UNIVERSIDADE FEDERAL DE PERNAMBUCO

Departamento de Eletrônica e Sistemas

Sensor de temperatura

Projeto 4

Isaac Neves Farias

Matheus José Araújo Oliveira

Pedro Henrique de Albuquerque Gomes

Recife, 30 de agosto de 2021

Sumário

1 Introdução

O objetivo deste projeto é a aplicação prática dos conceitos vistos em aula, e nesse caso o desenvolvimento de sistemas em VERILOG. Tendo isto em mente, o projeto consiste em um sensor de temperatura LM-75 de sensibilidade de 0.5 graus celsius que informa ao usuário a temperatura em três displays de 7 segmentos, e através do .LCD informa qual a situação do ambiente conforme está quente, frio ou normal. O buzzer é responsável por alertar se o clima está fora da normalidade, se o ambiente está quente ou frio ele executa um alarme sonoro, para cada uma das situações o alarme é diferente.

O sensor de temperatura foi desenvolvido no software Quartus e implementado na placa Cyclone IV.

Sobre a estrutura deste relatório, a seção 2, desenvolvimento, aprofunda nos passos tomados para o sensor funcionar nas especificações solicitadas. A seção é dividida em três etapas após uma visão geral do problema, em que abordam: sensor de temperatura, buzzer e exibição. Uma divisão esquemática pode ser observada na figura 01.

Além disso, na seção 3, manual de operação, é explicado a interface do usuário do sensor de temperatura implementado na placa Cyclone IV. Resultados & discussão, seção 4, traz a avaliação do produto final e discute sobre as dificuldades encontradas pela equipe durante o desenvolvimento do projeto. Por fim, na seção 5, conclusão, pensamentos finais são expressos sobre o projeto como um todo.

2 Desenvolvimento

A solução desenvolvida foi dividida em 3 partes:

1. Comunicação com o sensor LM75

Para base do código foi utilizado o algoritmo disponibilizado pelos professores. Basicamente, neste módulo é apresentado o processo de comunicação com o sensor de temperatura LM75.

O módulo em si é representado pela sigla I2C, que é responsável por realizar a comunicação e integração do sensor ao dispositivo.

2. Buzzer

No módulo buzzer é feita a ativação sonora do dispositivo, para o caso do ambiente se encontrar frio (temperatura menor que 27 graus celsius) é emitido um sinal de duração 100 milissegundos a cada um segundo.

No caso que o ambiente estiver quente (temperatura maior que 31 graus celsius), é emitido um sinal sonoro de 100 milissegundos a cada 250 milissegundos, sendo possível assim diferenciar o alarme para o quente e para o frio.

3. Exibição - LCD e display

O módulo de exibição é subdividido em LCD e display7seg. O módulo de LCD possui um sistema que é reutilizado do programa de VHDL, ele seta as configurações de escrita e então apresenta os comandos determinados pelo projeto a partir da temperatura analisada (quente, normal ou frio).

Já no módulo display7seg, é apenas apresentado ao usuário a temperatura lida pelo sensor em três displays de 7 segmentos, com a sensibilidade em 0.5 graus celsius.

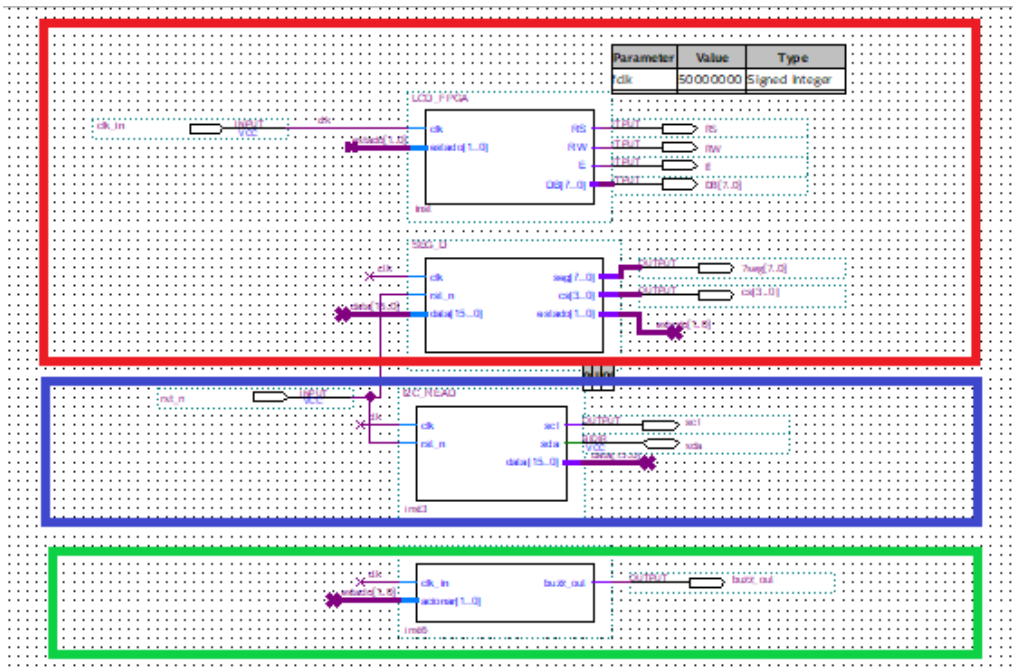

Figura 01: Projeto completo dividido nas partes do desenvolvimento.

2.1 Comunicação com o sensor LM75

Neste módulo é implementado o protocolo de comunicação I2C entre o FPGA e o sensor de temperatura LM75, presentes na placa CycloneIV.

O protocolo I2C é uma comunicação serial em um barramento que utiliza apenas duas linhas: a linha de dados seriais (SDA), e a linha de sincronização (SCL). Estas conectam os dispositivos presentes, sendo eles: o mestre, responsável por coordenar a comunicação, e os escravos que enviam os dados solicitados pelo mestre. No projeto desenvolvido o FPGA é o mestre, o sensor LM75 é o escravo e o código utilizado no FPGA é o coordenação da comunicação.

A Partir do estado neutro, isto é, SDA e SCL em valor digital alto, para dar início a comunicação com o sensor, o mestre leva SDA para o valor baixo. Figuras 02 e 03.

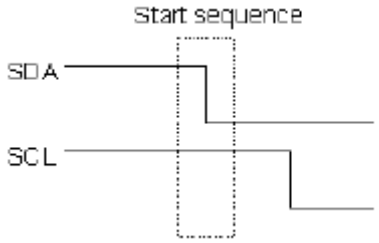

Figura 02: Ilustração da sequência de inicialização.

```
case(state)
    IDLE:  //fica nesse estado até o terminar o segundo atual
        begin
            sda_r    <= 1'b1;
            sda_link <= 1'b1;  //permite enviar no proximo estado
            if(timer_cnt == 26'd49_999_999)
                state <= START;  //vai para o START no segundo 1
            else
                state <= IDLE;
        end
    START: //colocamos o endereço de leitura no registrador que será enviado pelo sda
        begin
            if(`SCL_HIG)  //define os valores no tempo 1s + 1us
                begin
                    sda_r       <= 1'b0;
                    sda_link    <= 1'b1;  //permite enviar no proximo estado
                    address_reg <= `DEVICE_ADDRESS; //esse é o endereço do dispositivo
                    state             <= ADDRESS;
                    data_cnt          <= 4'd0;
                end
            else
                state <= START;
        end
```

Figura 03: Código em *Verilog* iniciando a comunicação.

Iniciada a comunicação o mestre deverá enviar o endereço de 8 bits do dispositivo ao qual quer se comunicar, no caso o LM75. Pelo datasheet do sensor o seu endereço é: 1 0 0 1 A2 A1 A0 R/W, onde A2 A1 A0 são o endereço do sensor, o que permite ter até 8 sensores LM75 no mesmo barramento, como só há 1 sensor o endereço é 000. Já o bit LSB refere-se ao modo de leitura (read) ou escrita (write), para apenas ler os dados este bit deve ser 1.

O endereço 10010001 (0x91) é enviado pelo mestre bit por bit pelo canal SDA a cada pulso de sincronização (SCL). O dispositivo que possui o endereço enviado, o sensor da

placa, responderá com um sinal *Acknowlegde* (ACK), mantendo a linha SDA em baixa durante um pulso de SCL. Figura 04.

```
ADDRESS:
    begin
        if(`SCL_LOW)  //começa no tempo 1s + 3us
            begin
                if(data_cnt == 4'd8)  //quando enviar todos os bits vai para o proximo estado
                    begin
                        state       <= ACK1;
                        data_cnt    <= 4'd0;
                        sda_r       <= 1'b1;
                        sda_link    <= 1'b0;  //permite receber no proximo estado
                    end
                else
                    begin
                        state   <= ADDRESS;  //repete o estado até enviar todos os bits
                        data_cnt <= data_cnt + 1'b1;
                        case(data_cnt)
                            4'd0: sda_r <= address_reg[7];
                            4'd1: sda_r <= address_reg[6];
                            4'd2: sda_r <= address_reg[5];
                            4'd3: sda_r <= address_reg[4];
                            4'd4: sda_r <= address_reg[3];
                            4'd5: sda_r <= address_reg[2];
                            4'd6: sda_r <= address_reg[1];
                            4'd7: sda_r <= address_reg[0];
                            default: ;
                        endcase
                    end
            end
        else
            state <= ADDRESS;
    end
ACK1: //caso o despositivo exista, ele responderá como um Acknowledge
    begin
        if(!sda && (`SCL_HIG))  //(sda mantido baixo até o scl alto)
            state <= READ1;
        else if(`SCL_NEG)       //scl negedge
            state <= READ1;
        else
            state <= ACK1;
    end
```

Figura 04: Código em *Verilog* enviando o endereço do sensor.

Daí, então, o sensor começa a enviar os dados de temperatura bit por bit. Como o protocolo só transmite dados em blocos de 8 bits e o sensor LM75 possui 16 bits de dados para enviar, o mestre (FPGA) deverá enviar um sinal ACK para o sensor após receber os 8 bits mais significativos. Figura 05.

```
READ1:  //recebe os 8 bits msb do sensor por sda
    begin
        if((`SCL_LOW) && (data_cnt == 4'd8))

            begin
                state        <= ACK2;
                data_cnt     <= 4'd0;
                sda_r        <= 1'b1;
                sda_link     <= 1'b1;  //permite enviar no proximo estado
            end
        else if(`SCL_HIG) //recebe os 8 bits durante scl alto
            begin
                data_cnt <= data_cnt + 1'b1;
                case(data_cnt)
                    4'd0: data_r[15] <= sda;
                    4'd1: data_r[14] <= sda;
                    4'd2: data_r[13] <= sda;
                    4'd3: data_r[12] <= sda;
                    4'd4: data_r[11] <= sda;
                    4'd5: data_r[10] <= sda;
                    4'd6: data_r[9]  <= sda;
                    4'd7: data_r[8]  <= sda;
                    default: ;
                endcase
            end
        else
            state <= READ1;
    end
ACK2: //envia acknowledge para o sensor
    begin
        if(`SCL_LOW)
            sda_r <= 1'b0;  //fallingedge
        else if(`SCL_NEG)
            begin
                sda_r        <= 1'b1;
                sda_link     <= 1'b0;  //permite receber no proximo estado
                state        <= READ2;
            end
        else
            state <= ACK2;
    end
READ2: //recebe os 8 bits lsb do sensor por sda
    begin
        if((`SCL_LOW) && (data_cnt == 4'd8))
            begin
                state        <= NACK;
                data_cnt     <= 4'd0;
                sda_r        <= 1'b1;
                sda_link     <= 1'b1;  //permite enviar no proximo estado
            end
        else if(`SCL_HIG)  //recebe os 8 bits durante scl alto
            begin
                data_cnt <= data_cnt + 1'b1;
                case(data_cnt)
                    4'd0: data_r[7] <= sda;
                    4'd1: data_r[6] <= sda;
                    4'd2: data_r[5] <= sda;
                    4'd3: data_r[4] <= sda;
```

Figura 05: Código em *Verilog* recebendo os dados do sensor.

Recebidos e armazenados os dados, o mestre encerra a comunicação enviando um sinal NACK, elevando SDA enquanto SCL está em alta. Assim permanece em repouso até o início do próximo ciclo. Este ciclo de comunicação se repete a cada segundo. Figuras 06 e 07.

```
    NACK: //aborta a comunicação com o sensor pt1
        begin
            if(`SCL_LOW)
                begin
                    state <= STOP;
                    sda_r <= 1'b0;  //envia sda 0 para no proximo posedge encerrar
                end
            else
                state <= NACK;
        end
    STOP:  //quando scl for alto reinicia
        begin
            if(`SCL_HIG)
                begin
                    state <= IDLE;  // volta ao inicio
                    sda_r <= 1'b1;  //envia sda 1 (posedge) enquanto scl alto,
                end
            else
                state <= STOP;
        end
    default: state <= IDLE;
endcase
```

Figura 06: Código em *Verilog* encerrando o ciclo de comunicação.

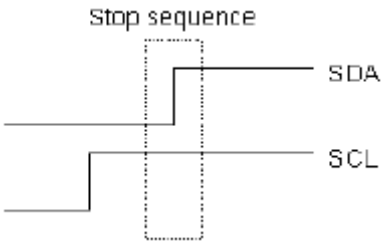

Figura 07: Ilustração da sequência de parada.

Como a linha SDA é bidirecional faz-se necessário o uso de *buffers* de 3 estados para evitar colisão de dados no barramento. Estes buffers tem como objetivo alterar o estado da porta da linha entre permitir envio de dados e alta impedância, o que permite o recebimento de dados. Figuras 08, 09 e 10.

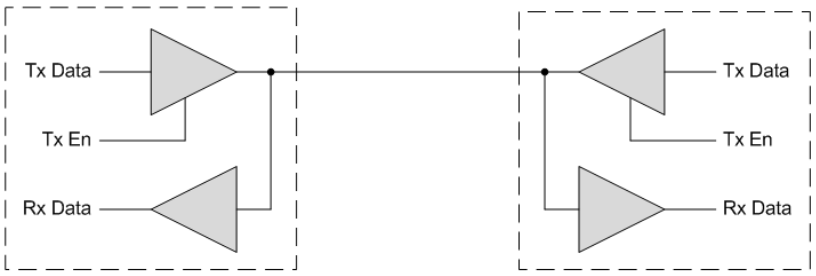

Figura 08: Diagrama de uma comunicação com uma linha bidirecional utilizando *buffers* de 3 estados (os triângulos).

Truth Table - Tri-State Buffer		
Tx Data	Tx Enable	Output
0	1	0
1	1	1
X	0	Z (high impedance)

Figura 09: Tabela verdade do *buffer* de 3 estados.

```
assign sda    = sda_link ? sda_r: 1'bz;
```

Figura 10: Código em *Verilog* do *buffer* de 3 estados.

2.2 Buzzer

O ativamento do buzzer é acionado sempre que o estado da temperatura sai do normal, tendo dois modos de acionamento que diferenciam o estado frio do quente.

Para esse processo de ativação se faz necessário dois sistemas contadores, um para definir a frequência do sinal sonoro (frequência de 2kHz) e outro para o intervalo de tempo em que acontece o "beep" do buzzer.

Para o ambiente frio esse beep possui duração 100 milissegundos a cada intervalo de um segundo. No caso que o ambiente estiver quente, o "beep" possui duração de 100 milissegundos e é ativado a cada 250 milissegundos.

Assim, foram implementados 4 contadores, um para o sinal sonoro e outros 3 para regular a ativação nos estados de temperatura, são eles: *clk_2khz* (som), *clk_10hz* (100ms), *clk_4hz* (250ms) e *clk_1hz* (1s), presentes nas Figuras 11 e 12.

```
case(acionar)
   2'b01: //quente
      begin
         if (clk_1hz==1)
            begin
               if(clk_10hz==1)
                  begin
                     buzzer_r<=clk_2khz;
                  end
               else
                  begin
                     buzzer_r<=1'b0;
                  end
            end
         else
            begin
               buzzer_r<=1'b0;
            end
      end
```

Figura 11: Código em *Verilog* para a ativação do buzzer no estado quente.

```
2'b10: //frio
   begin
      if (clk_4hz==1)
         begin
            if(clk_10hz==1)
               begin
                  buzzer_r<=clk_2khz;
               end
            else
               begin
                  buzzer_r<=1'b0;
               end
         end
      else
         begin
            buzzer_r<=1'b0;
         end
   end
default:
      begin
         buzzer_r<=1'b0;
      end
```

Figura 12: Código em *Verilog* para a ativação do buzzer no estado frio.

2.3 Exibição

2.3.1 Display de 7 segmentos & Decodificador

No módulo do display de 7 segmentos é feito primeiramente a declaração dos dados do sensor a serem expostos, do vetor recebido data [15:0] o valor natural é representado pelos bits de 14 a 8, valor decimal no bit 7 e o sinal no bit 15.

Após isso é implantado um esquema de delay utilizado na máquina de estados, e então a máquina de estados para a seleção do dígito que define em qual display o dígito é representado, Figura 13.

```
always@(posedge clk,negedge rst_n)  //maquina de estado dos digitos
   begin
      if(!rst_n)
         disp_dat<=2'd0;
      else if(delay_cnt==16'd5//0000)
         disp_dat<=disp_dat + 1'b1;
      else
         disp_dat<=disp_dat;
   end

always@(disp_dat)  //decodificador da maquina de estado
   begin
      case(disp_dat)
         2'b00: cs = 4'b1110;
         2'b01: cs = 4'b1101;
         2'b10: cs = 4'b1011;
         2'b11: cs = (dataout_buf > 4'd0) ? 4'b0111 : 4'b1111; // Enable hundreds digit
         default cs = 4'b1111;
      endcase
   end
```

Figura 13: Código em *Verilog* da máquina de estados para representação dos dígitos.

Por último é feita a decodificação dos dígitos , conforme nas Figuras 14 e 15.

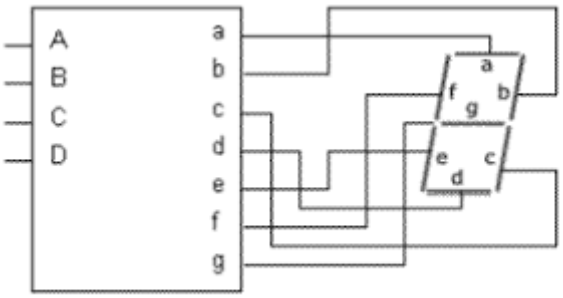

Figura 14: Esquemático de um decodificador e um display de sete segmentos.

```
always@(dataout_buf) //decodificador do 7seg
   begin
      case(dataout_buf)
         4'd0 : seg = 8'hc0; //0
         4'd1 : seg = 8'hf9; //1
         4'd2 : seg = 8'ha4; //2
         4'd3 : seg = 8'hb0; //3
         4'd4 : seg = 8'h99; //4
         4'd5 : seg = 8'h92; //5
         4'd6 : seg = 8'h82; //6
         4'd7 : seg = 8'hf8; //7
         4'd8 : seg = 8'h80; //8
         4'd9 : seg = 8'h90; //9
         default : seg =8'hc0; //0
      endcase
      if(disp_dat == 2'b01)
         seg = seg & 8'b01111111; // Add decimal point for second digit
   end
```

Figura 15: Código em *Verilog* do decodificador 7 segmentos.

2.3.2 LCD

Como já mencionado anteriormente, o código referente ao display LCD está configurado na linguagem de programação VHDL, devido ao reuso do projeto anterior. Dito isto, é possível analisar o seu funcionamento.

Nesse módulo é importante perceber que seu funcionamento depende de uma grande máquina de estados, exemplo na Figura 16, que possui estados de configuração (quando a entrada RS = 0) e de escrita ou leitura (quando RS = 1), o modo de escrita ou leitura é definido pelo parâmetro RW, como pode ser visto na Figura 17.

```
architecture hardware of LCD_FPGA is
type state is (FunctionSet1, FunctionSet2, FunctionSet3,
 FunctionSet4,FunctionSet5,FunctionSet6,FunctionSet7,FunctionSet8,FunctionSet9,Function
 FunctionSet13,FunctionSet14,FunctionSet15, ClearDisplay, DisplayControl, EntryMode,
 WriteData1, WriteData2, WriteData3, WriteData4, WriteData5,WriteData6,WriteData7,Write
 WriteData12,WriteData13,WriteData14,WriteData15,SetAddress1, ReturnHome);
signal pr_state, nx_state: state;
begin
```

Figura 16: Estados utilizados no processo de escrita do LCD.

```
entity LCD_FPGA is
generic (fclk: natural := 50_000_000);
port (
        clk          :     in bit;
        RS, RW       :    out bit;
        E            : buffer bit;
        DB           :    out bit_vector(7 downto 0);
        estado       :    in bit_vector(1 downto 0));
end LCD_FPGA;
```

Figura 17: Entradas e saídas na aplicação do LCD.

Esse comportamento de máquina de estados permite uma execução serial do código, onde cada dígito a ser escrito possui uma configuração predefinida, a representação de cada caractere é feita utilizando a tabela ASCII, pois é um sistema simples e que possui uma vasta gama de caracteres.

Para a representação diferente das palavras quente, frio e normal, basta receber do sistema o estado do quarto e a partir de uma condicional definir os dígitos a serem escritos.

3 Manual de operação

Nesta seção consta o manual de operação do sensor de temperatura. A Figura 18 apresenta uma legenda que determina o local de cada parte do FPGA citados no texto abaixo.

Nesta implementação não há necessidade de fato que o usuário opere a FPGA, no que se refere a botões e chaves, mas apenas saiba onde analisar os dados de interesse.

Os displays de LCD e 7 segmentos indicam qual a temperatura atual do ambiente (quarto) e se ele está frio, quente ou normal. O clima é considerado quente para uma temperatura maior que 31 graus celsius e frio quando for menor que 27 graus celsius, no intervalo entre esses dois valores o clima é considerado normal.

Existe também na placa, um buzzer que será responsável por emitir um alerta para quando o quarto estiver frio e outro alerta diferente para quando o quarto estiver quente, e por último o CI do sensor de temperatura, todos os itens destacados acima estão demarcados na foto.

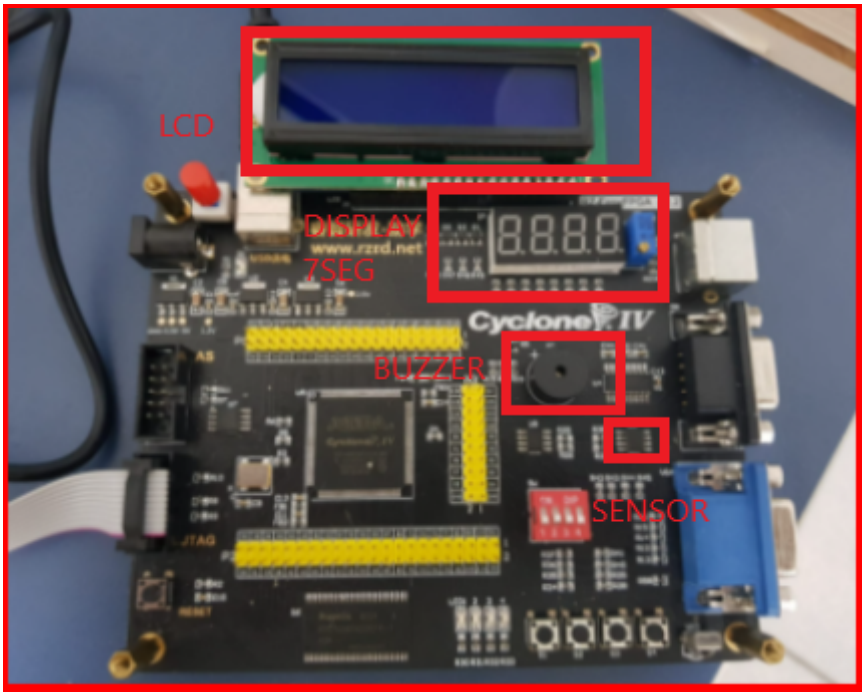

Figura 18: Esquemático Cyclone IV

4 Resultados & Discussão

De modo geral os requisitos para o funcionamento do projeto foram correspondidos.

Para melhor compreensão do sistema a visualização RLT de todo o projeto pode ser observada na Figura 19.

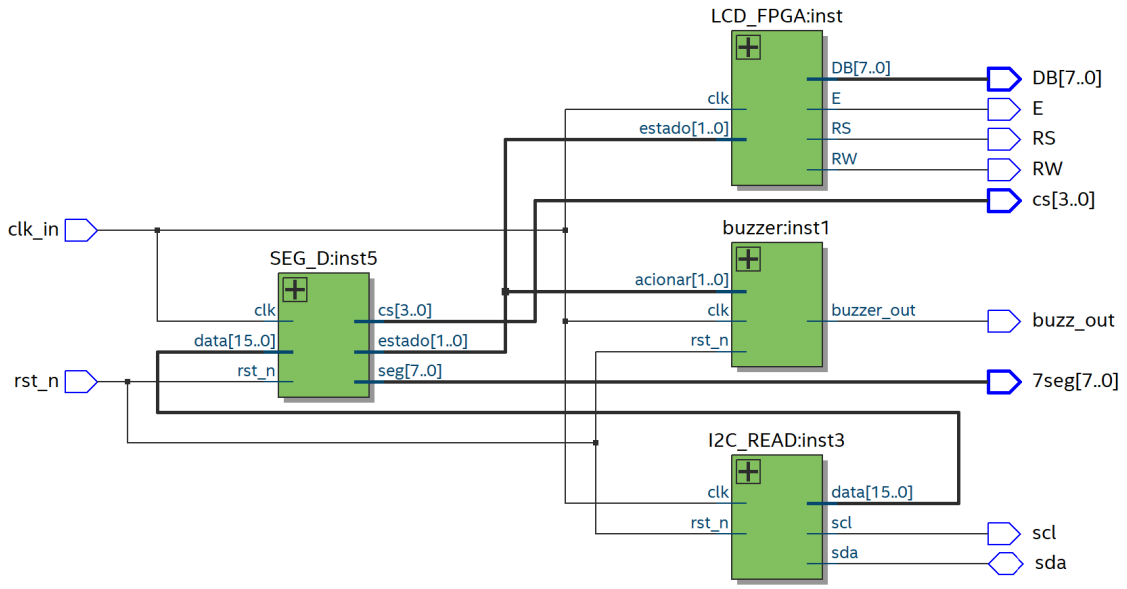

Figura 19: Visualização RLT do projeto

4.1 Principais problemas encontrados

Devido ao tamanho reduzido, o código disponibilizado pelo professor e a possibilidade de usar blocos de outros projetos em outras linguagens, foram encontrados poucos problemas na implementação deste projeto, foram eles:

1. Acionamento do buzzer: O buzzer aciona na frequência desejada para as condições de temperatura estabelecidas, porém não com o sistema de “beeps” requeridos.

2. Exibição no LCD: As letras do LCD se moviam para um lado. Este problema foi solucionado reconfigurando os estados iniciais para ter um estado de function set para cada letra que for ser escrita no LCD.

5 Conclusão

O resultado final apresentado foi satisfatório, visto o que é pedido pelo experimento. Por meio dessa prática foi possível desenvolver e otimizar o conceito de gerenciamento de projetos, assim como uma experiência em equipe.

Além disso, vale destacar que, devido à crescente complexidade dos projetos dos circuitos de eletrônica digital, fica evidente a importância de desenvolver habilidades em *HDLs*, sendo *VERILOG* uma linguagem bastante popular. A prática do presente projeto permitiu aos alunos a exploração e desenvolvimento de tais habilidades.

Por fim, o advento de sensores de temperatura em geral, trouxe um enorme impacto social e econômico no mundo. A partir do controle de temperatura é que se pode desenvolver sistemas auto regulados, principalmente no que se diz respeito a sistemas biológicos, tornando uma ferramenta imprescindível no processo de desenvolvimento e ciência na área.

Entretanto, se esse pensamento ainda for um pouco distante da vida cotidiana para alguns, basta pensar no dispositivo ar condicionado ou na geladeira. Equipamentos básicos para qualquer pessoa no mundo, e que também fazem o uso do controle de temperatura em seu funcionamento.