ID: 135521

Desenvolvimento de Sistemas Computacionais
Comunicação Digital

São José dos Campos - Brasil

Julho de 2023

ID: 135521

Desenvolvimento de Sistemas Computacionais
Comunicação Digital

Relatório apresentado à Universidade Federal de São Paulo como parte dos requisitos para aprovação na disciplina de Laboratório de Sistemas Computacionais: Comunicação Digital.

Docente: Prof. Dr. Lauro Paulo da Silva Neto

Universidade Federal de São Paulo - UNIFESP

Instituto de Ciência e Tecnologia - Campus São José dos Campos

São José dos Campos - Brasil

Julho de 2023

Resumo

Este projeto tem como objetivo desenvolver uma comunicação digital entre a placa *FPGA Altera DE2-115* e o sensor de som *KY-037*, após desenvolvimento da comunicação entre os dois dispositivos a placa irá traduzir os dados recebidos em código *Morse*. Espera-se criar um sistema entre os dois dispositivos capaz de capturar e analisar sons em tempo real, no qual o sensor de som faz captura e o *FPGA* faz a análise dos dados recebidos. Para alcançar esse objetivo, será necessário entender os princípios de funcionamento do sensor de som, suas características, além de conhecer as técnicas de comunicação digital entre os dois dispositivos. Ao longo deste trabalho, será apresentado o funcionameto dos componentes, o entendimento de código morse e a interação entre os dois hardware.

Palavras-chaves: Comunicação digital, Sensor de som, *FPGA*, *Morse*.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

A tecnologia vem avançando muito nos últimos anos e vem transformando a sociedade de maneira que nem podemos imaginar, em vários setores pode-se perceber sua atuação desde a indústria até a saúde e a educação. Neste contexto, utilização de sistemas embarcados tem sido uma solução frequentemente usada para o desenvolvimento de projetos eletrônicos, permitindo assim a criação de dispositivos que utilizam-se de um software que instrui um *hardware* exercer uma atividade específica. A placa *FPGA* é um sistema embarcado capaz de realizar uma descrição de hardware e realizar o desenvolvimento de projetos de alta complexidade.

O objetivo deste projeto é utilizar a placa *FPGA Altera DE2-115* em conjunto com o sensor de som *KY-037* para desenvolver um sistema. No qual faz o monitoramento de dados sonoros, esse dado é um código *Morse* e o *KY-037* tranformará em sinais digitais que será enviado para o *FPGA* realizar tradução. Para isso, será necessário compreender o funcionamento dos dispositivos envolvidos, espera-se obter um sistema capaz de capturar e analisar sons, trazendo a informação do código Morse como resultado.

A realização deste projeto visa contribuir, de certa forma, para o desenvolvimento de monitoramento sonoro de ambientes utilizando um sistema embarcado. Além disso, pode ser utilizado como base para a implementação de outros projetos que envolvam comunicação digital e tradução de dados sonoros.

2 Objetivos

2.1 Geral

O objetivo deste trabalho é desenvolver um sistema, no qual será capaz de capturar sinais sonoros, analisa- los e traduzir essa informação que será um código *Morse*.

2.2 Específico

- Descrever sobre o surgimento do código *Morse* e como fazer a leitura.
- Descrever sobre o funcionamento do sensor de som.
- Desenvolver um sistema embarcado capaz de capturar e analisar dados sonoros em tempo real.
- Realizar a comunicação digital entre a placa *FPGA* e o sensor de som.

3 Fundamentação Teórica

3.1 Código Morse

O Código *Morse* foi desenvolvido por Samuel Morse em 1835 e é um sistema de representação de letras e algarismos através de um sinal enviado de modo intermitente.(1)

Uma mensagem em Morse pode ser transmitida em pulsos (ou tons) curtos e longos:

- Pulsos elétricos.
- Ondas mecânicas.
- Sianis Visuais.
- Ondas eletromagnéticas.

O código Morse internacional é composto de seis elementos Sinal curto(*), Sinal longo(-), Intervalo entre caracteres, Intervalo curto, Intervalo médio e Intervalo longo 1.

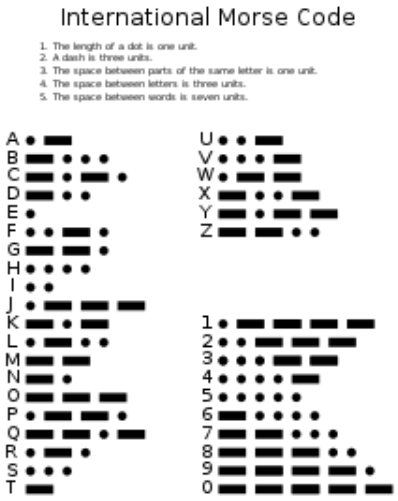
Tabela 1 – Exemplo de um Caracter

Caracter	Morse
J	* - - -

Fonte: Autor.

A partir dessas informações, será utilizado a seguinte referência 1 pré definida dos caracteres e seus respectivos código morses.

Figura 1 – Código Morse Caracteres



Fonte: Internacional Morse Code. (1)

3.2 Sensor de Som

O sensor de som é uma placa de circuito impresso com uma função específica. Ao detectar som, o módulo emite pulsos de energia que podem ser utilizados para acender ou apagar uma lâmpada, por exemplo. (2)

3.2.1 Módulo Sensor de Som *KY-037*

O sensor de som tem como função transformar a intensidade das ondas sonoras em tensões de 0 a 5V, utilizando o microfone para captar o som e identificar as vibrações das ondas no meio. Além disso, o sensor possui um Trimpot (Potenciômetro) que ajusta a sensibilidade do microfone a essas vibrações o sentido anti-horário diminui a sensibilidade e o sentido horário aumenta a sensibilidade.

O pino de leitura digital envia um sinal de nível lógico alto quando uma determinada intensidade do som é atingida, enquanto o pino analógico permite obter diferentes valores da intensidade do som. O componente possui também dois LEDs em sentidos opostos, sendo um para indicar que o módulo está energizado e outro para indicar que a saída está ativa, pois o sensor detectou som. (2)

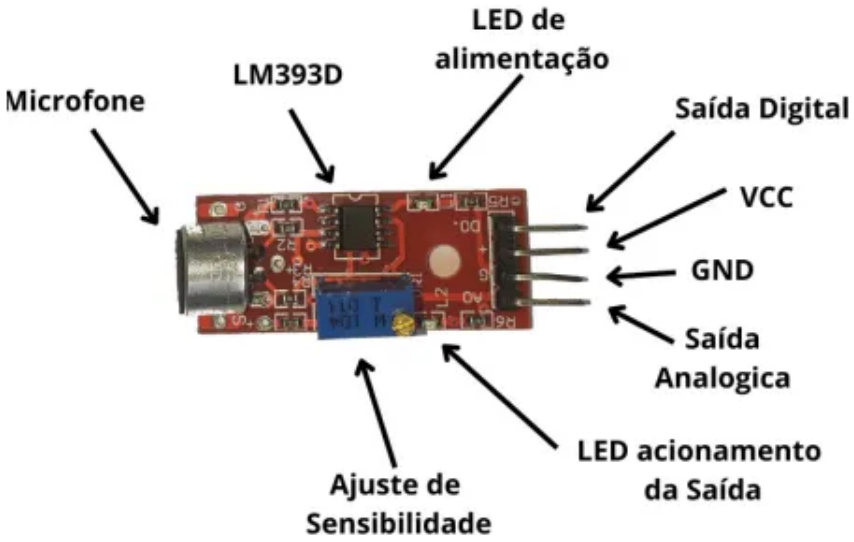
Os terminais do sensor são:

- VCC(+) – Tensão de entrada, entre 3,3 a 5 volts.
- GND(-) – O pino de O V do módulo, que é conectado ao GND do Arduino ou fonte.

- Saida Analogica(A0) – Pino de saída analogica (retorna o valor da intensidade do som captado).
- Saída Digital(D0)– Pino de saída digital (retorna HIGH ou LOW).

O Sensor *KY-037* está na imagem abaixo explicando cada um de seus componentes:

Figura 2 – Sensor *KY-037*



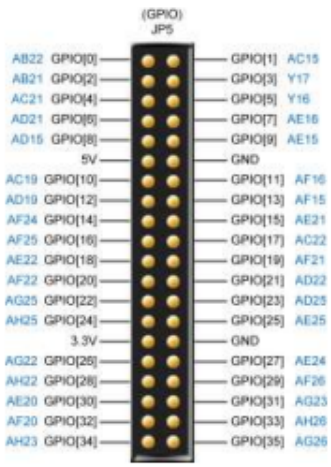
Fonte: Sensor de Som. (2)

4 Desenvolvimento

4.1 Comunicação entre o *FPGA* e o Sensor de Som

Inicialmente, é necessário realizar a conexão de três cabos nos pinos do *expansion header* (3) do *FPGA*. Um cabo deve ser conectado ao pino de *VCC* (3.3V), outro ao pino de *GND*, e o último deve ser conectado ao pino de entrada *GPIO[0]*. Este último pino será responsável por receber o sinal captado pelo sensor de som, que será utilizado para a tradução dos dados.

Figura 3 – Pinos GPIO



Fonte: Manual de2-115. (3)

4.2 Processamento do sinal digital

Devido ao fato de o sinal não se manter de forma contínua, foi necessário realizar a nivelação do sinal ao longo de todo o período em que ele se encontra no estado *HIGH*. Essa estabilidade é fundamental para uma análise correta do código Morse. Para alcançar isso, foi desenvolvido um código em *Verilog* que, ao receber um sinal *HIGH*, o mantém alto por um período adicional de tempo, garantindo assim um sinal estável e consistente. Essa abordagem no código *Verilog* permite evitar as oscilações no sinal recebido do sensor de som, garantindo uma detecção mais precisa dos elementos do código Morse.

O código 4.2 faz esse ajuste no sinal.

```
1 always @(posedge clock_debouced)
2 begin
3         if (reset == 0)
4         begin
```

```
5                   count = 0;
6                   debouced = 0;
7           end
8           else
9           begin
10                          if(sound_in == 1 && count == 0)
11                          begin
12                                  count = 1;
13                                  debouced = 1;
14                          end
15
16                          if(sound_in == 0 && count > 0)
17                          begin
18                                  count = count + 1;
19                          end
20
21                          if(sound_in == 1 && count > 0)
22                          begin
23                                  count = 1;
24                          end
25
26                          if(count > 5000)
27                          begin
28                                  count = 0;
29                                  debouced = 0;
30                          end
31
32          end
33
34  end
```

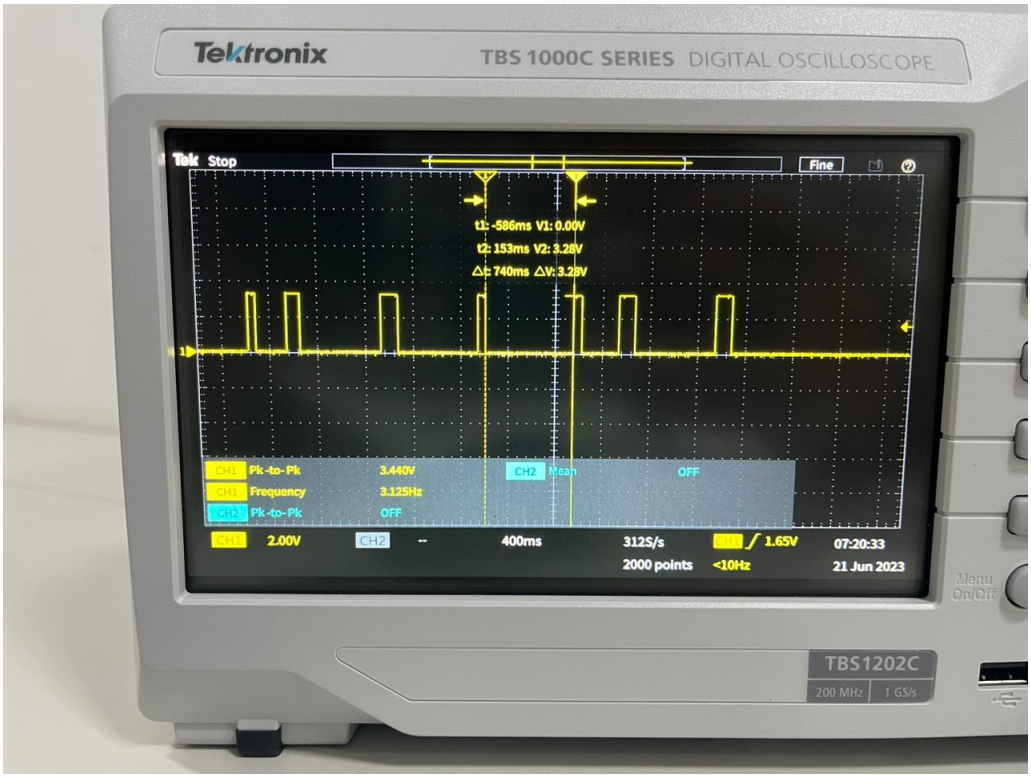
4.3 Verificação e análise do sinal

Para garantir a correta nivelação do sinal e determinar os intervalos de tempo correspondentes a pontos, traços, intervalos entre traços e pontos dentro de uma mesma letra, bem como o intervalo entre letras, foi utilizado um osciloscópio. Esse equipamento foi fundamental para realizar a análise dos períodos de duração do sinal gerado pelo aplicativo de celular que reproduzia os sons em código *Morse*.

Através do osciloscópio, foi possível visualizar e medir precisamente os intervalos de tempo, permitindo uma configuração adequada do sistema de recepção e tradução de código *Morse*, além de verificar se o nivelador de sinal tinhha funcionado corretamente.

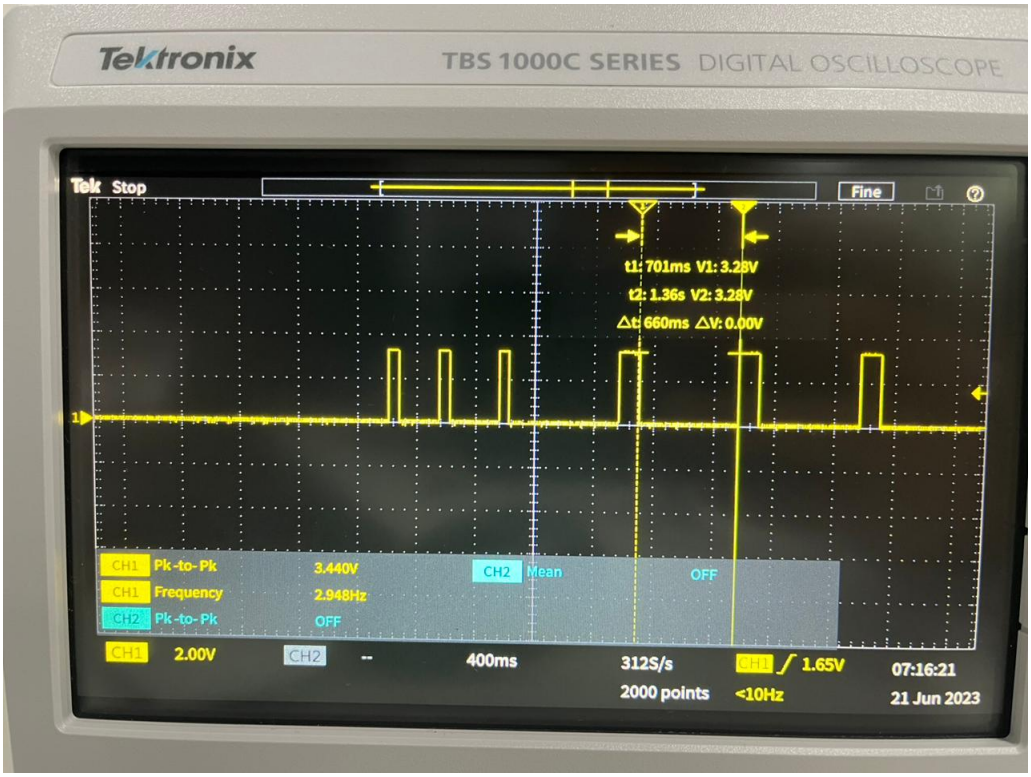
Nas imagens abaixo são os resultados obtidos através do osciloscópio.

Figura 4 – Espaço entre letras



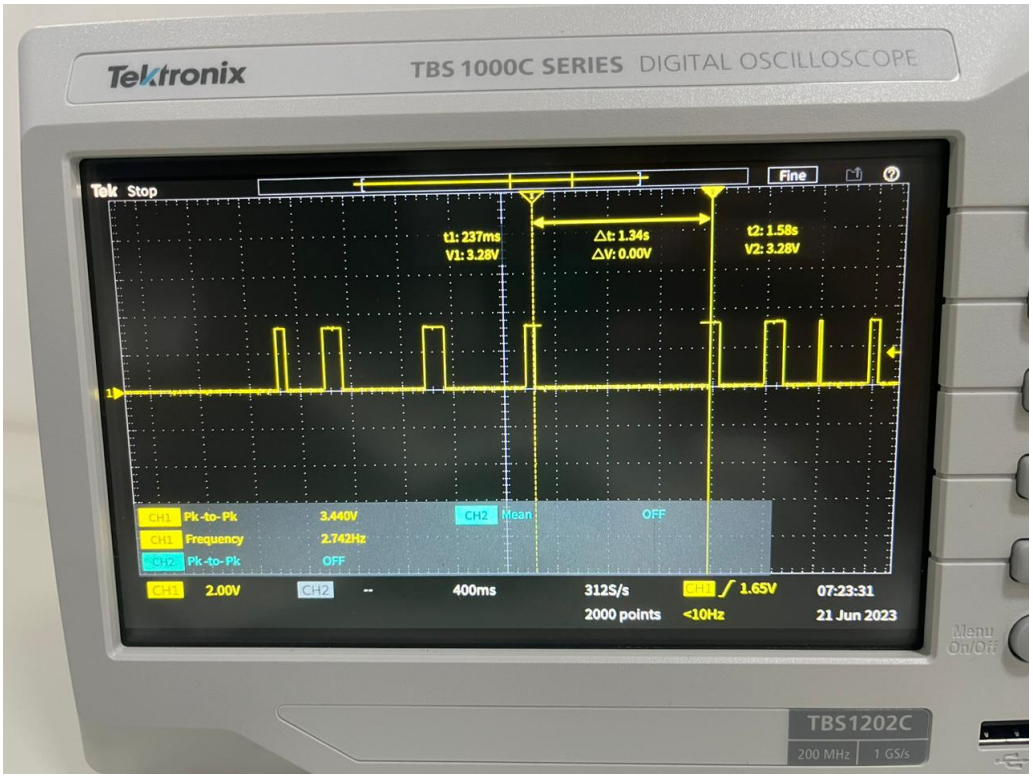
Fonte: Autor.

Figura 5 – Espaço entre ponto e traço na mesma letra



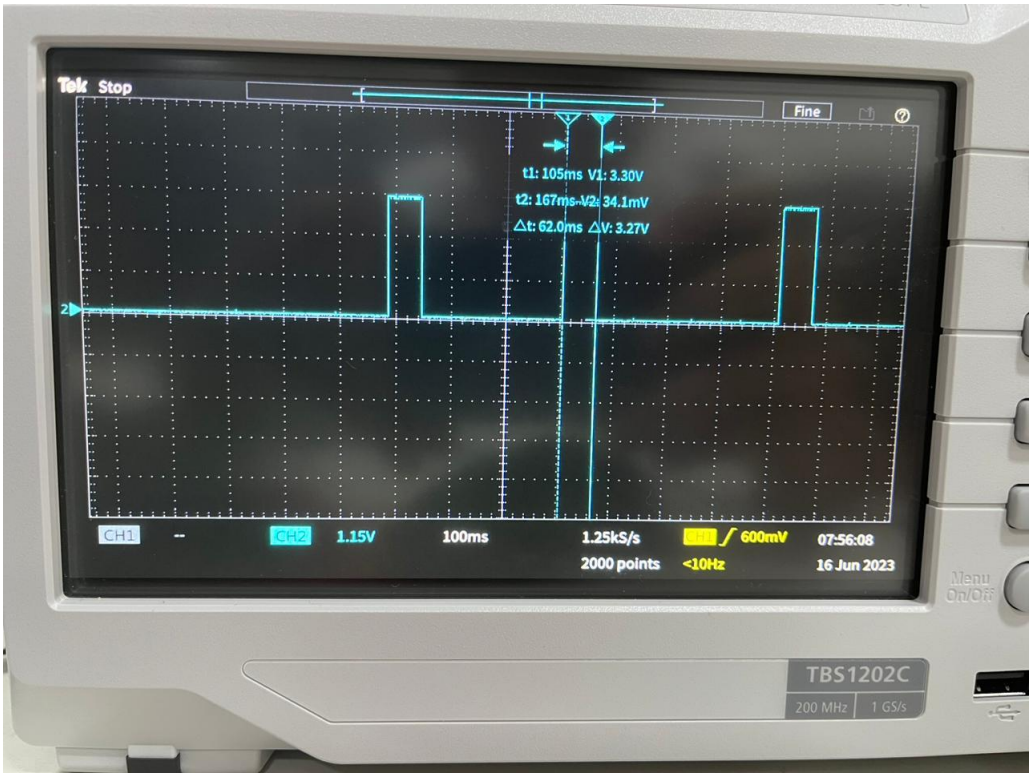
Fonte: Autor.

Figura 6 – Espaço entre palavras



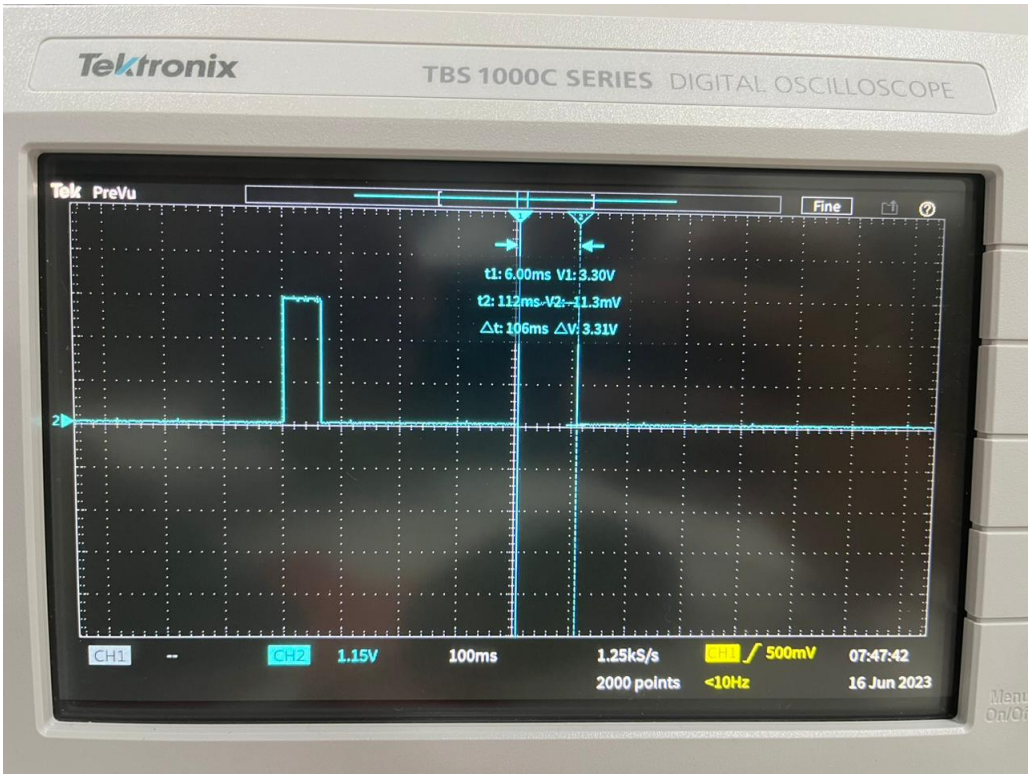
Fonte: Autor.

Figura 7 – Intervalo de um ponto



Fonte: Autor.

Figura 8 – Intervalo de um traço



Fonte: Autor.

Após a análise dos intervalos de tempo obtidos com o osciloscópio, foi desenvolvido um código para realizar a contagem desses intervalos e identificar em qual categoria eles se encaixam, como pontos, traços, intervalos entre traços e pontos dentro de uma mesma letra, bem como o intervalo entre letras.

Esse código implementado é responsável por processar os dados recebidos do sensor de som e realizar a interpretação correta do código Morse. Com base nos intervalos de tempo medidos, o código realiza a comparação com os valores de referência estabelecidos previamente para cada elemento do código Morse. O código 4.3 realiza esse processo.

```
1         parameter DEFINE_DOT_DASH = 248; // Se maior que esse valor e um traco se nao e
              um ponto
2         parameter DEFINE_SPACE_LETTER = 2553; // Se maior que esse valor e outra letra se
               nao esta na mesma letra
3         reg [15:0] count_dot_dash; // Contador para verificar se e ponto ou traco
4         reg [15:0] count_space; // Contador para verificar se esta na mesma letra ou se e
               outra letra
5         reg [15:0] morse_buffer; // Armazena um caracter em morse
6         reg [7:0] morse_character; // Armazena um caracter em ASCII para mandar para o
              LCD
7         reg [3:0] index; // index do registrador morse_buffer
8         reg aux,aux2, aux_ENB; // flags auxiliares
9
10  always @(posedge clock_morse)
11  begin
12        if (reset == 0)
```

```
13          begin
14                  morse_buffer = 0;
15                  morse_character = 0;
16                  count_dot_dash = 0;
17                  count_space = 0;
18                  index = 0;
19                  aux = 1;
20                  aux2 = 0;
21                  aux_ENB = 0;
22          end
23
24          if(debouced == 1)
25          begin
26                  count_dot_dash = count_dot_dash + 1;
27
28                  if(count_dot_dash < DEFINE_DOT_DASH )
29                  begin
30                          morse_buffer[index] = 1;
31                  end
32
33                  else
34                  begin
35                          morse_buffer[index + 1] = 1;
36                  end
37                  aux = 0;
38                  aux2 = 1;
39                  count_space=0;
40
41                  aux_ENB = 0;
42
43          end
44          else
45          begin
46                  if(aux == 0)
47                  begin
48                          if(count_dot_dash >= DEFINE_DOT_DASH)
49                                  index = index + 3;
50                          else
51                                  index = index + 2;
52
53                          count_dot_dash = 0;
54
55                          aux = 1;
56
57                  end
58
59                  if(aux2 == 1)
60                  begin
61                          count_space = count_space + 1;
62                  end
63
64                  if(count_space > DEFINE_SPACE_LETTER) // Registra no morse_character qual
                         caracter foi identificado
65                  begin
66                          morse_character = 8'd0;
67                          case (morse_buffer)
68
69                                  16'b0000000000001101: morse_character = 8'd97; //a
70                                  16'b0000000010101011: morse_character = 8'd98; //b
71                                  16'b0000000101101011: morse_character = 8'd99; //c
```

```
72                                  16'b0000000000101011: morse_character = 8'd100; //d
73                                  16'b0000000000000001: morse_character = 8'd101; //e
74                                  16'b0000000010110101: morse_character = 8'd102; //f
75                                  16'b0000000001011011: morse_character = 8'd103; //g
76                                  16'b0000000001010101: morse_character = 8'd104; //h
77                                  16'b0000000000000101: morse_character = 8'd105; //i
78                                  16'b0000001101101101: morse_character = 8'd106; //j
79                                  16'b0000000001101011: morse_character = 8'd107; //k
80                                  16'b0000000010101101: morse_character = 8'd108; //l
81                                  16'b0000000000011011: morse_character = 8'd109; //m
82                                  16'b0000000000001011: morse_character = 8'd110; //n
83                                  16'b0000000011011011: morse_character = 8'd111; //o
84                                  16'b0000000101101101: morse_character = 8'd112; //p
85                                  16'b0000001101011011: morse_character = 8'd113; //q
86                                  16'b0000000000101101: morse_character = 8'd114; //r
87                                  16'b0000000000010101: morse_character = 8'd115; //s
88                                  16'b0000000000000011: morse_character = 8'd116; //t
89                                  16'b0000000000110101: morse_character = 8'd117; //u
90                                  16'b0000000011010101: morse_character = 8'd118; //v
91                                  16'b0000000001101101: morse_character = 8'd119; //w
92                                  16'b0000000110101011: morse_character = 8'd120; //x
93                                  16'b0000001101101011: morse_character = 8'd121; //y
94                                  16'b0000000101011011: morse_character = 8'd122; //z
95
96                          endcase
97                          aux_ENB = 1;
98                          morse_buffer = 16'b0000000000000000;
99                          index = 0;
100                 end
101
102         end
103
104 end
```

5 Considerações Finais

O desenvolvimento do projeto de comunicação digital entre a placa *FPGA Altera DE2-115* e o sensor de som *KY-037*, com tradução dos dados em código *Morse*, proporcionou uma experiência enriquecedora e os resultados obtidos mostraram a viabilidade e eficiência dessa integração. Através do uso de técnicas de nivelação do sinal e a definição dos intervalos de tempo característicos do código *Morse*, foi possível estabelecer um sistema capaz de fazer a tradução dos caracteres em código *Morse*.

A utilização do osciloscópio como ferramenta de análise foi fundamental para garantir a correta identificação dos intervalos de tempo e a calibração adequada do sistema. Além disso, o desenvolvimento do código no sistema embarcado permitiu a contagem e classificação dos intervalos, resultando em uma interpretação do código *Morse* recebido.

No entanto, é importante ressaltar que o trabalho apresenta algumas limitações, como a dependência da qualidade do sinal sonoro captado pelo sensor de som e a necessidade de um ambiente sem interferências externas para uma tradução precisa. Esses pontos podem ser aprimorados em trabalhos futuros, buscando soluções para melhorar a robustez e confiabilidade do sistema.

Referências

1 WIKIPEDIA. *Código Morse*. 2023. <https://pt.wikipedia.org/wiki/Codigo_Morse>. Acessado em 19 de abril de 2023. Citado 2 vezes nas páginas 11 e 12.

2 VIDADES portal. *Sensor de Som*. 2023. <https://portal.vidadesilicio.com.br/sensor-de-som-acendendo-um-led-arduino/>. Acessado em 19 de abril de 2023. Citado 2 vezes nas páginas 12 e 13.

3 ALTERA. *Manual FPGA*. 2023. <https://www.terasic.com.tw/attachment/archive/502/DE2_115_User_manual.pdf>. Acessado em 9 de julho de 2023. Citado na página 15.

Apêndices

APÊNDICE A – Código Inicial

```
module sound_sensor(
    input wire sound_in,
    output reg sound_detected
);

always @(posedge sound_in) begin
    sound_detected <= 1;
end

endmodule
```

APÊNDICE B – Código Tradutor de Morse

```
1
2  module morseTranslator(
3          input clk,
4          input   sound_in,
5          input reset,
6          output wire [15:0] sound_detected,
7          output wire lcd_rs,
8          output wire lcd_rw,
9          output wire lcd_e,
10         output wire lcd_on,
11         output wire [7:0]lcd_db
12 );
13         wire clock_debouced;
14         wire clock_morse;
15         wire clock_lcd;
16         DivisorDeFrequencia #(.DIV_VALUE(50)) divide_lcd(.clk(clk),.divided_clk(clock_lcd
               ));
17         DivisorDeFrequencia #(.DIV_VALUE(100)) divide(.clk(clk),.divided_clk(
               clock_debouced));
18         DivisorDeFrequencia #(.DIV_VALUE(7000)) counter(.clk(clk),.divided_clk(
               clock_morse));
19
20         reg [15:0] count;
21         reg debouced;
22
23 always @(posedge clock_debouced)
24 begin
25         if (reset == 0)
26         begin
27                 count = 0;
28                 debouced = 0;
29         end
30         else
31         begin
32                         if(sound_in == 1 && count == 0)
33                         begin
34                                 count = 1;
35                                 debouced = 1;
36                         end
37
38                         if(sound_in == 0 && count > 0)
39                         begin
40                                 count = count + 1;
41                         end
42
43                         if(sound_in == 1 && count > 0)
44                         begin
45                                 count = 1;
46                         end
47
48                         if(count > 5000)
49                         begin
50                                 count = 0;
51                                 debouced = 0;
```

```
52                          end
53
54          end
55
56  end
57          parameter DEFINE_DOT_DASH = 248;
58          parameter DEFINE_SPACE_LETTER = 2553;
59          reg [15:0] count_dot_dash;
60          reg [15:0] count_space;
61          reg [15:0] morse_buffer;
62          reg [7:0] morse_character;
63          reg [3:0] index;
64          reg aux,aux2, aux_ENB;
65
66  always @(posedge clock_morse)
67  begin
68          if (reset == 0)
69          begin
70                  morse_buffer = 0;
71                  morse_character = 0;
72                  count_dot_dash = 0;
73                  count_space = 0;
74                  index = 0;
75                  aux = 1;
76                  aux2 = 0;
77                  aux_ENB = 0;
78          end
79
80          if(debouced == 1)
81          begin
82                  count_dot_dash = count_dot_dash + 1;
83
84                  if(count_dot_dash < DEFINE_DOT_DASH )
85                  begin
86                          morse_buffer[index] = 1;
87                  end
88
89                  else
90                  begin
91                          morse_buffer[index + 1] = 1;
92                  end
93                  aux = 0;
94                  aux2 = 1;
95                  count_space=0;
96
97                  aux_ENB = 0;
98
99          end
100         else
101         begin
102                 if(aux == 0)
103                 begin
104                         if(count_dot_dash >= DEFINE_DOT_DASH)
105                                 index = index + 3;
106                         else
107                                 index = index + 2;
108
109                         count_dot_dash = 0;
110
111                         aux = 1;
```

```
112
113                 end
114
115                 if(aux2 == 1)
116                 begin
117                         count_space = count_space + 1;
118                 end
119
120                 if(count_space > DEFINE_SPACE_LETTER)
121                 begin
122                         morse_character = 8'd0;
123                         case (morse_buffer)
124
125                                 16'b0000000000001101: morse_character = 8'd97; //a
126                                 16'b0000000010101011: morse_character = 8'd98; //b
127                                 16'b0000000101101011: morse_character = 8'd99; //c
128                                 16'b0000000000101011: morse_character = 8'd100; //d
129                                 16'b0000000000000001: morse_character = 8'd101; //e
130                                 16'b0000000010110101: morse_character = 8'd102; //f
131                                 16'b0000000001011011: morse_character = 8'd103; //g
132                                 16'b0000000001010101: morse_character = 8'd104; //h
133                                 16'b0000000000000101: morse_character = 8'd105; //i
134                                 16'b0000001101101101: morse_character = 8'd106; //j
135                                 16'b0000000001101011: morse_character = 8'd107; //k
136                                 16'b0000000010101101: morse_character = 8'd108; //l
137                                 16'b0000000000011011: morse_character = 8'd109; //m
138                                 16'b0000000000001011: morse_character = 8'd110; //n
139                                 16'b0000000011011011: morse_character = 8'd111; //o
140                                 16'b0000000101101101: morse_character = 8'd112; //p
141                                 16'b0000001101011011: morse_character = 8'd113; //q
142                                 16'b0000000000101101: morse_character = 8'd114; //r
143                                 16'b0000000000010101: morse_character = 8'd115; //s
144                                 16'b0000000000000011: morse_character = 8'd116; //t
145                                 16'b0000000000110101: morse_character = 8'd117; //u
146                                 16'b0000000011010101: morse_character = 8'd118; //v
147                                 16'b0000000001101101: morse_character = 8'd119; //w
148                                 16'b0000000110101011: morse_character = 8'd120; //x
149                                 16'b0000001101101011: morse_character = 8'd121; //y
150                                 16'b0000000101011011: morse_character = 8'd122; //z
151
152                         endcase
153                         aux_ENB = 1;
154                         morse_buffer = 16'b0000000000000000;
155                         index = 0;
156                 end
157
158         end
159
160 end
161
162 lcd2(.CLK(clock_lcd), .LCD_RS(lcd_rs), .LCD_RW(lcd_rw), .LCD_E(lcd_e), .LCD_DB(lcd_db), .
        DATA(morse_character), .OPER(1), .ENB(aux_ENB), .RST(~reset));
163
164 assign lcd_on = 1;
165 assign sound_detected = morse_buffer;
166
167 endmodule
```