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 TERCEIRA 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 TERCEIRA 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 terceira prática, a utilização da FPGA foi para formular um music player em VHDL. Ele funciona de modo que ao apertar os push-buttons da FPGA, cada push-button tem uma devida função no play, stop, pause e passar para a próxima música. No final, as músicas serão tocadas pelo buzzer, a duração da música será mostrada pelo display e o nome da música, mais o artista que a canta, será mostrado no LCD.

O objetivo do trabalho é a utilização da plataforma quartus e, por meio da linguagem VHDL, elaborar o sistema pedido na terceira prática. Para atingir esse objetivo, é necessário que: o primeiro push-button mute a música(o som continua, mas não é ouvido), o segundo push-button começe, ou pare, a música e o display, o terceiro push-button pule para a próxima música, que começa parada, o quarto push-button pare a música, zere o display e volte a música para o início, o tempo da música apareça no display, o número da música apareça nos LEDS e o nome da música, com o seu cantor ou cantora, apareça no LCD.

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 mega-sena.

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, os quatro push-buttons e o clock da placa, sendo este último utilizado em praticamente todos os blocos do projeto. Os push-buttons são conectados ao bloco controle, que tem como função principal aplicar o debounce aos push-buttons.

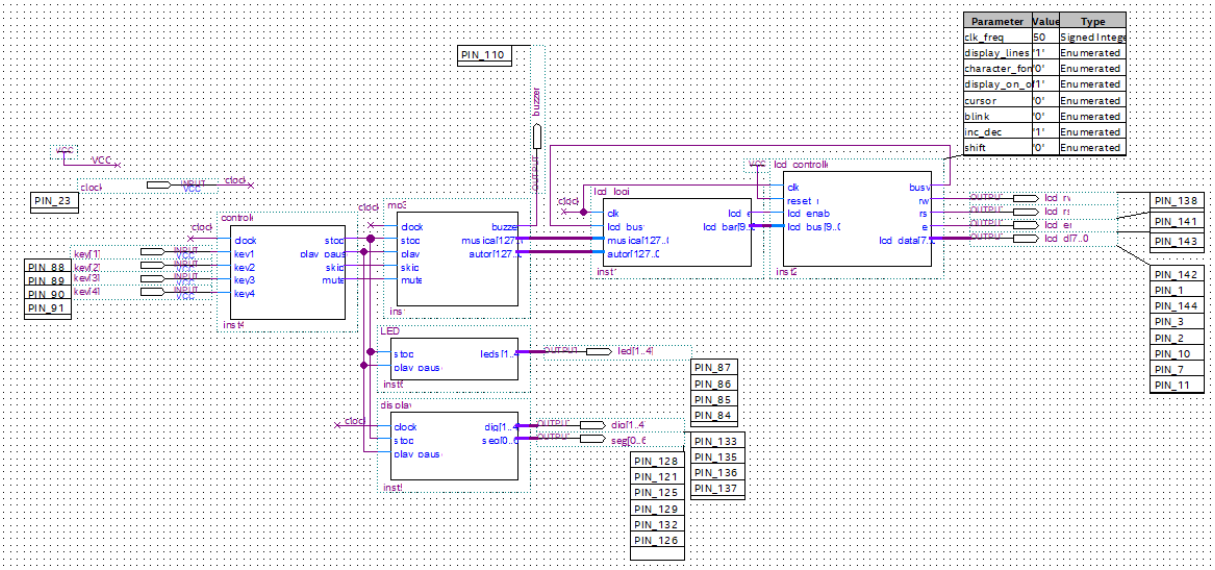

Figura 1 – Representação esquemática do projeto completo.

As saídas do controle são direcionadas para os blocos do mp3, led e display. O display exibe a contagem de quanto tempo a música está tocando, resetando a contagem quando o stop é pressionado e pausando a contagem quando o pause é ativado.

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 LCD-Logic, para que estas informações sejam exibidas no LCD.

2.2 Controle

Na Figura 2 pode-se observar a representação do controle. Ele desempenha um papel crucial ao aplicar a técnica de debounce nos quatro push-buttons de pressão, assegurando que qualquer ruído ou flutuação nos sinais seja filtrado. Em seguida, ele encaminha os sinais que atingiram estabilidade para as funções subsequentes, que então os empregam de maneira confiável. Essa abordagem garante que apenas os sinais consistentes e genuínos sejam passados adiante, contribuindo para um funcionamento preciso e confiável do sistema.

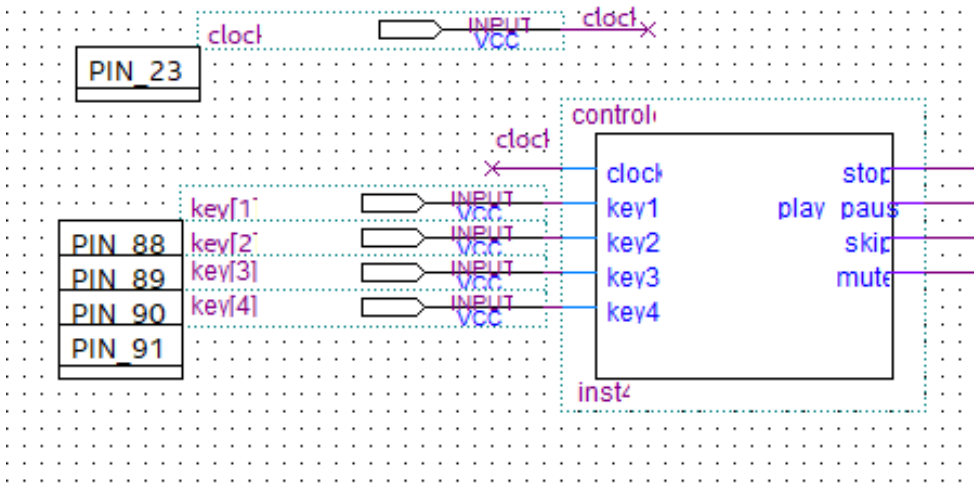

Figura 2 – Representação do controle.

Na Figura 3 pode-se observar a chamada do debouncer que está no package. O primeiro e o terceiro push-button tem como resultado o próprio mute e o skip(responsável por mudar as músicas), mas o segundo e o quarto push-button tem saídas que ainda serão modificadas por conta do pause/play e stop.

```
12  ARCHITECTURE estrutural OF controle IS
13      SIGNAL b2, toggle2, b4, toggle4 : std_logic;
14  BEGIN
15      debouncer1 : debouncer PORT MAP (
16          clk => clock,
17          button => key1,
18          result => mute
19      );
20
21      debouncer2 : debouncer PORT MAP (
22          clk => clock,
23          button => key2,
24          result => b2
25      );
26
27      debouncer3 : debouncer PORT MAP (
28          clk => clock,
29          button => key3,
30          result => skip
31      );
32
33      debouncer4 : debouncer PORT MAP (
34          clk => clock,
35          button => key4,
36          result => b4
37      );
38
```

Figura 3 – Parte do código do controle.

Na Figura 4, é possível observar a lógica que possibilita a manutenção do valor de saída do debouncer após o botão ser solto. Em outras palavras, estam sendo simulado um **tff** na saída do debouncer, no caso dos botões de play/pause e stop. Foram utilizados dois process, um para o botão 2 e outro para o botão 4. O process do botão 2 serve para verificar se houve uma borda de descida no botão. Caso ela tenha acontecido, inverte o valor do signal toggle 2, que é equivalente a saída do play/pause. No process do botão 4, é verificada a borda de descida do botão 4. Caso ela tenha acontecido, inverte o valor do signal toggle 4, que equivalente a saída do stop. Além disso, caso o toggle 4 esteja em nivel lógico alto, ele faz com que o toggle 2 vá para o nivel lógico baixo.

```
    PROCESS (b2)
        BEGIN
        IF (toggle4 = '1') THEN
            toggle2 <= '0';
        ELSIF FALLING_EDGE(b2) THEN
            IF (toggle2 = '0') THEN
                toggle2 <= '1';
            ELSE
                toggle2 <= '0';
            END IF;
        END IF;
    END PROCESS;

    play_pause <= toggle2;

    PROCESS (b4)
        BEGIN
        IF FALLING_EDGE(b4) THEN
            IF (toggle4 = '0') THEN
                toggle4 <= '1';
            ELSE
                toggle4 <= '0';
            END IF;
        END IF;
    END PROCESS;
    stop <= toggle4;

END estrutural;
```

Figura 4 – Parte do código do controle.

2.2.1 Debouncer

No controle foi utilizado o debouncer. Ele pode ser visto na Figura 5. O processo de debouncing, utilizado para garantir a estabilidade dos sinais provenientes dos push-buttons de pressão, emprega uma estratégia com dois flip-flops dispostos em uma configuração de buffer. Nesse contexto, foi aplicado uma operação lógica **XOR** entre esses flip-flops, com o objetivo de identificar qualquer diferença nos sinais capturados. Quando a saída dessa operação **XOR** proporciona um valor de 0, isto é, quando os dois flip-flops produzem estados idênticos, desencadea-se uma contagem que opera ao longo de um período de $10.5ms$.

```
12
13  ARCHITECTURE logic OF debouncer IS
14      --VARIABLE counter_size : INTEGER := 19; --counter size (19 bits dá 10.5 ms com cl
15      SIGNAL flipflops : std_logic_vector (1 DOWNTO 0);
16      SIGNAL counter_set : std_logic;
17      SIGNAL counter_out : std_logic_vector (counter_size DOWNTO 0) := (OTHERS => '0');
18  BEGIN
19
20      counter_set <= flipflops(0) xor flipflops(1);
21
22      PROCESS(clk)
23      BEGIN
24          IF(clk'EVENT and clk = '1') THEN
25              flipflops(0) <= not button;
26              flipflops(1) <= flipflops(0);
27              IF (counter_set = '1') THEN
28                  counter_out <= (OTHERS => '0');
29              ELSIF (counter_out(counter_size) = '0') THEN
30                  counter_out <= counter_out + 1;
31              ELSE
32                  result <= flipflops(1);
33              END IF;
34          END IF;
35      END PROCESS;
36  END logic;
```

Figura 5 – Parte do código do debouncer.

Ao atingir o término desse intervalo de contagem, toma-se a decisão de transmitir o resultado gerado pelo segundo flip-flop para a saída do sistema, como um sinal processado e estável. No entanto, se a operação **XOR** resultar em um valor de 1, o que indica que os estados dos flip-flops estão divergentes, procede-se ao reset da contagem em andamento. Em seguida, adota-se uma abordagem de espera, onde aguarda-se que o sinal de entrada se estabilize completamente antes de continuar o processamento.

Por meio dessa metodologia cuidadosamente estruturada, conseguimos mitigar quaisquer oscilações transitórias ou ruídos presentes nos sinais provenientes dos push-buttons de pressão, assegurando que somente informações consistentes e confiáveis sejam encaminhadas para as etapas subsequentes do sistema. Isso contribui para um desempenho global mais preciso e confiável do sistema como um todo.

2.3 Mp3

Na Figura 6 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.

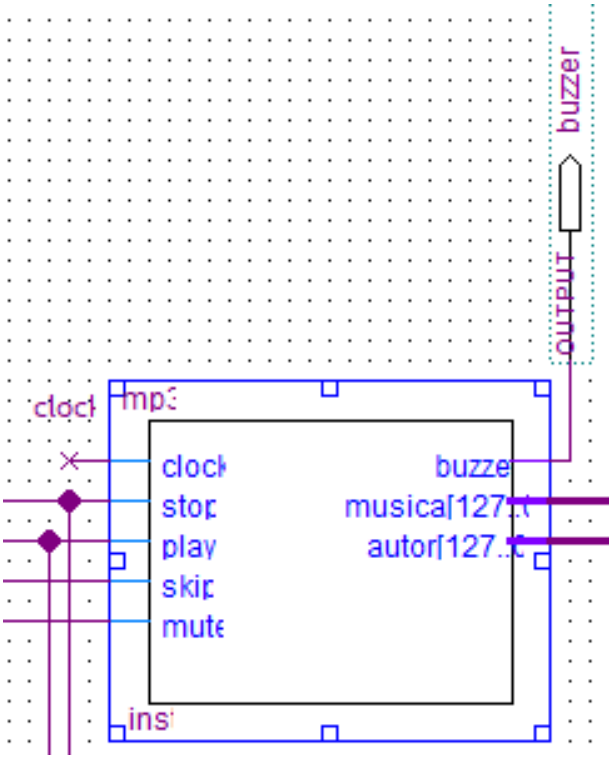

Figura 6 – Representação do Mp3.

Dentro dessa máquina de estados, cuja representação visual está ilustrada na Figura 7, adota-se uma abordagem estruturada, utilizando o push-button de **Skip** como elemento-chave para a transição entre os diferentes estados. Cada estado é projetado para corresponder a uma música específica. Nestes estados, os push-buttons de pressão estão interligados com os blocos de controle de cada música, permitindo a interação para a seleção das faixas musicais.

```
120     L3: PROCESS (musica_atual)
121     BEGIN
122        CASE musica_atual IS
123           WHEN m1 =>
124              musica1play <= play;
125              musica1stop <= stop;
126              t2 <= musica1t2;
127              t3 <= musica1t3;
128              t5 <= musica1t5;
129              musica <= musica1_1;
130              autor <= musica1_2;
131           WHEN m2 =>
132              musica2play <= play;
133              musica2stop <= stop;
134              t2 <= musica2t2;
135              t3 <= musica2t3;
136              t5 <= musica2t5;
137              musica <= musica2_1;
138              autor <= musica2_2;
139           WHEN m3 =>
140              musica3play <= play;
141              musica3stop <= stop;
142              t2 <= musica3t2;
143              t3 <= musica3t3;
144              t5 <= musica3t5;
145              musica <= musica3_1;
146              autor <= musica3_2;
147           WHEN m4 =>
148              musica4play <= play;
149              musica4stop <= stop;
```

Figura 7 – Parte do código do Mp3.

Além disso, conectam-se os nomes das músicas e seus respectivos autores ao LCD, onde são apresentados. 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 LCD 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 Package pkg_buzzer

No Mp3 foi usado dois packages. Uma parte do primeiro pode ser visto na Figura 8. Esse package teve função de instanciar certos códigos para que eles pudessem ser utilizados em mais de uma aplicação. Os códigos instanciados foram os do: debouncer, contador, divisor de clock, temporizador e as 4 músicas utilizadas.

```
COMPONENT debouncer
    PORT(
        clk, button : IN std_logic;
        result: OUT std_logic);
END COMPONENT;

COMPONENT contador
    PORT (clock, enable, clear : IN std_logic;
          overflow : IN std_logic_vector (3 DOWNTO 0);
          contagem: OUT std_logic);
END COMPONENT;

COMPONENT divisor_clock
    PORT (Clk_in : IN std_logic;
          Overflow : IN std_logic_vector (27 DOWNTO 0);
          Clk_out: OUT std_logic);
END COMPONENT;

COMPONENT temporizador IS
PORT ( Clk, Disparo :IN std_logic;
       Overflow : IN std_logic_vector (27 DOWNTO 0);
       Q :OUT std_logic );
END COMPONENT;

COMPONENT furElise IS
PORT (   Clk_out, Disparo  : OUT std_logic;
         Temp_out, Freq_out : OUT std_logic_vector (27 DOWNTO 0);
         Clk_in, Duracao, Stop_in, Play_in : IN std_logic
      );
END COMPONENT;
```

Figura 8 – Uma parte do package pkg_buzzer.

2.3.2 Package lcd_vhdl

O segundo package que foi utilizado, pode ser visto, uma parte, na Figura 9. Esse package teve função de instanciar o lcd controller, além de mandar algumas instruções para o lcd em relação ao seu funcionamento e em como ele mostra os caracteres na sua tela.

```
PACKAGE BODY lcd_vhdl_package IS
    --converte uma string em uma array de vetores de 8bits
    FUNCTION to_std_logic_vector( s : string ) RETURN std_logic_vector
    IS --variavel auxiliar para armazenamento temporário
        VARIABLE r : std_logic_vector( 0 TO s'LENGTH * 8 - 1) ;
    BEGIN
        FOR i IN 1 TO s'HIGH LOOP --percorre todos os caracteres da string
            --converte cada caractere em um vetor de 8bits
            --e armazena na variável auxiliar em ordem crescente
            r((i - 1) * 8  TO i * 8 - 1) := std_logic_vector( to_unsigned( character'POS(s(i)) , 8 ) ) ;
        END loop ;
    RETURN r ; --retorna a array de vetores de 8bits
    END FUNCTION ;
    --inverte a sequência de caracteres numa string
    FUNCTION reverse( s : string ) RETURN string
    IS --variavel auxiliar para armazenamento temporário
        VARIABLE r : string(s'HIGH DOWNTO s'LOW) ;
    BEGIN
        FOR i IN 1 TO s'HIGH LOOP --percorre todos os caracteres da string
            --inverte a posição de cada caractere
            --eg. 8bits r(7) := s(0) e r(0):= s(7)
            r(s'HIGH + 1 - i) := s(i) ;
        END LOOP ;
    RETURN r ;
    END FUNCTION ;

END PACKAGE BODY lcd_vhdl_package ;
```

Figura 9 – Uma parte do package lcd_vhdl.

2.3.3 Temporizador

O temporizador pode ser visto na Figura 10. 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  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.ALL;
3  USE IEEE.numeric_std.ALL;
4  ENTITY temporizador IS
5  PORT ( Clk, Disparo :IN std_logic;
6         Overflow : IN std_logic_vector (27 DOWNTO 0);
7         Q :OUT std_logic := '0');
8  END temporizador;
9  ARCHITECTURE sem_rearme OF temporizador IS
10    SIGNAL cnt : integer := 0;
11 BEGIN
12    PROCESS (Clk)
13    BEGIN --reset assíncrono
14       IF rising_edge(Clk) THEN
15          IF (Disparo = '1' AND cnt = 0) THEN
16             cnt <= to_integer(unsigned(Overflow)); --carrega o temp.
17          ELSIF (Disparo = '0' AND cnt = 0) THEN cnt <= 0;--retem
18          ELSE cnt <= cnt - 1 ; --decrementa
19          END IF;
20       END IF;
21       --nível alto durante o período de contagem
22       IF cnt /= 0 THEN Q <= '1';
23       ELSE Q <= '0';
24       END IF;
25    END PROCESS;
26 END sem_rearme;
```

Figura 10 – 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.

2.3.4 Divisor de clock

O divisor de clock pode ser visto na Figura 11. A utilização do divisor de clock desempenha o papel de selecionar 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  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.ALL;
3  USE IEEE.numeric_std.ALL;
4
5  ENTITY divisor_clock IS
6     PORT (Clk_in   : IN std_logic;
7           Overflow : IN std_logic_vector (27 DOWNTO 0);
8           Clk_out  : OUT std_logic := '0');
9  END divisor_clock;
10
11 ARCHITECTURE clock OF divisor_clock IS
12    SIGNAL toggle : std_logic := '0';
13    SIGNAL cnt : integer := 0;
14 BEGIN
15    PROCESS(Clk_in)
16    BEGIN
17       IF rising_edge(Clk_in) THEN
18          IF cnt < to_integer(unsigned(Overflow)) THEN
19             cnt <= cnt + 1; --incrementa
20             toggle <= toggle;
21          ELSE
22             cnt <= 0; --reinicia
23             toggle <= not toggle; --alterna
24          END IF;
25       END IF;
26    END PROCESS;
27    Clk_out <= toggle;
28 END clock;
```

Figura 11 – Código do divisor de clock.

Inicialmente, o vetor de 28 Bits é convertido em um formato do tipo **integer**, o qual viabiliza a execução de uma contagem progressiva. Essa 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.

2.3.5 Fur Elise

Para fazer a música Fur Elise foi necessário fazer uma máquina de estados, que pode ser vista, em parte, na Figura 12. 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. Além disso, cada nota da música foi inserida manualmente no PROCESS L3, 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.

```
460   L3: PROCESS (Clk_in, estado_atual)
461   BEGIN
462      IF rising_edge(Clk_in) THEN
463         CASE estado_atual IS
464            WHEN s0 => nota(0, ov_t2); --s0 apenas inicia a prox nota
465            WHEN s1 => nota(E4, ov_t1_2);
466            WHEN s2 => nota(D_4, ov_t1_2);
467            WHEN s3 => nota(E4, ov_t1_2);
468            WHEN s4 => nota(D_4, ov_t1_2);
469            WHEN s5 => nota(E4, ov_t1_2);
470            WHEN s6 => nota(I3, ov_t1_2);
471            WHEN s7 => nota(D4, ov_t1_2);
472            WHEN s8 => nota(C4, ov_t1_2);
473            WHEN s9 => nota(H3, ov_t1);
474            WHEN s10 => nota(0, ov_t1_2);
475            WHEN s11 => nota(C3, ov_t1_2);
476            WHEN s12 => nota(E3, ov_t1_2);
477            WHEN s13 => nota(H3, ov_t1_2);
478            WHEN s14 => nota(I3, ov_t1);
479            WHEN s15 => nota(0, ov_t1_2);
480            WHEN s16 => nota(E3, ov_t1_2);
481            WHEN s17 => nota(G_3, ov_t1_2);
482            WHEN s18 => nota(I3, ov_t1_2);
483            WHEN s19 => nota(C4, ov_t1);
484            WHEN s20 => nota(0, ov_t1_2);
485            WHEN s21 => nota(E3, ov_t1_2);
486            WHEN s22 => nota(E4, ov_t1_2);
487            WHEN s23 => nota(D_4, ov_t1_2);
```

Figura 12 – Uma parte do código da música Fur Elise.

2.3.6 Over The Waves

Para fazer a música Over The Waves foi necessário fazer uma máquina de estados, que pode ser vista, em parte, na Figura 13. 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.

```
387   L3: PROCESS (Clk_in, estado_atual)
388   BEGIN
389      IF rising_edge(Clk_in) THEN
390         CASE estado_atual IS
391            WHEN s0 => nota(0, ov_t2); --s0 apenas inicia a prox nota
392            WHEN s1 => nota(E3, ov_t2);
393            WHEN s2 => nota(E3, ov_t1);
394            WHEN s3 => nota(D_3, ov_t1);
395            WHEN s4 => nota(E3, ov_t1);
396            WHEN s5 => nota(G3, ov_t1);
397            WHEN s6 => nota(C4, ov_t2);
398            WHEN s7 => nota(C4, ov_t2);
399            WHEN s8 => nota(I3, ov_t1);
400            WHEN s9 => nota(C4, ov_t1);
401            WHEN s10 => nota(D4, ov_t1);
402            WHEN s11 => nota(C4, ov_t1);
403            WHEN s12 => nota(I3, ov_t1);
404            WHEN s13 => nota(C4, ov_t1);
405            WHEN s14 => nota(E3, ov_t1);
406            WHEN s15 => nota(G3, ov_t1);
407            WHEN s16 => nota(I3, ov_t4);
408            WHEN s17 => nota(I3, ov_t2);
409            WHEN s18 => nota(F3, ov_t2);
410            WHEN s19 => nota(F3, ov_t1);
411            WHEN s20 => nota(E3, ov_t1);
412            WHEN s21 => nota(F3, ov_t1);
413            WHEN s22 => nota(G3, ov_t1);
414            WHEN s23 => nota(I3, ov_t4);
415            WHEN s24 => nota(H_3, ov_t1);
```

Figura 13 – Uma parte do código da música Over The Waves.

2.3.7 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 14. 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.

```
305     BEGIN
306         IF rising_edge(Clk_in) THEN
307             CASE estado_atual IS
308                 WHEN s0 => nota(0, ov_t2); --s0 apenas inicia a prox nota
309                 WHEN s1 => nota(F3, ov_t2);
310                 WHEN s2 => nota(F4, ov_t2);
311                 WHEN s3 => nota(E4, ov_t1);
312                 WHEN s4 => nota(C4, ov_t1_2);
313                 WHEN s5 => nota(D4, ov_t1_2);
314                 WHEN s6 => nota(E4, ov_t1);
315                 WHEN s7 => nota(F4, ov_t1);
316                 WHEN s8 => nota(F3, ov_t2);
317                 WHEN s9 => nota(D4, ov_t2);
318                 WHEN s10 => nota(C4, ov_t4);
319                 WHEN s11 => nota(D3, ov_t2);
320                 WHEN s12 => nota(H_3, ov_t2);
321                 WHEN s13 => nota(H3, ov_t1);
322                 WHEN s14 => nota(F3, ov_t1_2);
323                 WHEN s15 => nota(G3, ov_t1_2);
324                 WHEN s16 => nota(H3, ov_t1);
325                 WHEN s17 => nota(H_3, ov_t1);
326                 WHEN s18 => nota(G3, ov_t1);
327                 WHEN s19 => nota(E3, ov_t1_2);
328                 WHEN s20 => nota(F3, ov_t1_2);
329                 WHEN s21 => nota(G3, ov_t1);
330                 WHEN s22 => nota(H3, ov_t1);
331                 WHEN s23 => nota(F3, ov_t4);
332                 WHEN s24 => nota(0, ov_t4);
333 --Não é necessário WHEN OTHERS pois
334 ---o controle é feito no processo L2
335             END CASE;
```

Figura 14 – Uma parte do código da música Over The Rainbow.

2.3.8 Frere Jacque

Para fazer a música Frere Jacque foi necessário fazer uma máquina de estados, que pode ser vista, em parte, na Figura 15. 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.

```
388     BEGIN
389        IF rising_edge(Clk_in) THEN
390           CASE estado_atual IS
391              WHEN s0 => nota(0, ov_t2); --s0 apenas inicia a prox nota
392              WHEN s1 => nota(F3, ov_t1);
393              WHEN s2 => nota(G3, ov_t1);
394              WHEN s3 => nota(H3, ov_t1);
395              WHEN s4 => nota(F3, ov_t1);
396              WHEN s5 => nota(F3, ov_t1);
397              WHEN s6 => nota(G3, ov_t1);
398              WHEN s7 => nota(H3, ov_t1);
399              WHEN s8 => nota(F3, ov_t1);
400              WHEN s9 => nota(H3, ov_t1);
401              WHEN s10 => nota(H_3, ov_t1);
402              WHEN s11 => nota(C4, ov_t2);
403              WHEN s12 => nota(H3, ov_t1);
404              WHEN s13 => nota(H_3, ov_t1);
405              WHEN s14 => nota(C4, ov_t2);
406              WHEN s15 => nota(C4, ov_t1_2);
407              WHEN s16 => nota(D4, ov_t1_2);
408              WHEN s17 => nota(C4, ov_t1_2);
409              WHEN s18 => nota(H_3, ov_t1_2);
410              WHEN s19 => nota(H3, ov_t1);
411              WHEN s20 => nota(F3, ov_t1);
412              WHEN s21 => nota(C4, ov_t1_2);
413              WHEN s22 => nota(D4, ov_t1_2);
414              WHEN s23 => nota(C4, ov_t1_2);
415              WHEN s24 => nota(H_3, ov_t1_2);
416              WHEN s25 => nota(H3, ov_t1);
417              WHEN s26 => nota(F3, ov_t1);
418              WHEN s27 => nota(G4, ov_t1);
```

Figura 15 – Uma parte do código da música Frere Jacque.

2.4 LEDS

Na Figura 16 tem-se a representação do LED, que tem como objetivo acender e/ou apagar os 4 LEDS presentes na placa.

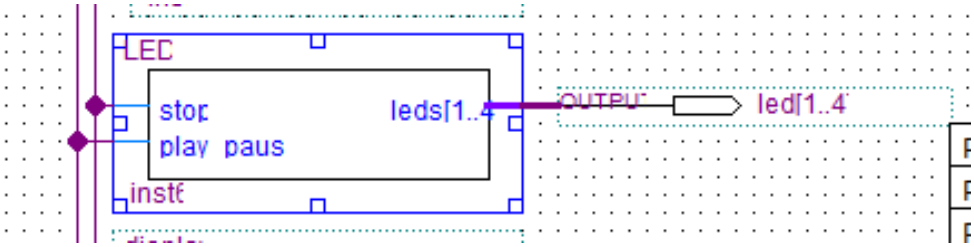

Figura 16 – Representação do LED.

Seu código pode ser visto na Figura 17. 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.

```
1   LIBRARY IEEE;
2   USE IEEE.std_logic_1164.ALL;
3   USE IEEE.pkg_buzzer.ALL;
4
5   ENTITY LED IS
6       PORT (
7           stop, play_pause : IN std_logic;
8           leds : OUT std_logic_vector (1 TO 4)
9           );
10  END LED;
11
12  ARCHITECTURE estrutural OF LED IS
13
14  BEGIN
15      leds(1) <= not stop;
16      leds(2) <= not play_pause;
17      leds(3) <= play_pause;
18      leds(4) <= not (not stop and not play_pause and play_pause);
19  END estrutural;
20
```

Figura 17 – Código do LED.

2.5 Display

Na Figura 18 tem-se a representação do display, que tem como objetivo fazer a contagem das durações das músicas.

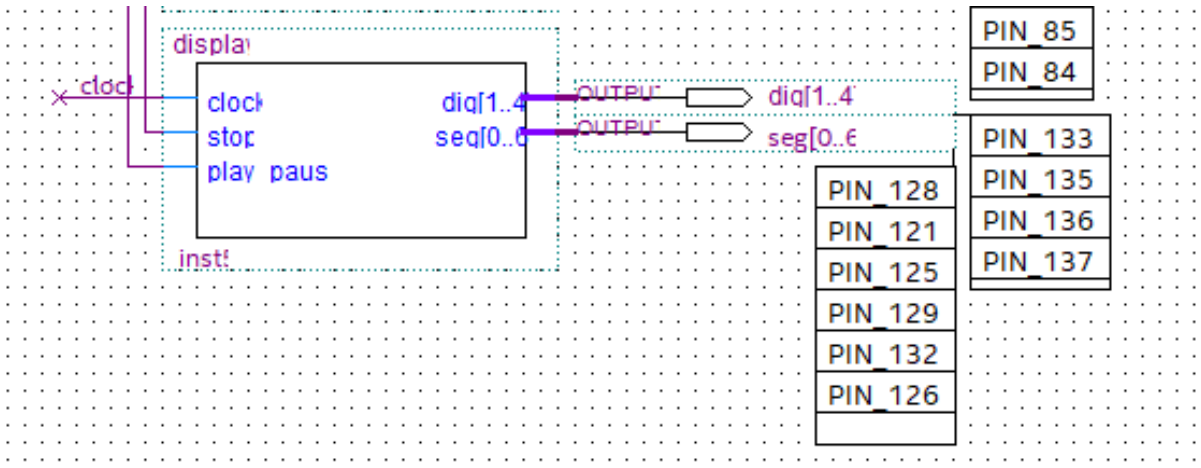

Figura 18 – Representação do display.

Uma parte do seu código pode ser visto na Figura 19. Esse display possui dois divisores de clock, um desses divisores foi utilizado para realizar a contagem do décimo de segundo ou seja, um clock de 10Hz e o outro para alternar rapidamente entre os quatro dígitos do display de sete segmentos para manté-los todos ligados simultâneamente aos olhos de quem observar a placa funcionando, para isso foi utilizada uma alta frequência.

```
48    BEGIN
49       CASE alterna IS
50          WHEN 0 =>
51             dig(1) <= '0';
52             dig(2) <= '1';
53             dig(3) <= '1';
54             dig(4) <= '1';
55             temp   <= dec_seg;
56             seg7 <= '1';
57          WHEN 1 =>
58             dig(1) <= '1';
59             dig(2) <= '0';
60             dig(3) <= '1';
61             dig(4) <= '1';
62             temp   <= uni_seg;
63             seg7 <= '0';
64          WHEN 2 =>
65             dig(1) <= '1';
66             dig(2) <= '1';
67             dig(3) <= '0';
68             dig(4) <= '1';
69             temp   <= dez_seg;
70             seg7 <= '1';
71          WHEN 3 =>
72             dig(1) <= '1';
73             dig(2) <= '1';
74             dig(3) <= '1';
75             dig(4) <= '0';
76             temp   <= uni_min;
77             seg7 <= '0';
78       END CASE;
```

Figura 19 – Uma parte do código do display.

Há um case no código informando para o display quais os segmentos deverão acender caso um certo número seja apontado pelo contador, por exemplo, para exibir o número zero, todos os segmentos exceto o último deverá acender

O valor a ser exibido no display advém de um contador instanciado no arquivo do display que funciona de forma assíncrona com quatro SIGNALS do tipo inteiro, de forma que sejam representados os dígitos para décimo de segundo, unidade e dezena de segundo e unidade de minuto.

2.5.1 Contador

No Display foi utilizado o contador, que pode ser visto, uma parte, na Figura 20. No contador, a lógica utilizada foi que, inicialmente, quando o stop é pressionado, todos os números do display serão zerados. Após isso, quando o play for pressionado, ou seja, houver um rising edge no pulso, a contagem do décimo de segundo começa. Quando o décimo de segundo chegar em 9, ele é zerado e a unidade do segundo recebe um incremento de 1. Esse ciclo continua até a unidade de segundo chegar em 9, pois quando isso acontece, a unidade de segundo é zerada e a

dezena do segundo recebe um incremento de 1. Quando a dezena do segundo chega em 5, ela zera e a unidade do minuto recebe um incremento de 1. A contagem máxima desse contador é de 9:59.9.

```
18
19      PROCESS(pulso)
20      BEGIN
21          play <= pulso and play_pause;
22          IF (stop = '1') THEN
23              dec_seg <= 0;
24              uni_seg <= 0;
25              dez_seg <= 0;
26              uni_min <= 0;
27          ELSIF(rising_edge(play)) THEN
28              dec_seg <= dec_seg + 1;
29              IF (dec_seg = 9) THEN
30                  dec_seg <= 0;
31                  uni_seg <= uni_seg + 1;
32                  IF (uni_seg = 9) THEN
33                      uni_seg <= 0;
34                      dez_seg <= dez_Seg + 1;
35                      IF (dez_seg = 5) THEN
36                          dez_seg <= 0;
37                          uni_min <= uni_min + 1;
38                          IF (uni_min = 9) THEN
39                              uni_min <= 0;
40                          END IF;
41                      END IF;
42                  END IF;
43              END IF;
```

Figura 20 – Uma parte do código do contador.

O botão de play serve de enable para esse código e o pulso é o clock de 10 Hz que é uma instância do divisor de clock e permite a contagem do décimo de segundo.

2.6 LCD

Foi disponibilizado para o grupo um arquivo LCD que já funcionava. Por conta disso, foram feitas algumas mudanças no arquivo que foi dado. Na Figura 21 tem-se a o LCD no circuito.

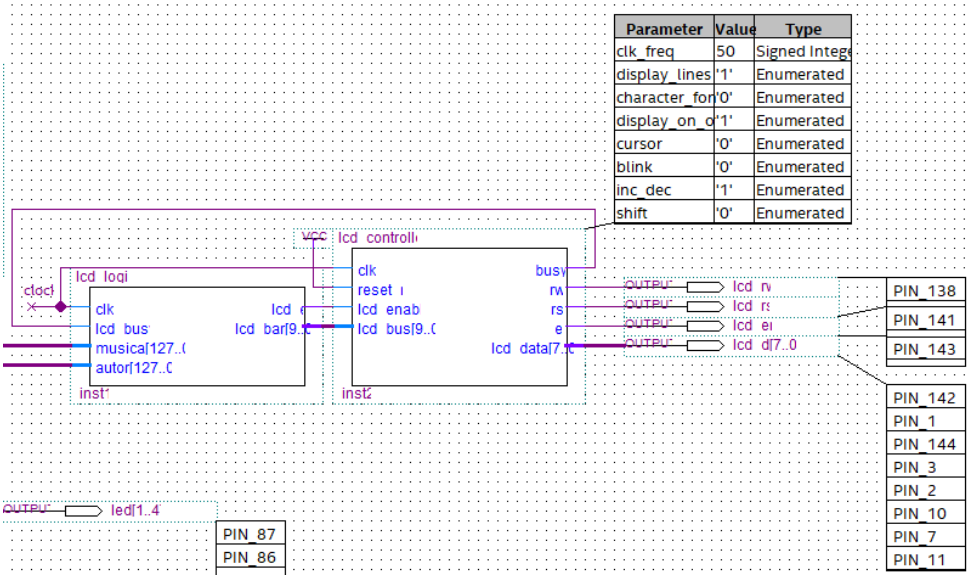

Figura 21 – Representação do LCD.

Note-se que lcd_controler não foi modificado. O que foi modificado foi lcd_logic. As principais mudanças foram na formação das linhas 1 e 2 do LCD. A linha 1 do LCD receberá o nome da música e a linha 2 do LCD receberá o nome do cantor da música.

Cada linha do LCD tem 16 caracteres. Esses caracteres recebem o que pode ser visto na Figura 22. Nele, a linha 1 e a linha 2 só vão ser enviados a um vetor de bits música e autor. A seleção dessa música e cantor foi explicada na seção de Mp3.

```
17      --barramento de dados do display
18      SIGNAL L1 : std_logic_vector (127 DOWNTO 0);
19      SIGNAL L2 : std_logic_vector (127 DOWNTO 0);
20
21      --constantes
22      CONSTANT musica1_1 : std_logic_vector (127 DOWNTO 0) := to_std_logic_vector("    Fur Elise   ");--16 caracteres!!!
23      CONSTANT musica1_2 : std_logic_vector (127 DOWNTO 0) := to_std_logic_vector("    Beethoven   ");--16 caracteres!!!
24      CONSTANT musica2_1 : std_logic_vector (127 DOWNTO 0) := to_std_logic_vector(" Over The Waves ");--16 caracteres!!!
25      CONSTANT musica2_2 : std_logic_vector (127 DOWNTO 0) := to_std_logic_vector("    P. Rosas    ");--16 caracteres!!!
26      CONSTANT musica3_1 : std_logic_vector (127 DOWNTO 0) := to_std_logic_vector("Over The Rainbow");--16 caracteres!!!
27      CONSTANT musica3_2 : std_logic_vector (127 DOWNTO 0) := to_std_logic_vector("   Yip Harburg  ");--16 caracteres!!!
28      CONSTANT musica4_1 : std_logic_vector (127 DOWNTO 0) := to_std_logic_vector("  Frere Jacque  ");--16 caracteres!!!
29      CONSTANT musica4_2 : std_logic_vector (127 DOWNTO 0) := to_std_logic_vector("  Desconhecido  ");--16 caracteres!!!
30    BEGIN
31      --atribuição continua das saidas registradas
32      lcd_e <= lcd_enable;
33      lcd_bar <= lcd_bus;
34
35      --seleção do conteúdo através do key1
36      L1 <= musica;
37      L2 <= autor;
38
39    --Sequenciamento do envio de cada caractere de L1 e L2
40    PROCESS(clk)
41      VARIABLE char  :  INTEGER RANGE 0 TO 34 := 0; --6 bits
42    BEGIN
43      IF rising_edge(clk) THEN
44        IF (lcd_busy = '0' AND lcd_enable = '0') THEN
```

Figura 22 – Caracteres nas linhas do LCD.

3 MANUAL DE OPERAÇÃO

Quando o hardware é sintetizado na FPGA, o display inicializa em 0000 com a primeira música selecionada no LCD, sem emitir som, como pode ser visto na Figura 23.

Figura 23 – Início do hardware.

Na placa tem-se 4 push-buttons, como pode ser visto na Figura 24

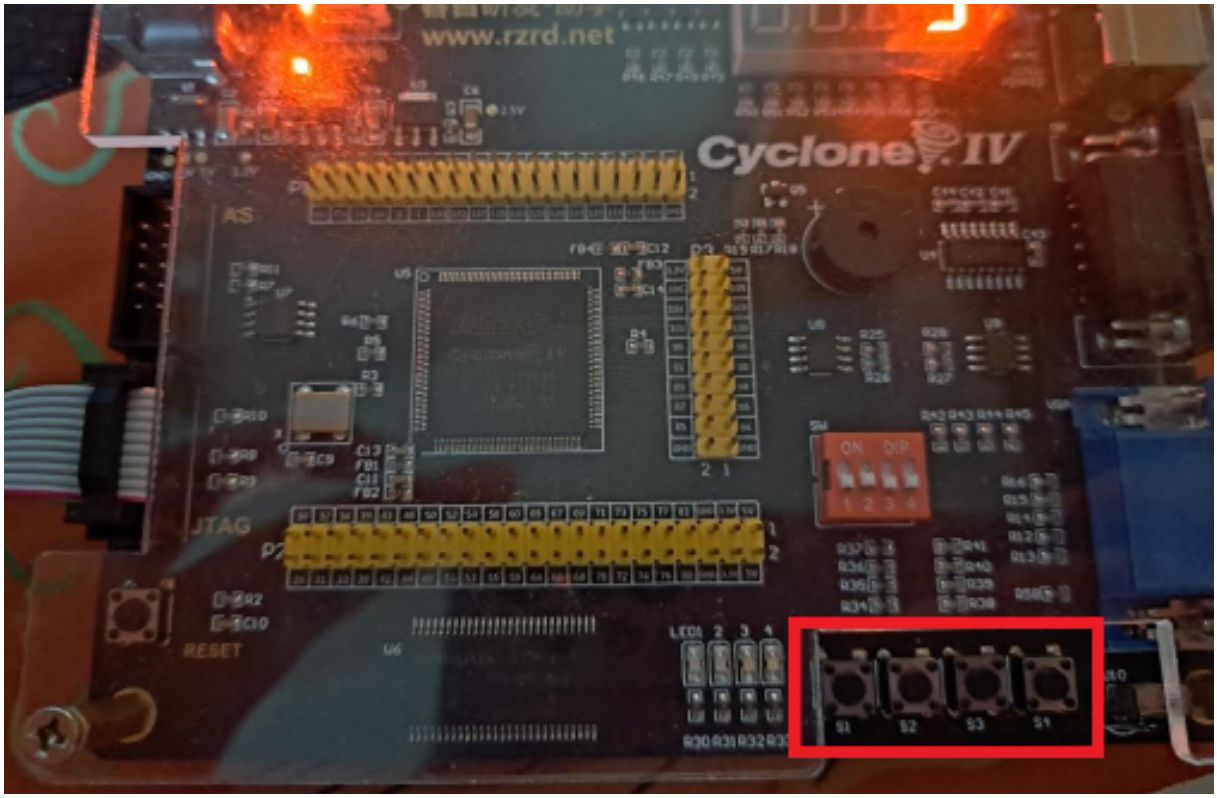

Figura 24 – Local dos 4 push-buttons.

Se pressionado o segundo push-button, mostrado na Figura 25, o display começará uma contagem, que se refere ao tempo de reprodução da música, a primeira música, Fur Elise de Beethoven começa a tocar e o segundo LED irá acender, como pode ser visto na Figura 26.

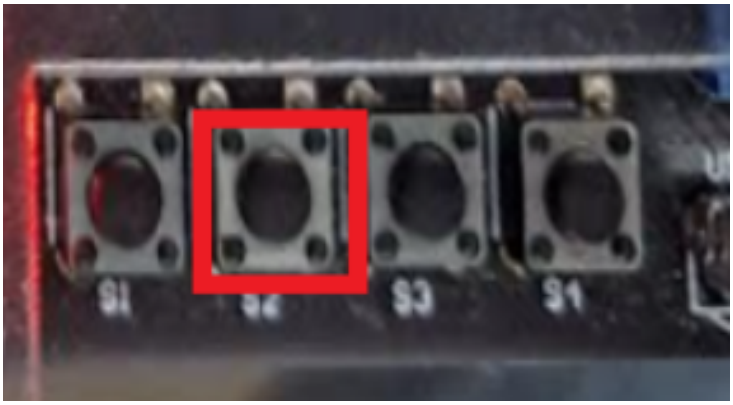

Figura 25 – Push-button 2.

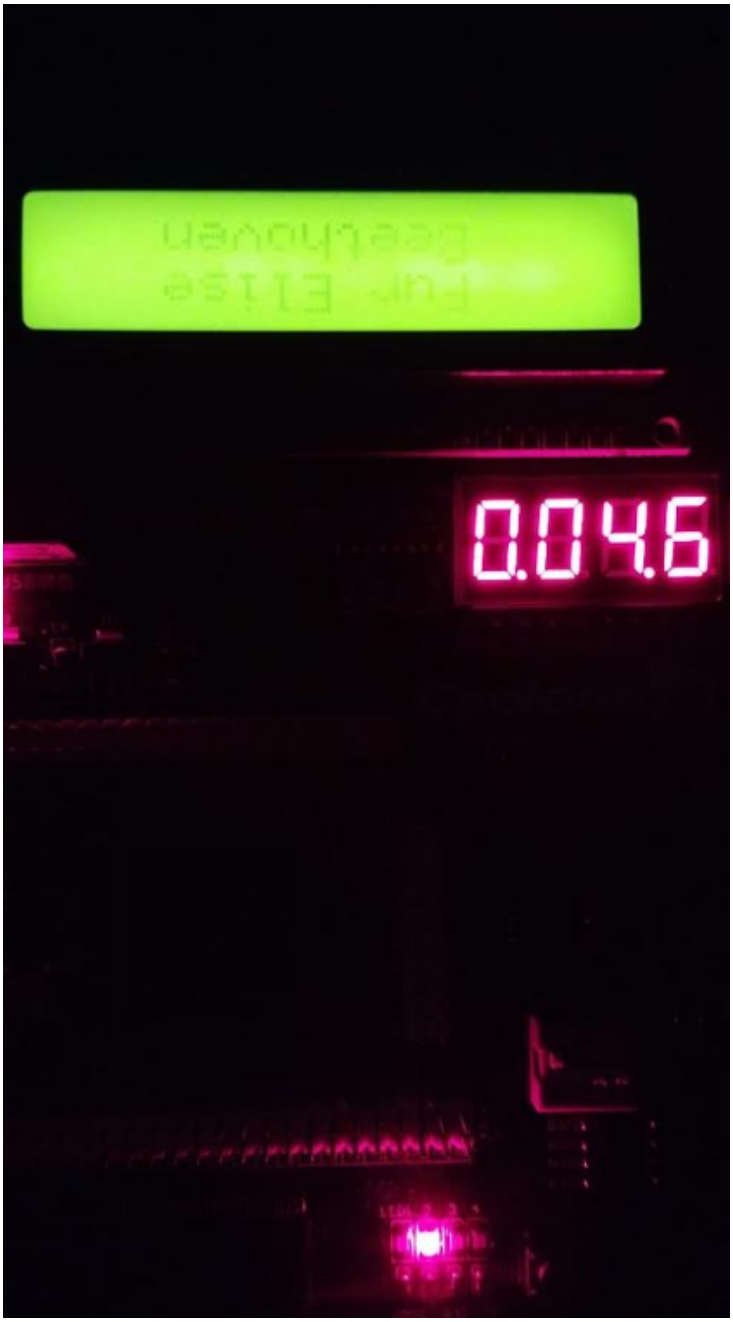

Figura 26 – Início da contagem da música Fur Elise.

Se o segundo push-button for pressionado novamente, a música entrará num pause exatamente na nota que estava tocando quando o segundo push-button foi pressionado. A contagem também entrará num pause, o segundo LED apagará e o terceiro LED acenderá, como visto na Figura 27. Para despausar a música, continuar a contagem do display e acender novamente o LED, basta pressionar o segundo push-button novamente.

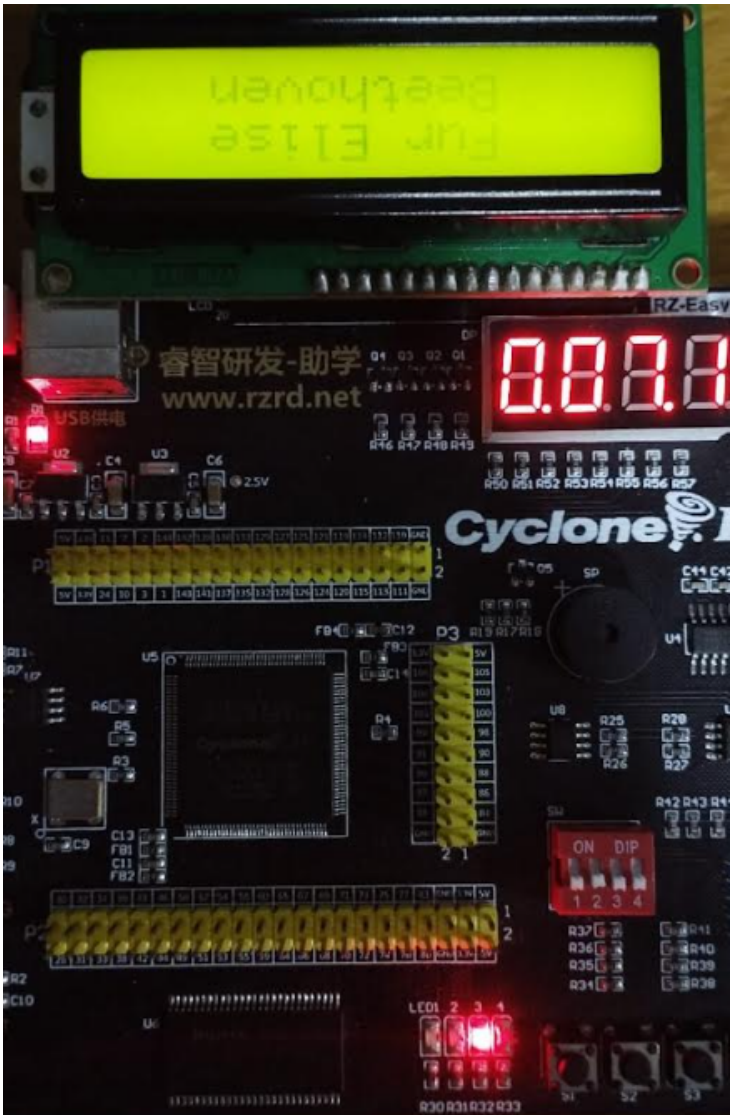

Figura 27 – Pausa na contagem da música Fur Elise.

Enquanto o primeiro push-button, mostrado na Figura 28, estiver pressionado, o buzzer da música será mutado e o primeiro LED ficará acesso, como mostrado na Figura 29. Isso significa que, se a música estiver tocando, ela não será escutada pelo usuário, mas o display continuará contando normalmente. Se o push-button for liberado, a música voltará a tocar normalmente e o LED apagará.

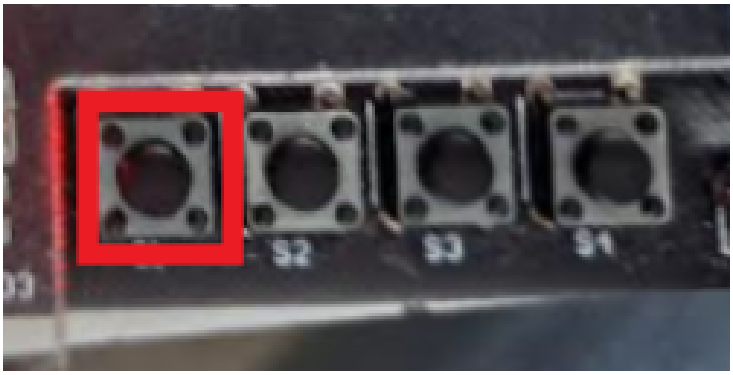

Figura 28 – Push-button 1.

Figura 29 – LED atrelado ao primeiro push-button acesso.

Se pressionado o terceiro push-button, mostrado na Figura 30, a música será trocada para a próxima e o LCD mostrará quem é essa nova música.

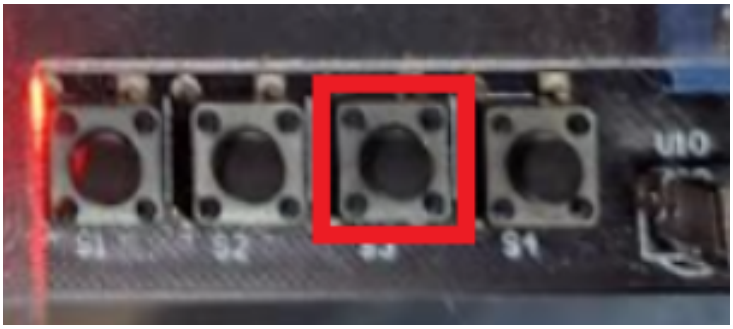

Figura 30 – Push-button 3.

Porém, a música só será trocada se, no momento em que se pressionar o push-button, a música atual estiver no stop, que será visto mais a frente. Se a música estiver no stop e o terceiro push-button for pressionado, o LCD troca o nome para a próxima música, o display volta para 0000 e a nova música é selecionada, mas não toca imediatamente, como mostrado na Figura 31. Para essa música começar a tocar, o usuário deve pressionar o segundo push-button.

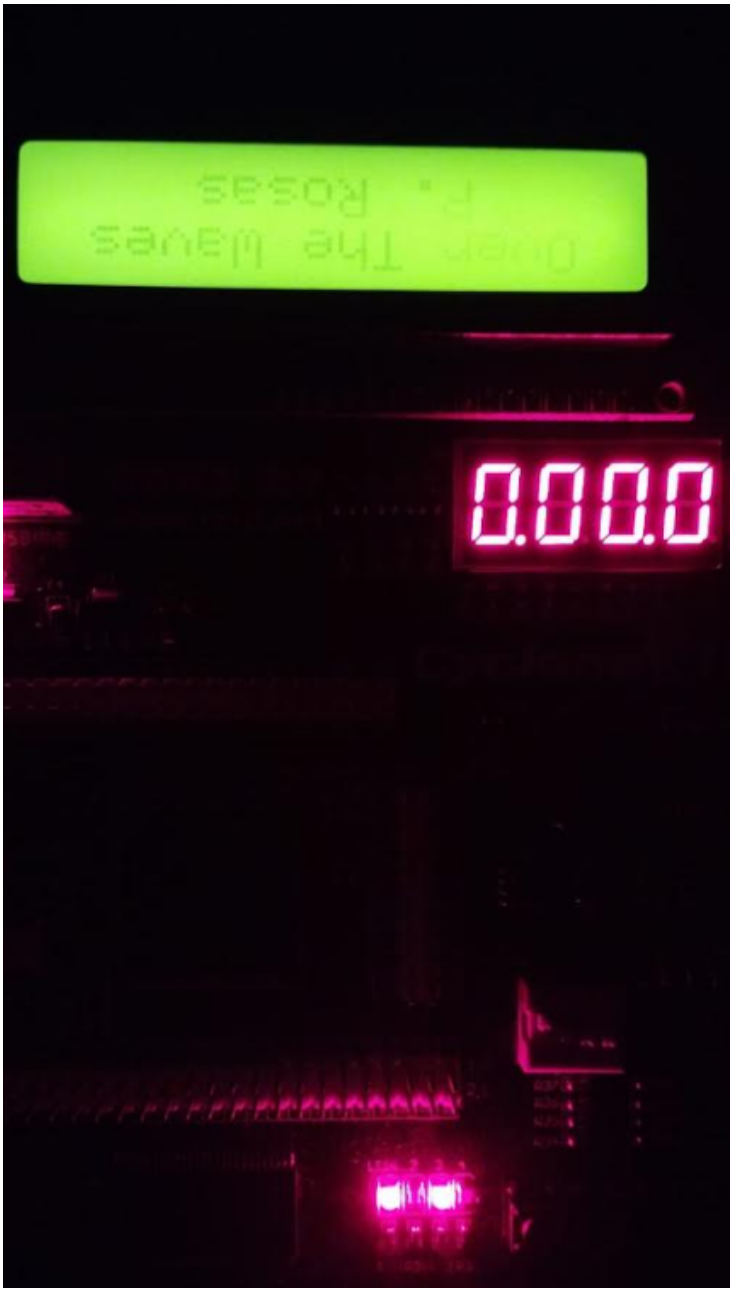

Figura 31 – Nova música selecionada.

A troca das músicas, pelo pressionamento do terceiro push-button, é um processo cíclico. Ou seja, a primeira troca é para Over The Waves, a segunda troca é para Over The Rainbow, a terceira troca é para Frere Jacque, a quarta troca é para Fur Elise e assim continua o ciclo. Todas essas trocas são vistas pelo LCD, pois ele mostra o nome da música e seu cantor, toda vez que a música é trocada.

Se o quarto push-button, mostrado na Figura 32, for pressionado, a música será pausada e reiniciada e o display volta para 0000. Para fazer a música tocar novamente, é necessário pressionar o segundo push-button.

Figura 32 – Push-button 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.

4 RESULTADOS

Os requerimentos do projeto foram finalizados. Eles são:

- Foi implementado um debouncer para os quatro push-buttons.
- O primeiro push-button habilita ou desabilita o mute.
- O segundo push-button faz tanto a música dar play, quanto faz a música dar pause.
- O terceiro push-button seleciona a próxima música a ser tocada.
- O quarto push-button faz o stop da música.
- Foi implementado, no display de sete segmentos, um contador para contar o tempo de duração das músicas.
- 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.
- Foi implementado um LCD que mostra o nome da música e o seu cantor, para cada uma das músicas implementadas.

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 VHDL, além disso, foi encontrado um desafio ao tentar importar algum arquivo VHDL dentro de outro.

5.1.2 Soluções

Para solucionar esses problemas foi feita a análise dos projetos exemplos disponibilizados no classroom, juntamente com os slides das aulas.

5.2 Controle

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

5.2.1 Desafios

Foi encontrada a necessidade de utilizar um debouncer para cada botão utilizado no circuito. Além disso, outro desafio encontrado foi a necessidade de manter a saída dos botões de play, pause e stop.

5.2.2 Soluções

Para solucionar esses problemas foi implementado o bloco de controle, que serve para aplicar o debouncer em todos os botões e para os botões de play, pause e stop funcionarem como o clock de um flip-flop tipo t. O arquivo de controle pode ser visto na subseção 2.2.

5.3 Mp3

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

5.3.1 Desafios

Encontrou-se a necessidade de um arquivo que contivesse todas as músicas, selecionasse a musica que irá ser tocada e fizesse a conexão da música atual com o divisor de clock e com o temporizador.

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 4 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, com isso, 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.

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 controlador. 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 Display

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

5.5.1 Desafios

Um dos desafios no display foi como fazer a contagem, já que uma contagem dependia da outra e a primeira contagem a se fazer era do décimo de segundo e essa era a contagem mais rápida.

5.5.2 Solução

Para solucionar esse problema foi criado uma seleção de if elses que eram sensíveis a um pulso, como pode ser visto na seção 2.5. Esse pulso iniciava a contagem do décimo de segundo e, consequentemente, das outras contagens.

5.6 LCD

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

5.6.1 Desafios

Um dos desafios da implementação do LCD foi em como fazer a música e o cantor aparecerem na linha 1 e na linha 2, respectivamente.

5.6.2 Soluções

Para solucionar esse problema, foi implementado no LCD logic somente duas entradas de dois vetores de 128 bits. A definição de quem seria essas entradas foi definida no Mp3, pois nele já havia uma lógica para seleção das músicas. Assim, foi acrescentado uma saída no Mp3 que dizia qual música e qual cantor estavam tocando e essa saída se tornou a entrada no LCD logic.

6 CONCLUSÃO

Pode-se concluir que, em geral, o aprendizado de como reproduzir sons no buzzer foi atingido. O debouncer de cada push-button foi implementado. Os quatro push-buttons, seus LEDS e suas devidas funções foram implementadas e o LCD também foi implementado.

Houve dificuldades na relação teoria versus prática em várias ocasiões. No circuito completo, a implementação do VHDL foi um problema no começo, pois ainda existiam dúvidas sobre como implementar a linguagem. Para executar alguns IF's houve dificuldades até que foi compreendido que eles precisavam estar dentro de um process. Outra dificuldade, foi na implementação do stop. Nela, foi difícil implementar o stop para reiniciar a música e só começar a tocar a música de novo se o play fosse ativado.

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.