

## **PBLE02**

Board bring-up e validação de protótipos eletrônicos

## **Manual**

Grupo 1A

**Itajubá, 2024**

## Identificação dos membros

Numeração	Nome	Matrícula	Responsabilidade
01	Fabio Henrique Poncio Alves	2022006701	Documentação
02	Gabriel Del Monte Schiavi Noda	2022014552	Software
03	Marcos Cláudio Donato Silva	2021025471	Hardware

Tabela 1: Integrantes do grupo

# Sumário

<b>1</b>	<b>Introdução</b>	<b>8</b>
<b>2</b>	<b>Requisitos de projeto</b>	<b>9</b>
<b>3</b>	<b>Ambiente de desenvolvimento</b>	<b>10</b>
<b>4</b>	<b>Esquema elétrico</b>	<b>11</b>
4.1	Subcircuito de alimentação . . . . .	11
4.2	Subcircuito de interação com o usuário . . . . .	12
4.3	Subcircuito de comunicação serial . . . . .	14
4.4	Subcircuito de operação . . . . .	15
4.5	Subcircuitos dos periféricos . . . . .	16
4.5.1	Entrada diferencial . . . . .	16
4.5.2	Conversor digital-analógico . . . . .	17
4.5.3	Relógio de tempo real . . . . .	19
4.6	Placa impressa . . . . .	21
4.7	Relatório de verificação de erros do projeto elétrico . . . . .	22
<b>5</b>	<b>Placa de circuito impresso</b>	<b>23</b>
5.1	Desenho da PCI . . . . .	24
5.2	Relatório de verificação de erros do projeto . . . . .	25
<b>6</b>	<b>Características gerais</b>	<b>27</b>
6.1	Mapa de pinos . . . . .	27
6.2	Alimentação e consumo . . . . .	28
<b>7</b>	<b>Códigos-fonte</b>	<b>30</b>
7.1	ADC . . . . .	30
7.2	button . . . . .	30
7.3	config . . . . .	31
7.4	DAC_MCP4725 . . . . .	31
7.5	defines . . . . .	33
7.6	delay . . . . .	33
7.7	event . . . . .	34
7.8	I2C . . . . .	36
7.9	LCD . . . . .	38
7.10	LED . . . . .	40
7.11	output . . . . .	41
7.12	PWM . . . . .	43
7.13	RTC_DS1307 . . . . .	44
7.14	serial . . . . .	46
7.15	stateMachine . . . . .	47
7.16	variables . . . . .	49
7.17	main . . . . .	52

<b>8</b>	<b>Diagramas propostos</b>	<b>53</b>
8.1	Diagrama de classes . . . . .	53
8.2	Diagrama de estados . . . . .	54

# **Lista de Figuras**

1	Folha de topo do projeto . . . . .	11
2	Subcírculo de alimentação no Kicad . . . . .	11
3	Subcírculo de alimentação na placa . . . . .	12
4	Subcírculo de interação com o usuário: teclado no Kicad . . . . .	12
5	Subcírculo de interação com o usuário: LCD no Kicad . . . . .	13
6	Subcírculo de interação com o usuário: LEDs no Kicad . . . . .	13
7	Subcírculo de interação com o usuário: LCD na placa . . . . .	14
8	Subcírculo de interação com o usuário: botões e LEDs na placa . . . . .	14
9	Subcírculo de comunicação serial no Kicad . . . . .	15
10	Subcírculo de comunicação serial na placa . . . . .	15
11	Subcírculo de operação no Kicad . . . . .	16
12	Subcírculo de operação na placa . . . . .	16
13	Subcírculo da entrada diferencial no Kicad . . . . .	17
14	Subcírculo da entrada diferencial na placa . . . . .	17
15	Subcírculo do conversor digital-analógico no Kicad . . . . .	18
16	Subcírculo do conversor digital-analógico na placa . . . . .	19
17	Subcírculo do relógio de tempo real no Kicad . . . . .	20
18	Subcírculo do relógio de tempo real na placa . . . . .	20
19	Vista inferior da placa impressa . . . . .	21
20	Vista superior da placa impressa . . . . .	21
21	Relatório de verificação de erros do projeto elétrico . . . . .	22
22	Imagen frontal da PCI . . . . .	24
23	Imagen inferior da PCI . . . . .	25
24	Imagen das trilhas da PCI . . . . .	25
25	Relatório de verificação de erros do projeto . . . . .	26
26	Header file para o funcionamento do ADC . . . . .	30
27	Source file para o funcionamento do ADC . . . . .	30
28	Header file para o funcionamento dos botões . . . . .	30
29	Source file para o funcionamento dos botões . . . . .	31
30	Arquivo de configuração para o microcontrolador . . . . .	31
31	Header file para o funcionamento do DAC . . . . .	31
32	Source file para o funcionamento do DAC . . . . .	32
33	Arquivo de defines para o microcontrolador . . . . .	33
34	Header file para o funcionamento de delays . . . . .	33
35	Source file para o funcionamento de delays . . . . .	34
36	Header file para o funcionamento dos eventos . . . . .	34
37	Source file para o funcionamento dos eventos . . . . .	35
38	Header file para o funcionamento da comunicação I2C . . . . .	36
39	Source file para o funcionamento da comunicação I2C (parte 1) . . . . .	37
40	Source file para o funcionamento da comunicação I2C (parte 2) . . . . .	38
41	Header file para o funcionamento do LCD . . . . .	38
42	Source file para o funcionamento do LCD . . . . .	39
43	Header file para o funcionamento dos LEDs . . . . .	40
44	Source file para o funcionamento dos LEDs . . . . .	40
45	Header file para o funcionamento da saída do microcontrolador . . . . .	41
46	Source file para o funcionamento da saída do microcontrolador (parte 1) . . . . .	41

47	Source file para o funcionamento da saída do microcontrolador (parte 2) . . . . .	42
48	Source file para o funcionamento da saída do microcontrolador (parte 3) . . . . .	43
49	Header file para o funcionamento do PWM . . . . .	43
50	Source file para o funcionamento do PWM . . . . .	44
51	Header file para o funcionamento do RTC . . . . .	44
52	Source file para o funcionamento do RTC . . . . .	45
53	Header file para o funcionamento da comunicação serial . . . . .	46
54	Source file para o funcionamento da comunicação serial . . . . .	46
55	Header file para o funcionamento da máquina de estados . . . . .	47
56	Source file para o funcionamento da máquina de estados (parte 1) . . . . .	47
57	Source file para o funcionamento da máquina de estados (parte 2) . . . . .	48
58	Header file para o funcionamento das variáveis utilizadas . . . . .	49
59	Source file para o funcionamento das variáveis utilizadas (parte 1) . . . . .	50
60	Source file para o funcionamento das variáveis utilizadas (parte 2) . . . . .	50
61	Source file para o funcionamento das variáveis utilizadas (parte 3) . . . . .	51
62	Source file para o funcionamento das variáveis utilizadas (parte 4) . . . . .	51
63	Source file para a aplicação principal do projeto . . . . .	52
64	Diagrama de classes proposto para a aplicação . . . . .	53
65	Diagrama de estados proposto para a aplicação . . . . .	54

## **Lista de Tabelas**

1	Integrantes do grupo . . . . .	2
2	Requisitos de projeto . . . . .	9
3	Ambiente de desenvolvimento . . . . .	10
4	Requisitos para a PCI . . . . .	23
5	Lista de componentes . . . . .	24
6	Conexões do microcontrolador . . . . .	27
7	Conexões do conversor USB-Serial . . . . .	27
8	Conexões do relógio de tempo real . . . . .	27
9	Conexões do conversor D/A . . . . .	28
10	Conexões da entrada analógica . . . . .	28
11	Conexões dos pinos de expansão . . . . .	28
12	Conexões do LCD . . . . .	28

# **1 Introdução**

Este manual tem como objetivo descrever e direcionar etapas e ferramentas necessárias para o uso e reprodução do projeto de uma placa de circuito impresso desenvolvida anteriormente na disciplina *PBLE01* ministrada pelo professor Rodrigo de Paula Rodrigues. Nesta versão, atualizada durante a disciplina *PBLE02* ministrada pelo professor Rodrigo Maximiano Antunes de Almeida, serão encontradas melhorias e correções da primeira versão da placa produzida.

## 2 Requisitos de projeto

Nesta seção serão apresentados os requisitos técnicos referentes ao esquema elétrico.

Requisito	Descrição
R2 - Alimentação	1 - Tensão de entrada na faixa de 7 a 12 [V] DC; 2 - Conector de alimentação do tipo <i>jack J4</i> ; 3 - Proteção contra tensão reversa; 4 - Regulador linear com saída de 5 [V]; Diodo emissor de luz para presença de alimentação.
R3 - Operação	1 - Microcontrolador <i>PIC184550</i> ; 2 - Barra de pinos de gravação ICSP; 3 - Chave táctil de reinício.
R4 - Interação com o usuário	1 - Teclado numérico de cinco teclas; 2 - Visor de 16x2 com luminosidade controlada por trimmer; 3 - Quatro diodos emissores de luz.
R5 - Periféricos e expansão	1 - Relógio de tempo real; 2 - Conversor digital para analógico com conector de 2 terminais para saída da placa; 3 - Barra de expansão com dois sinais analógicos; 4 - Barra de expansão para os sinais I2C, sinal de referência e de alimentação; 5 - Barra de expansão de sinais para pinos não utilizados, de modo a contemplar os sinais de 0 [V], 5 [V] e pinos de comunicação I2C.
R6 - Comunicação	1 - Conversor USB-serial.

Tabela 2: Requisitos de projeto

### 3 Ambiente de desenvolvimento

Nesta seção serão apresentados os softwares utilizados durante o desenvolvimento do projeto.

Numeração	Software	Versão
01	Kicad	7.0
02	MPLabX IDE	6.20
03	XC8	2.46

Tabela 3: Ambiente de desenvolvimento

## 4 Esquema elétrico

Nesta seção, será apresentado todo o esquema elétrico da placa confeccionada. Considerando que não foi necessário realizar nenhum alteração nas conexões, os dados contidos neste manual e na primeira versão realizada serão exatamente iguais.

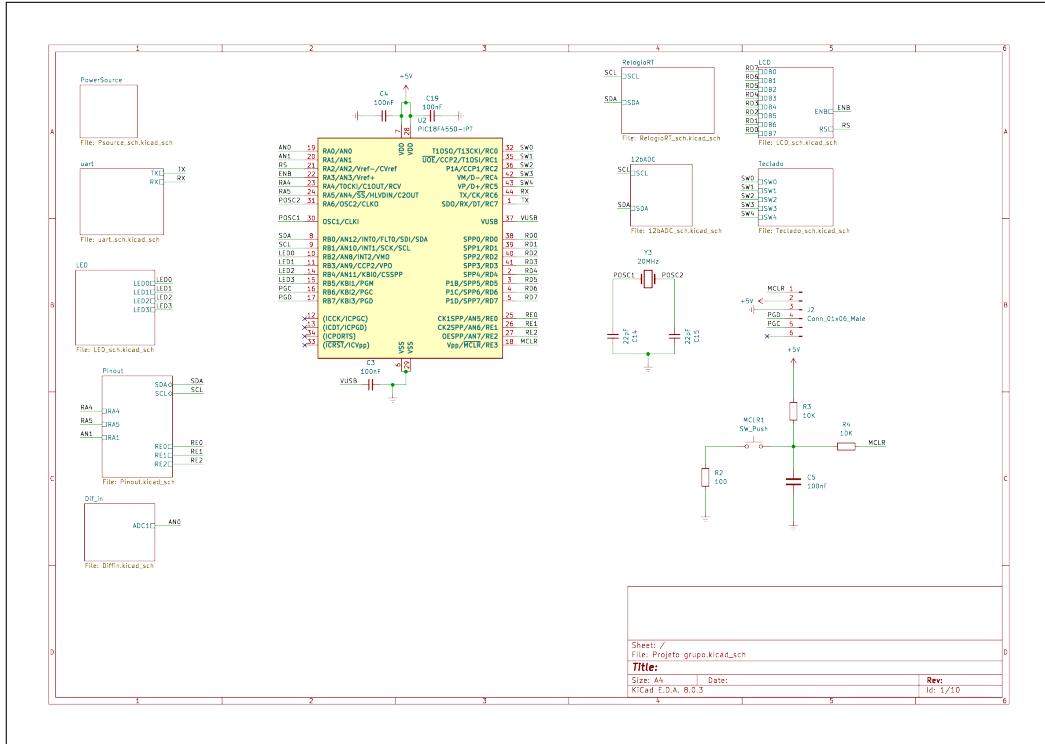


Figura 1: Folha de topo do projeto

#### 4.1 Subcircuito de alimentação

O subcircuito representado nesta subseção regula a tensão de entrada do circuito em 5 [V], de modo a possuir um diodo de prevenção de corrente reversa e um LED para a indicação de presença de alimentação. Vale ressaltar que os valores de capacitores e suas conexões são especificados pelo fabricante.

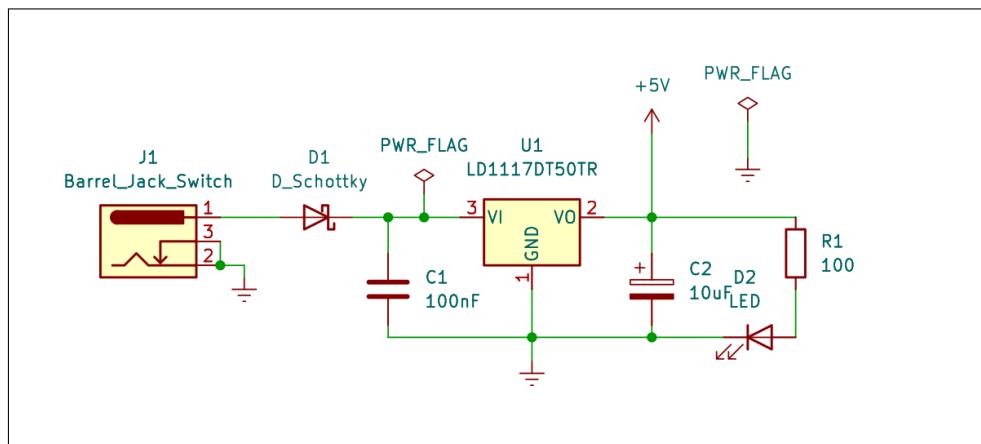


Figura 2: Subcircuito de alimentação no Kicad

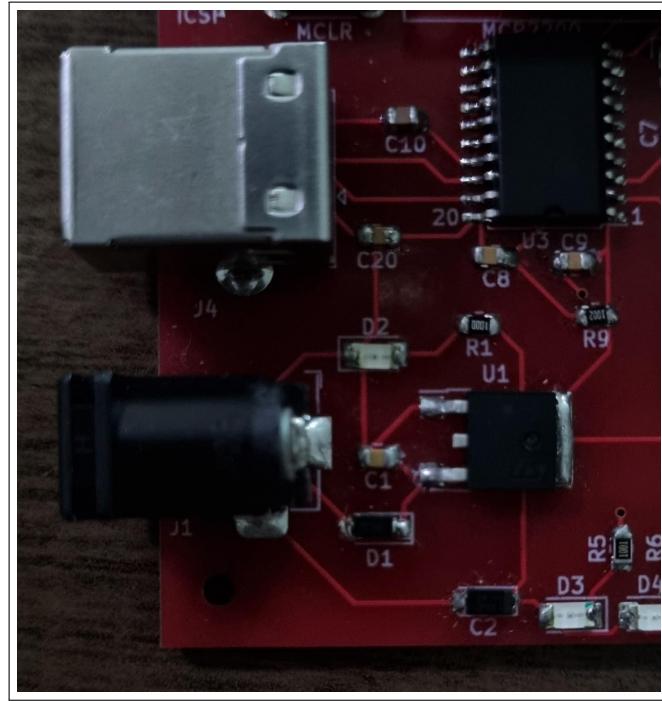


Figura 3: Subcircuito de alimentação na placa

## 4.2 Subcircuito de interação com o usuário

Nesta subseção, serão representados os circuitos relacionados ao teclado, LCD e LEDs utilizados durante a interação com o usuário.

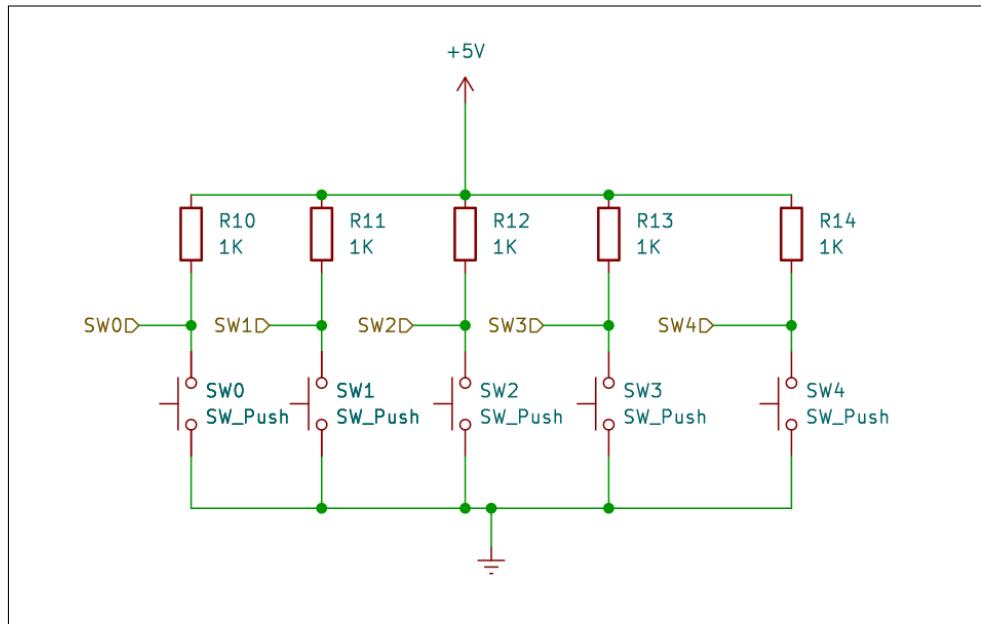


Figura 4: Subcircuito de interação com o usuário: teclado no Kicad

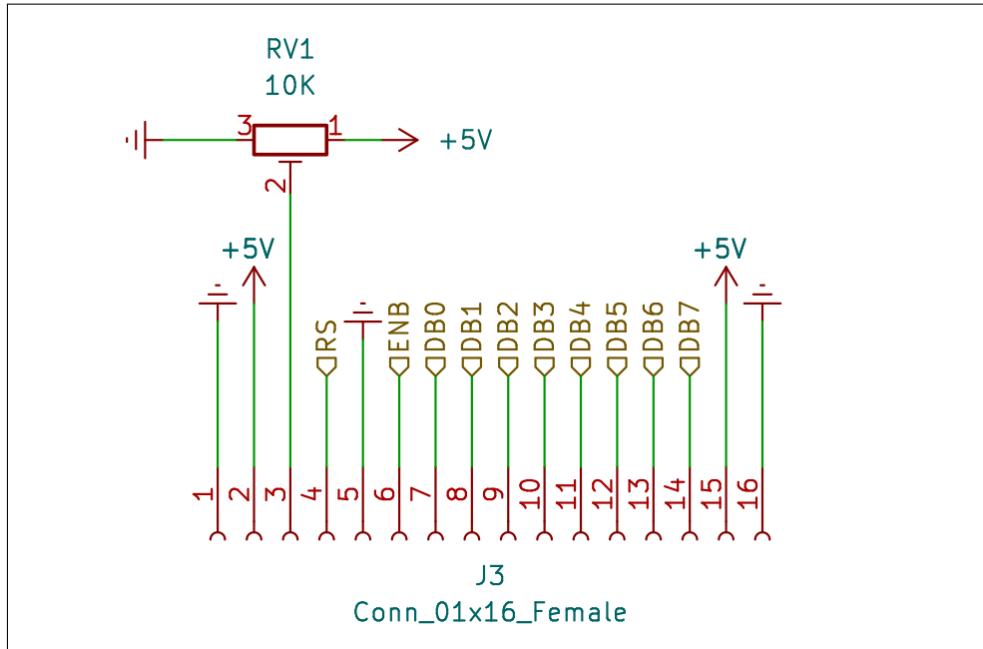


Figura 5: Subcircuito de interação com o usuário: LCD no Kicad

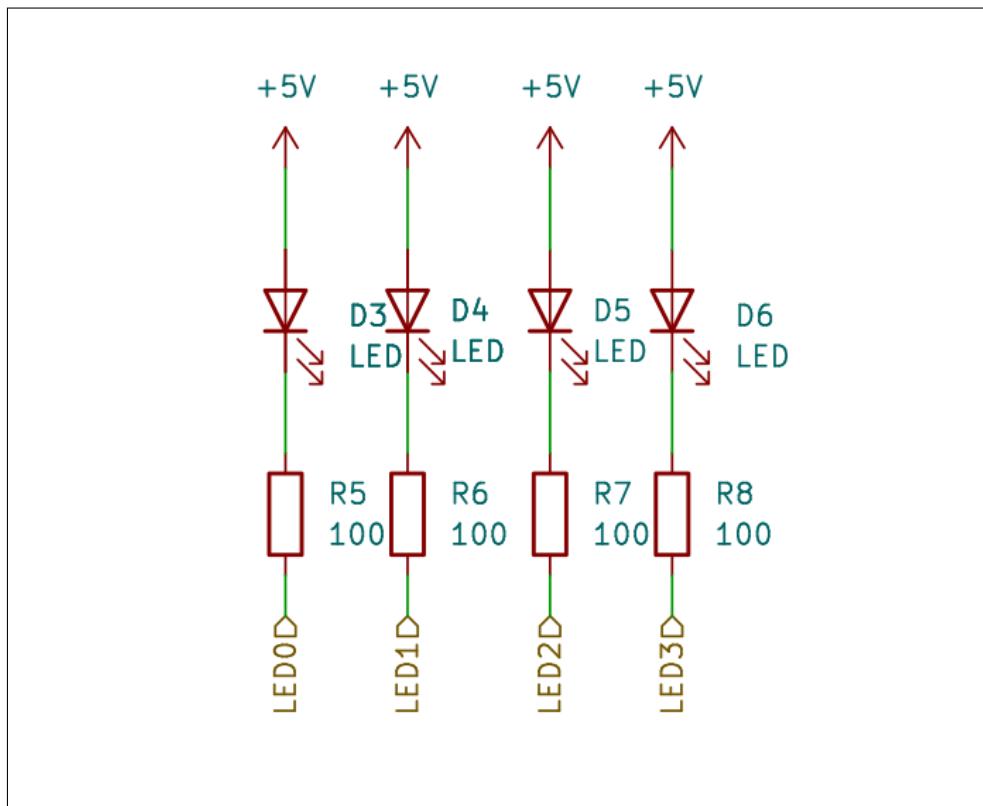


Figura 6: Subcircuito de interação com o usuário: LEDs no Kicad



Figura 7: Subcírculo de interação com o usuário: LCD na placa

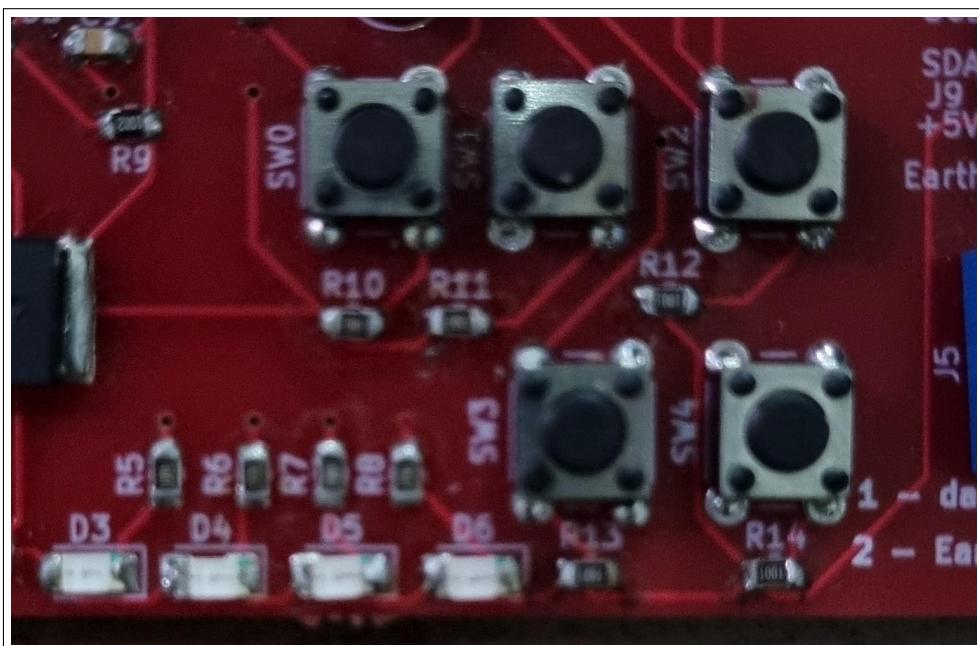


Figura 8: Subcírculo de interação com o usuário: botões e LEDs na placa

### 4.3 Subcírculo de comunicação serial

Nesta subseção, será representado o circuito de comunicação serial, que conta com um conversor USB-Serial (*MCP2200*), um conector USB e um oscilador externo de 12 MHz, necessários para realizar a comunicação serial com o usuário.

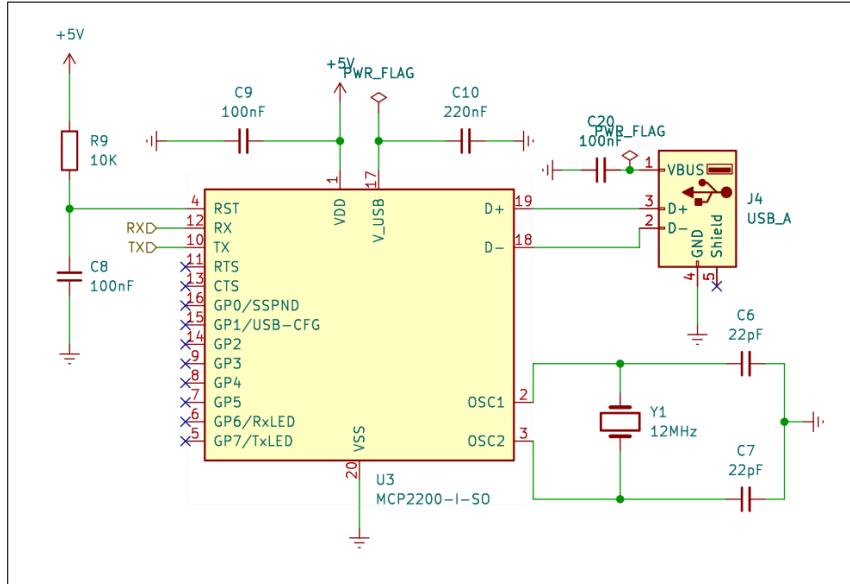


Figura 9: Subcircuito de comunicação serial no Kicad

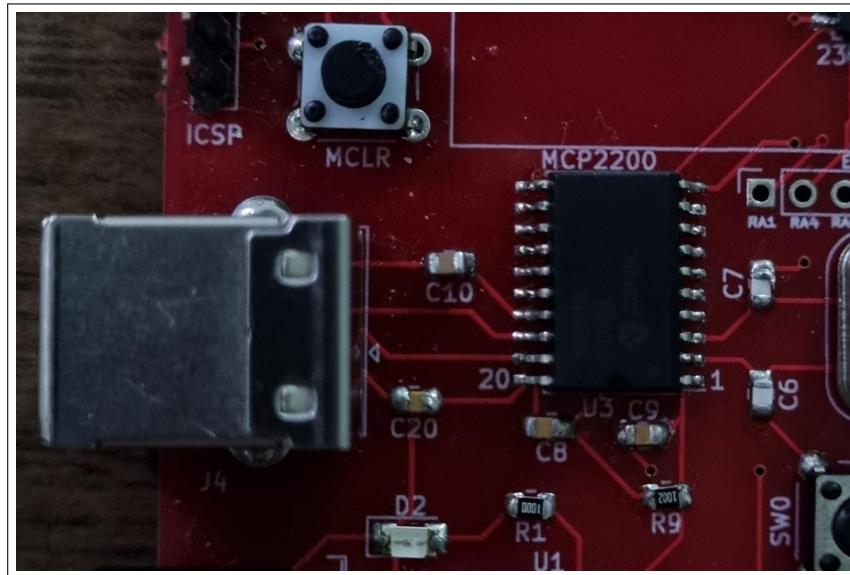


Figura 10: Subcircuito de comunicação serial na placa

#### 4.4 Subcircuito de operação

Nesta subseção, será representado o circuito de operação, que conta com uma barra de pinos para conectar o ICSP de gravação, um botão de reinício do tipo *pull-down*, além do microcontrolador utilizado e seu circuito de alimentação e oscilador externo.

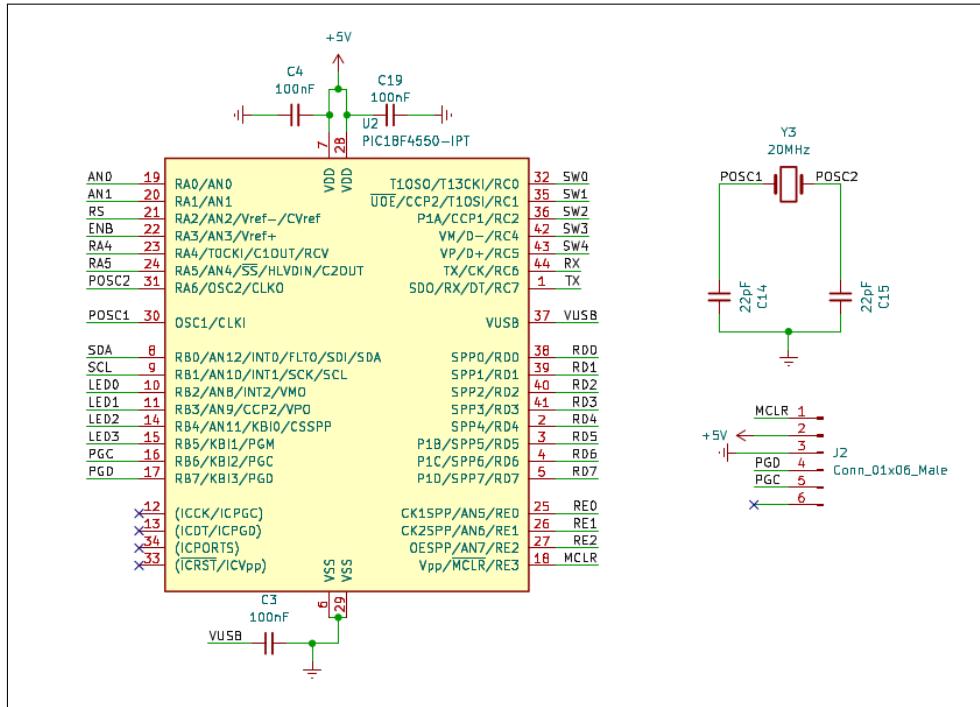


Figura 11: Subcircuito de operação no Kicad

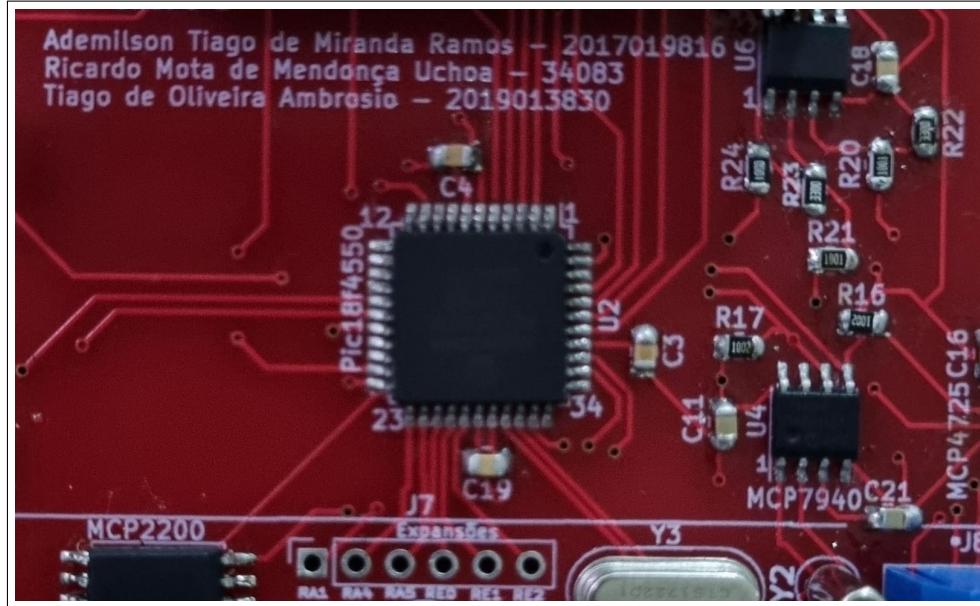


Figura 12: Subcircuito de operação na placa

## 4.5 Subcircuitos dos periféricos

### 4.5.1 Entrada diferencial

Nesta subsubseção, será representado o circuito de operação do amplificador diferencial, que transforma uma tensão de entrada de 0 a 10 [V] em uma tensão de 0 a 3 [V], necessário para a leitura analógica do microcontrolador.

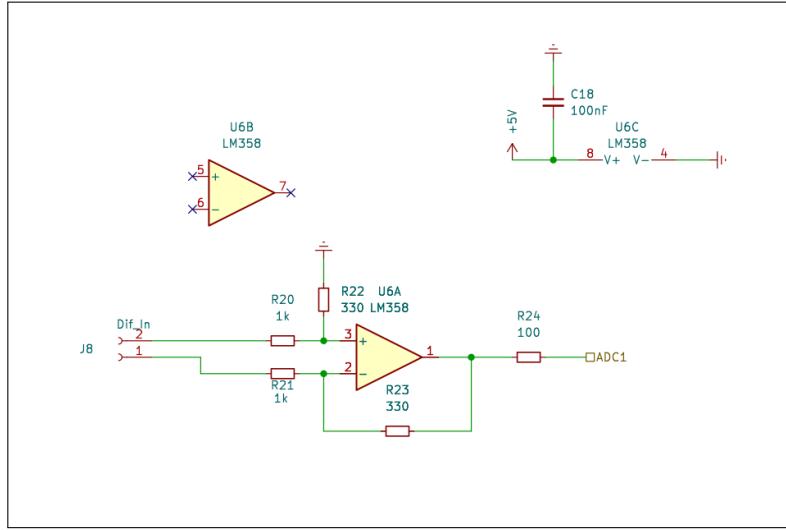


Figura 13: Subcircuito da entrada diferencial no Kicad

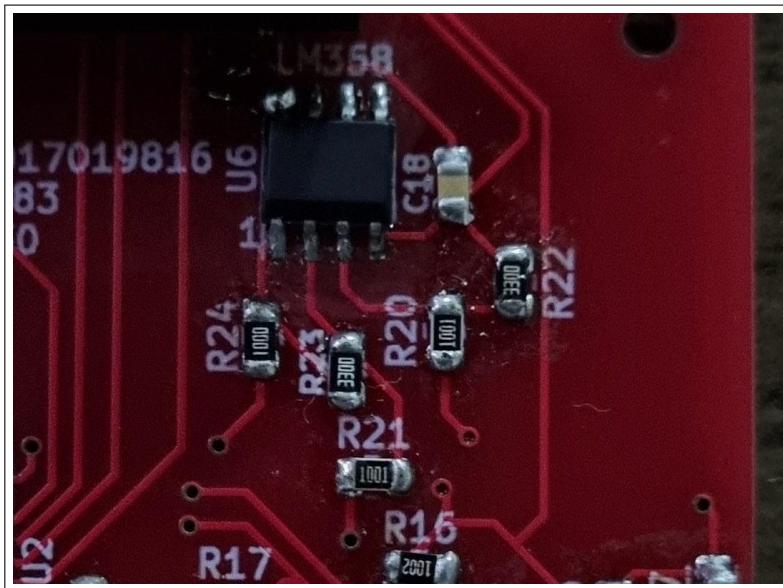


Figura 14: Subcircuito da entrada diferencial na placa

#### 4.5.2 Conversor digital-analógico

Nesta subsubseção, será representado o circuito de operação do conversor digital-analógico, que recebe, via comunicação I2C, um valor do microcontrolador e transforma em um valor analógico de tensão.

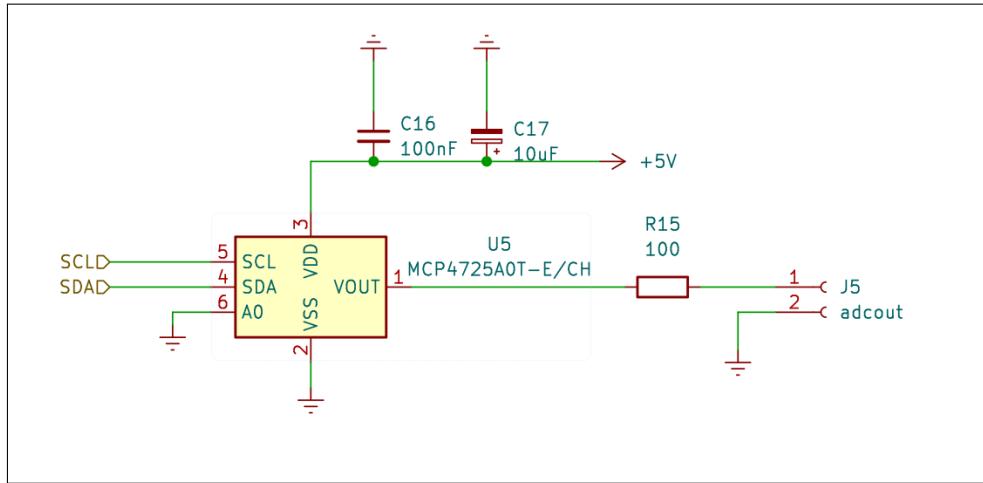


Figura 15: Subcircuito do conversor digital-analógico no Kicad

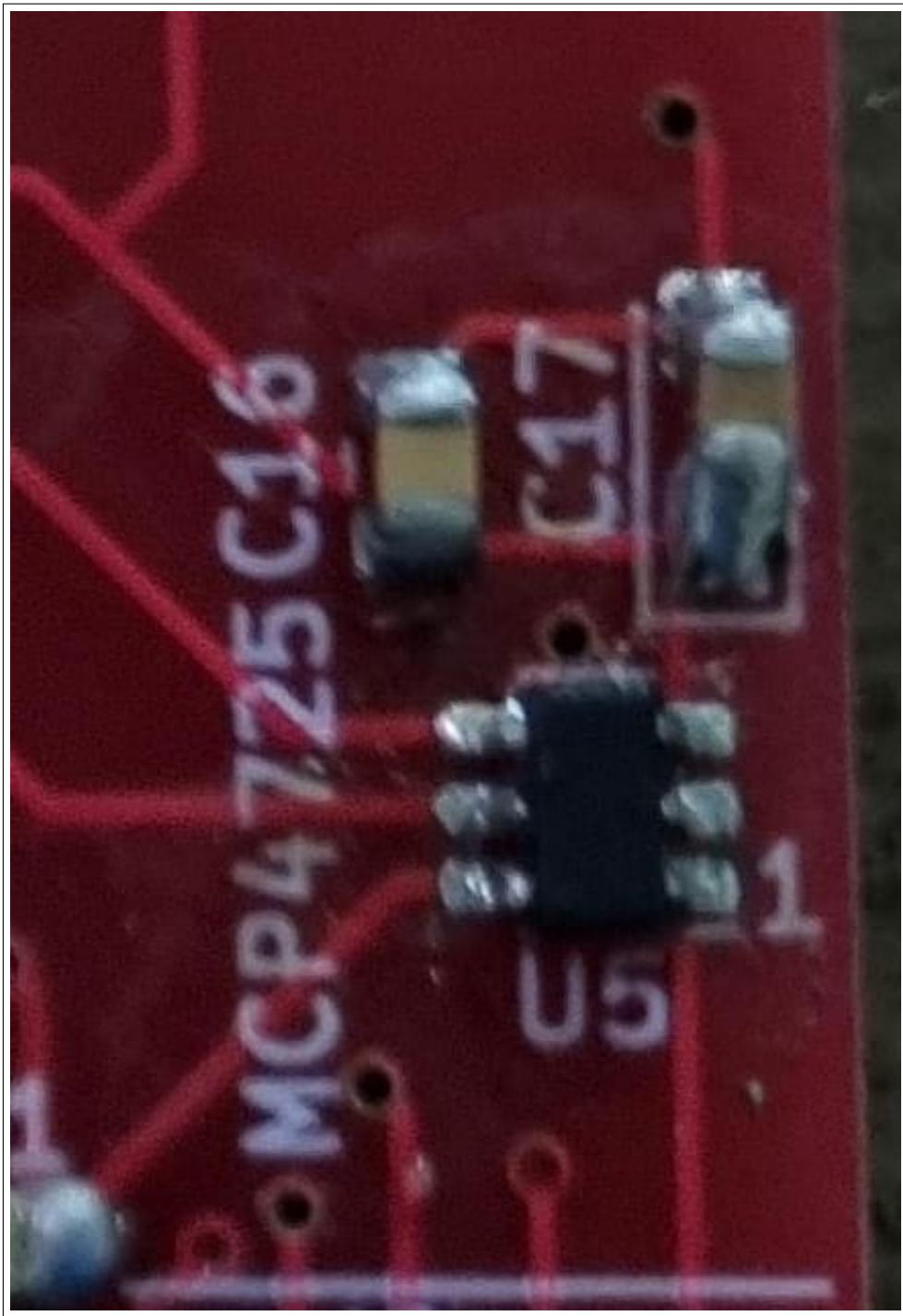


Figura 16: Subcírcuito do conversor digital-analógico na placa

#### 4.5.3 Relógio de tempo real

Nesta subsubseção, será representado o circuito de operação do relógio de tempo real, que se comunica via I2C com o microcontrolador e portanto, também será representado o circuito de comunicação I2C, que possui dois resistores de  $10\text{ k}\Omega$  do tipo *pull-up*.

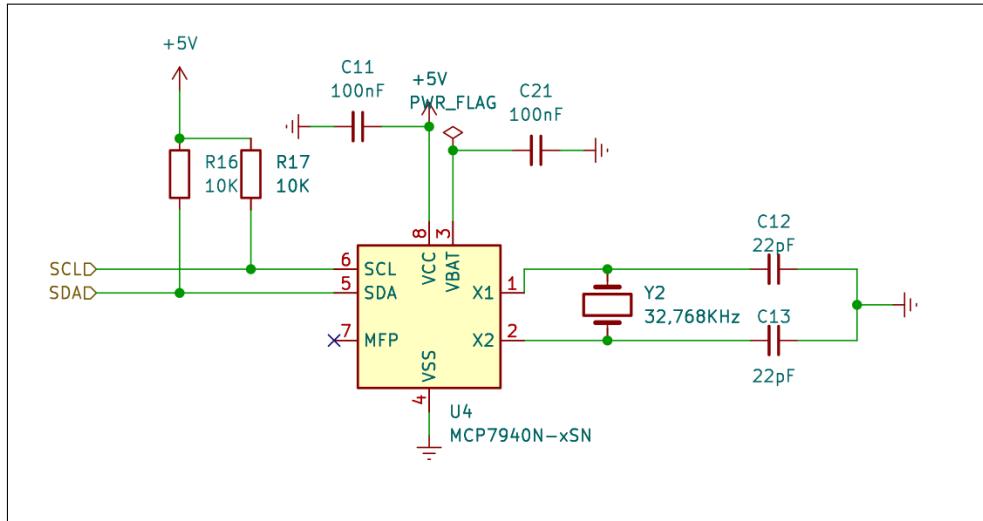


Figura 17: Subcircuito do relógio de tempo real no Kicad

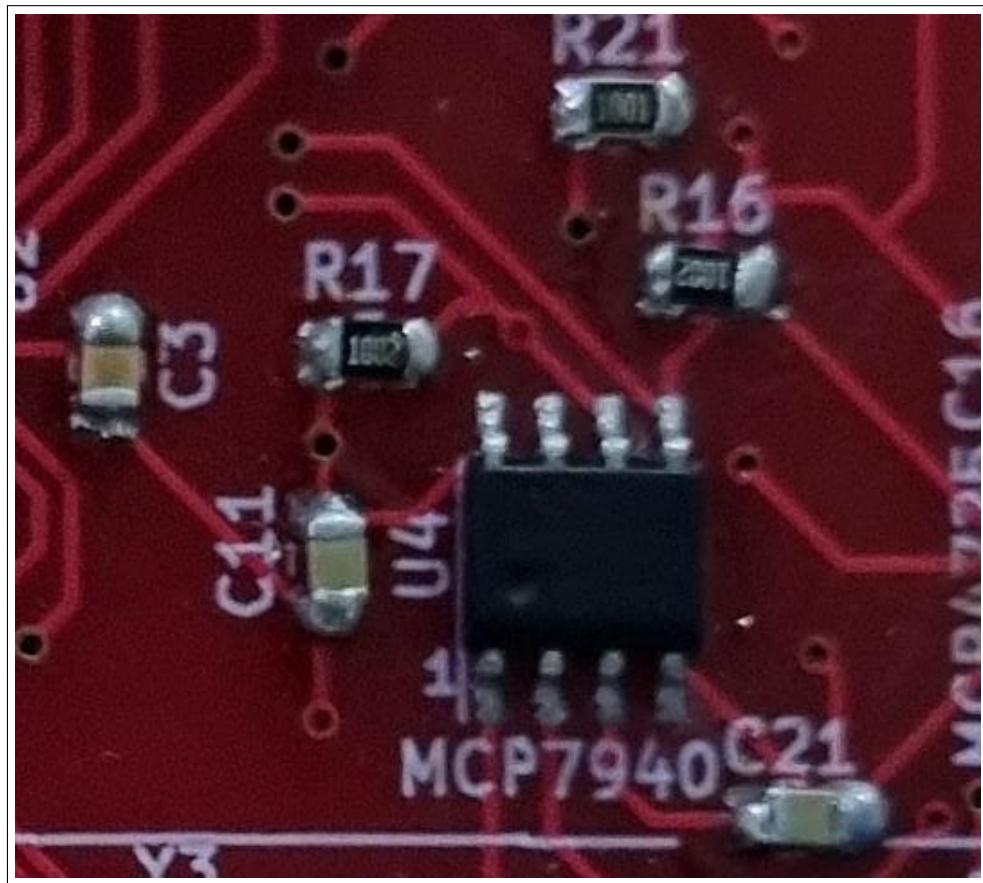


Figura 18: Subcircuito do relógio de tempo real na placa

## 4.6 Placa impressa

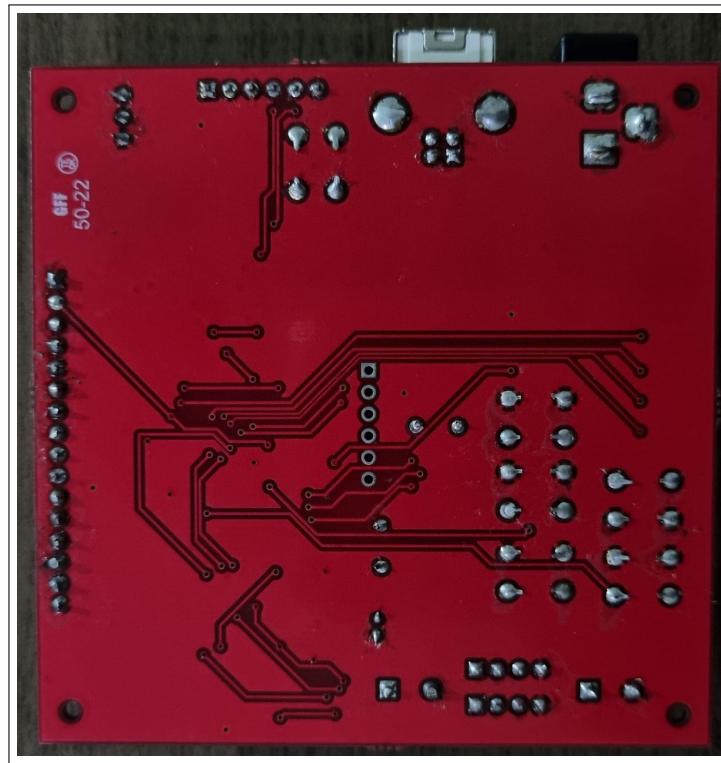


Figura 19: Vista inferior da placa impressa

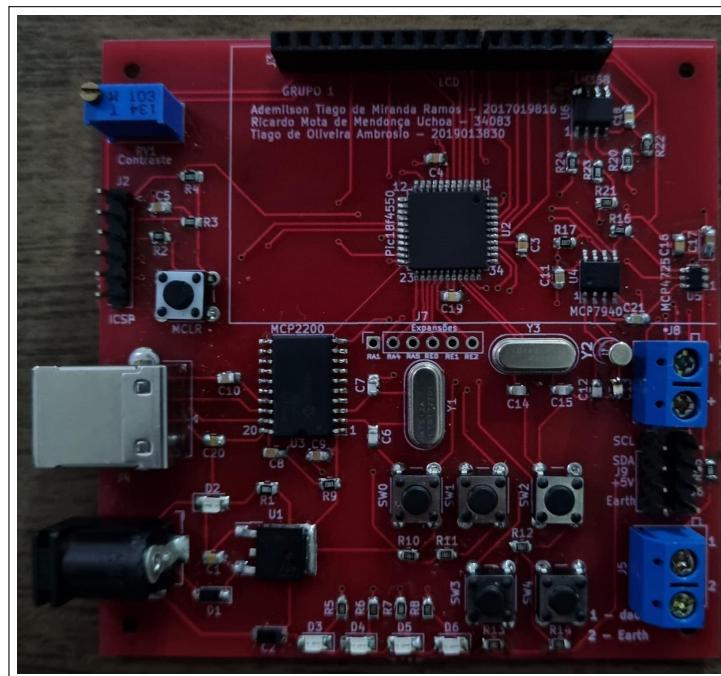


Figura 20: Vista superior da placa impressa

## 4.7 Relatório de verificação de erros do projeto elétrico

Abaixo nesta subseção, será representado o relatório de verificação de erros do projeto elétrico da placa confeccionada.

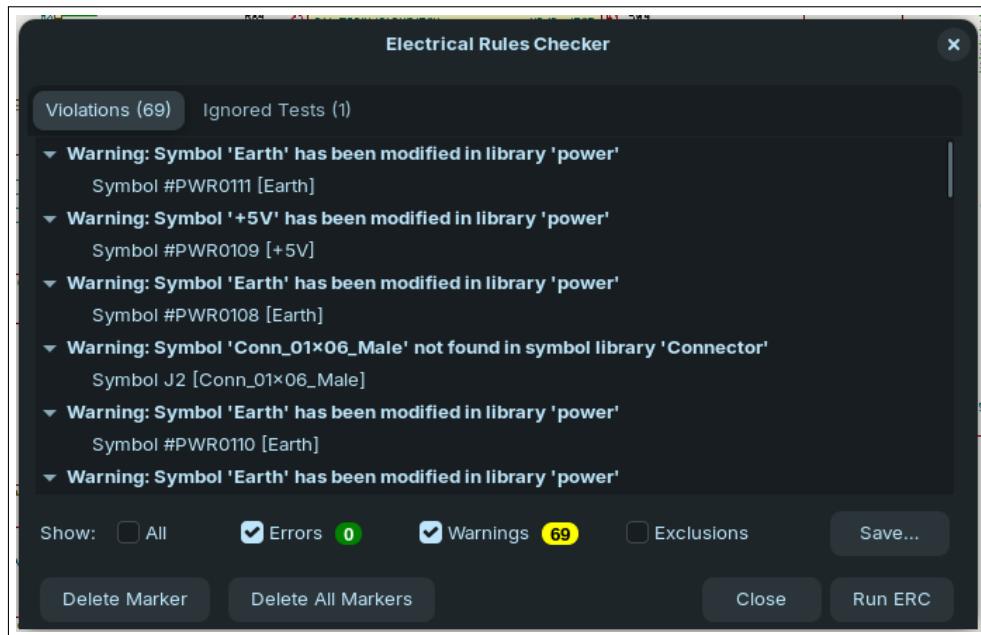


Figura 21: Relatório de verificação de erros do projeto elétrico

## 5 Placa de circuito impresso

Pensando na criação da PCI, foram considerados os seguintes requisitos da tabela a seguir:

Requisito	Classe	Especificação
R1	Características físicas da PCI	1 - Dimensão de até $8x8\text{ cm}^2$ ; 2 - Dupla face; 3 - Face inferior majoritariamente como plano de terra; 4 - Capacitores de supressão de tensão; 5 - Camada de texto com identificação de cada componente utilizado; 6 - Quatro furos de fixação dispostos em seus cantos; 7 - Identificar todas as conexões de entrada e saída.
R8	Espaçamento e dimensões de trilhas e afins	1 - Arquivos de fabricação em formato <i>Gerber RS274X</i> ; 2 - Mínima largura para trilhas de sinais: 8 mils; 3 - Mínima largura para trilhas de alimentação: 12 mils; 4 - Mínima largura entre trilhas, furos e ilhas: 8 mils; 5 - Mínimo diâmetro de furo de vias: 12 mils; 6 - Mínimo diâmetro de ilhas de vias: 25 mils; 7 - Não utilizar microvias;

Tabela 4: Requisitos para a PCI

Além disso, foram utilizados os seguintes componentes para a fabricação da placa:

Item	Modelo	Fabricante
Processador	PIC18F4550-I/PT	Microchip
Relógio de tempo real	MCP7940N-I/SN	Microchip
Transceptor USB-serial	MCP2200-I/SO	Microchip
Conversor digital para analógico	MCP4725A0T-E/CH	Microchip
Resistores	SMD 0805	Diversos
Capacitores	SMD 0805	Diversos
Capacitores	F931C106KAA	AVX
Trimpot de 10 kΩ	P160KN-0QC15B100K	TT Electronics
Trimmer de 10 kΩ	3296W-1-103RLF	Bourns Inc.
Barra de pinos	PPTC101LFBN-RC	Sullins Connector Solutions
Conecotor de energia	PJ-002A	CUI Devices
Diodos emissores de luz	LTST-C150GKT	Lite On
Amplificador operacional	LM358DR/LM358DG	On Semi
Conecotor USB	897-43-004-90-000000	Mill-Max
Chaves tátceis	1825910-6	TE Connectivity
Regulador de tensão	LD1117DT50TR	STMicroelectronics
Cristal de 32.768 kHz	AB38T-32.768KHZ	ABRACON
Cristal de 20 MHz e/ou 12MHz	ATS20A	CTS Electronic Components
Conecotor para entradas diferenciais	OSTTA024163	On Shore Technology Inc
Conecotor para barra de expansão	3-644456-2	TE Connectivity
Diodo retificador	1N4001RLG 1N5819HW-7	On Semi Diodes Incorporated
Visor de 16x2 pontos	JHD162A	-

Tabela 5: Lista de componentes

## 5.1 Desenho da PCI

Nesta subseção, serão mostrados os desenhos da PCI confeccionada.

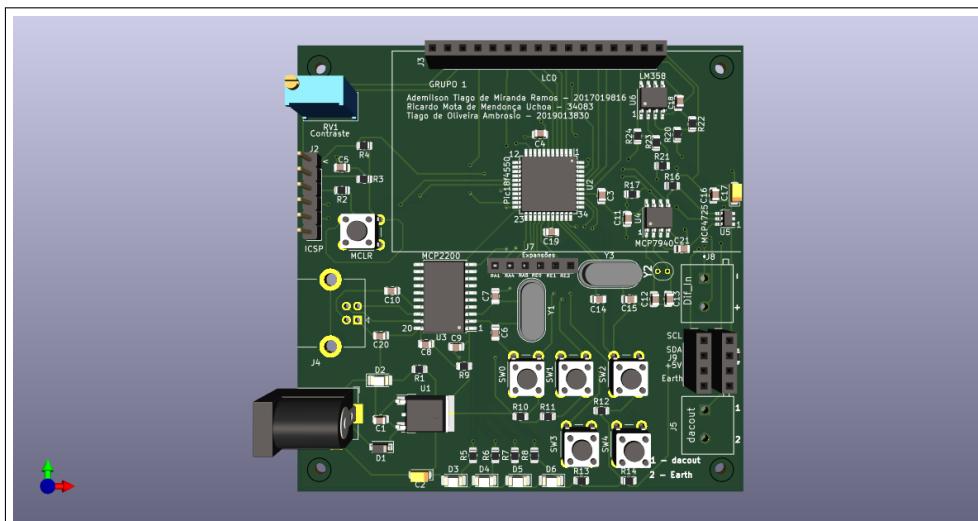


Figura 22: Imagem frontal da PCI

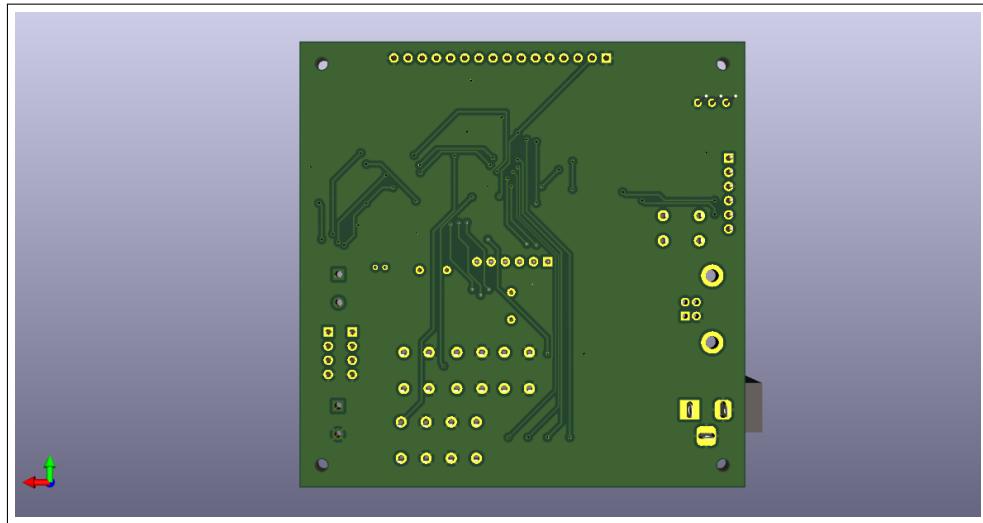


Figura 23: Imagem inferior da PCI

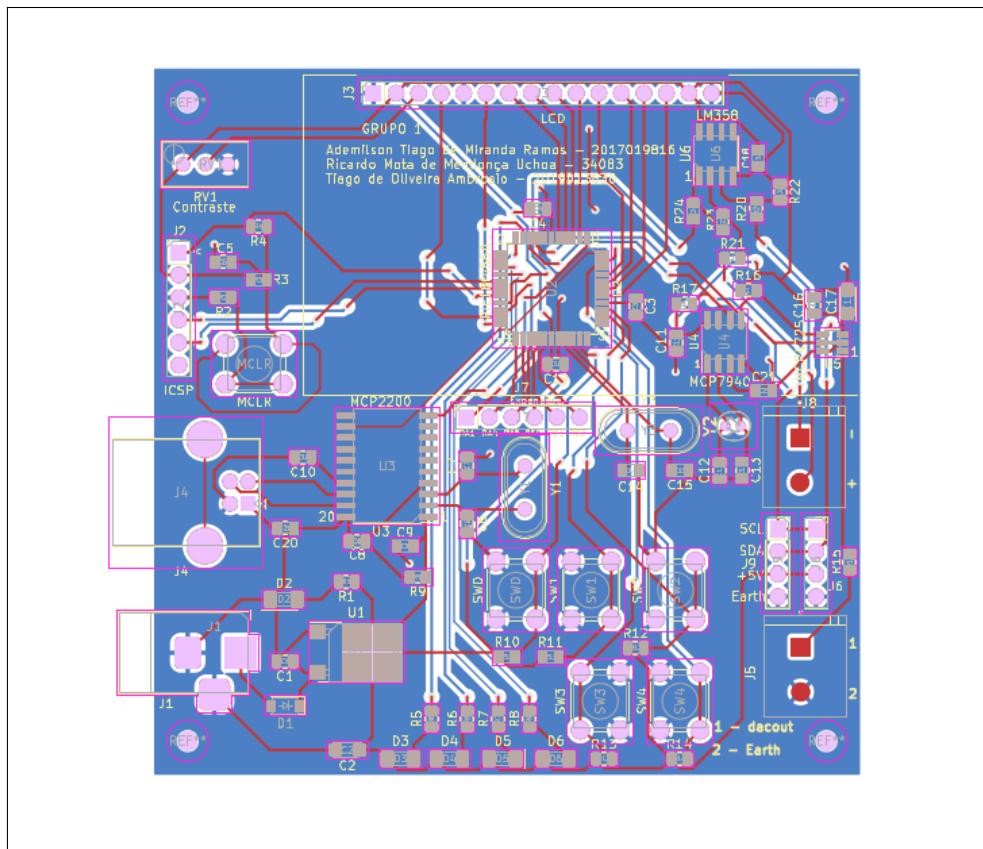


Figura 24: Imagem das trilhas da PCI

## 5.2 Relatório de verificação de erros do projeto

Nesta subseção, será representado o relatório de verificação de erros do projeto.



Figura 25: Relatório de verificação de erros do projeto

## 6 Características gerais

### 6.1 Mapa de pinos

Nesta subseção serão apresentados os mapas de pinos para o microcontrolador e para os circuitos auxiliares, como é o caso de circuitos de interação com o usuário.

Pino	Conexão	Pino	Conexão	Pino	Conexão	Pino	Conexão
01	UART	12	N/C	23	N/C	34	N/C
02	LCD	13	N/C	24	N/C	35	Teclado
03	LCD	14	LEDs	25	N/C	36	Teclado
04	LCD	15	LEDs	26	N/C	37	GND
05	LCD	16	ICSP	27	N/C	38	LCD
06	GND	17	ICSP	28	5 [V]	39	LCD
07	5 [V]	18	ICSP/Reset	29	GND	40	LCD
08	RTC/DAC	19	Diferencial IN	30	Oscilador	41	LCD
09	RTC/DAC	20	N/C	31	Oscilador	42	Teclado
10	LEDs	21	LCD	32	Teclado	43	Teclado
11	LEDs	22	LCD	33	N/C	44	UART

Tabela 6: Conexões do microcontrolador

Componente	Pino	Conexão	Pino	Conexão
U3 (MCP2200)	1 - VDD	5 [V]	11 - RTS	N/C
	2 - Oscilador	Crystal Y1	12 - RX	Micro(TX)
	3 - Oscilador	Crystal Y1	13 - CTS	N/C
	4 - Reset	5 [V]	14 - GP2	N/C
	5 - GP7	N/C	15 - GP1	N/C
	6 - GP6	N/C	16 - GP0	N/C
	7 - GP5	N/C	17 - V_USB	GND
	8 - GP4	N/C	18 - D-	USB(D-)
	9 - GP3	N/C	19 - D+	USB(D+)
	10 - TX	Micro(RX)	20 - VSS	GND
USB	1 - VBUS	GND	4 - GND	GND
	2 - D-	MCP2200(D-)	5 - Shield	N/C
	3 - D+	MCP2200(D+)	-	-

Tabela 7: Conexões do conversor USB-Serial

Componente	Pino	Conexão	Pino	Conexão
U4 (MCP7940N)	1 - X1	Crystal Y2	5 - SDA	SDA
	2 - X2	Crystal Y2	6 - SCL	SCL
	3 - VBAT	GND	7 - MFP	N/C
	4 - VSS	GND	8 - VCC	5 [V]

Tabela 8: Conexões do relógio de tempo real

Componente	Pino	Conexão	Pino	Conexão
U5 (MCP4725)	1 - VOUT	J5 (pino 1)	4 - SDA	SDA
	2 - VSS	GND	5 - SCL	SCL
	3 - VDD	5 [V]	6 - A0	GND
J5	1	MCP4725 (VOUT)	2	GND

Tabela 9: Conexões do conversor D/A

Componente	Pino	Conexão	Pino	Conexão
U6 (LM358)	1	Micro (pino 19)	5	N/C
	2	J8 (pino 1)	6	N/C
	3	J8 (pino 2)	7	N/C
	4	GND	8	5 [V]
J8	1	LM358 (pino 3)	2	LM358 (pino 2)

Tabela 10: Conexões da entrada analógica

Componente	Pino	Conexão	Pino	Conexão
J6	1	Micro (pino 20)	4	Micro (pino 25)
	2	Micro (pino 23)	5	Micro (pino 26)
	3	Micro (pino 24)	6	Micro (pino 27)
J7	1	SCL	3	5 [V]
	2	SDA	4	GND
J9	1	SCL	3	5 [V]
	2	SDA	4	GND

Tabela 11: Conexões dos pinos de expansão

Componente	Pino	Conexão	Pino	Conexão
J3 (LCD)	1	GND	9	Micro (pino 3)
	2	5 [V]	10	Micro (pino 2)
	3	RV1 (pino 2)	11	Micro (pino 41)
	4	Micro (pino 21)	12	Micro (pino 40)
	5	GND	13	Micro (pino 39)
	6	Micro (pino 22)	14	Micro (pino 38)
	7	Micro (pino 5)	15	5 [V]
	8	Micro (pino 4)	16	GND
RV1	1	5 [V]	3	GND
	2	LCD (pino 3)	-	-

Tabela 12: Conexões do LCD

## 6.2 Alimentação e consumo

Como característica de alimentação, tem-se que a placa deve possuir uma alimentação de, pelo menos, 7 [V], enquanto o regulador de tensão suporta uma entrada de até 15 [V] que será

regulada para 5 [V]. Além disso, o regulador consegue gerar até um máximo de 800 mA, porém o circuito necessita de apenas 470 mA e suporta até 12 W de potência, enquanto o circuito deve consumir apenas 2.5 W durante seu funcionamento.

## 7 Códigos-fonte

Nesta seção serão apresentados todos os códigos-fonte utilizados para o funcionamento da placa, desde os periféricos até o funcionamento da interface homem-máquina (IHM) proposta.

### 7.1 ADC

```
1  ifndef ADC_H
2  define ADC_H
3
4      void adcInit(void);
5
6      unsigned int adcRead(void);
7
8  endif /* ADC_H */
```

Figura 26: Header file para o funcionamento do ADC

```
1  #include <xc.h>
2  #include <pic18f4550.h>
3  #include "defines.h"
4  #include "ADC.h"
5
6  void adcInit(void) {
7      BitSet(TRISA, 0); // Set the RA0 as input
8      ADCON0 = 0b00000001; // Select the Channel 0
9      ADCON1 = 0b00001110; // Just AN0 is analog and the reference is based on VSS/VDD
10     ADCON2 = 0b10101010; // Time of conversion: 12 TAD, FOSC/32
11     return;
12 }
13
14 unsigned int adcRead(void) {
15     unsigned int valueRead;
16     ADCON0 |= 0b00000010; // Initialize the conversion
17     while (BitTst(ADCON0, 1)); // Wait to finish
18     valueRead = ADRESH ; // Read the register
19     valueRead <= 8;
20     valueRead += ADRESL; // Read the register
21     return valueRead;
22 }
```

Figura 27: Source file para o funcionamento do ADC

### 7.2 button

```
1  ifndef BUTTON_H
2  define BUTTON_H
3
4      void buttonInit(void);
5
6  endif /* BUTTON_H */
```

Figura 28: Header file para o funcionamento dos botões

```

1  #include <xc.h>
2  #include <pic18f4550.h>
3  #include "defines.h"
4  #include "button.h"
5
6  void buttonInit(void) {
7      BitClr(UCON, 3);
8      BitSet(UCFG, 3);
9      TRISC = 0xFF;
10     return;
11 }
```

Figura 29: Source file para o funcionamento dos botões

### 7.3 config

```

1  ifndef CONFIG_H
2  define CONFIG_H
3
4      include "defines.h"
5
6      define PIC18F4550
7
8  ifdef PIC18F4550
9      pragma config MCLRE = ON
10     pragma config FOSC = HS
11     pragma config WDT = OFF
12     pragma config LVP = OFF
13     endif
14
15  endif /* CONFIG_H */
```

Figura 30: Arquivo de configuração para o microcontrolador

### 7.4 DAC\_MCP4725

```

1  ifndef DAC_MCP4725_H
2  define DAC_MCP4725_H
3
4      void DAC_MCP4725_I2C_Master_Init(uint32_t clock);
5
6      void DAC_MCP4725_I2C_Master_Wait(void);
7
8      void DAC_MCP4725_I2C_Start(void);
9
10     void DAC_MCP4725_I2C_Stop(void);
11
12     void DAC_MCP4725_I2C_Repeated_Start(void);
13
14     void DAC_MCP4725_I2C_Master_Write(uint8_t data);
15
16     uint8_t DAC_MCP4725_I2C_Master_Read(uint8_t ACK);
17
18     void DAC_MCP4725_SetVoltage(uint16_t voltage);
19
20     uint16_t DAC_MCP4725_Read(void);
21
22  endif /* DAC_MCP4725_H */
```

Figura 31: Header file para o funcionamento do DAC

```

1 #include <xc.h>
2 #include <pic18f4550.h>
3 #include "config.h"
4 #include "defines.h"
5 #include "DAC_MCP4725.h"
6
7 void DAC_MCP4725_I2C_Master_Init(uint32_t clock) {
8     BitSet(TRISB, 0);
9     BitSet(TRISB, 1);
10    SSPSTATbits.SMP = 1;
11    SSPCON1bits.SSPEEN = 1;
12    SSPCON1bits.SSPM = 0b1000;
13    SSPADD = (uint8_t)((FREQ/(4.0 * clock)) - 1);
14    SSPCON2 = 0x00;
15    return;
16 }
17
18 void DAC_MCP4725_I2C_Master_Wait(void) {
19     while ((SSPCON2 & 0b00011111) || (SSPSTAT & 0b00000100));
20     return;
21 }
22
23 void DAC_MCP4725_I2C_Start(void) {
24     DAC_MCP4725_I2C_Master_Wait();
25     SSPCON2bits.SEN = 1;
26     return;
27 }
28
29 void DAC_MCP4725_I2C_Stop(void) {
30     DAC_MCP4725_I2C_Master_Wait();
31     SSPCON2bits.PEN = 1;
32     return;
33 }
34
35 void DAC_MCP4725_I2C_Repeated_Start(void) {
36     DAC_MCP4725_I2C_Master_Wait();
37     SSPCON2bits.RSEN = 1;
38     return;
39 }
40
41 void DAC_MCP4725_I2C_Master_Write(uint8_t data) {
42     DAC_MCP4725_I2C_Master_Wait();
43     SSPBUF = data;
44     return;
45 }
46
47 uint8_t DAC_MCP4725_I2C_Master_Read(uint8_t ACK) {
48     uint8_t data;
49     DAC_MCP4725_I2C_Master_Wait();
50     SSPCON2bits.RCEN = 1;
51     DAC_MCP4725_I2C_Master_Wait();
52     data = SSPBUF;
53     DAC_MCP4725_I2C_Master_Wait();
54     SSPCON2bits.ACKDT = (ACK) ? 0 : 1;
55     SSPCON2bits.ACKEN = 1;
56     return data;
57 }
58
59 void DAC_MCP4725_SetVoltage(uint16_t voltage) {
60     DAC_MCP4725_I2C_Start();
61     DAC_MCP4725_I2C_Master_Wait();
62     DAC_MCP4725_I2C_Master_Write(0b11000000);
63     DAC_MCP4725_I2C_Master_Wait();
64     DAC_MCP4725_I2C_Master_Write(0b01000000);
65     DAC_MCP4725_I2C_Master_Wait();
66     DAC_MCP4725_I2C_Master_Write((uint8_t)(voltage >> 4));
67     DAC_MCP4725_I2C_Master_Wait();
68     DAC_MCP4725_I2C_Master_Write((uint8_t)((voltage & 0b1111) << 4));
69     DAC_MCP4725_I2C_Master_Wait();
70     DAC_MCP4725_I2C_Stop();
71     DAC_MCP4725_I2C_Master_Wait();
72     return;
73 }
74
75 uint16_t DAC_MCP4725_Read(void) {
76     uint8_t byte2;
77     uint8_t byte3;
78     uint8_t byte4;
79     uint8_t byte5;
80     uint8_t byte6;
81     uint16_t data;
82     DAC_MCP4725_I2C_Start();
83     DAC_MCP4725_I2C_Master_Wait();
84     DAC_MCP4725_I2C_Master_Write(0b11000001);
85     DAC_MCP4725_I2C_Master_Wait();
86     byte2 = DAC_MCP4725_I2C_Master_Read(4);
87     byte3 = DAC_MCP4725_I2C_Master_Read(3);
88     byte4 = DAC_MCP4725_I2C_Master_Read(2);
89     byte5 = DAC_MCP4725_I2C_Master_Read(1);
90     byte6 = DAC_MCP4725_I2C_Master_Read(0);
91     data = (((uint16_t)byte5 << 8) | byte6);
92     DAC_MCP4725_I2C_Stop();
93     DAC_MCP4725_I2C_Master_Wait();
94     return data;
95 }

```

Figura 32: Source file para o funcionamento do DAC

## 7.5 defines

```
1  ifndef DEFINES_H
2  define DEFINES_H
3
4      #define PIC18F4550
5
6      #define FREQ 20000000
7
8      #define SPEED 25
9
10     #define NUMBER_LANGUAGES 6
11
12     #define MAX_CHARACTERS 16
13
14     #define MAX_STRING_LENGTH 15
15
16     #define MIN_ADC_VALUE 0000
17     #define MAX_ADC_VALUE 1000
18
19     #define BitSet(arg,bit) ((arg) |= (1<<bit))
20     #define BitClr(arg,bit) ((arg) &= ~(1<<bit))
21     #define BitFlp(arg,bit) ((arg) ^= (1<<bit))
22     #define BitTst(arg,bit) ((arg) & (1<<bit))
23
24 endif /* DEFINES_H */
```

Figura 33: Arquivo de defines para o microcontrolador

## 7.6 delay

```
1  ifndef DELAY_H
2  define DELAY_H
3
4      void Delay40us(void);
5
6      void Delay2ms(void);
7
8      void Delay10ms(void);
9
10     void Delay100ms(void);
11
12 endif /* DELAY_H */
```

Figura 34: Header file para o funcionamento de delays

```

1  #include "delay.h"
2
3  void Delay40us(void) {
4      unsigned char index;
5      for(index = 0; index < 25; index++) {
6          __asm("NOP");
7      }
8      return;
9  }
10
11 void Delay2ms(void) {
12     unsigned char index;
13     for(index = 0; index < 50; index++)
14         Delay40us();
15     return;
16 }
17
18 void Delay10ms(void) {
19     unsigned char index;
20     for(index = 0; index < 5; index++)
21         Delay2ms();
22     return;
23 }
24
25 void Delay100ms(void) {
26     unsigned char index;
27     for(index = 0; index < 10; index++)
28         Delay10ms();
29     return;
30 }

```

Figura 35: Source file para o funcionamento de delays

## 7.7 event

```

1  ifndef EVENT_H
2  define EVENT_H
3
4  enum {
5      EV_UP,
6      EV_DOWN,
7      EV_LEFT,
8      EV_RIGHT,
9      EV_BASE,
10     EV_ALARM,
11     EV_NOEVENT
12 };
13
14 void eventInit(void);
15
16 unsigned char eventButton(void);
17
18 unsigned char eventSerial(unsigned char character);
19
20 unsigned char eventRead(void);
21
22 endif /* EVENT_H */

```

Figura 36: Header file para o funcionamento dos eventos

```

1 #include <xc.h>
2 #include <pic18f4550.h>
3 #include "event.h"
4 #include "defines.h"
5 #include "ADC.h"
6 #include "button.h"
7 #include "LCD.h"
8 #include "serial.h"
9 #include "variables.h"
10 #include "delay.h"
11
12 static unsigned char isCommand;
13
14 void eventInit(void) {
15     adcInit();
16     buttonInit();
17     serialInit();
18     isCommand = 0;
19     return;
20 }
21
22 unsigned char eventButton(void) {
23     static unsigned char oldEvent = EV_NOEVENT;
24     unsigned char event = EV_NOEVENT;
25     if (!BitTst(PORTC, 0)) // SW0
26         event = EV_LEFT;
27     if (!BitTst(PORTC, 1)) // SW1
28         event = EV_BASE;
29     if (!BitTst(PORTC, 2)) // SW2
30         event = EV_RIGHT;
31     if (!BitTst(PORTC, 4)) // SW3
32         event = EV_DOWN;
33     if (!BitTst(PORTC, 5)) // SW4
34         event = EV_UP;
35     if (event != oldEvent) {
36         oldEvent = event;
37         return event;
38     }
39 }
40
41
42 unsigned char eventSerial(unsigned char character) {
43     unsigned char event = EV_NOEVENT;
44     if (character == 'W' || character == 'w')
45         event = EV_UP;
46     if (character == 'A' || character == 'a')
47         event = EV_LEFT;
48     if (character == 'S' || character == 's')
49         event = EV_DOWN;
50     if (character == 'D' || character == 'd')
51         event = EV_RIGHT;
52     if (character == 'R' || character == 'r')
53         event = EV_BASE;
54     return event;
55 }
56
57 unsigned char eventRead(void) {
58     unsigned char event = EV_NOEVENT;
59     unsigned char character = serialRead();
60     setValueADC(adcRead());
61     if (getValueADC() > getAlarmLevelHigh() || getValueADC() < getAlarmLevelLow())
62         event = EV_ALARM;
63     if (character == '/') {
64         isCommand = 1;
65     }
66     if (!isCommand)
67         event = (eventSerial(character) != EV_NOEVENT) ? eventSerial(character) : eventButton();
68     else {
69         if (character == '\r' || character == '\n') {
70             isCommand = 0;
71             executeProtocol();
72             resetSerialBuffer();
73         }
74         else
75             if (character != 0)
76                 addSerialBuffer(character);
77     }
78     Delay10ms();
79 }
80

```

Figura 37: Source file para o funcionamento dos eventos

## 7.8 I2C

```
1  #ifndef I2C_H
2  #define I2C_H
3
4  void I2C_delay(void);
5  unsigned char read_SCL(void);
6  void clear_SCL(void);
7  unsigned char read_SDA(void);
8  void clear_SDA(void);
9
10 void I2C_Init(void);
11 void I2C_Start(void);
12 void I2C_Stop(void);
13 void I2C_WriteBit(unsigned char bit);
14 unsigned char I2C_WriteByte(unsigned char start, unsigned char stop, unsigned char byte);
15 unsigned char I2C_ReadBit(void);
16 unsigned char I2C_ReadByte(unsigned char nack, unsigned char stop);
17
18 #endif /* I2C_H */
```

Figura 38: Header file para o funcionamento da comunicação I2C

```

1  #include <xc.h>
2  #include <pic18f4550.h>
3  #include "defines.h"
4  #include "I2C.h"
5
6  #define SDA() BitTst(PORTB, 0)
7  #define SDA_OFF() BitClr(PORTB, 0)
8  #define SDA_IN() BitSet(TRISB, 0)
9  #define SDA_OUT() BitClr(TRISB, 0)
10
11 #define SCL() BitTst(PORTB, 1)
12 #define SCL_OFF() BitClr(PORTB, 1)
13 #define SCL_IN() BitSet(TRISB, 1)
14 #define SCL_OUT() BitClr(TRISB, 1)
15
16 unsigned char started = 0;
17
18 void I2C_delay(void) {
19     int index = 0;
20     for (index = 0; index < SPEED / 2; index++)
21         __asm("NOP");
22     return;
23 }
24
25 unsigned char read_SCL(void) {
26     SCL_IN();
27     return !SCL();
28 }
29
30 void clear_SCL(void) {
31     SCL_OUT();
32     SCL_IN();
33     return;
34 }
35
36 unsigned char read_SDA(void) {
37     SDA_IN();
38     return !SDA();
39 }
40
41 void clear_SDA(void) {
42     SDA_OUT();
43     SDA_OFF();
44     return;
45 }
46
47 void I2C_Init(void) {
48     SDA_IN();
49     SCL_IN();
50     return;
51 }
52
53 void I2C_Start(void) {
54     if (started) {
55         read_SDA();
56         I2C_delay();
57         while (!read_SCL());
58         I2C_delay();
59     }
60     if (!read_SDA()) {
61         // Error
62     }
63     clear_SDA();
64     I2C_delay();
65     clear_SCL();
66     started = 1;
67     return;
68 }
69
70 void I2C_Stop(void) {
71     clear_SDA();
72     I2C_delay();
73     while (!read_SCL());
74     I2C_delay();
75     if (!read_SDA()) {
76         // Error
77     }
78     I2C_delay();
79     started = 0;
80     return;
81 }
82
83 void I2C_WriteBit(unsigned char bit) {
84     if (bit)
85         read_SDA();
86     else
87         clear_SDA();
88     I2C_delay();
89     while (!read_SCL());
90     if (bit && !read_SDA()) {
91         // Error
92     }
93     I2C_delay();
94     clear_SCL();
95     I2C_delay();
96     return;
97 }

```

Figura 39: Source file para o funcionamento da comunicação I2C (parte 1)

```

98  unsigned char I2C_WriteByte(unsigned char start, unsigned char stop, unsigned char byte) {
99    unsigned char bit;
100   unsigned char nack;
101   if (start) {
102     I2C_Start();
103   }
104   for (bit = 0; bit < 8; bit++) {
105     I2C_WriteBit((byte & 0x80) != 0);
106     byte <<= 1;
107   }
108   nack = I2C_ReadBit();
109   if (stop) {
110     I2C_Stop();
111   }
112   return nack;
113 }
114
115
116 unsigned char I2C_ReadBit(void) {
117   unsigned char bit;
118   read_SDA();
119   I2C_delay();
120   while (!read_SCL());
121   bit = read_SDA();
122   I2C_delay();
123   clear_SCL();
124   I2C_delay();
125   return bit;
126 }
127
128 unsigned char I2C_ReadByte(unsigned char nack, unsigned char stop) {
129   unsigned char byte = 0;
130   unsigned char bit;
131   for (bit = 0; bit < 8; bit++) {
132     byte = (unsigned char)(byte << 1) | I2C_ReadBit();
133   }
134   I2C_WriteBit(nack);
135   if (stop) {
136     I2C_Stop();
137   }
138   return byte;
139 }

```

Figura 40: Source file para o funcionamento da comunicação I2C (parte 2)

## 7.9 LCD

```

1 #ifndef LCD_H
2 #define LCD_H
3
4   void lcdInit(void);
5
6   void lcdCommand(unsigned char cmd);
7
8   void lcdData(unsigned char value);
9
10  void lcdString(char *message);
11
12  void lcdInt(int value);
13
14 #endif /* LCD_H */

```

Figura 41: Header file para o funcionamento do LCD

```

1  #include <xc.h>
2  #include <pic18f4550.h>
3  #include "defines.h"
4  #include "LCD.h"
5  #include "delay.h"
6
7  #define RS 2
8  #define EN 3
9
10 void lcdInit(void) {
11     ADCON1 = 0b00001110;
12     BitClr(TRISA, RS);
13     BitClr(TRISA, EN);
14     TRISD = 0x00;
15     Delay10ms();
16     lcdCommand(0x38);
17     Delay2ms();
18     lcdCommand(0x38);
19     Delay2ms();
20     lcdCommand(0x38);
21     lcdCommand(0x38);
22     lcdCommand(0x06);
23     lcdCommand(0x0F);
24     lcdCommand(0x01);
25     return;
26 }
27
28
29 void lcdCommand(unsigned char command) {
30     BitClr(PORTA, RS);
31     PORTD = (unsigned char)(
32         ((command & 0b00000001) << 7) |
33         ((command & 0b00000010) << 5) |
34         ((command & 0b00000100) << 3) |
35         ((command & 0b00001000) << 1) |
36         ((command & 0b00010000) >> 1) |
37         ((command & 0b00100000) >> 3) |
38         ((command & 0b01000000) >> 5) |
39         ((command & 0b10000000) >> 7));
40     BitSet(PORTA, EN);
41     BitClr(PORTA, EN);
42     if((command == 0x02) || (command == 0x01))
43         Delay2ms();
44     else
45         Delay40us();
46     return;
47 }
48
49 void lcdData(unsigned char value) {
50     BitSet(PORTA, RS);
51     PORTD = (unsigned char)(
52         ((value & 0b00000001) << 7) |
53         ((value & 0b00000010) << 5) |
54         ((value & 0b00000100) << 3) |
55         ((value & 0b00001000) << 1) |
56         ((value & 0b00010000) >> 1) |
57         ((value & 0b00100000) >> 3) |
58         ((value & 0b01000000) >> 5) |
59         ((value & 0b10000000) >> 7));
60     BitSet(PORTA, EN);
61     BitClr(PORTA, EN);
62     BitClr(PORTA, RS);
63     Delay40us();
64     return;
65 }
66
67 void lcdString(char *message) {
68     int index = 0;
69     while (message[index] != 0) {
70         lcdData(message[index]);
71         index++;
72     }
73     return;
74 }
75
76 void lcdInt(int value) {
77     if (value < 0) {
78         value = value * (-1);
79         lcdData('-');
80     }
81     lcdData((value / 1000) % 10 + 48);
82     lcdData((value / 100) % 10 + 48);
83     lcdData((value / 10) % 10 + 48);
84     lcdData((value / 1) % 10 + 48);
85     return;
86 }

```

Figura 42: Source file para o funcionamento do LCD

## 7.10 LED

```
1 ifndef LED_H
2 define LED_H
3
4     void ledInit(void);
5
6     void ledON(char led);
7
8     void allLedON(void);
9
10    void ledOFF(char led);
11
12    void allLedOFF(void);
13
14 endif /* LED_H */
```

Figura 43: Header file para o funcionamento dos LEDs

```
1 include <xc.h>
2 include <pic18f4550.h>
3 include "defines.h"
4 include "LED.h"
5
6 void ledInit(void) {
7     TRISB = 0x00;
8     PORTB = 0xFF;
9     return;
10}
11
12 void ledON(char led) {
13     BitClr(PORTB, led);
14     return;
15}
16
17 void allLedON(void) {
18     PORTB = 0x00;
19     return;
20}
21
22 void ledOFF(char led) {
23     BitSet(PORTB, led);
24     return;
25}
26
27 void allLedOFF(void) {
28     PORTB = 0xFF;
29     return;
30}
```

Figura 44: Source file para o funcionamento dos LEDs

## 7.11 output

```

1 #ifndef OUTPUT_H
2 #define OUTPUT_H
3
4     void outputInit(void);
5
6     void outputPrint(int numberScreen, char language);
7
8 #endif /* OUTPUT_H */

```

Figura 45: Header file para o funcionamento da saída do microcontrolador

```

1 #include "output.h"
2 #include "defines.h"
3 #include "LCD.h"
4 #include "serial.h"
5 #include "stateMachine.h"
6 #include "variables.h"
7
8 typedef struct {
9     char message[NUMBER_LANGUAGES][MAX_CHARACTERS + 1];
10 } LCD_Screen;
11
12 /*
13 State stateMachine[] = {
14     {STATE_MENU, STATE_ALARMLOW, menuFunction}, // MENU
15     {STATE_MENU, STATE_MENU, alarmFunction}, // ALARMLOW
16     {STATE_MENU, STATE_ALARMHIGH, alarmFunction}, // ALARMHIGH
17     {STATE_MENU, STATE_ALARMHIGH, STATE_HOUR, languageFunction}, // LANGUAGE
18     {STATE_MENU, STATE_LANGUAGE, STATE_MINUTE, timeFunction}, // HOUR
19     {STATE_MENU, STATE_HOUR, STATE_SECOND, timeFunction}, // MINUTE
20     {STATE_MENU, STATE_MINUTE, STATE_MENU, timeFunction}, // SECOND
21     {STATE_MENU, STATE_WARNING, STATE_WARNING, warningFunction}, // WARNING
22 };
23 */
24
25 LCD_Screen const screens[STATE_END] = {
26     {"Bem-vindo!", "Welcome!", "Bienvenido!", "Bienvenue!", "Willkommen!", "Benvenuto!"}, // STATE_MENU
27     {"Alarme (inf)", "Alarm (low)", "Alarma (baja)", "Alarme (bas)", "Alarm (niedrig)", "Allarme (basso)"}, // STATE_ALARMLOW
28     {"Alarme (sup)", "Alarm (high)", "Alarma (alta)", "Alarme (haut)", "Alarm (hoch)", "Allarme (alto)"}, // STATE_ALARMHIGH
29     {"Alterar idioma", "Change language", "Cambiar idioma", "Changer langue", "Sprache ändern", "Cambia lingua"}, // STATE_LANGUAGE
30     {"Hora", "Hour", "Hora", "Heure", "Stunde", "Ora"}, // STATE_HOUR
31     {"Minutos", "Minutes", "Minutos", "Minutes", "Minuten", "Minuti"}, // STATE_MINUTE
32     {"Segundos", "Seconds", "Segundos", "Secondes", "Sekunden", "Secondi"}, // STATE_SECOND
33     {"CUIDADO!", "WARNING!", "ADVERTENCIA!", "ATTENTION!", "WARNUNG!", "ATTENZIONE!"}, // STATE_WARNING
34 };
35
36 static const char languages[NUMBER_LANGUAGES][16] = {
37     "Portugues",
38     "English",
39     "Espanol",
40     "Francais",
41     "Deutsch",
42     "Italiano",
43 };
44
45 void showDisplay(LCD_Screen *screen, int numberScreen, char language) {
46     lcdCommand(0x01);
47     lcdCommand(0x80);
48     lcdString(screen->message[language]);
49     lcdCommand(0xC0);
50     switch (numberScreen) {
51         case STATE_MENU:
52             lcdData(((getRTCHours() / 10) % 10) + 48);
53             lcdData((getRTCHours() % 10) + 48);
54             lcdData(':');
55             lcdData(((getRTCMinutes() / 10) % 10) + 48);
56             lcdData((getRTCMinutes() % 10) + 48);
57             lcdData(':');
58             lcdData(((getRTCSSeconds() / 10) % 10) + 48);
59             lcdData((getRTCSSeconds() % 10) + 48);
60             lcdString(" ");
61             break;
62         case STATE_ALARMLOW:
63             lcdData(['L']);
64             lcdInt(getAlarmLevelLow());
65             lcdData(['J']);
66             lcdData('.');
67             lcdInt((int)getValueADC());
68             lcdData(' ');
69             lcdInt(getAlarmLevelHigh());
70             lcdString(" ");
71             break;
72         case STATE_ALARMHIGH:
73             lcdInt(getAlarmLevelLow());
74             lcdData(['J']);
75             lcdInt((int)getValueADC());
76             lcdData('.');
77             lcdData(['L']);
78             lcdInt(getAlarmLevelHigh());
79             lcdData(['J']);
80             lcdString(" ");
81             break;
82         case STATE_LANGUAGE:
83             lcdData(' ');
84             lcdString(languages[getLanguage()]);
85             lcdData(' ');
86             break;
87     }
88 }

```

Figura 46: Source file para o funcionamento da saída do microcontrolador (parte 1)

```

87     case STATE_HOUR:
88         lcdData(['']);
89         lcdData(((getRTCHours() / 10) % 10) + 48);
90         lcdData((getRTCHours() % 10) + 48);
91         lcdData(['']);
92         lcdData([':']);
93         lcdData(((getRTCMinutes() / 10) % 10) + 48);
94         lcdData((getRTCMinutes() % 10) + 48);
95         lcdData(['']);
96         lcdData(((getRTCSecs() / 10) % 10) + 48);
97         lcdData((getRTCSecs() % 10) + 48);
98         lcdString("      ");
99         break;
100    case STATE_MINUTE:
101        lcdData(((getRTCHours() / 10) % 10) + 48);
102        lcdData((getRTCHours() % 10) + 48);
103        lcdData(['']);
104        lcdData(['']);
105        lcdData(((getRTCMinutes() / 10) % 10) + 48);
106        lcdData((getRTCMinutes() % 10) + 48);
107        lcdData(['']);
108        lcdData([':']);
109        lcdData(((getRTCSecs() / 10) % 10) + 48);
110        lcdData((getRTCSecs() % 10) + 48);
111        lcdString("      ");
112        break;
113    case STATE_SECOND:
114        lcdData(((getRTCHours() / 10) % 10) + 48);
115        lcdData((getRTCHours() % 10) + 48);
116        lcdData(['']);
117        lcdData(((getRTCMinutes() / 10) % 10) + 48);
118        lcdData((getRTCMinutes() % 10) + 48);
119        lcdData(['']);
120        lcdData([':']);
121        lcdData(((getRTCSecs() / 10) % 10) + 48);
122        lcdData((getRTCSecs() % 10) + 48);
123        lcdData(['']);
124        lcdString("      ");
125        break;
126    case STATE_WARNING:
127        lcdString("!!!!!!!");
128        break;
129    }
130 }
131 return;
132 }

133 void showSerial(LCD_Screen *screen, int numberScreen, char language) {
134     serialClear();
135     serialSendMessage(screen->message[language]);
136     switch (numberScreen) {
137         case STATE_MENU:
138             serialSendChar((getRTCHours() / 10) % 10 + 48);
139             serialSendChar((getRTCHours() % 10) + 48);
140             serialSendChar([':']);
141             serialSendChar(((getRTCMinutes() / 10) % 10) + 48);
142             serialSendChar((getRTCMinutes() % 10) + 48);
143             serialSendChar([':']);
144             serialSendChar(((getRTCSecs() / 10) % 10) + 48);
145             serialSendChar((getRTCSecs() % 10) + 48);
146             serialClear();
147             serialSendChar('\r');
148             serialSendChar('\n');
149             break;
150         case STATE_ALARMLOW:
151             serialSendChar([':']);
152             serialSendInt(getAlarmLevelLow(), 4);
153             serialSendChar([':']);
154             serialSendChar([':']);
155             serialSendInt((int)getValueADC(), 4);
156             serialSendChar([':']);
157             serialSendInt(getAlarmLevelHigh(), 4);
158             serialClear();
159             serialSendChar('\r');
160             serialSendChar('\n');
161             break;
162         case STATE_ALARMHIGH:
163             serialSendInt(getAlarmLevelLow(), 4);
164             serialSendChar([':']);
165             serialSendInt((int)getValueADC(), 4);
166             serialSendChar([':']);
167             serialSendChar([':']);
168             serialSendInt(getAlarmLevelHigh(), 4);
169             serialSendChar([':']);
170             serialClear();
171             serialSendChar('\r');
172             serialSendChar('\n');
173             break;
174         case STATE_LANGUAGE:
175             serialSendChar([':']);
176             serialSendMessage(languages[getLanguage()]);
177             serialSendChar([':']);
178             serialClear();
179             serialSendChar('\r');
180             serialSendChar('\n');
181             break;
182     }
183 }
```

Figura 47: Source file para o funcionamento da saída do microcontrolador (parte 2)

```

182     case STATE_HOUR:
183         serialSendChar('[');
184         serialSendChar(((getRTCHours() / 10) % 10) + 48);
185         serialSendChar(']');
186         serialSendChar(',');
187         serialSendChar('[');
188         serialSendChar(((getRTCMinutes() / 10) % 10) + 48);
189         serialSendChar((getRTCMinutes() % 10) + 48);
190         serialSendChar(',');
191         serialSendChar(((getRTCSeconds() / 10) % 10) + 48);
192         serialSendChar((getRTCSeconds() % 10) + 48);
193         serialClear();
194         serialSendChar('\r');
195         serialSendChar('\n');
196         break;
197     case STATE_MINUTE:
198         serialSendChar(((getRTCHours() / 10) % 10) + 48);
199         serialSendChar((getRTCHours() % 10) + 48);
200         serialSendChar(',');
201         serialSendChar(',');
202         serialSendChar(((getRTCMinutes() / 10) % 10) + 48);
203         serialSendChar((getRTCMinutes() % 10) + 48);
204         serialSendChar(',');
205         serialSendChar(',');
206         serialSendChar(((getRTCSeconds() / 10) % 10) + 48);
207         serialSendChar((getRTCSeconds() % 10) + 48);
208         serialClear();
209         serialSendChar('\r');
210         serialSendChar('\n');
211         break;
212     case STATE_SECOND:
213         serialSendChar(((getRTCHours() / 10) % 10) + 48);
214         serialSendChar((getRTCHours() % 10) + 48);
215         serialSendChar(',');
216         serialSendChar(((getRTCMinutes() / 10) % 10) + 48);
217         serialSendChar((getRTCMinutes() % 10) + 48);
218         serialSendChar(',');
219         serialSendChar(',');
220         serialSendChar(((getRTCSeconds() / 10) % 10) + 48);
221         serialSendChar((getRTCSeconds() % 10) + 48);
222         serialSendChar(',');
223         serialClear();
224         serialSendChar('\r');
225         serialSendChar('\n');
226         break;
227     case STATE_WARNING:
228         serialSendMessage("|||||");
229         serialClear();
230         serialSendChar('\r');
231         serialSendChar('\n');
232         break;
233     }
234 }
235 }
236
237 void outputInit(void) {
238     lcdInit();
239     serialInit();
240     return;
241 }
242
243 void outputPrint(int numberScreen, char language) {
244     LCD_Screen *screen = &screens[numberScreen];
245     showDisplay(screen, numberScreen, language);
246     showSerial(screen, numberScreen, language);
247     return;
248 }

```

Figura 48: Source file para o funcionamento da saída do microcontrolador (parte 3)

## 7.12 PWM

```

1 #ifndef PWM_H
2 #define PWM_H
3
4     void pwmInit(void);
5
6     void pwmSet1(unsigned char percentage);
7
8     void pwmSet2(unsigned char percentage);
9
10    void pwmFrequency(unsigned int frequency);
11
12 #endif /* PWM_H */

```

Figura 49: Header file para o funcionamento do PWM

```

1  #include <xc.h>
2  #include <pic18f4550.h>
3  #include "defines.h"
4  #include "PWM.h"
5
6  void pwmInit(void) {
7      BitClr(TRISC, 1);           // Set RC1 as output
8      BitClr(TRISC, 2);           // Set RC2 as output
9      T2CON |= 0b00000011;        // Set the prescaler of TIMER2 as 1:16
10     BitSet(T2CON, 2);
11     CCP1CON |= 0b00001100;     // Set CCP1 as PWM
12     CCP2CON |= 0b00001100;     // Set CCP2 as PWM
13     return;
14 }
15
16 void pwmSet1(unsigned char percentage) {
17     unsigned int value = ((unsigned int)percentage) *(PR2+1);
18     value = value / 25;
19     value &= 0x03ff;
20     CCPRL = (unsigned char)value >> 2;
21     CCP1CON |= (value & 0x0003) << 4;
22     return;
23 }
24
25 void pwmSet2(unsigned char percentage) {
26     unsigned int value = ((unsigned int)percentage) *(PR2+1);
27     value = value / 25;
28     value &= 0x03ff;
29     CCPRL = (unsigned char)value >> 2;
30     CCP2CON |= (value & 0x0003) << 4;
31     return;
32 }
33
34 void pwmFrequency(unsigned int frequency) {
35     PR2 = (unsigned char)(125000 / (frequency)) - 1;
36     return;
37 }
--
```

Figura 50: Source file para o funcionamento do PWM

## 7.13 RTC\_DS1307

```

1  ifndef RTC_DS1307_H
2  define RTC_DS1307_H
3
4  define SEC    0
5  define MIN   1
6  define HOUR  2
7  define WEEKDAY 3
8  define DAY   4
9  define MONTH 5
10 define YEAR  6
11
12 define getSeconds() (bcd2dec(RTC_DS1307_ReadData(SEC)& 0x7f))
13 define getMinutes() (bcd2dec(RTC_DS1307_ReadData(MIN)& 0x7f))
14 define getHours() (bcd2dec(RTC_DS1307_ReadData(HOUR)& 0x5f))
15 define getWeekDay() (bcd2dec(RTC_DS1307_ReadData(WEEKDAY)& 0x07))
16 define getDays() (bcd2dec(RTC_DS1307_ReadData(DAY)& 0x5f))
17 define getMonths() (bcd2dec(RTC_DS1307_ReadData(MONTH)& 0x3f))
18 define getYears() (bcd2dec(RTC_DS1307_ReadData(YEAR)& 0xff))
19
20 define setSeconds(v) (RTC_DS1307_WriteData((unsigned char)dec2bcd(v)|0x80,SEC))
21 define setMinutes(v) (RTC_DS1307_WriteData((unsigned char)dec2bcd(v),MIN))
22 define setHours(v) (RTC_DS1307_WriteData((unsigned char)dec2bcd(v),HOUR))
23 define setWeekDay(v) (RTC_DS1307_WriteData((unsigned char)dec2bcd(v),WEEKDAY))
24 define setDays(v) (RTC_DS1307_WriteData((unsigned char)dec2bcd(v),DAY))
25 define setMonths(v) (RTC_DS1307_WriteData((unsigned char)dec2bcd(v),MONTH))
26 define setYears(v) (RTC_DS1307_WriteData((unsigned char)dec2bcd(v),YEAR))
27
28 int dec2bcd(int value);
29
30 int bcd2dec(int value);
31
32 void RTC_DS1307_Init(void);
33
34 void RTC_DS1307_StartClock(void);
35
36 void RTC_DS1307_WriteData(unsigned char value, int address);
37
38 int RTC_DS1307_ReadData(int address);
39
40 endif /* RTC_DS1307_H */
```

Figura 51: Header file para o funcionamento do RTC

```

1  #include <xc.h>
2  #include <pic18f4550.h>
3  #include "config.h"
4  #include "defines.h"
5  #include "RTC_DS1307.h"
6  #include "I2C.h"
7
8  #define DS1307_CTRL_ID 0b11011110
9
10 #define I2C_WRITE 0
11 #define I2C_READ 1
12
13 int dec2bcd(int value) {
14     return ((value / 10 * 16) + (value % 10));
15 }
16
17 int bcd2dec(int value) {
18     return ((value / 16 * 10) + (value % 16));
19 }
20
21 void RTC_DS1307_Init(void) {
22     I2C_Init();
23     return;
24 }
25
26 void RTC_DS1307_StartClock(void) {
27     int seconds;
28     seconds = RTC_DS1307_ReadData(SEC);
29     RTC_DS1307_WriteData(0x7f & seconds, SEC);
30     return;
31 }
32
33 void RTC_DS1307_WriteData(unsigned char value, int address) {
34     I2C_WriteByte(1, 0, (unsigned char)(DS1307_CTRL_ID | I2C_WRITE));
35     I2C_WriteByte(0, 0, (unsigned char)address);
36     I2C_WriteByte(0, 1, (unsigned char)value);
37     return;
38 }
39
40 int RTC_DS1307_ReadData(int address) {
41     int result;
42     I2C_WriteByte(1, 0, (unsigned char)(DS1307_CTRL_ID | I2C_WRITE));
43     I2C_WriteByte(0, 0, (unsigned char)address);
44     I2C_WriteByte(1, 0, (unsigned char)(DS1307_CTRL_ID | I2C_READ));
45     result = I2C_ReadByte(1, 1);
46     return result;
47 }

```

Figura 52: Source file para o funcionamento do RTC

## 7.14 serial

```
1 #ifndef SERIAL_H
2 #define SERIAL_H
3
4     void serialInit(void);
5
6     void serialClear(void);
7
8     void serialSendChar(unsigned char character);
9
10    void serialSendMessage(char *message);
11
12    void serialSendInt(int value, int numberDigits);
13
14    unsigned char serialRead(void);
15
16 #endif /* SERIAL_H */
```

Figura 53: Header file para o funcionamento da comunicação serial

```
1 #include <xc.h>
2 #include <pic18f4550.h>
3 #include "defines.h"
4 #include "serial.h"
5
6 void serialInit(void) {
7     TXSTA = 0b00101100; // Configure the transmission of data by serial
8     RCSTA = 0b10010000; // Configure the receive of data by serial
9     BAUDCON = 0b00001000;
10    SPBRGH = 0b00000000;
11    SPBRG = 0b00100010;
12    BitSet(TRISC, 6);
13    BitSet(TRISC, 7);
14    return;
15 }
16
17 void serialClear() {
18     serialSendMessage("\r\n");
19     serialSendChar('\r');
20     serialSendChar('\n');
21     serialSendChar('\r');
22     return;
23 }
24
25 void serialSendChar(unsigned char character) {
26     while (!BitTst(PIR1, 4)); // Wait until busy
27     TXREG = character;
28     return;
29 }
30
31 void serialSendMessage(char *message) {
32     int index = 0;
33     while (message[index] != '\0') {
34         while (!BitTst(PIR1, 4)); // Wait until busy
35         TXREG = message[index];
36         index++;
37     }
38     return;
39 }
40
41 void serialSendInt(int value, int numberDigits) {
42     int index = 1;
43     int auxiliar;
44     for (auxiliar = 1; auxiliar < numberDigits; auxiliar++)
45         index *= 10;
46     if (value < 0) {
47         value *= (-1);
48         serialSendChar('-');
49     }
50     while(index > 0) {
51         serialSendChar((value / index) % 10 + 48);
52         index /= 10;
53     }
54     return;
55 }
56
57 unsigned char serialRead(void) {
58     char valueRead = 0;
59     if (BitTst(RCSTA, 1)) {
60         BitClr(RCSTA, 4);
61         BitSet(RCSTA, 4);
62     }
63     if (BitTst(PIR1, 5)) {
64         valueRead = RCREG;
65     }
66     return valueRead;
67 }
```

Figura 54: Source file para o funcionamento da comunicação serial

## 7.15 stateMachine

```

1  ifndef STATEMACHINE_H
2  define STATEMACHINE_H
3
4  enum {
5      STATE_MENU,
6      STATE_ALARMLOW,
7      STATE_ALARMHIGH,
8      STATE_LANGUAGE,
9      STATE_HOUR,
10     STATE_MINUTE,
11     STATE_SECOND,
12     STATE_WARNING,
13     STATE_END
14 };
15
16 void stateMachineInit(void);
17
18 void stateMachineLoop(void);
19
20 void menuFunction(const unsigned char event);
21
22 void alarmFunction(const unsigned char event);
23
24 void languageFunction(const unsigned char event);
25
26 void timeFunction(const unsigned char event);
27
28 void warningFunction(const unsigned char event);
29
30 endif /* STATEMACHINE_H */

```

Figura 55: Header file para o funcionamento da máquina de estados

```

1 #include "stateMachine.h"
2 #include "event.h"
3 #include "LED.h"
4 #include "output.h"
5 #include "variables.h"
6
7 typedef void (*Function)(const unsigned char event);
8
9 typedef struct {
10     unsigned char baseState;
11     unsigned char previousState;
12     unsigned char nextState;
13     Function function;
14 } State;
15
16 State stateMachine[] = {
17     {STATE_MENU, STATE_SECOND, STATE_ALARMLOW, menuFunction}, // MENU
18     {STATE_MENU, STATE_MENU, STATE_ALARMHIGH, alarmFunction}, // ALARMLOW
19     {STATE_MENU, STATE_ALARMLOW, STATE_LANGUAGE, alarmFunction}, // ALARMHIGH
20     {STATE_MENU, STATE_ALARMHIGH, STATE_LANGUAGE, languageFunction}, // LANGUAGE
21     {STATE_MENU, STATE_LANGUAGE, STATE_MINUTE, timeFunction}, // HOUR
22     {STATE_MENU, STATE_HOUR, STATE_SECOND, timeFunction}, // MINUTE
23     {STATE_MENU, STATE_MINUTE, STATE_MENU, timeFunction}, // SECOND
24     {STATE_MENU, STATE_WARNING, STATE_WARNING, warningFunction}, // WARNING
25 };
26
27 void stateMachineInit() {
28     setState(STATE_MENU);
29     return;
30 }
31
32 void stateMachineLoop() {
33     unsigned char event = eventRead();
34     allledOff();
35     if ((getState() >= STATE_MENU) & (getState() < STATE_END)) {
36         if (event == EV_ALARM) {
37             if (getState() != STATE_WARNING)
38                 setPreviousState(getState());
39             setState(STATE_WARNING);
40         }
41         State *currentState = &stateMachine[getState()];
42         currentState->function(event);
43         if (event == EV_BASE)
44             setState(currentState->baseState);
45         if (event == EV_LEFT)
46             setState(currentState->previousState);
47         if (event == EV_RIGHT)
48             setState(currentState->nextState);
49     }
50     outputPrint(getState(), getLanguage());
51     return;
52 }
53
54 void menuFunction(const unsigned char event) {
55     return;
56 }
57
58 void alarmFunction(const unsigned char event) {
59     switch (getState()) {
60         case STATE_ALARMLOW:
61             if (event == EV_UP):

```

Figura 56: Source file para o funcionamento da máquina de estados (parte 1)

```

62     setAlarmLevelLow(getAlarmLevelLow() + 1);
63     if (event == EV_DOWN)
64         setAlarmLevelLow(getAlarmLevelLow() - 1);
65     break;
66 case STATE_ALARMHIGH:
67     if (event == EV_UP)
68         setAlarmLevelHigh(getAlarmLevelHigh() + 1);
69     if (event == EV_DOWN)
70         setAlarmLevelHigh(getAlarmLevelHigh() - 1);
71     break;
72 }
73 return;
74 }
75
76 void languageFunction(const unsigned char event) {
77     if (event == EV_UP)
78         setLanguage(getLanguage() + 1);
79     if (event == EV_DOWN) {
80         setLanguage(getLanguage() - 1);
81     }
82     return;
83 }
84
85 void timeFunction(const unsigned char event) {
86     switch (getState()) {
87     case STATE_HOUR:
88         if (event == EV_UP)
89             setRTCHours(getRTCHours() + 1);
90         if (event == EV_DOWN)
91             setRTCHours(getRTCHours() - 1);
92         break;
93     case STATE_MINUTE:
94         if (event == EV_UP)
95             setRTCMinutes(getRTCMinutes() + 1);
96         if (event == EV_DOWN)
97             setRTCMinutes(getRTCMinutes() - 1);
98         break;
99     case STATE_SECOND:
100        if (event == EV_UP)
101            setRTCSecs(getRTCSecs() + 1);
102        if (event == EV_DOWN)
103            setRTCSecs(getRTCSecs() - 1);
104        break;
105    }
106    return;
107 }
108
109 void warningFunction(const unsigned char event) {
110     if (event != EV_ALARM)
111         setState(getPreviousState());
112     allledON();
113     return;
114 }

```

Figura 57: Source file para o funcionamento da máquina de estados (parte 2)

## 7.16 variables

```
1  ifndef VARIABLES_H
2  define VARIABLES_H
3
4      #include <xc.h>
5
6      void variablesInit(void);
7
8      char getState(void);
9
10     void setState(char newState);
11
12     unsigned char getPreviousState(void);
13
14     void setPreviousState(unsigned char state);
15
16     int getAlarmLevelHigh(void);
17
18     void setAlarmLevelHigh(int newAlarmLevelHigh);
19
20     int getAlarmLevelLow(void);
21
22     void setAlarmLevelLow(int newAlarmLevelLow);
23
24     char getLanguage(void);
25
26     void setLanguage(char newLanguage);
27
28     uint16_t getValueADC(void);
29
30     void setValueADC(uint16_t newValueADC);
31
32     void setValueADC(uint16_t newValueADC);
33
34     int getRTCHours(void);
35
36     void setRTCHours(int newHours);
37
38     int getRTCMinutes(void);
39
40     void setRTCMinutes(int newMinutes);
41
42     int getRTCSeconds(void);
43
44     void setRTCSeconds(int newSeconds);
45
46     char isStarted(void);
47
48     void setStarted(void);
49
50     int limitValue(int value, int limitInf, int limitSup);
51
52     unsigned char strcmp_local(const char *first, const char *second, unsigned char findex, unsigned char sindex);
53
54     void resetSerialBuffer(void);
55
56     void addSerialBuffer(char value);
57
58     void executeProtocol(void);
59
60 #endif /* VARIABLES_H */
```

Figura 58: Header file para o funcionamento das variáveis utilizadas

```

1  #include <xc.h>
2  #include <pic18f4550.h>
3  #include "defines.h"
4  #include "RTC_DS1307.h"
5  #include "serial.h"
6  #include "variables.h"
7
8  #define ALARM_LOW 0x22
9  #define ALARM_HIGH 0x24
10 #define LANGUAGE 0x26
11 #define PREVIOUS_STATE 0x28
12 #define STARTED 0x40
13
14 static char state;
15 static int alarmLevelHigh;
16 static int alarmLevelLow;
17 static uint16_t valueADC;
18
19 static char serialBuffer[16];
20 static char bufferPosition;
21
22 static const char alarmProtocol[4] = "/ALM";
23 static const char *languageProtocol[6] = {"/LPT", "/LEN", "/LEP", "/LFR", "/LDE", "/LIT"};
24
25 void variablesInit(void) {
26     setState(0);
27     bufferPosition = 0;
28     resetSerialBuffer();
29     if (isStarted()) {
30         setAlarmLevelLow(getAlarmLevelLow());
31         setAlarmLevelHigh(getAlarmLevelHigh());
32     } else {
33         setAlarmLevelLow(0);
34         setAlarmLevelHigh(1000);
35         setStarted();
36         setLanguage(0);
37     }
38     return;
39 }
40
41 char getState(void) {
42     return state;
43 }
44
45 void setState(char newState) {
46     state = newState;
47     return;
48 }
49
50 unsigned char getPreviousState(void) {
51     return (unsigned char)RTC_DS1307_ReadData(PREVIOUS_STATE);
52 }
53
54 void setPreviousState(unsigned char state) {
55     RTC_DS1307_WriteData(state, PREVIOUS_STATE);
56     return;
57 }

```

Figura 59: Source file para o funcionamento das variáveis utilizadas (parte 1)

```

59 int getAlarmLevelHigh(void) {
60     return ((RTC_DS1307_ReadData(ALARM_HIGH + 1) << 8) + RTC_DS1307_ReadData(ALARM_HIGH));
61 }
62
63
64 void setAlarmLevelHigh(int newAlarmLevelHigh) {
65     alarmLevelHigh = limitValue(newAlarmLevelHigh, MIN_ADC_VALUE, MAX_ADC_VALUE);
66     if (alarmLevelHigh > getAlarmLevelLow()) {
67         RTC_DS1307_WriteData(alarmLevelHigh & 0xFF, ALARM_HIGH);
68         RTC_DS1307_WriteData((alarmLevelHigh >> 8) & 0xFF, ALARM_HIGH + 1);
69     }
70     return;
71 }
72
73 int getAlarmLevelLow(void) {
74     return ((RTC_DS1307_ReadData(ALARM_LOW + 1) << 8) + RTC_DS1307_ReadData(ALARM_LOW));
75 }
76
77 void setAlarmLevelLow(int newAlarmLevelLow) {
78     alarmLevelLow = limitValue(newAlarmLevelLow, MIN_ADC_VALUE, MAX_ADC_VALUE);
79     if (alarmLevelLow < getAlarmLevelHigh()) {
80         RTC_DS1307_WriteData(alarmLevelLow & 0xFF, ALARM_LOW);
81         RTC_DS1307_WriteData((alarmLevelLow >> 8) & 0xFF, ALARM_LOW + 1);
82     }
83     return;
84 }
85
86 char getLanguage(void) {
87     return (RTC_DS1307_ReadData(LANGUAGE) % NUMBER_LANGUAGES);
88 }
89
90 void setLanguage(char newLanguage) {
91     if (getLanguage() == 0 && newLanguage == 0xFF) {
92         newLanguage = NUMBER_LANGUAGES - 1;
93     }
94     RTC_DS1307_WriteData(newLanguage % NUMBER_LANGUAGES, LANGUAGE);
95     return;
96 }
97
98 uint16_t getValueADC(void) {
99     return valueADC;
100 }
101
102 void setValueADC(uint16_t newValueADC) {
103     valueADC = (uint16_t)((float)newValueADC / 1023.0) * 1000;
104     return;
105 }
106
107 int getRTCHours(void) {
108     return getHours();
109 }
110
111 void setRTCHours(int newHours) {
112     if (!(getRTCHours() == 0 && newHours < getRTCHours()) || (getRTCHours() == 11 && getRTCHours() < newHours))
113         setHours(newHours);
114     return;
115 }

```

Figura 60: Source file para o funcionamento das variáveis utilizadas (parte 2)

```

147     int getRTCMinutes(void) {
148         return getMinutes();
149     }
150
151     void setRTCMinutes(int newMinutes) {
152         if (!(getRTCMinutes() == 0 && newMinutes < getRTCMinutes()) || (getRTCMinutes() == 59 && getRTCMinutes() < newMinutes))
153             setMinutes(newMinutes);
154         return;
155     }
156
157     int getRTCSeconds(void) {
158         return getSeconds();
159     }
160
161     void setRTCSeconds(int newSeconds) {
162         if (!(getRTCSeconds() == 0 && newSeconds < getRTCSeconds()) || (getRTCSeconds() == 59 && getRTCSeconds() < newSeconds))
163             setSeconds(newSeconds);
164         return;
165     }
166
167     char isStarted(void) {
168         return (RTC_DS1307_ReadData(STARTED) == 0x01);
169     }
170
171     void setStarted(void) {
172         RTC_DS1307_WriteData(0x01, STARTED);
173         return;
174     }
175
176     int limitValue(int value, int limitInf, int limitSup) {
177         return (value > limitSup) ? limitSup : (value < limitInf) ? limitInf : value;
178     }
179
180     unsigned char strcmp_local(const char *first, const char *second, unsigned char findex, unsigned char sindex) {
181         unsigned char index;
182         unsigned char value = 1;
183         for (index = findex; index <= sindex; index++) {
184             if (first[index] != second[index]) {
185                 value = 0;
186                 break;
187             }
188         }
189         return value;
190     }
191
192     void resetSerialBuffer(void) {
193         char index;
194         for (index = 0; index < 15; index++)
195             serialBuffer[index] = 48;
196         serialBuffer[15] = '\0';
197         bufferPosition = 0;
198         return;
199     }
200
201     void addSerialBuffer(char value) {
202         if (bufferPosition < 15)
203             if (value != '\r' && value != '\n' && value != '\0')
204                 serialBuffer[bufferPosition++] = value;

```

Figura 61: Source file para o funcionamento das variáveis utilizadas (parte 3)

```

174         serialBuffer[bufferPosition++] = value;
175     else
176         bufferPosition = 0;
177     serialClear();
178     serialSendMessage("Buffer: ");
179     serialSendMessage(serialBuffer);
180     serialClear();
181     serialSendChar('\r');
182     serialSendChar('\n');
183     return;
184 }
185
186 void executeProtocol(void) {
187     unsigned char index;
188     int value;
189     // LANGUAGE: /LXXX
190     if (serialBuffer[1] == 'L') {
191         for (index = 0; index < 6; index++)
192             if (strcmp_local(serialBuffer, languageProtocol[index], 2, 3)) {
193                 setlanguage(index);
194                 break;
195             }
196         return;
197     }
198     // ALARM: /ALMX[XXXX]
199     if (strcmp_local(serialBuffer, alarmProtocol, 0, 3)) {
200         value = ((serialBuffer[0] - 48) * 1000 + (serialBuffer[7] - 48) * 100 + (serialBuffer[8] - 48) * 10 + (serialBuffer[9] - 48) * 1);
201         if (serialBuffer[4] == '[' && serialBuffer[5] == ']' && serialBuffer[10] == ']')
202             setAlarmLevelLow(value);
203         if (serialBuffer[4] == '[' && serialBuffer[5] == ']' && serialBuffer[10] == ']')
204             setAlarmLevelHigh(value);
205         return;
206     }
207     // HOUR: /H:XXM:XXS:XX
208     if (serialBuffer[1] == 'H' && serialBuffer[2] == ':' && serialBuffer[5] == 'M' && serialBuffer[6] == ':' && serialBuffer[9] == 'S' && serialBuffer[10] == ':') {
209         value = ((serialBuffer[3] - 48) * 10 + serialBuffer[4] - 48);
210         setRTCHours(value);
211         value = ((serialBuffer[7] - 48) * 10 + serialBuffer[8] - 48);
212         setRTCMinutes(value);
213         value = ((serialBuffer[11] - 48) * 10 + serialBuffer[12] - 48);
214         setRTCSseconds(value);
215     }
216 }
217 }
218

```

Figura 62: Source file para o funcionamento das variáveis utilizadas (parte 4)

## 7.17 main

```
1  #include <xc.h>
2  #include <pic18f4550.h>
3  #include "config.h"
4  #include "defines.h"
5  #include "ADC.h"
6  #include "button.h"
7  #include "DAC_MCP4725.h"
8  #include "delay.h"
9  #include "I2C.h"
10 #include "LCD.h"
11 #include "LED.h"
12 #include "PWM.h"
13 #include "RTC_DS1307.h"
14 #include "serial.h"
15
16 #include "event.h"
17 #include "output.h"
18 #include "stateMachine.h"
19 #include "variables.h"
20
21 void initAllPeripheral(void) {
22     adcInit();
23     buttonInit();
24     lcdInit();
25     ledInit();
26     serialInit();
27     setRTCHours(0);
28     setRTCMinutes(0);
29     setRTCSeconds(0);
30     return;
31 }
32
33 void initAllFunctions(void) {
34     eventInit();
35     outputInit();
36     stateMachineInit();
37     variablesInit();
38     return;
39 }
40
41 void main(void) {
42     char index = 0;
43     initAllPeripheral();
44     initAllFunctions();
45     while (1) {
46         stateMachineLoop();
47         Delay100ms();
48         for (index = 0; index < 3; index++)
49             Delay10ms();
50     }
51     return;
52 }
```

Figura 63: Source file para a aplicação principal do projeto

# 8 Diagramas propostos

Nesta seção serão apresentados os diagramas de "classes" e estados propostos para o funcionamento da IHM.

## 8.1 Diagrama de classes

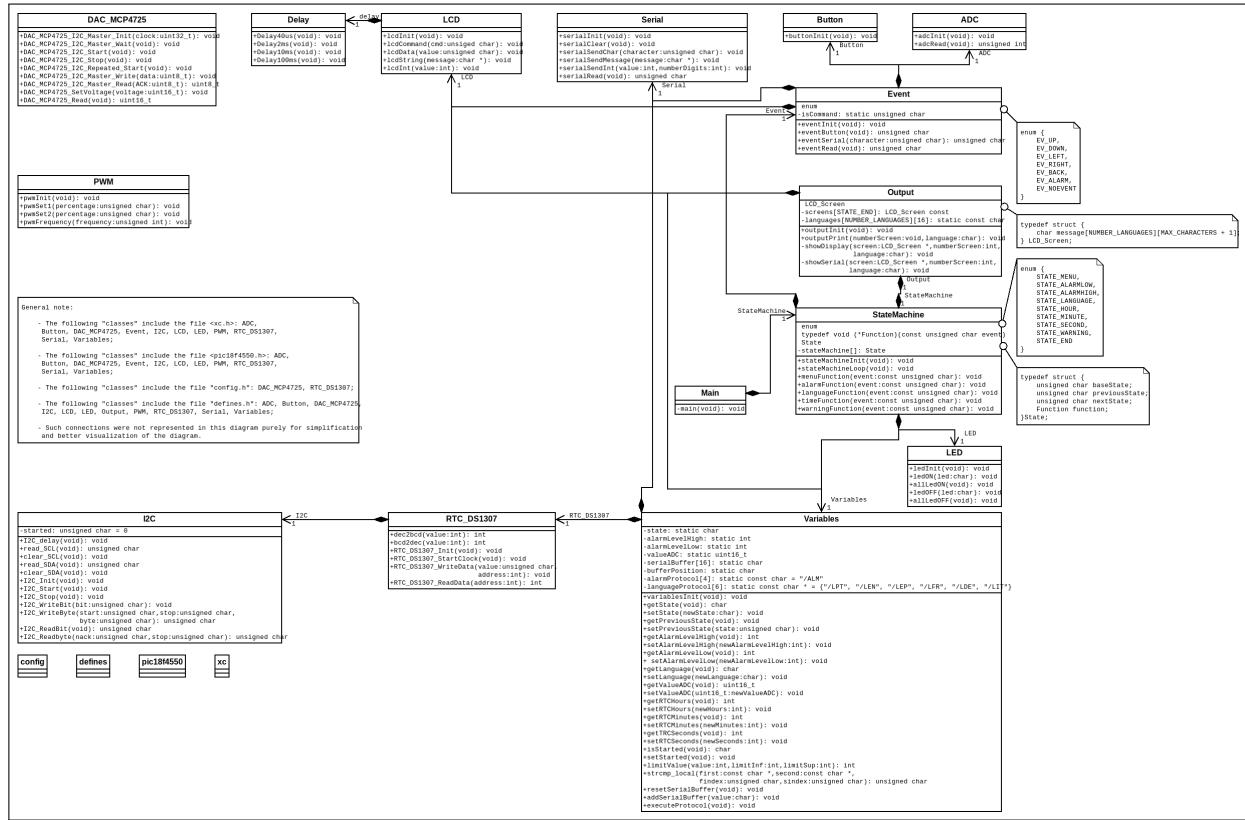


Figura 64: Diagrama de classes proposto para a aplicação

## 8.2 Diagrama de estados

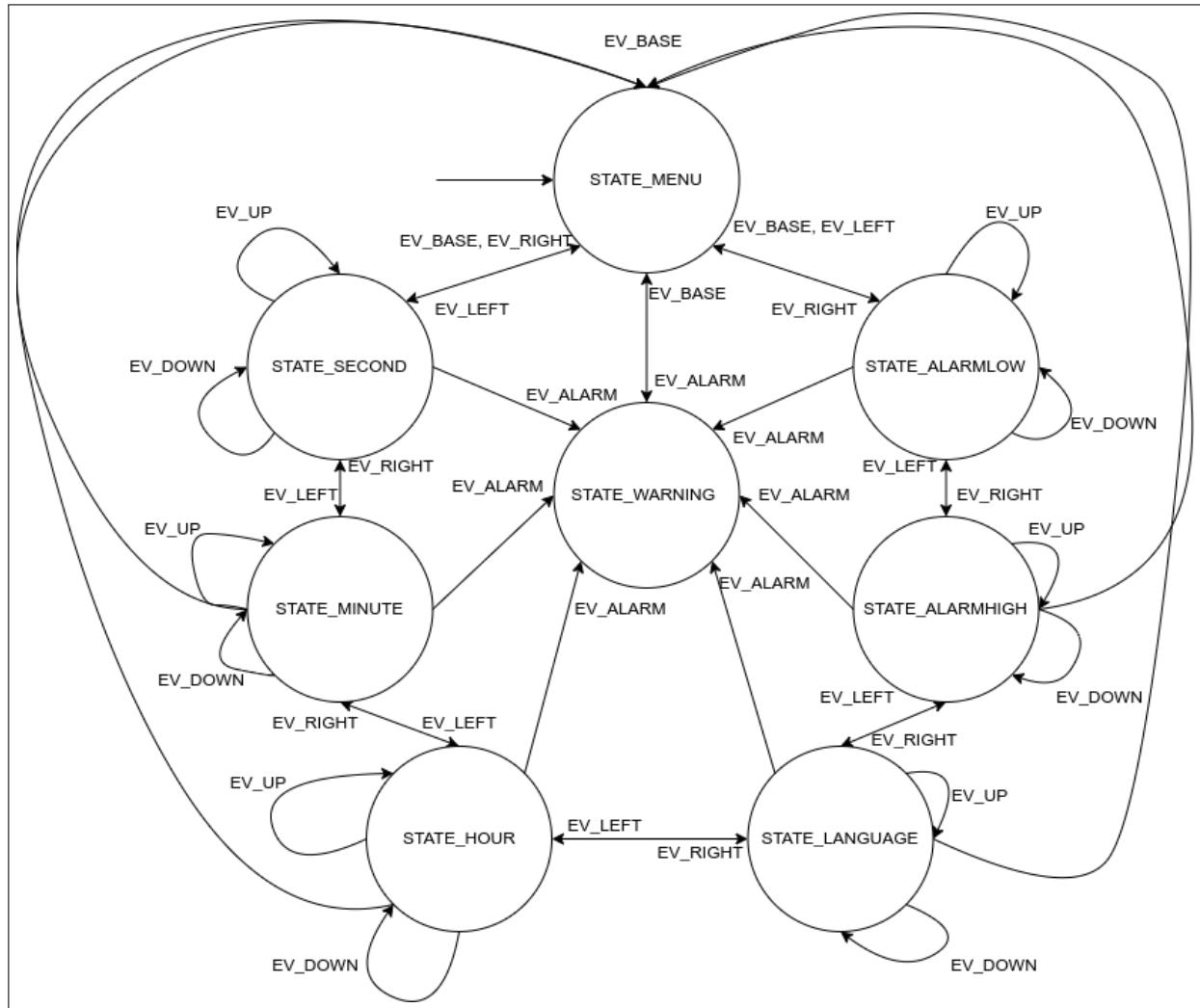


Figura 65: Diagrama de estados proposto para a aplicação