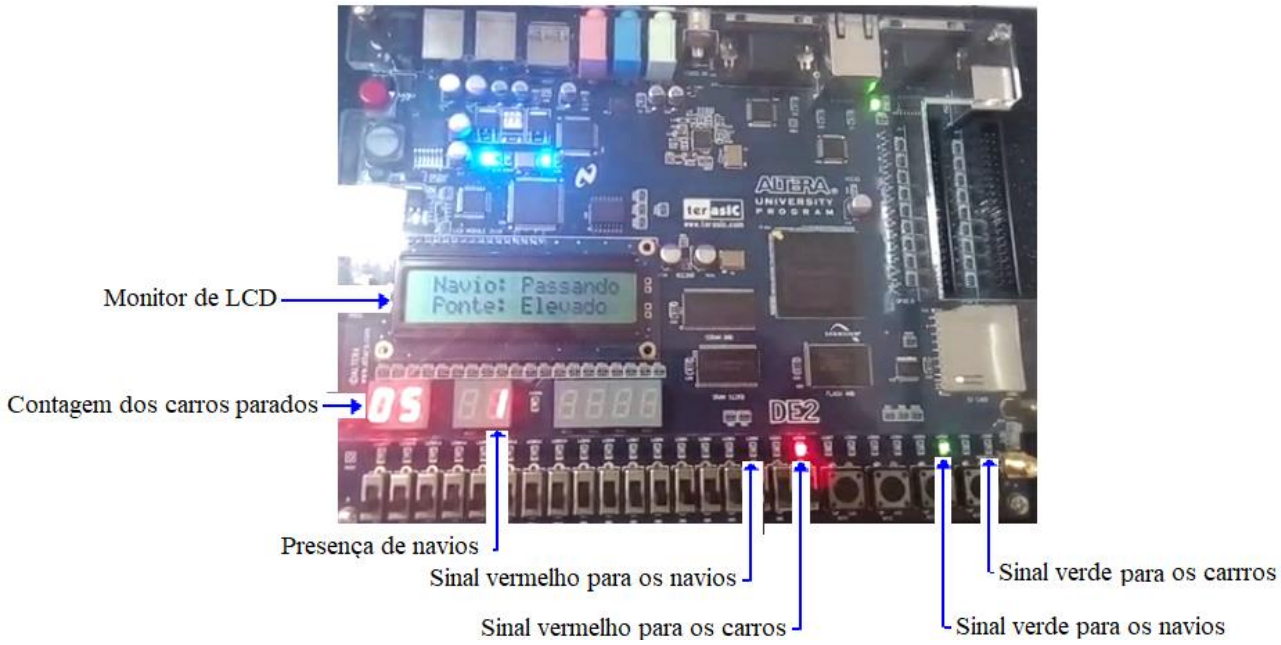

Figura 4-3 Terceiro Estado

4.4 – Quarto Estado: Ponte Descendo.

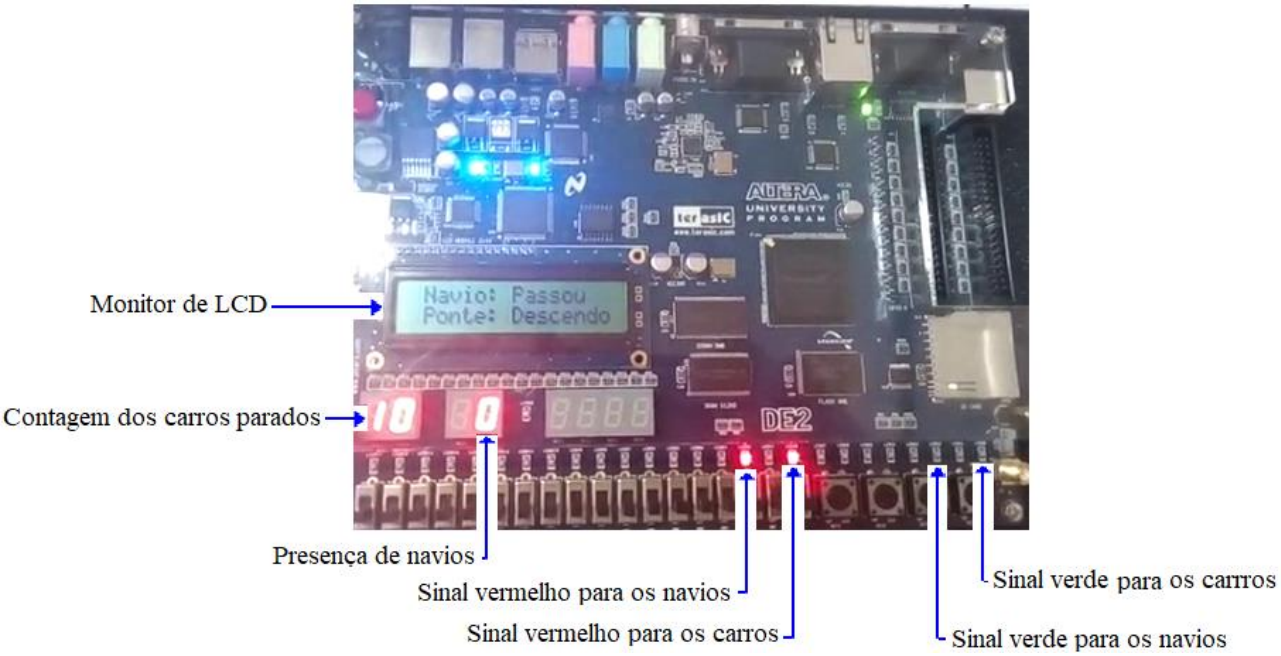

Figura 4-3 Quarto Estado

Quando a ponte estiver totalmente baixa, o fluxo de carros é liberado ate que um navio chegue para atravessar a ponte e todos os estados acima se repetem novamente.

Uma das etapas do projeto era criar um comando configurando um botão (switch), para quando apertado mostrar no monitor de LCD a quantidade de navios que tivesse passado ate o momento, este foi problema que não conseguimos resolver. A quantidade de carros que chegava na ponte, e ficava aguardando enquanto a ponte baixava era

mostrado no display de sete segmentos de 5 em 5 carros, porém não conseguimos fazer com que esta quantidade de carros fosse decrementada quando o sinal ficava verde.

5. Conclusão.

Por meio deste trabalho foi possível aprender sobre implementação de sistemas digitais em FPGA, controle de memória e técnicas iniciais sobre processamento de dados usando lógica combinacional e lógica sequencial. Uma das etapas mais importantes do trabalho foi à implementação de um código em Verilog e a utilização do FPGA para simular a ponte, passagem de carros, navios e sinais que controlavam o fluxo de veículos com o uso do monitor de LCD, LEDs e displays de sete segmentos.

6. Referências.

https://www.jdoodle.com/execute-verilog-online

ftp://ftp.altera.com/up/pub/Intel_Material/9.0/Tutorials/VHDL/Quartus_II_Introduction.pdf

http://www.rennes.supelec.fr/ren/perso/jweiss/fpga/altera/carte_de2.php/

https://stackoverflow.com/

ftp://ftp.altera.com/up/pub/Webdocs/DE2_UserManual.pdf

Anexos

```
/*
 SW8 (GLOBAL RESET) resets LCD
ENTITY LCD_Display IS
-- Enter number of live Hex hardware data values to display
-- (do not count ASCII character constants)
        GENERIC(Num_Hex_Digits: Integer:= 2);
-----------------------------------------------------------------------
-- LCD Displays 16 Characters on 2 lines
-- LCD_display string is an ASCII character string entered in hex for
-- the two lines of the  LCD Display   (See ASCII to hex table below)
-- Edit LCD_Display_String entries above to modify display
-- Enter the ASCII character's 2 hex digit equivalent value
-- (see table below for ASCII hex values)
-- To display character assign ASCII value to LCD_display_string(x)
-- To skip a character use 8'h20" (ASCII space)
-- To dislay "live" hex values from hardware on LCD use the following:
--       make array element for that character location 8'h0" & 4-bit field from
Hex_Display_Data
--      state machine sees 8'h0" in high 4-bits & grabs the next lower 4-bits from
Hex_Display_Data input
--   and performs 4-bit binary to ASCII conversion needed to print a hex digit
--    Num_Hex_Digits must be set to the count of hex data characters (ie. "00"s) in the
display
--   Connect hardware bits to display to Hex_Display_Data input
-- To display less than 32 characters, terminate string with an entry of 8'hFE"
--  (fewer characters may slightly increase the LCD's data update rate)
-----------------------------------------------------------------
--                    ASCII HEX TABLE
--  Hex                                              Low Hex Digit
-- Value  0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
------\-----------------------------------------------------------------
--H  2 |  SP  !   "   #   $   %   &   '   (   )   *   +   ,   -   .   /
```

```
--i  3 |  0   1   2   3   4   5   6   7   8   9   :   ;   <   =   >   ?
--g  4 |  @   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O
--h  5 |  P   Q   R   S   T   U   V   W   X   Y   Z   [   \   ]   ^   _
--   6 |  `   a   b   c   d   e   f   g   h   i   j   k   l   m   n   o
--   7 |  p   q   r   s   t   u   v   w   x   y   z   {   |   }   ~ DEL
------------------------------------------------------------------------
-- Example "A" is row 4 column 1, so hex value is 8'h41"
-- *see LCD Controller's Datasheet for other graphics characters available
*/

module LCD_Display(iCLK_50MHZ, iRST_N, hex1, hex0,
LCD_RS,LCD_E,LCD_RW,DATA_BUS, segdisplay, segdisplay2 , segdisplay3, LG, LR);

input iCLK_50MHZ, iRST_N;
input [3:0] hex1, hex0;
output LCD_RS, LCD_E, LCD_RW;
inout [7:0] DATA_BUS;
output reg [6:0] segdisplay;
output reg [6:0] segdisplay2;
output reg [6:0] segdisplay3;
output reg [8:0] LG ;
output reg [17:0] LR ;

parameter
HOLD = 4'h0,
FUNC_SET = 4'h1,
DISPLAY_ON = 4'h2,
MODE_SET = 4'h3,
Print_String = 4'h4,
LINE2 = 4'h5,
RETURN_HOME = 4'h6,
DROP_LCD_E = 4'h7,
RESET1 = 4'h8,
RESET2 = 4'h9,
RESET3 = 4'ha,
DISPLAY_OFF = 4'hb,
```

```
DISPLAY_CLEAR = 4'hc;

reg [3:0] state, next_command;
// Enter new ASCII hex data above for LCD Display
reg [7:0] DATA_BUS_VALUE;
reg [7:0] Next_Char;
reg [19:0] CLK_COUNT_400HZ;
reg [4:0] CHAR_COUNT;
reg active;
reg CLK_400HZ, LCD_RW_INT, LCD_E, LCD_RS;
integer  checker = 0, clock = 0, one_second = 50000000, sleep_time = 3, ship_time = 4,
atual_time = 0, step = 0, amount_ship = 0, incre = 1, car_amount = 0, contadorseg = 1,
ledcar = 0, ledship = 0,amount_car = 0,al,al2,incre2=1,teste=0,car_time=1,increship=0;

// BIDIRECTIONAL TRI STATE LCD DATA BUS
assign DATA_BUS = (LCD_RW_INT? 8'bZZZZZZZZ: DATA_BUS_VALUE);

//Define the ports used for LCD string
//LCD_display_string u1(
//.index(CHAR_COUNT),
//.out(Next_Char),
//.hex1(hex1),
//.hex0(hex0));

//contador do display
always@(posedge iCLK_50MHZ)begin
        case(amount_ship)
                4:begin
                                        case(increship)
                                1: begin
                                        segdisplay <= 7'b1111001;
                                end
                                2:begin
                                        segdisplay <= 7'b0100100;
                                        end
                                3:begin
```

```
                segdisplay <= 7'b0110000;
                end
            4:begin
                segdisplay <= 7'b0011001;
            end
            5:begin
                segdisplay <= 7'b0010010;
            end
            6:begin
                segdisplay <= 7'b0000010;
                end
            7:begin
                segdisplay <= 7'b1111000;
                end
            8:begin
                segdisplay <= 7'b0000000;
                end
            9:begin
                segdisplay <= 7'b0010000;
            end
        default: begin segdisplay <= 7'b1000000;
            end
            endcase
            end
    1: begin
        segdisplay <= 7'b1111001;
        end
    2:begin
        segdisplay <= 7'b0100100;
        end

    default: begin segdisplay <= 7'b1000000;
    end
endcase
end
```

```
always@(amount_car)begin
	case(amount_car)
		5:
			begin
			segdisplay3 <= 7'b0010010;
			end
		10:
			begin
			segdisplay2 <= 7'b1111001;
			segdisplay3 <= 7'b1000000;
			end
		15:
			begin
			segdisplay2 <= 7'b1111001;
			segdisplay3 <= 7'b0010010;
			end
		20:
			begin
			segdisplay2 <= 7'b0100100;
			segdisplay3 <= 7'b1000000;
			end
		25:
			begin
			segdisplay2 <= 7'b0100100;
			segdisplay3 <= 7'b0010010;
			end
		default: begin
			segdisplay3 <= 7'b1000000;
			segdisplay2 <= 7'b1000000;

			end
	endcase
end

always @(posedge iCLK_50MHZ)begin
	case(step)
```

```
        0: begin
                LR <= 17'b00000000000000100;
                LG <= 9'b000000001;

                end
        1: begin
                LR <= #9000 17'b00000000000000101;
                LG <= #9000 9'b000000000;
                end
        2:begin
                LR <= #9000 17'b00000000000000001;
                LG <= #9000 9'b000000100;
                end
         3:begin
                LR <= #150000000 17'b00000000000000101;
                LG <= #900000 9'b000000000;
                end
        endcase
end

assign LCD_RW = LCD_RW_INT;


always@(posedge iCLK_50MHZ)begin
        clock <= clock + 1;

        //EXIBE CONTAGEM DE CARROS TOTAIS
                if(hex0 == 4'b0010)begin
                        step<=-1;
                        amount_ship<=4 ;
                end
                if(hex0 == 4'b0000)begin
                        step<=0;
                end
                        if (clock == one_second * 10 * incre && step>=0)begin
                                        amount_ship <= amount_ship + 1;
```

```
                    incre<= incre+1;
                    increship<=increship+1;
                end

                if (clock == one_second * 4 * incre2 )begin
                        if(step > 0)begin
                        amount_car <= amount_car + 5;
                        car_amount <= car_amount + amount_car;
                        end
                        incre2<= incre2+1;
                end

                if (clock == one_second * contadorseg * 3)begin
                    if (step == 0 && amount_car>0)begin
                        amount_car <= #900000 amount_car - 5;
                    end
                    contadorseg<=contadorseg+1;
                end

            if(amount_ship > 0 || step > 0)begin

                //Normal
                if (clock == one_second *  (sleep_time + atual_time) && step ==
0)begin
                    sleep_time <= 3;
                    atual_time <= atual_time + sleep_time;
                    step <= 1;
                    checker <= checker + 1;
                end

                //Levantando ponte
                if (clock == one_second *  (sleep_time + atual_time) && step ==
1)begin
                    sleep_time <= amount_ship * ship_time; //Quantidade de
navios x o tempo do navio
                    atual_time <= atual_time + sleep_time;
```

```
                        step <= 2;
                        checker <= checker + 2;
                    end

                    if (clock == one_second *  (sleep_time/amount_ship          +
atual_time) && step == 2)begin
                        amount_ship <= amount_ship - 1;
                    end

                    //Ponte levantada
                    if (clock == one_second *  (sleep_time + atual_time) && step ==
2)begin
                        step <= 3;
                        sleep_time <= 3;
                        atual_time <= atual_time + sleep_time;
                        checker <= checker + 3;
                    end
                    //Ponte Descendo ponte
                    if (clock == one_second *  (sleep_time + atual_time) && step ==
3)begin
                        step <= 0;
                        sleep_time <= 3;
                        atual_time <= atual_time + sleep_time;
                        checker <= checker + 4;
                    end
                end
                else begin
                        if (clock == one_second *(1 + atual_time))begin
                                atual_time <= atual_time + 1;
                            end
                end
            end

//Esse always cria um clock de 40MHz
always @(posedge iCLK_50MHZ or negedge iRST_N)
    if (!iRST_N) // Reseta o contador e o clock
```

```
        begin
          CLK_COUNT_400HZ <= 20'h00000;
          CLK_400HZ <= 1'b0;
        end
        else if (CLK_COUNT_400HZ < 20'h0F424)
        begin
          CLK_COUNT_400HZ <= CLK_COUNT_400HZ + 1'b1;
        end
        else
        begin
          CLK_COUNT_400HZ <= 20'h00000;
          CLK_400HZ <= ~CLK_400HZ;
        end
// State Machine to send commands and data to LCD DISPLAY

always @(posedge CLK_400HZ or negedge iRST_N) //Uso do clock de 400Mhz
        if (!iRST_N)
        begin
         state <= RESET1; //If reset button is pressed the state is changed to RESET1
        end
        else
        case (state)
        RESET1:
// Set Function to 8-bit transfer and 2 line display with 5x8 Font size
// see Hitachi HD44780 family data sheet for LCD command and timing details
        begin
          LCD_E <= 1'b1; //Starts data READ / WRITE
          LCD_RS <= 1'b0;
          LCD_RW_INT <= 1'b0;
          DATA_BUS_VALUE <= 8'h38;
          state <= DROP_LCD_E;
          next_command <= RESET2;
          CHAR_COUNT <= 5'b00000;
        end
        RESET2:
        begin
```

```
          LCD_E <= 1'b1;
          LCD_RS <= 1'b0;
          LCD_RW_INT <= 1'b0;
          DATA_BUS_VALUE <= 8'h38;
          state <= DROP_LCD_E;
          next_command <= RESET3;
        end
        RESET3:
        begin
          LCD_E <= 1'b1;
          LCD_RS <= 1'b0;
          LCD_RW_INT <= 1'b0;
          DATA_BUS_VALUE <= 8'h38;
          state <= DROP_LCD_E;
          next_command <= FUNC_SET;
        end
// EXTRA STATES ABOVE ARE NEEDED FOR RELIABLE PUSHBUTTON RESET OF LCD

        FUNC_SET:
        begin
          LCD_E <= 1'b1;
          LCD_RS <= 1'b0;
          LCD_RW_INT <= 1'b0;
          DATA_BUS_VALUE <= 8'h38;
          state <= DROP_LCD_E;
          next_command <= DISPLAY_OFF;
        end

// Turn off Display and Turn off cursor
        DISPLAY_OFF:
        begin
          LCD_E <= 1'b1;
          LCD_RS <= 1'b0;
          LCD_RW_INT <= 1'b0;
          DATA_BUS_VALUE <= 8'h08;
```

```
        state <= DROP_LCD_E;
        next_command <= DISPLAY_CLEAR;
      end

// Clear Display and Turn off cursor
      DISPLAY_CLEAR:
      begin
        LCD_E <= 1'b1;
        LCD_RS <= 1'b0;
        LCD_RW_INT <= 1'b0;
        DATA_BUS_VALUE <= 8'h01;
        state <= DROP_LCD_E;
        next_command <= DISPLAY_ON;
      end

// Turn on Display and Turn off cursor
      DISPLAY_ON:
      begin
        LCD_E <= 1'b1;
        LCD_RS <= 1'b0;
        LCD_RW_INT <= 1'b0;
        DATA_BUS_VALUE <= 8'h0C;
        state <= DROP_LCD_E;
        next_command <= MODE_SET;
      end

// Set write mode to auto increment address and move cursor to the right
      MODE_SET:
      begin
        LCD_E <= 1'b1;
        LCD_RS <= 1'b0;
        LCD_RW_INT <= 1'b0;
        DATA_BUS_VALUE <= 8'h06;
        state <= DROP_LCD_E;
        next_command <= Print_String;
      end
```

```
//----------------------------------------------------------------------
        Print_String: // Write ASCII hex character in first LCD character location
        begin
        case(CHAR_COUNT)
                        5'h00: Next_Char <= 8'h4E; //N
                        5'h01: Next_Char <= 8'h61; //a
                        5'h02: Next_Char <= 8'h76; //v
                        5'h03: Next_Char <= 8'h69; //i
                        5'h04: Next_Char <= 8'h6F; //o
                        5'h05: Next_Char <= 8'h3A; //:
                        5'h07: Next_Char <= 8'h4E; //N
                        5'h08: Next_Char <= 8'h65; //e
                        5'h09: Next_Char <= 8'h6E; //n
                        5'h0A: Next_Char <= 8'h68; //h
                        5'h0B: Next_Char <= 8'h75; //u
                        5'h0C: Next_Char <= 8'h6D; //m
                        5'h10: Next_Char <= 8'h50; //P
                        5'h11: Next_Char <= 8'h6F; //o
                        5'h12: Next_Char <= 8'h6E; //n
                        5'h13: Next_Char <= 8'h74; //t
                        5'h14: Next_Char <= 8'h65; //e
                        5'h15: Next_Char <= 8'h3A; //:
                         5'h17: Next_Char <= 8'h4E; //N
                         5'h18: Next_Char <= 8'h6F; //o
                         5'h19: Next_Char <= 8'h72; //r
                         5'h1A: Next_Char <= 8'h6D; //m
                         5'h1B: Next_Char <= 8'h61; //a
                         5'h1C: Next_Char <= 8'h6C; //l
                default: Next_Char <= 8'h20;
        endcase

                case(step)
                        1:begin

                                #9000 state <= DISPLAY_CLEAR;
```

```
                case (CHAR_COUNT)
                        5'h00:#9000 Next_Char <= 8'h4E; //N
                        5'h01:#9000 Next_Char <= 8'h61; //a
                        5'h02:#9000 Next_Char <= 8'h76; //v
                        5'h03:#9000 Next_Char <= 8'h69; //i
                        5'h04:#9000 Next_Char <= 8'h6F; //o
                        5'h05:#9000 Next_Char <= 8'h3A; //:
                        5'h06:#9000 Next_Char <= 8'h45; //E
                        5'h07:#9000 Next_Char <= 8'h73; //s
                        5'h08:#9000 Next_Char <= 8'h70; //p
                        5'h09:#9000 Next_Char <= 8'h65; //e
                        5'h0A:#9000 Next_Char <= 8'h72; //r
                        5'h0B:#9000 Next_Char <= 8'h61; //a
                        5'h0C:#9000 Next_Char <= 8'h6E; //n
                        5'h0D:#9000 Next_Char <= 8'h64; //d
                        5'h0E:#9000 Next_Char <= 8'h6F; //o

                        5'h10:#9000 Next_Char <= 8'h50; //P
                        5'h11:#9000 Next_Char <= 8'h6F; //o
                        5'h12:#9000 Next_Char <= 8'h6E; //n
                        5'h13:#9000 Next_Char <= 8'h74; //t
                        5'h14:#9000 Next_Char <= 8'h65; //e
                        5'h15:#9000 Next_Char <= 8'h3A; //:
                        5'h17:#9000 Next_Char <= 8'h45; //E
                        5'h18:#9000 Next_Char <= 8'h6C; //l
                        5'h19:#9000 Next_Char <= 8'h65; //e
                        5'h1A:#9000 Next_Char <= 8'h76; //v
                        5'h1B:#9000 Next_Char <= 8'h61; //a
                        5'h1C:#9000 Next_Char <= 8'h6E; //n
                        5'h1D:#9000 Next_Char <= 8'h64; //d
                        5'h1E:#9000 Next_Char <= 8'h6F; //o
                        default: Next_Char <= 8'h20; //Blank spaces
                endcase
                end
        2:begin
```

```
#9000 state <= DISPLAY_CLEAR;
        case (CHAR_COUNT)
                5'h00:#9000 Next_Char <= 8'h4E; //N
                5'h01:#9000 Next_Char <= 8'h61; //a
                5'h02:#9000 Next_Char <= 8'h76; //v
                5'h03:#9000 Next_Char <= 8'h69; //i
                5'h04:#9000 Next_Char <= 8'h6F; //o
                5'h05:#9000 Next_Char <= 8'h3A; //:
                5'h07:#9000 Next_Char <= 8'h50; //P
                5'h08:#9000 Next_Char <= 8'h61; //a
                5'h09:#9000 Next_Char <= 8'h73; //s
                5'h0A:#9000 Next_Char <= 8'h73; //s
                5'h0B:#9000 Next_Char <= 8'h61; //a
                5'h0C:#9000 Next_Char <= 8'h6E; //n
                5'h0D:#9000 Next_Char <= 8'h64; //d
                5'h0E:#9000 Next_Char <= 8'h6F; //o


                5'h10:#9000 Next_Char <= 8'h50; //P
                5'h11:#9000 Next_Char <= 8'h6F; //o
                5'h12:#9000 Next_Char <= 8'h6E; //n
                5'h13:#9000 Next_Char <= 8'h74; //t
                5'h14:#9000 Next_Char <= 8'h65; //e
                5'h15:#9000 Next_Char <= 8'h3A; //:
                5'h17:#9000 Next_Char <= 8'h45; //E
                5'h18:#9000 Next_Char <= 8'h6C; //l
                5'h19:#9000 Next_Char <= 8'h65; //e
                5'h1A:#9000 Next_Char <= 8'h76; //v
                5'h1B:#9000 Next_Char <= 8'h61; //a
                5'h1C:#9000 Next_Char <= 8'h64; //d
                5'h1D:#9000 Next_Char <= 8'h6F; //o
                default: Next_Char <= 8'h20; //Blank spaces
 endcase
end

3:begin
```

```
#9000 state <= DISPLAY_CLEAR;
        case (CHAR_COUNT)
                5'h00:#40000 Next_Char <= 8'h4E; //N
                5'h01:#40000 Next_Char <= 8'h61; //a
                5'h02:#40000 Next_Char <= 8'h76; //v
                5'h03:#40000 Next_Char <= 8'h69; //i
                5'h04:#40000 Next_Char <= 8'h6F; //o
                5'h05:#40000 Next_Char <= 8'h3A; //:
                5'h07:#40000 Next_Char <= 8'h50; //P
                5'h08:#40000 Next_Char <= 8'h61; //a
                5'h09:#40000 Next_Char <= 8'h73; //s
                5'h0A:#40000 Next_Char <= 8'h73; //s
                5'h0B:#40000 Next_Char <= 8'h6F; //o
                5'h0C:#40000 Next_Char <= 8'h75; //u

                5'h10:#40000 Next_Char <= 8'h50; //P
                5'h11:#40000 Next_Char <= 8'h6F; //o
                5'h12:#40000 Next_Char <= 8'h6E; //n
                5'h13:#40000 Next_Char <= 8'h74; //t
                5'h14:#40000 Next_Char <= 8'h65; //e
                5'h15:#40000 Next_Char <= 8'h3A; //:

                5'h17:#40000 Next_Char <= 8'h44; //D
                5'h18:#40000 Next_Char <= 8'h65; //e
                5'h19:#40000 Next_Char <= 8'h73; //s
                5'h1A:#40000 Next_Char <= 8'h63; //c
                5'h1B:#40000 Next_Char <= 8'h65; //e
                5'h1C:#40000 Next_Char <= 8'h6E; //n
                5'h1D:#40000 Next_Char <= 8'h64; //d
                5'h1E:#40000 Next_Char <= 8'h6F; //o
                default: Next_Char <= 8'h20; //Blank spaces
 endcase
end

-1:begin
```

```
                #9000 state <= DISPLAY_CLEAR;
                case (CHAR_COUNT)
                        5'h00:begin
                        #40000 Next_Char <= 8'h30;end//0
                        5'h00:begin
                        #40000 Next_Char <= 8'h31;end //1
                        5'h00:begin
                        #40000 Next_Char <= 8'h32;end //2
                        5'h00:begin
                        #40000 Next_Char <= 8'h33;end //3
                        5'h00:begin
                        #40000 Next_Char <= 8'h34;end //4
                        5'h00:begin
                        #40000 Next_Char <= 8'h35;end //5
                        5'h00:begin
                        #40000 Next_Char <= 8'h36;end //6
                        5'h00:begin
                        #40000 Next_Char <= 8'h37;end //7
                        5'h00:begin
                        #40000 Next_Char <= 8'h38;end //8
                        5'h00:begin
                        #40000 Next_Char <= 8'h39;end //9
                        5'h00:begin
                        #40000 Next_Char <= 8'h31;
                        #40000 Next_Char <= 8'h30;
                        end //10
                        5'h00:begin
                        end
                        default: Next_Char <= 8'h20; //Blank spaces

                endcase

            end
endcase

 state <= DROP_LCD_E;
```

```
        LCD_E <= 1'b1;
        LCD_RS <= 1'b1; // activate RECEIVE/SEND
        LCD_RW_INT <= 1'b0;

        if (Next_Char[7:4] != 4'h0)// ASCII character to output
              DATA_BUS_VALUE <= Next_Char; // Convert 4-bit value to an ASCII hex
digit
        else if (Next_Char[3:0]>9)// ASCII A...F
              DATA_BUS_VALUE <= {4'h4,Next_Char[3:0]-4'h9};
        else // ASCII 0...9
              DATA_BUS_VALUE <= {4'h3,Next_Char[3:0]};

        // Loop to send out 32 characters to LCD Display  (16 by 2 lines)
        if ((CHAR_COUNT < 31) && (Next_Char != 8'hFE))
          CHAR_COUNT <= CHAR_COUNT + 1'b1;
        else
          CHAR_COUNT <= 5'b00000;

        // Jump to second line?
        if (CHAR_COUNT == 15)
          next_command <= LINE2;
        else if ((CHAR_COUNT == 31) || (Next_Char == 8'hFE))
          next_command <= RETURN_HOME; // Return to first line?
        else
          next_command <= Print_String;
      end

//-------------------------------------------------------------------
      LINE2: // Set write address to line 2 character 1
      begin
        LCD_E <= 1'b1;
        LCD_RS <= 1'b0;
        LCD_RW_INT <= 1'b0;
        DATA_BUS_VALUE <= 8'hC0;
        state <= DROP_LCD_E;
        next_command <= Print_String;
```

```
        end
//-----------------------------------------------------------------
        RETURN_HOME: // Return write address to first postion on line 1
        begin
          LCD_E <= 1'b1;
          LCD_RS <= 1'b0;
          LCD_RW_INT <= 1'b0;
          DATA_BUS_VALUE <= 8'h80;
          state <= DROP_LCD_E;
          next_command <= Print_String;
        end
//-----------------------------------------------------------------
        DROP_LCD_E: // Falling edge loads inst/data to LCD controller
        begin
          LCD_E <= 1'b0; // Stops READ / WRITE
          state <= HOLD;
        end
//-----------------------------------------------------------------
        HOLD: // Hold LCD inst/data valid after falling edge of E line
        begin
          state <= next_command;
        end
        endcase


endmodule

module LCD_display_string(index,out,hex0,hex1);
input [4:0] index;
input [3:0] hex0,hex1;
output [7:0] out;
reg [7:0] out;

endmodule
```