UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
DEPARTAMENTO DE ELETRÔNICA E SISTEMAS
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

GABRIELA LEITE PEREIRA
BRUNO FRANCA GUIMARÃES
PEDRO LUCAS DE SOUZA LEÃO
HENRIQUE PEDRO DA SILVA

RELATÓRIO DA QUARTA PRÁTICA DE ELETRÔNICA DIGITAL

RECIFE

2023

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
DEPARTAMENTO DE ELETRÔNICA E SISTEMAS
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

GABRIELA LEITE PEREIRA
BRUNO FRANCA GUIMARÃES
PEDRO LUCAS DE SOUZA LEÃO
HENRIQUE PEDRO DA SILVA

RELATÓRIO DA QUARTA PRÁTICA DE ELETRÔNICA DIGITAL

INFORMAÇÕES DA DISCIPLINA
Curso : ENGENHARIA ELETRÔNICA – CTG
Disciplina: ELETRÔNICA DIGITAL 1A
Código: ES441, Turma: EB, Semestre: 2023.1

DOCENTE RESPONSÁVEL: DR. MARCO AURÉLIO BENEDETTI RODRIGUES
DOCENTE ESTAGIÁRIO: MSC. MALKIÇEDHEQ B. C. SILVA

RECIFE

2023

SUMÁRIO

1 INTRODUÇÃO

Nessa quarta prática, a FPGA foi utilizada para formular um music player em Verilog. Este funciona de modo que, se utilizando de um controle remoto, é possível controlar as ações do tocador de música. Cada botão tem uma devida função no play, stop, pause (que serão mostrados pelos LEDS) e skip. Ao final, as músicas deverão reproduzidas pelo buzzer, a duração da música deverá ser mostrada pelo VGA, além do nome da música, o artista que a canta (caso este seja conhecido), o nome da disciplina, o nome dos integrantes do grupo e a temperatura no ambiente, que é possível ser medida pela utilização do LM75A.

O objetivo do trabalho é a utilização da plataforma quartus e, por meio da linguagem Verilog, elaborar o sistema pedido na quarta prática. Para atingir esse objetivo, é necessário que: o primeiro botão escolhido no controle remoto mute a música(a música mantém a reprodução, porém sem acionar o buzzer), o segundo botão escolhido no controle remoto começe, ou pare, a música e a contagem da sua duração, o terceiro botão escolhido no controle remoto pule para a próxima música, que começa parada, o quarto botão escolhido no controle remoto pare a música, zere a contagem da duração da música e volte a música para o início, o tempo da música apareça na interface do VGA, junto com o nome da música, o seu cantor ou cantora, o nome dos integrantes do grupo, o nome da disciplina e a temperatura ambiente.

Este relatório estará dividido em 5 seções. A primeira é o desenvolvimento, no qual será explicado as etapas em que foi feito so processo com suas devidas explicações. A segunda é o manual de operações, no qual será explicado todo o funcionamento do hardware projetado na fpga. A terceira é os resulatdos, que terá, em ordem, o que foi obtido como resposta ao longo do desenvolvimento do projeto. A quarta é a discussão dos resultados, que abordará do porque desses resultados terem sido atingidos. A última é a conclusão que irá discutir, de forma resumida, se os resultados obtidos foram os solicitados no projeto.

2 DESENVOLVIMENTO

Neste capítulo, será discutido, inicialmente, uma visão geral do circuito completo e como seus componentes se comportam em conjunto para gerar uma lógica que toca duas músicas e mostra suas informações, além de outras, numa interface VGA.

Após isso, detalhes de cada componente do circuito serão discutidos.

2.1 O Circuito Completo

Neste circuito, visto na Figura 1, tem-se o bdf completo do projeto. Foram utilizados como entradas gerais do programa, o clock da placa, sendo este utilizado em praticamente todos os blocos do projeto, o rst_r e o ir.

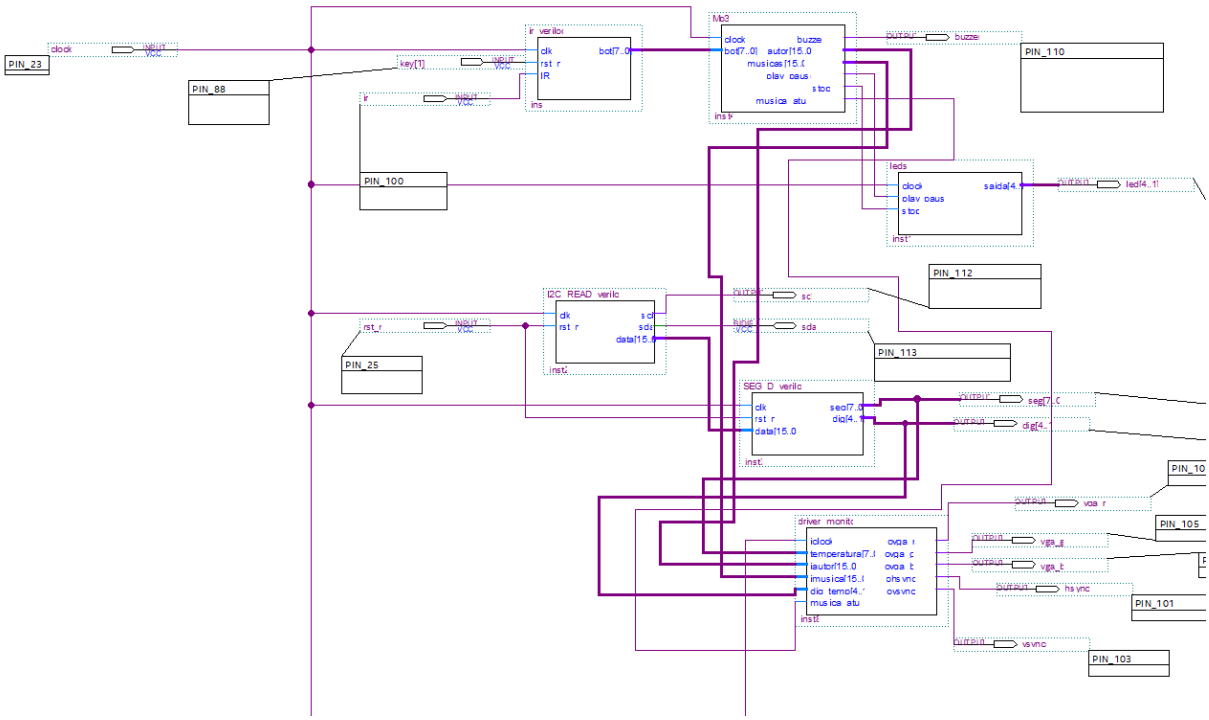

Figura 1 – Representação esquemática do projeto completo.

O ir verilog é o bloco que manda a informação do controle remoto para o Mp3. Assim, pode-se saber qual botão foi apertado e que ação deve ser feito após o aperto dele.

A saída do ir verilog é direcionada para o bloco do Mp3. A saída do Mp3 é direcionada para os LEDS e para o VGA. Além do Mp3 o VGA recebe também a saída do LM75A, que possui o sensor de temperatura.

O bloco dos leds tem as instruções para acender o primeiro led durante o stop, acender o segundo led durante o play, o terceiro durante o pause e o quarto led em um caso especial que

pode ser visto na seção destinada ao bloco dos leds.

Por fim, o bloco do Mp3, que contém uma máquina de estados para efetuar a seleção das músicas que serão reproduzidas no buzzer, envia o nome da música atual bem como o nome de seu autor para o bloco do VGA, para que estas informações sejam visualizadas na sua interface.

2.2 Ir verilog

Na Figura 2 pode-se observar a representação do ir verilog. Ele desempenha um papel crucial ao desenvolver a lógica para o funcionamento do controle remoto. Essa abordagem garante que sinais consistentes e genuínos sejam passados adiante.

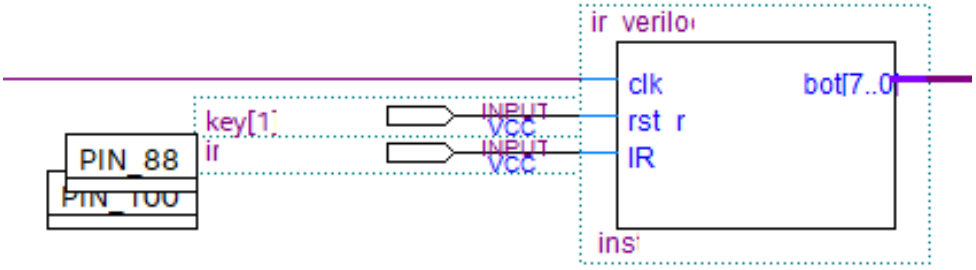

Figura 2 – Representação do ir verilog.

Na Figura 3 pode-se observar os dois contadores necessários para o funcionamento do controle remoto. Esses dois contadores servem para criar um clock de 38 kHz.

```
//---------------------------------------------------------------------------
always @ (posedge clk)
  if (!rst_n) //reset assíncrono
    cnt1 <= 11'd0; //reinicia o contador 1
  else if (irda_change) //na transição de borda de irda
    cnt1 <= 11'd0; //reinicia o contador 1
  else if (cnt1 == 11'd1750) //caso contador estoure
    cnt1 <= 11'd0; //reinicia o contador 1
  else
    cnt1 <= cnt1 + 1'b1; // incrementa o contador 1
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
always @ (posedge clk)
  if (!rst_n) //reset assíncrono
    cnt2 <= 9'd0; //reinicia o contador 2
  else if (irda_change) //na transição de borda de irda
    cnt2 <= 9'd0; //reinicia o contador 2
  else if (cnt1 == 11'd1750) //1750 pulso nível baixo e 1750 pulsos nível alto
    cnt2 <= cnt2 +1'b1; //incrementa o contador 2
 //---------------------------------------------------------------------------
```

Figura 3 – Parte do código do ir verilog.

Após os dois contadores, e antes do códigos conseguir guardar os dados que vem do pressionamento dos botões do controle remoto, tem-se que aguardar dois pulsos. O primeiro pulso é de 9 ms e o segundo pulso é de 4.5 ms. Depois desses dois pulsos pode-se tem a certeza que o que vem são os dados referentes aos botões.

Na Figura 4, é possível observar a lógica que possibilita a manutenção do valor de saída do ir verilog. Os dados que são pertinentes ao pressionamneto do botão tem 32 bits. Esses 32

bits são divididos em 4 grupos de 8 bits cada: c_comando, comando, c_endereço e endereço. Desses 4 grupos somente o grupo do comando tem a informação que será necessária para diferir qual botão foi pressionado. Por conta disso, após a separação dos 32 bits, foi feito um case para definir qual botão foi pressionado.

```
197   //Separa cada byte do protocolo de 32bits ----------------------
198   always @ (posedge clk)
199     if (!rst_n) //reset assíncrono
200       begin
201         c_comando <= 8'b0;
202         comando  <= 8'b0;
203         endereco <= 8'b0;
204         c_endereco <= 8'b0;
205       end
206     else if ((data_cnt ==6'd32) & irda_reg1)
207       begin
208         c_comando <= get_data[7:0];  //complemento do comando
209         comando <= get_data[15:8]; //comando
210         endereco <= get_data[23:16];//endereco
211         c_endereco <= get_data[31:24];//complemento do endereco
212       end
213      else comando <= 0;
214
215
216    always@(comando)
217    begin
218       case(comando)
219          8'h68: bot = 8'b00;  //escolhe botao 0(mute)
220          8'h30: bot = 8'b01;  //escolhe botao 1(play/pause)
221          8'h18: bot = 8'b10;  //escolhe botao 2(skip)
222          8'h7A: bot = 8'b11;  //escolhe botao 3(stop)
223          default: bot = 8'b11111111;
224       endcase
225    end
```

Figura 4 – Parte do código do ir verilog.

Assim, a saída do bloco ir verilog será um pulso com o valor correspondente ao botão pressionado. Esse pulso vai para o Mp3 e ajuda na escolha e manipulação das 2 músicas.

2.3 Mp3

Na Figura 5 tem-se a representação do Mp3. O bloco Mp3 desempenha o papel de chaveamento entre músicas por meio de uma máquina de estados de Moore.

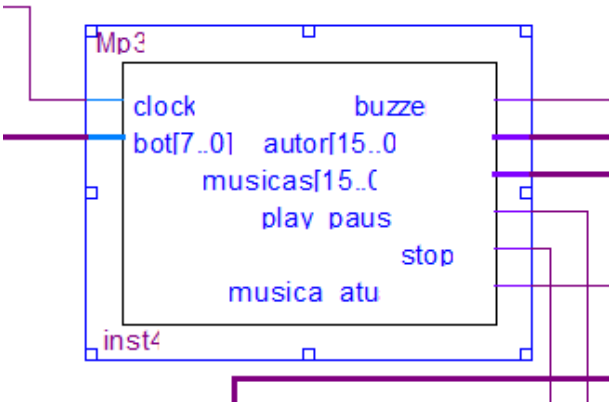

Figura 5 – Representação do Mp3.

Antes da máquina de estados Moore, tem-se o uso de um case, que pode ser visto na Figura 6, para definir a lógica do stop, play_pause, skip e mute. Se pressionado o primeiro botão escolhido do controle remoto(0), o registrador silêncio é selecionado(A música não seria mais

escutada). Se pressionado o segundo botão(1), e o stop não ter sido pressionado anteriormente, o play_pause é selecionado(A música dá play). Se o terceiro botão(2) for presionado, e o stop ter já sido selecionado logo antes, o skip é selecionado(Troca-se de música). O quarto botão(3) seleciona o stop e faz o play_pause ser desselecionado, ou seja, entra no pause e a música é reiniciada.

```
63      always @(posedge clock)
64      begin
65         case(bot)
66            8'b00: silencio <= !silencio;//1'b1;
67            8'b01: begin
68                     if (!stop) play_pause <= !play_pause;
69                     else play_pause <= play_pause;
70                   end
71            8'b10: begin
72                     if (stop) skip <= 1'b1;
73                     else skip <= 1'b0;
74                   end
75            8'b11: begin
76                     stop <= !stop;//1'b1;
77                     play_pause <= 1'b0;
78                   end
79            default begin
80                     silencio <= silencio;
81                     play_pause <= play_pause;
82                     stop <= stop;
83                     skip <= 1'b0;
84                    end
85         endcase
86      end
```

Figura 6 – Parte do código do Mp3.

Dentro da máquina de estados dita anteriormente nessa seção, cuja representação visual está ilustrada na Figura 7, adota-se uma abordagem estruturada, utilizando o botão de **Skip** (botão 2 do controle remoto) como elemento-chave para a transição entre os diferentes estados. Cada estado é projetado para corresponder a uma música específica.

```
167     //FSM Lógica para controle do estado atual
168     always @ (posedge skip)
169     begin : L1
170        if (stop) estado_atual <= prox_estado;
171        else estado_atual <= estado_atual;
172     end
173
174     //FSM Lógica para controle do próximo estado
175     always @ (posedge clock)//Combinacional
176     begin : L2
177        case (estado_atual)
178           s0: prox_estado <= s1;//próximo estado
179           s1: prox_estado <= s0;//próximo estado
180           default: prox_estado <= s0; //recupera de estado inválido
181        endcase
182     end
183
184     //FSM Lógica para controle das saídas
185     always @ (estado_atual)//Combinacional
186     begin : L3
187        case (estado_atual)
188           s0: begin
189              musica1play <= play_pause;
190              musica1stop <= stop;
191              t2 <= musica1t2;
192              t3 <= musica1t3;
193              t5 <= musica1t5;
194              musicas <= musica1_1;
195              autor <= musica1_2;
196              musica_atual <= s0;
197           end
198           s1: begin
199              musica2play <= play_pause;
200              musica2stop <= stop;
201              t2 <= musica2t2;
202              t3 <= musica2t3;
203              t5 <= musica2t5;
204              musicas <= musica2_1;
205              autor <= musica2_2;
206              musica_atual <= s1;
207           end
208        endcase
```

Figura 7 – Parte do código do Mp3.

Além disso, conectam-se os nomes das músicas e seus respectivos autores ao VGA, além da informação de qual música está realmente tocando no momento, onde essas informações são apresentadas. Esse visor de informações fornece um contexto valioso ao usuário, exibindo detalhes relevantes sobre a música que está sendo tocada naquele momento.

É importante destacar que cada elemento desse arranjo opera em harmonia, proporcionando uma experiência de audição otimizada e agradável. A combinação do chaveamento realizado pelo bloco Mp3, da estrutura de estados de Moore e do feedback visual oferecido pelo VGA eleva a qualidade da interação do usuário com o sistema musical, garantindo que a música seja não apenas ouvida, mas também compreendida e apreciada em seu contexto completo.

2.3.1 Temporizador

O temporizador pode ser visto na Figura 8. O código do temporizador desempenha o papel de garantir que cada nota musical seja tocada pelo tempo apropriado e, ao mesmo tempo, permite que o módulo de reprodução musical compreenda o momento exato para progredir para a nota subsequente.

```
1   module temporizador
2   (
3       output Q = 1'b0,
4       input Clk,
5       input Disparo,
6       input [27:0] Overflow
7   );
8       reg [27:0] cnt = 0;
9
10      always @(posedge Clk)
11      begin
12          if ((Disparo) && (cnt == 0))
13              cnt <= Overflow;
14          else if ((!Disparo) && (cnt == 0)) cnt <= 0;
15          else cnt <= cnt - 1'b1;
16      end
17
18      assign Q = (cnt != 0) ? 1'b1 : 1'b0;
19
20  endmodule
21
```

Figura 8 – Código do temporizador.

Através de uma sincronização precisa, o temporizador assegura que cada nota seja sustentada pelo período de tempo desejado, contribuindo para a fidelidade e expressividade da reprodução musical. Isso é especialmente importante para criar uma experiência auditiva agradável e autêntica.

Além disso, o momento do estouro do temporizador é empregado como um sinal de referência para a transição na máquina de estados da música. Quando o temporizador atinge seu limite, isso atua como um marco que indica ao sistema musical que é o momento exato de avançar para a próxima nota. Essa abordagem garante uma sincronização precisa entre a duração de cada nota e a progressão da peça musical como um todo.

2.3.2 Divisor de clock

O divisor de clock pode ser visto na Figura 9. A utilização do divisor de clock desempenha o papel de gerar as frequências a serem reproduzidas pelo buzzer. Esse divisor requer dois parâmetros como entrada: o clock a ser dividido e um vetor de 28 Bits, que atua como o módulo da operação de divisão.

```
1   module divisor_clock
2   (
3       output reg Clk_out = 1'b0,
4       input Clk_in,
5       input [27:0] Overflow
6   );
7       reg [27:0] cnt = 0;
8
9       always @(posedge Clk_in)
10      begin
11          if (cnt < Overflow) begin
12              cnt <= cnt + 1'b1;
13              Clk_out <= Clk_out;
14          end
15          else begin
16              cnt <= 0;
17              Clk_out <= ~Clk_out;
18          end
19      end
20
21  endmodule
22
```

Figura 9 – Código do divisor de clock.

Inicialmente, a contagem é realizada mediante a detecção do sinal **rising_edge** do clock de entrada, garantindo um sincronismo preciso.

No momento em que essa contagem atinge o valor determinado pelo módulo de divisão, ocorre um duplo evento: a contagem é redefinida para seu estado inicial e, simultaneamente, o nível lógico do sinal **Clk_out** é alterado. Essa mudança de nível lógico tem como propósito sinalizar o término da operação de divisão do clock.

2.3.3 Fur Elise

Para fazer a música Fur Elise foi necessário fazer uma máquina de estados, que pode ser vista, em parte, na Figura 10. A máquina de estados tem por objetivo ciclar as notas das músicas fazendo com que uma nota seja tocada após a outra em loop, isso acontece pois em cada estado da máquina de estados, foi definido qual será o próximo estado de forma cíclica. Cada estado da máquina de estado está sensível aos botões, de modo que se o play_pause for pressionado, a música para na exata nota que estava tocando antes de se apertar o botão, ou começa na nota que estava tocando quando se deu o aperto do botão. Se o botão de stop for pressionado a música é reiniciada e colocada no pause. Além disso, cada nota da música foi inserida manualmente no L3,

por meio de uma task, que envia a informação do tempo de duração da nota para o temporizador que fará com que a frequência da nota seja soada pelo buzzer durante um tempo específico definido manualmente, como dito anteriormente.

```
377     //FSM Lógica para controle das saídas
378     always @ (estado)//Combinacional
379     begin : L3
380        case (estado) //s0 apenas inicia a prox nota
381           s0: nota(0,`ov_t2);  //freq atual, dur prox
382           s1: nota(`E4,`ov_t1_2);//freq atual, dur prox
383           s2: nota(`D_4,`ov_t1_2);//freq atual, dur prox
384           s3: nota(`E4,`ov_t1_2);//freq atual, dur prox
385           s4: nota(`D_4,`ov_t1_2);//freq atual, dur prox
386           s5: nota(`E4,`ov_t1_2);//freq atual, dur prox
387           s6: nota(`I3,`ov_t1_2);//freq atual, dur prox
388           s7: nota(`D4,`ov_t1_2);//freq atual, dur prox
389           s8: nota(`C4,`ov_t1_2);//freq atual, dur prox
390           s9: nota(`H3,`ov_t1);//freq atual, dur prox
391           s10: nota(0,`ov_t1_2);//freq atual, dur prox
392           s11: nota(`C3,`ov_t1_2);//freq atual, dur prox
393           s12: nota(`E3,`ov_t1_2);//freq atual, dur prox
394           s13: nota(`H3,`ov_t1_2);//freq atual, dur prox
395           s14: nota(`I3,`ov_t1);//freq atual, dur prox
396           s15: nota(0,`ov_t1_2);//freq atual, dur prox
397           s16: nota(`E3,`ov_t1_2);//freq atual, dur prox
398           s17: nota(`G_3,`ov_t1_2);//freq atual, dur prox
399           s18: nota(`I3,`ov_t1_2);//freq atual, dur prox
400           s19: nota(`C4,`ov_t1);//freq atual, dur prox
401           s20: nota(0,`ov_t1_2);//freq atual, dur prox
402           s21: nota(`E3,`ov_t1_2);//freq atual, dur prox
403           s22: nota(`E4,`ov_t1_2);//freq atual, dur prox
404           s23: nota(`D_4,`ov_t1_2);//freq atual, dur prox
405           s24: nota(`E4,`ov_t1_2);//freq atual, dur prox
406           s25: nota(`D_4,`ov_t1_2);
407           s26: nota(`E4,`ov_t1_2);
408           s27: nota(`I3,`ov_t1_2);
409           s28: nota(`D4, `ov_t1_2);
410           s29: nota(`C4, `ov_t1_2);
411           s30: nota(`H3, `ov_t1);
412           s31: nota(0, `ov_t1_2);
413           s32: nota(`C3, `ov_t1_2);
414           s33: nota(`E3, `ov_t1_2);
415           s34: nota(`H3, `ov_t1_2);
416           s35: nota(`I3, `ov_t1);
417           s36: nota(0, `ov_t1_2);
418           s37: nota(`E3, `ov_t1_2);
419           s38: nota(`C4, `ov_t1_2);
420           s39: nota(`I3, `ov_t1_2);
421           s40: nota(`H3, `ov_t2);
422           s41: nota(0, `ov_t1);
```

Figura 10 – Uma parte do código da música Fur Elise.

2.3.4 Over The Rainbow

Para fazer a música Over The Rainbow foi necessário fazer uma máquina de estados, que pode ser vista, em parte, na Figura 11. A máquina de estados dessa música funciona da mesma maneira da máquina de estado da música Fur Elise. O que muda é o número de estados, as notas utilizadas e a duração dessas notas.

```
263     //FSM Lógica para controle das saídas
264     always @ (estado)//Combinacional
265     begin : L3
266        case (estado) //s0 apenas inicia a prox nota
267           s0: nota(0, `ov_t2); //s0 apenas inicia a prox nota
268           s1: nota(`F3, `ov_t2);
269           s2: nota(`F4, `ov_t2);
270           s3: nota(`E4, `ov_t1);
271           s4: nota(`C4, `ov_t1_2);
272           s5: nota(`D4, `ov_t1_2);
273           s6: nota(`E4, `ov_t1);
274           s7: nota(`F4, `ov_t1);
275           s8: nota(`F3, `ov_t2);
276           s9: nota(`D4, `ov_t2);
277           s10: nota(`C4, `ov_t4);
278           s11: nota(`D3, `ov_t2);
279           s12: nota(`H_3, `ov_t2);
280           s13: nota(`H3, `ov_t1);
281           s14: nota(`F3, `ov_t1_2);
282           s15: nota(`G3, `ov_t1_2);
283           s16: nota(`H3, `ov_t1);
284           s17: nota(`H_3, `ov_t1);
285           s18: nota(`G3, `ov_t1);
286           s19: nota(`E3, `ov_t1_2);
287           s20: nota(`F3, `ov_t1_2);
288           s21: nota(`G3, `ov_t1);
289           s22: nota(`H3, `ov_t1);
290           s23: nota(`F3, `ov_t4);
291           s24: nota(0, `ov_t4);
292
293        endcase
294     end
```

Figura 11 – Uma parte do código da música Over The Rainbow.

2.4 LEDS

Na Figura 12 tem-se a representação do LED, que tem como objetivo acender e/ou apagar os 4 LEDS presentes na placa.

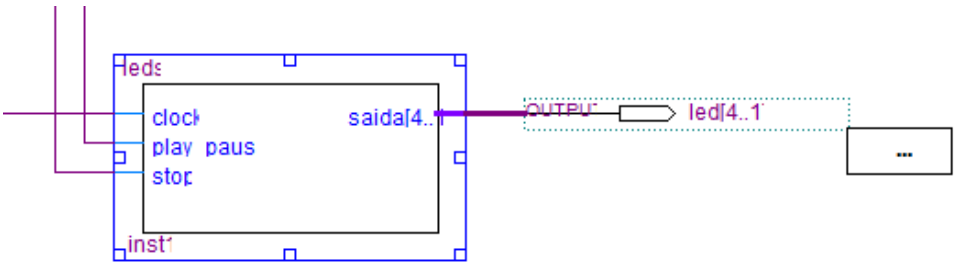

Figura 12 – Representação do LED.

Seu código pode ser visto na Figura 13. Ele funciona para acender os 4 LEDS a partir de certos comandos. O primeiro LED se acenderá quando o stop for pressionado. O segundo LED acende quando o play for selecionado e o terceiro LED acende quando o pause for selecionado. O LED 1 e 2 é notado pois a placa tem ativo baixo, ou seja, ela acende o LED no 0, não no 1. O quarto LED só acenderá se nenhum dos outros estados for verdadeiro, acusando uma falha no código.

```
42
43      always @(posedge clock)
44      begin
45         if (play_pause) saida[2] <= !play_pause;
46         else if (!play_pause) saida[3] <= !saida[2];
47         else if (stop) saida[1] <= !stop;
48         else saida[4] <= !((!play_pause)&(play_pause)&(!stop));
49      end
50
51
```

Figura 13 – Código do LED.

2.5 LM75A

Foi disponibilizado para o grupo um arquivo LM75A que já funcionava. Por conta disso, foram feitas algumas mudanças no arquivo que foi dado. Na Figura 14 tem-se a represntação do LM75A no circuito.

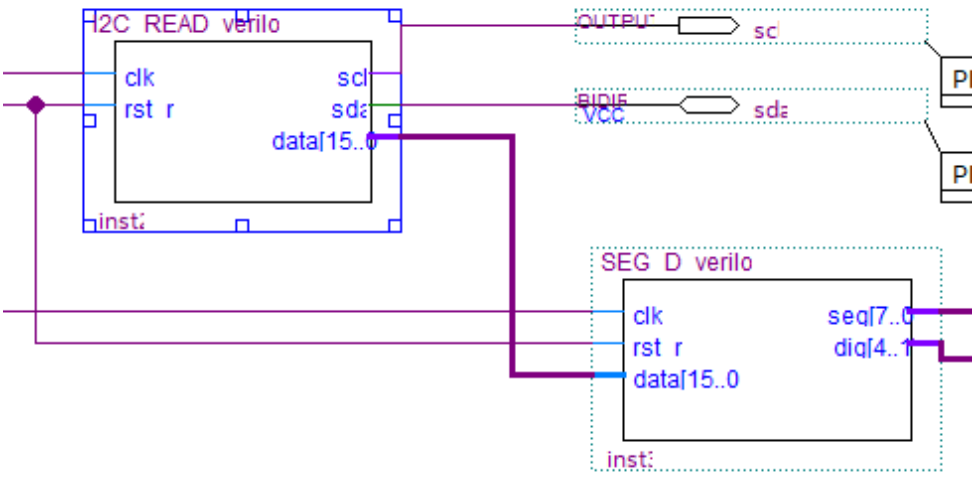

Figura 14 – Representação do LM75A.

2.5.1 I2C Read Verilog

O I2C read tem como modo de funcionamento o sistema escravo-mestre. Nesse I2C o sensor de temperatura se comporta como um escravo, ou seja, só faz a leitura da temperatura e muda seu valor. Já o mestre é a FPGA, pois ela pede a temperatura e manda instruções para o escravo. Portanto, o sensor de temperatura só manda a temperatura se o mestre pedir.

O sistema I2C funciona em torno do uso do sda e do scl. Os sda são o sinal de dados I2C. Ele é um sinal de dados bidirecional usado para transmitir ou receber todos os dados do barramento. Já o scl é o relógio I2C gerado pelo mestre. Embora o escravo nunca gere o sinal de clock, ele pode manter o clock baixo, paralisando o barramento até que esteja pronto para enviar dados.

As instruções que o mestre pode pedir estão definidas em uma máquina de estados, que pode ser vista em parte na Figura 15. Essa máquina de estados tem 9 estados e são eles que

definem o funcionamento do I2C além das instruções que o mestre passa para o escravo.

```
73      localparam  IDLE     = 9'b0_0000_0000,
74                  START    = 9'b0_0000_0010, //inicia comunicação com escravo
75                  ADDRESS  = 9'b0_0000_0100, //envia endereço do escravo
76                  ACK1     = 9'b0_0000_1000, //confirmação pelo escravo
77                  READ1    = 9'b0_0001_0000, //leitura do 1byte MSB (temperatura)
78                  ACK2     = 9'b0_0010_0000, //confirmação pelo mestre
79                  READ2    = 9'b0_0100_0000, //leitura do 1byte LSB (temperatura)
80                  NACK     = 9'b0_1000_0000, //mestre não responde ao escravo
81                  STOP     = 9'b1_0000_0000; //finaliza comunicação com escravo
82
83      //Endereço do dispositivo _ operação leitura
84      `define DEVICE_ADDRESS 8'b1001_0001
85
86      //B5 : Descrição da FSM
87      always@(posedge clk or negedge rst_n) begin : B5
88         if (!rst_n)
89            begin
90               data_r      <= 16'd0; //limpa o registrador de dados
91               sda_r       <= 1'b1;  //transmite um bit 1
92               sda_link    <= 1'b1;  //habilita saída SDA (escrita)
93               estado      <= IDLE;  //reinicia a FSM
94               addr_reg    <= 8'd0;  //endereço inicial 0000_0000
95               data_cnt    <= 4'd0;  //reinicia contador de dados
96            end
97         else
98            case(estado)
99               IDLE: //Estado Inicial
100                  begin
101                     sda_r    <= 1'b1; //transmite um bit 1
102                     sda_link <= 1'b1; //habilita saída SDA (escrita)
103                     if (timer_cnt ==  `TIMER_OVER) //periodo concluido
104                        estado <= START; //inicia a FSM
105                     else  estado <= IDLE;
106                  end
107               START://Mestre inicia comunicação com escravo
108                  begin
109                     if (`SCL_HIG) begin //caso SCL nível alto
```

Figura 15 – Uma parte do código do I2C read.

2.5.2 SEG D Verilog

Esse código tem como objetivo transformar os dados que chegaram do I2C read em valores bínarios e mandar eles para o VGA. Para isso tem-se que fazer, primeiramente, um clock de 1 kHz. Esse clock é utilizado na habilitação das 4 posições que vão aparecer no VGA correspondentes a temperatura. Depois, é utilizado um outro case para a decodificação dos 7 segmentos.

Após essas lógicas, os dados recebidos do I2C são compilados em 4 partes. Cada parte equivale a um número. O último número, será ou 0 ou 5, pois esse último dígito refere-se se a temperatura é exata(24.0), ou se ela tem um meio após a dezena e unidade(24.5). O primeiro número será ou 0, ou 1, pois o sensor não habilita leitura de mais de cem graus celsius. Já os dois números do meio, são as decodificações da dezena e da unidade da leitura da temperarura. Uma parte desse código pode ser visto na Figura 16.

```
63          //data[7] decimal da temperatura (0 ->.0º, 1->.5º)
64          4'b1110:  dataout_buf = data[7] ? 4'h5 : 4'h0;
65          //dado display 3 (unidade da temperatura)
66          4'b1101: begin //data[15:8] MSB+LSB da temperatura, data[15] sinal (0 -> +, 1-> -)
67             if (data[15]) dataout_buf = 4'd0; //caso temp negativa valor exibido 0
68             else
69                if (data[14:8] >= 10 && data[14:8] < 20) dataout_buf = data[14:8] - 7'd10;
70                else if  (data[14:8] >= 20 && data[14:8] < 30) dataout_buf = data[14:8] - 7'd20;
71                else if  (data[14:8] >= 30 && data[15:8] < 40) dataout_buf = data[14:8] - 7'd30;
72                else if  (data[14:8] >= 40 && data[15:8] < 50) dataout_buf = data[14:8] - 7'd40;
73                else if  (data[14:8] >= 50 && data[15:8] < 60) dataout_buf = data[14:8] - 7'd50;
74                else if  (data[14:8] >= 60 && data[15:8] < 70) dataout_buf = data[14:8] - 7'd60;
75                else if  (data[14:8] >= 70 && data[15:8] < 80) dataout_buf = data[14:8] - 7'd70;
76                else if  (data[14:8] >= 80 && data[15:8] < 90) dataout_buf = data[14:8] - 7'd80;
77                else if  (data[14:8] >= 90 && data[15:8] < 100) dataout_buf = data[14:8] - 7'd90;
78                else if  (data[14:8] >= 100 && data[15:8] < 110) dataout_buf = data[14:8] - 7'd100;
79                else if  (data[14:8] >= 110 && data[15:8] < 120) dataout_buf = data[14:8] - 7'd110;
80                else if  (data[14:8] >= 120) dataout_buf = data[14:8] - 7'd120;
81                else dataout_buf = data[11:8]; //apenas unidade
82          end
83          //dado display 2 (dezena da temperatura)
```

Figura 16 – Uma parte do código do Seg D Verilog.

2.6 VGA

Na Figura 17 tem-se a representação do VGA, que tem como objetivo mostrar o nome do grupo, a disciplina, o nome da música que está tocando e o seu autor e a temperatura ambiente.

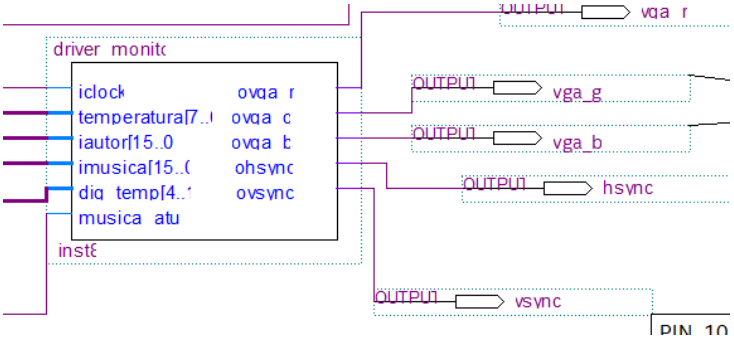

Figura 17 – Representação do VGA.

2.6.1 Horizontal sync

Na Figura 18 tem-se uma parte do código, que auxilia na leitura da linha horizontal da interface VGA. A área visível horizontal é maior, logo o clock que percorre toda a linha horizontal precisa ser mais rápido que o clock que percorre a linha vertical. Essa sincronização é importante, pois é ela que garante que o monitor esteja sincronizado corretamente.

```
//*************************VGA hsync*********************************
//varredura horizontal de pixels
always @(posedge vga_clk)
begin
   if (hcount_ov)
      hcount <= 10'd0;
   else
      hcount <= hcount + 10'd1;
   end
assign hcount_ov = (hcount == hpixel_end);
//---------------------------------------------
```

Figura 18 – Uma parte do código da sincronização horizontal.

2.6.2 Vertical sync

Na Figura 19 tem-se uma parte do código, que auxilia na leitura da linha vertical da interface VGA. A área visível vertical é menor, logo o clock que percorre toda a linha vertical precisa ser mais lento que o clock que percorre a linha horizontal. Essa sincronização é importante, pelo mesmo motivo da sincronização horizontal.

```
97      //************************VGA vsync*******************************
98      //varredura de coluna -----------------------
99      always @(posedge vga_clk)
100     begin
101        if (hcount_ov)
102        begin
103           if (vcount_ov)
104           vcount <= 10'd0;
105        else
106           vcount <= vcount + 10'd1;
107        end
108     end
109     assign  vcount_ov = (vcount == vline_end);
110     //-----------------------------------------
```

Figura 19 – Uma parte do código da sincronização vertical.

2.6.3 Pixel generator

Nesse arquivo foi feito a modelagem das letras e números que aparecem na interface VGA. Na Figura 20, tem-se um exemplo de uma lógica para a construção da letra L. Primeiramente, foi feito um laço interativo que faz uma linha vertical começando de cima até em baixo. Para isso, no laço interativo a posição y, responsável pela altura, é a que muda com a mudança do i. Depois disso foi feita um outro laço para construir uma linha horizontal em baixo. Dessa vez, é a posição x que muda com a mudança do i.

```
task draw_L(input [10:1] POSX, POSY, THICK, input [3:1] COLOR);
    begin
         integer i;
            for(i=0; i<9; i=i+1) begin
               draw_sqr(POSX, POSY+i*THICK, THICK, COLOR);
            end

            for(i=0; i<5; i=i+1) begin
               draw_sqr(POSX+i, POSY+8*THICK, THICK, COLOR);
            end
    end
endtask
```

Figura 20 – Código para construção do L.

Outras letras, e números também, foram formadas usando tasks com lógicas diferentes. No final, essas letras e números foram chamados individualmente para formar a interface do VGA. Uma parte desse código pode ser visto na Figura 21. Esse código faz parte de um always sensível à mudança do clock da FPGA. Dentro dele, foram definidos parâmetros que garantiam a

localização correta das informações dentro da interface, além do uso das tasks para definir cada número e cada letra.

```
192
193         draw_bg(WHITE);
194
195         // Alunos
196
197           draw_A(pos_alunos,90, 2, RED);
198           draw_L(pos_alunos+22,90, 2, RED);
199           draw_U(pos_alunos+44,90, 2, RED);
200           draw_N(pos_alunos+66,90, 2, RED);
201           draw_O(pos_alunos+88,90, 2, RED);
202           draw_S(pos_alunos+110,90, 2, RED);
203
204         // Bruno
205
206           draw_B(pos_alunos,120, 2, RED);
207           draw_R(pos_alunos+22,120, 2, RED);
208           draw_U(pos_alunos+44,120, 2, RED);
209           draw_N(pos_alunos+66,120, 2, RED);
210           draw_O(pos_alunos+88,120, 2, RED);
211
212        // Gabriela
213
214           draw_G(pos_alunos,150, 2, RED);
215           draw_A(pos_alunos+22,150, 2, RED);
216           draw_B(pos_alunos+44,150, 2, RED);
217           draw_R(pos_alunos+66,150, 2, RED);
218           draw_I(pos_alunos+88,150, 2, RED);
219           draw_E(pos_alunos+110,150, 2, RED);
220           draw_L(pos_alunos+132,150, 2, RED);
221           draw_A(pos_alunos+154,150, 2, RED);
222
223           // Henrique
224
225           draw_H(pos_alunos,180, 2, RED);
226           draw_E(pos_alunos+22,180, 2, RED);
227           draw_N(pos_alunos+44,180, 2, RED);
228           draw_R(pos_alunos+66,180, 2, RED);
229           draw_I(pos_alunos+88,180, 2, RED);
230           draw_Q(pos_alunos+110,180, 2, RED);
231           draw_U(pos_alunos+132,180, 2, RED);
232           draw_E(pos_alunos+154,180, 2, RED);
233
```

Figura 21 – Código para construção da interface VGA.

3 MANUAL DE OPERAÇÃO

Quando o hardware é sintetizado na FPGA, a interface VGA inicializa com a primeira música selecionada, sem emitir som, com o nome dos integrantes do grupo, o nome da disciplina e a temperatura ambiente, como pode ser visto na Figura 22.

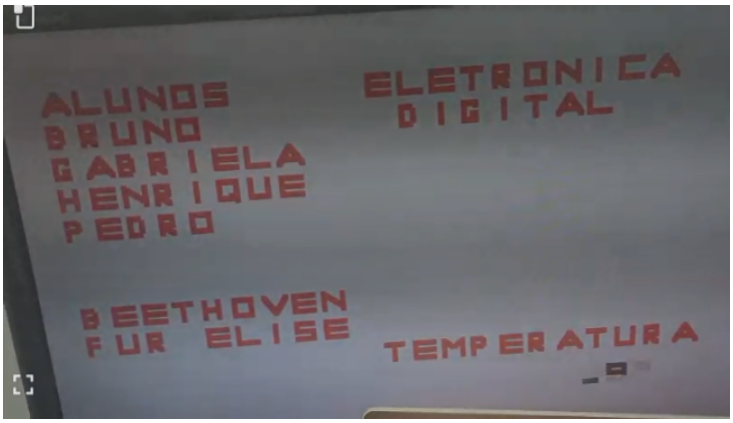

Figura 22 – Início do hardware.

Na controle remoto será utilizado os 4 botões em destaque na Figura 23

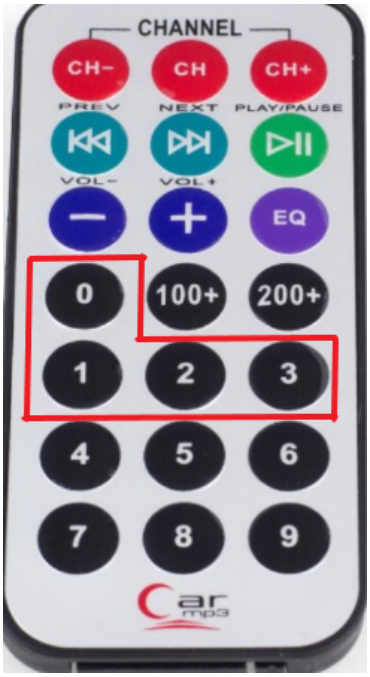

Figura 23 – Botões que serão utilizados no controle remoto.

Se pressionado o segundo botão selecionado, mostrado na Figura 24, a primeira música, Fur Elise de Beethoven, começa a tocar e o segundo LED irá acender, como pode ser visto na Figura 25.

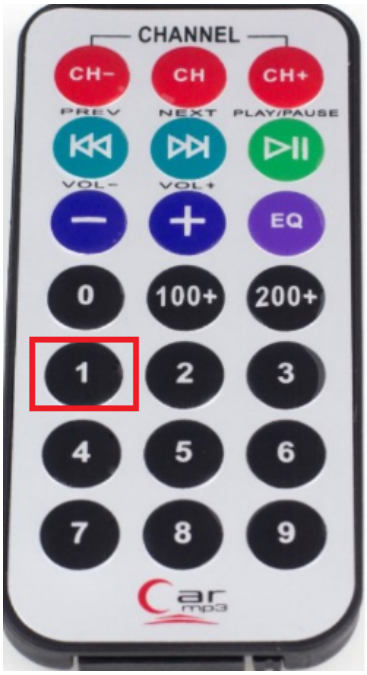

Figura 24 – Botão selecionado 2.

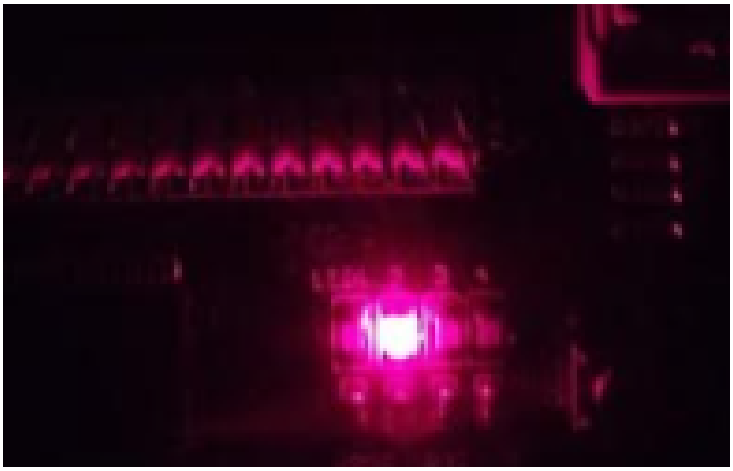

Figura 25 – Início da música Fur Elise.

Se o segundo botão selecionado for pressionado novamente, a música entrará num pause exatamente na nota que estava tocando quando o segundo botão selecionado foi pressionado. O segundo LED apagará e o terceiro LED acenderá, como visto na Figura 26. Para despausar a música e acender novamente o segundo LED, basta pressionar o segundo botão selecionado novamente.

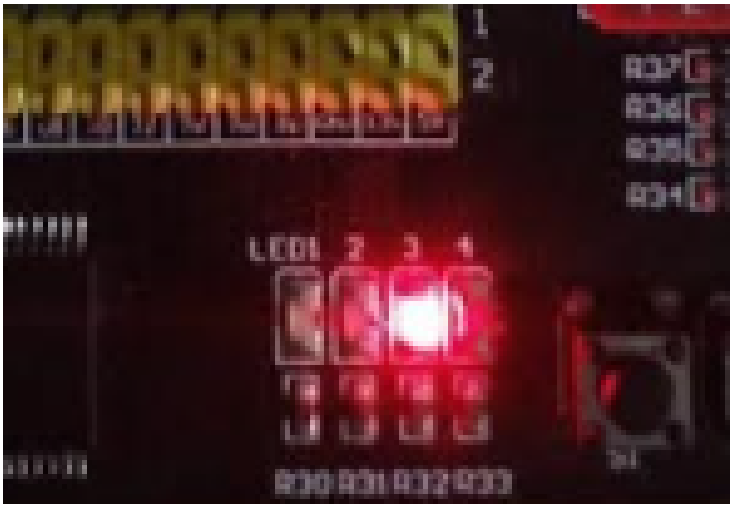

Figura 26 – Pausa na música Fur Elise.

Quando o primeiro botão selecionado, mostrado na Figura 27, for pressionado, o buzzer da música será mutado. Isso significa que, se a música estiver tocando, ela não será escutada pelo usuário. Se o primeiro botão selecionado for pressionado novamente, a música voltará a tocar normalmente.

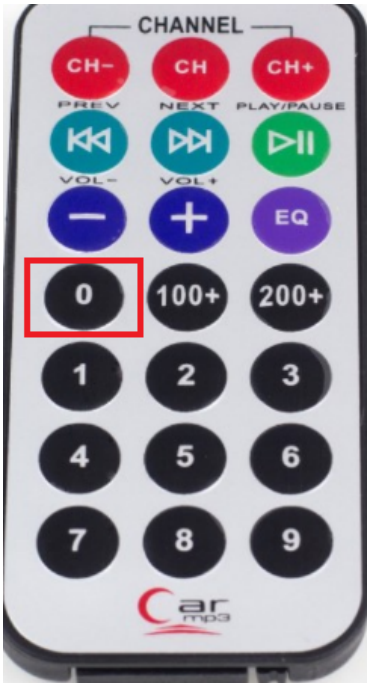

Figura 27 – Botão selecionado 1.

Se pressionado o terceiro botão selecionado, mostrado na Figura 28, a música será trocada para a próxima e a interface VGA mostrará quem é essa nova música.

Figura 28 – Botão selecionado 3.

Porém, a música só será trocada se, no momento em que se pressionar o o terceiro botão selecionado, a música atual estiver no stop, que será visto mais a frente. Se a música estiver no stop, o primeiro led se acenderá, como pode ser visto na Figura 29, e o led atrelado ao pause também se acenderá. Se a música estiver no stop e, além disso, o terceiro botão selecionado for pressionado, a interface VGA só trocará o nome para a próxima música e seu cantor e a nova música é selecionada, mas não toca imediatamente, como mostrado na Figura 30. Para essa música começar a tocar, o usuário deve pressionar o segundo botão selecionado.

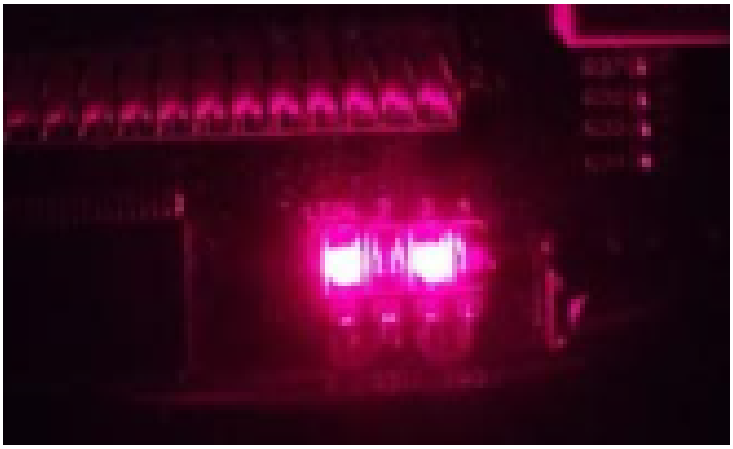

Figura 29 – Led atrelado ao stop e pause acesso.

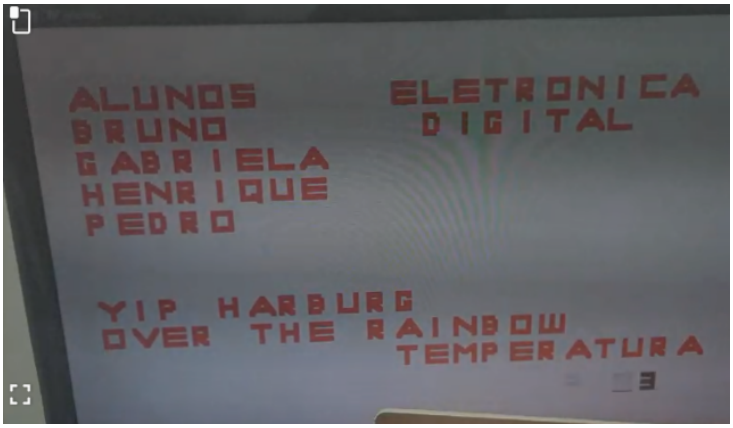

Figura 30 – Nova música selecionada.

A troca das músicas, pelo pressionamento do terceiro push-button, é um processo cíclico. Ou seja, a primeira troca é para Over The Rainbow, a segunda troca é para Fur Elise e assim continua o ciclo, pois somente duas músicas estão programadas para tocar. Todas essas trocas são vistas na interface VGA, pois ela mostra, entre outras coisas, o nome da música e seu cantor toda vez que a música é trocada.

Se o quarto botão selecionado, mostrado na Figura 31, for pressionado, a música será pausada e reiniciada. Para fazer a música tocar novamente, é necessário pressionar o segundo botão selecionado.

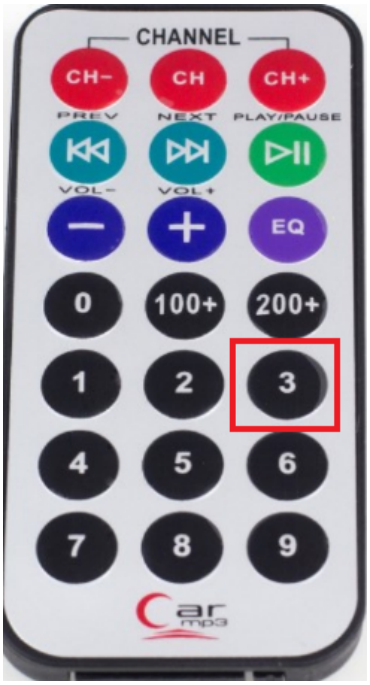

Figura 31 – Botão selecionado 4.

Se o último LED estiver acesso, isso significa que a música está em algum estado que não seja stop, pause ou play. Nesse estado, é possível indentificar uma falha no sistema.

Durante toda o uso da FPGA, as informações na interface VGA referentes ao nome dos integrantes do grupo e nome da disciplina não serão mudados. O nome da música e do cantor mudará na interface a partir do momento que o usuário faz o uso dos botões selecionados. A informação da temperatura na interface muda de acordo com a leitura do LM75 da temperatura ambiente, porém essa leitura não é clara por conta de erros na parte da implementação do VGA.

4 RESULTADOS

Os requerimentos do projeto foram finalizados. Eles são:

- O primeiro botão escolhido no controle remoto habilita ou desabilita o mute.
- O segundo botão escolhido no controle remoto faz tanto a música dar play, quanto faz a música dar pause.
- O terceiro botão escolhido no controle remoto seleciona a próxima música a ser tocada.
- O quarto botão escolhido no controle remoto faz o stop da música.
- Foi implementado, na interface VGA um registrador da temperatura ambiente e de informações sobre a música e o grupo.
- Os LEDS foram implementados para acender quando o play, stop e pause estiverem ativados, sendo o quarto LED ativado quando nenhum desses comandos estiverem sendo utilizados.

4.1 Vídeo

Vídeo do funcionamento da placa

O vídeo acima mostra o funcionamento prático do projeto implementado à placa, junto com a narração de como ela funciona.

5 DISCUSSÃO DOS RESULTADOS

Neste capitulo será discutido os desafios, e soluções que levaram ao funcionamento do projeto de acordo com as especificações.

5.1 O Circuito Completo

A seguir apresentam-se os desafios e as soluções do circuito completo.

5.1.1 Desafios

O maior desafio para implementar todo o circuito foi aprender sintaxe do Verilog. Além disso, foi encontrado um desafio ao tentar entender as saídas hexadecimais, bínarias e inteiras de certos blocos.

5.1.2 Soluções

Para solucionar esses problemas foi feita a análise dos projetos exemplos disponibilizados no classroom, juntamente com os slides das aulas. Esse aprendizado foi importante para a compreensão das ligações entre os blocos e seus respectivos inputs e outputs.

5.2 Ir verilog

A seguir apresentam-se os desafios e as soluções do ir verilog.

5.2.1 Desafios

Um dos desafios foi entender como o sinal sai do ir verilog depois de pressionado o botão no controle remoto.

5.2.2 Soluções

Para solucionar esses problemas foi implementado um else no código do ir verilog quando os 32 bits são separados em 4 grupos. Isso foi feito com o objetivo de transformar o comando que sai do controle remoto ao apertar um botão em um pulso.

5.3 Mp3

A seguir apresentam-se os desafios e as soluções do Mp3.

5.3.1 Desafios

O principal desafio foi fazer a conexão entre musica que irá ser tocada e a música atual com o divisor de clock e com o temporizador. Outro desafio foi na implementação do clock das músicas, que para essa prática foi de 16 MHz.

5.3.2 Soluções

Para isso, foi feito o arquivo que pode ser visto na subseção 2.3. Esse bloco possui uma máquina de estados com 2 estados, onde cada estado representa uma música. Ao apertar o botão de passar para a próxima musica enquanto o stop está ativado, o estado atual passa para o próximo estado. Assim, a música atual é alterada. As músicas só serão conectadas ao divisor de clock e ao temporizador, quando a máquina de estados estiver no estado desta música. Com isso, ao apertar o botão de play, a música atual será enviada para o buzzer. Também foi feito o uso do divisor de clock para criar o novo clock de 16 MHz.

5.4 LEDS

A seguir apresentam-se os desafios e as soluções dos LEDS.

5.4.1 Desafios

Encontrou-se a necessidade de mostrar nos leds quando o stop estiver ativo, quando o play estiver ativo, quando o pause estiver ativo e quando nenhuma dessas entradas estiverem ativas, indicando que há uma falha no sistema.

5.4.2 Soluções

Para isso, foi feito o arquivo que pode ser ser visto na subseção 2.4. Esse bloco tem como entrada a saída stop e play/pause do Mp3. O led 1 é aceso quando o stop estiver ligado, o led 2 é aceso quando o play estiver ligado, o led 3 é aceso quando o pause estiver ligado e o led 4 é aceso quando nenhuma das entradas estiverem ligadas.

5.5 LM75A

A seguir apresentam-se os desafios e as soluções do LM75A.

5.5.1 Desafios

Um dos desafios na implementação do LM75A foi fazer a sua saída ir para o VGA e o seu valor aparecer na interface. Além disso, outro desafio foi na compreensão do código.

5.5.2 Solução

Para tentar solucionar esses problemas, foi feita uma pesquisa sobre o sistema I2C e uma leitura do material de apoio disponibilizado pelo professor. Além disso, a entrada do VGA foi convertida para receber a saída do do seg d verilog. Porém essa saída trocava valores muito rapidamente e o valor atual da temperatura não foi registrado com precisão.

Uma solução para esse problema talvez fosse pegar as informações de temperatura não do seg d verilog, mas sim direto do bloco do I2C read. Assim, teria-se as informações atuais sem nenhuma interferência da frequência.

5.6 VGA

A seguir apresentam-se os desafios e as soluções do VGA.

5.6.1 Desafios

Um dos desafios da implementação do VGA foi formar as letras que foram utilizadas na interface, além de conseguir informar a temperatura ambiente e o contador na interface.

5.6.2 Soluções

Foram utilizados os pixels disponíveis para formar as letras e números e, além disso, um estudo nos materiais de apoio e nos slides foi feito para garantir que a saída dos blocos Mp3 e LM75A e as entradas do VGA fossem equivalentes.

Como dito na seção anterior, uma possível solução para o aparecimento da temperatura na interface VGA seja, ao invês de tentar pegar as informações direto da saída do bloco seg d verilog, pegar as informações da saída do I2C read.

Já sobre o desafio do contador, ele poderia ser resolvido se fosse feito um contador, sensível a mudança do stop e do play/pause, e colocar seus números em certas possições da interface VGA na qual pudesse ser feito a diferenciação dos segundos, minutos e horas com o auxílio das letras hrs, min e seg.

6 CONCLUSÃO

Pode-se concluir que, em geral, o aprendizado de como construir um music player, junto com o aprendizado do uso do controle remoto foi atingido. O VGA e o LM75A foram compreendidos quanto a sua funcionalidade, mas a sua execução e nível de hardware no projeto foi falha, pois não foi atingido o objetivo completo deles. Os quatro botões no controle remoto foram implementados, assim como seus LEDS.

Houve dificuldades na relação teoria versus prática em várias ocasiões. No circuito completo, a implementação do Verilog foi um problema no começo, pois ainda existiam dúvidas sobre como implementar a linguagem. Para executar algumas instâncias houve dificuldades até que foi compreendido a lógica correta. Outra dificuldade, foi na implementação das músicas. O clock pedido foi de 16 MHz, então antes de fazer a música funcionar um divisor de clock teve que ser implementado. Porém a maior dificuldade do projeto foi na implementação da interface VGA. Apesar do grupo ter entendido sua função e seu funcionamento, a implementação dela foi falha, pois não foi possível implementar em sua interface um contador e o valor correto da temperatura ambiente.

A realização do relatório foi muito importante para o grupo. Isso porque, foi pela realização dele, que foi possível ter discussões sobre como melhorar o projeto e sobre o que cada arquivo do quartus fazia. Também foi possível, para cada integrante do grupo, compreender melhor todas as partes do projeto, já que a elaboração do relatório incentivou o diálogo entre todos os integrantes do grupo e, consequentemente, levou a um melhor trabalho na equipe.