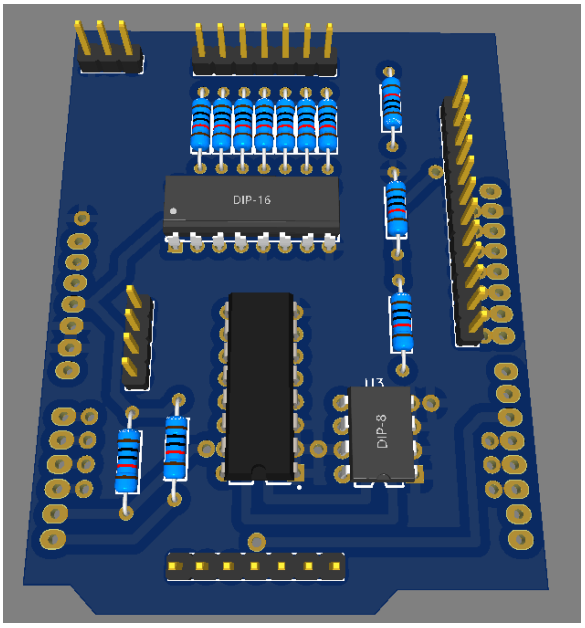
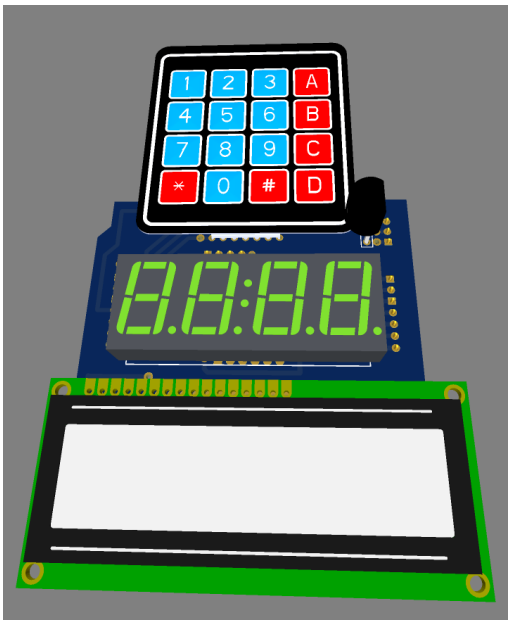
Arduino Datalogger Shield

Projetado e programado por Raphael do Espirito Santo Nascimento



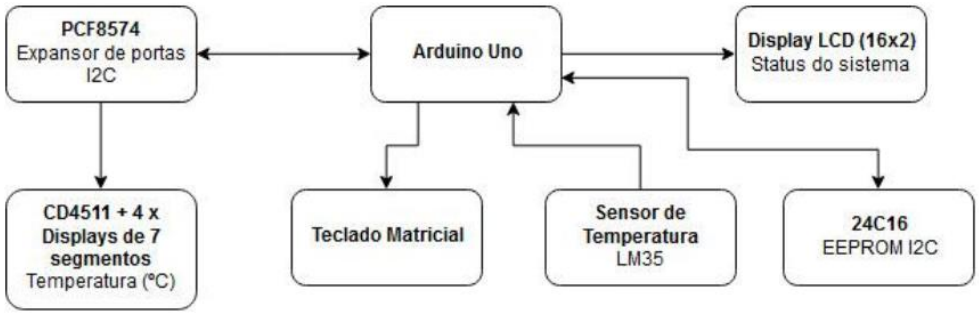


Características

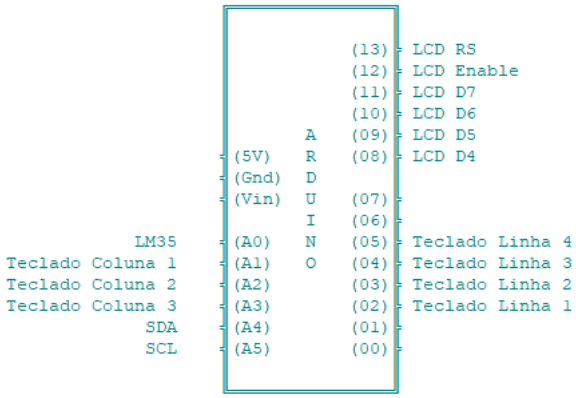
- Mede a temperatura através de um LM35 (datasheet)
- Guarda as informações em uma memória EEPROM da família 24C
- Display 4 dígitos de 7 segmentos controlado por decodificador BCD (CD4511) e um multiplicador de portas (PCF8574)
- Display LCD 16x02 controlado por barramento de 4 bits
- Controle de funções através de um teclado Matricial 3x4
- Envio dos dados por comunicação Serial

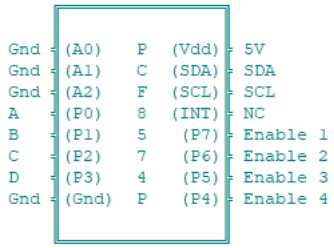
O Sistema

A figura abaixo mostra o diagrama de blocos do sistema:



As conexões, que podem ser conferidas também nos esquemáticos, foram:



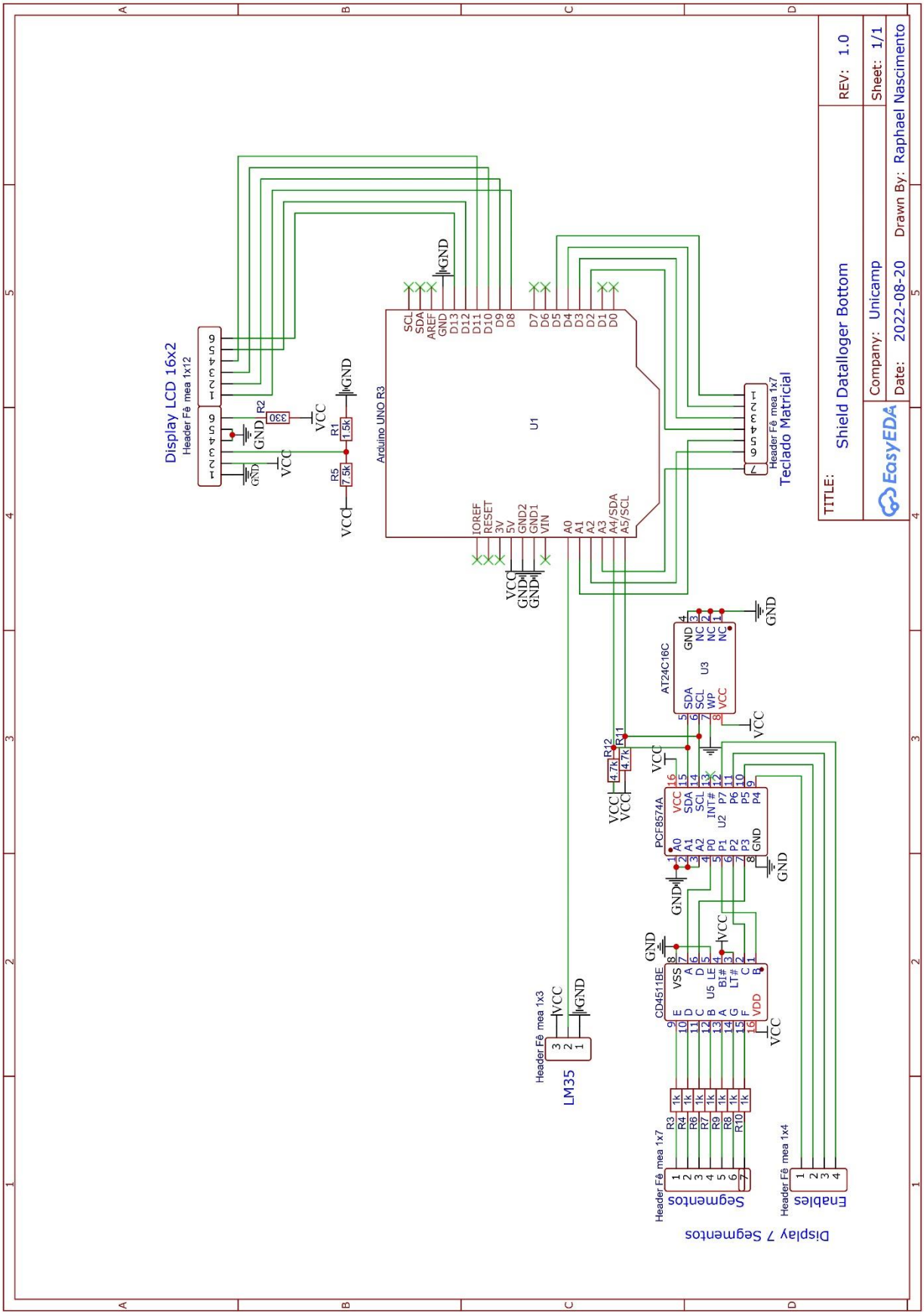


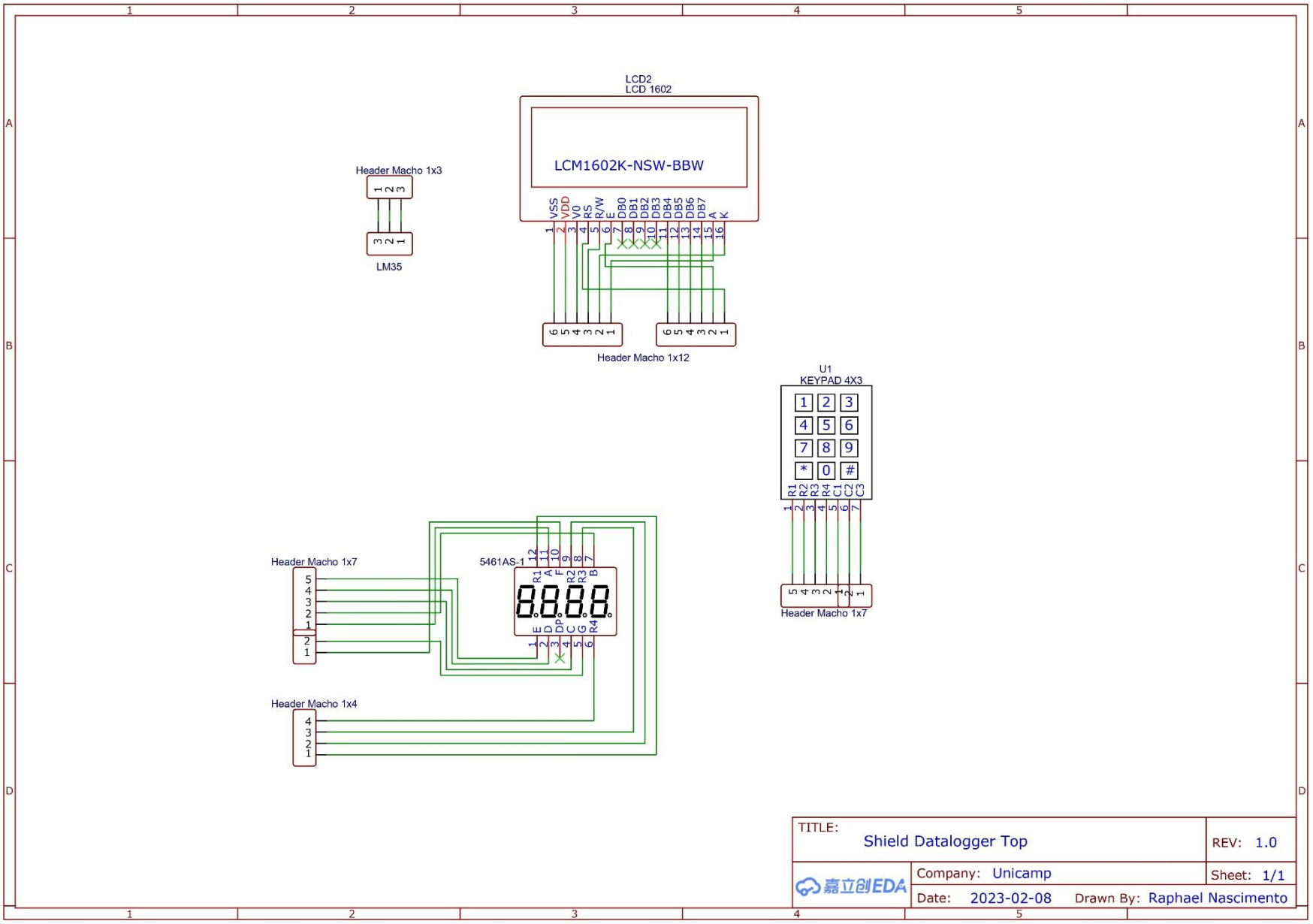
As funções existentes são:

1. Apaga toda a memória, com aviso no display (zera o contador de medidas armazenadas)
2. Mostra no display o número de dados gravados e o número de medições disponíveis
3. Inicia a coleta periódica, se houver espaço na memória
4. Finaliza a coleta periódica, exibindo quantas coletas foram feitas
5. Envia pela porta serial os dados coletados, mostra mensagem no display

Todas as funções devem ser confirmadas com a tecla '#' ou cancelada com a tecla '*'

Esquemáticos




LCD2
LCD 1602
LCM1602K-NSW-BBW
VSS VDD V0 RS R/W E DB0 DB1 DB2 DB3 DB4 DB5 DB6 DB7 A K
Header Macho 1x12
Header Macho 1x3
LM35
U1
KEYPAD 4X3
1 2 3
4 5 6
7 8 9
* 0 #
R1 R2 R3 R4 C1 C2 C3
Header Macho 1x7
5461AS-1
R1 A F R2 R3 B
E D DP C G R4
Header Macho 1x7
Header Macho 1x4
TITLE:
Shield Datalogger Top
REV: 1.0
Company: Unicamp
Sheet: 1/1
Date: 2023-02-08
Drawn By: Raphael Nascimento
嘉立创EDA

Materias

Para a montagem são necessários os seguintes itens:

07 - Resistor 1k Ohms
02 - Resistor 4.7k Ohms
01 - Resistor 220 Ohms
01 - Diodo
01 - CI CD4511
01 - CI PCF8574
01 - CI 224C16
02 - Soquete CI 16 pinos
01 - Soquete CI 8 pinos
55 - Pinos Macho (1 fileira)
01 - Pino Fêmea (1x3)
01 - Pino Fêmea (1x4)
02 - Pino Fêmea (1x7)
01 - Pino Fêmea (1x12)
01 - Teclado Matricial 4x3
01 - Display 7 segmentos Catodo Comum (4 dígitos)
01 - LM35
01 - Display LCD 16x02

Código

```
/*****************************************************************
*************************************************************

Projeto feito por Raphael Nascimento para a disciplina de EA076 -
Sistemas Embarcados, como PAD para o 1S2023

O programa simula um datalogger, medindo temperaturas a cada 2
segundos e salvando-as em uma memória EEPROM do tipo 24C16, com
capacidade para 2048 palavras de 8 bits, que são utilizadas aos pares,
sendo os primeiros 1023 pares ocupados por medidas de
temperatura e o último par guardará a quantidade de medidas feitas.
A aquisição da temperatura será feita através do LM35 e lida
analogicamente através da porta A0 do microcontrolador, sendo que a
cada 10 mV lido representa 1°C.
A todo momento, a ultima medição feita será exibida no conjunto de
display de 7 segmentos, onde seus segmentos estão conectados
no CI CD4511, que por sua vez, junto com os pinos de enable estão no
expansor de portas PCF8574. O display é ligado com um nível
baixo em suas portas de enable, por ser um display catodo comum.
As funções do datalogger podem ser acessadas por meio de um teclado
matricial e visualizadas através de um display LCD 16x02.


As funções existentes são:
      1 - Apaga toda a memória, com aviso no display (zera o contador
de medidas armazenadas)
      2 - Mostra no display o número de dados gravados e o número de
medições disponíveis
      3 - Inicia a coleta periódica, se houver espaço na memória
      4 - Finaliza a coleta periódica, exibindo quantas coletas foram
feitas
      5 - Envia pela porta serial os dados coletados, mostra mensagem
no display

Todas as funções devem ser confirmadas com a tecla '#' ou cancelada
com a tecla '*'


*****************************************************************
*************************************************************/

#include <Wire.h>
#include <LiquidCrystal.h>


//Variaveis relacionadas a medição de temperatura
float temperatura;
int contadorTemperatura;
unsigned char coletando;

//Variaveis relacionadas aos displays de 7 segmentos
unsigned char dezenaTemperatura;
unsigned char unidadeTemperatura;
unsigned char decimalTemperatura;
unsigned char centesimalTemperatura;
enum displays {dezena, unidade, decimal, centesimal} digitos;

//Variaveis relacionadas ao uso da EEPROM
```

```
#define endEEPROM 0x50
int quantOcupada;

//Variaveis relacionadas a execução das funções
enum funcoes {semFuncao, reset, status, start, stop, transferir,
escolherValores, enviarValores} funcao;

//Variaveis relacionadas a impressão dos valores pela serial
int digitosImpressao;
int impressao;

//Variaveis relacionadas ao teclado
char teclado [4][3] = {
  '1', '2', '3',
  '4', '5', '6',
  '7', '8', '9',
  '*', '0', '#'
};
char teclaReconhecida;
char tecla;

//Variaveis relacionadas ao display LCD
LiquidCrystal lcd_1(13, 12, 8, 9, 10, 11);


//FUNÇÕES EEPROM

void escreverEEPROM(int dado, int posicao){
      /************************************************************
************************************************************
      O objetivo desta função é de escrever um par de valores na
memória EEPROM
      Apesar da memória apresentar roll-over a cada 16 endereços, como
os valores são escitos aos pares é certeza que não ocorrerá
      este problema neste programa, portanto os valores são escritos
aos pares sem a preocupação de enviar o segundo endereço.

      O endereço da memória 24C16 possui 11 bits, portanto será
separado em dois bytes, onde o byte mais significativo irá conter
      somente 3 bits, que serão concatenados com o endereço do CI e o
byte menos significativo será enviado em seguida em uma
      operação de escrita.

      O dado a ser enviado também é dividido em dois bytes, que serão
enviados logo em seguida

      A transmissão então é encerrada e é garantido que irá ser
esperado o tempo mínimo de 5ms requerido pela memória antes da
      próxima operação de escrita
      ************************************************************
************************************************************/

      unsigned char endMSB = posicao >> 8;
      unsigned char endLSB = posicao & 0xFF;

      unsigned char dadoMSB = dado >> 8;
      unsigned char dadoLSB = dado & 0xFF;


      Wire.beginTransmission(endEEPROM | endMSB);
      Wire.write (endLSB);
```

```
      Wire.write (dadoMSB);
      Wire.write (dadoLSB);

      Wire.endTransmission();

      _delay_ms(5);
}

int lerEEPROM(int posicao){
      /*************************************************************
************************************************************
      Esta função serve para ler dois bytes consecutivos da memória e
concatena-los em uma única variável

      A operação de leitura é feita com o processo de uma escrita
simulada (dummy write), onde é enviado o endereço da EEPROM
      concatenado com os 3 bits mais significativos do endereço de
leitura, seguido o byte menos significativo e do fim da
      transmissão. Então é reiniciada a transmissão como leitura
      *************************************************************
************************************************************/



      int leitura = 0xFFFF;


      unsigned char endMSB = posicao >> 8;
      unsigned char endLSB = posicao & 0xFF;


      Wire.beginTransmission(endEEPROM | endMSB);
      Wire.write (endLSB);

      Wire.endTransmission();

      Wire.requestFrom((endEEPROM | endMSB), 2);

      leitura = Wire.read() << 8;
      leitura |= Wire.read();

      return leitura;
}


//FUNÇÃO DO DISPLAY DE 7 SEGMENTOS
void mostrarDigitos(){
      /*************************************************************
************************************************************
      Esta função exibe os 4 dígitos enviados a ela nos displays de 7
segmentos, multiplexando a exibição no tempo.
      Cada vez que o programa passa por essa função, um dos dígitos é
exibido e os outros são apagados, de acordo com a informação
      enviada para o expansor de portas PCF8574
      O nibble mais significativo corresponde aos sinais de enable do
conjunto de displays e o nibble menos significativo
      corresponde ao valor que é enviado ao CD4511
      *************************************************************
************************************************************/
```

```
	Wire.beginTransmission(0x20);
	switch (digitos){
		default:
		case dezena:
			Wire.write(dezenaTemperatura | (0x07 << 4));
			digitos = unidade;
			break;
		case unidade:
			Wire.write(unidadeTemperatura | (0x0B << 4));
			digitos = decimal;
			break;
		case decimal:
			Wire.write(decimalTemperatura | (0x0D << 4));
			digitos = centesimal;
			break;
		case centesimal:
			Wire.write(centesimalTemperatura | (0x0E << 4));
			digitos = dezena;
			break;

	}
	Wire.endTransmission();

}


//FUNÇÕES DO TECLADO MATRICIAL
char varreduraTeclado(){
	/*************************************************************
************************************************************
	A função serve para identificar se alguma tecla está sendo
pressionada, por meio de varredura
	As linhas estão conectadas em portas definidas como saída, sendo
que em cada loop somente uma delas será colocada em nível
	baixo, enquanto que a restante continua em nível alto
	As colunas estão conectadas em portas definidas como entrada,
com o resistor de pull-up ativo
	Deste modo, quando uma tecla é pressionada, se sua linha estiver
em nível baixo o valor baixo aparecerá na coluna
	correspondente, que será lido pelo loop aninhado.
	Quando a primeira tecla pressionada for encontrada, o caractere
existente na tecla, que foi definido na variavel 'teclado'
	será retornado pela função, que encerrará sua execução.
	Caso nenhuma tecla esteja pressionada, o valor de '-1' será
enviado
	*************************************************************
************************************************************/

	for (char linhas = 2; linhas <= 5; linhas++){
		PORTD |= 0x3C;
		PORTD &= ~(1 << linhas);

		for (char colunas = 1; colunas <= 3; colunas++){
			if (((PINC & 0x0E) & (1 << colunas)) == 0) {
				return teclado[linhas-2][colunas-1];
			}
		}
	}

	return -1;
}
```

```
void verificarTeclado(){
      /*****************************************************************
*************************************************************
      Esta função serve para garantir que a função seja executada uma
única vez, mesmo com o pressionamento prolongado de uma tecla
      A variavel 'teclaAtual' guarda a ultima varredura do teclado,
que caso seja válida (diferente de '-1') e não tenha sido
      tratada ainda ('teclaReconhecida' igual a 0) passará o seu valor
para uma variavel global, para ser tratada em outra função
      Além disso, função atual seja a de escolher valores, o que
representa a escolha de dados a ser enviado pela porta serial,
      o valor digitado será exibido no display LCD e guardado em uma
variavel para impressao após o botão de confirmar
      Caso a varredura retorne um valor não válido é reiniciada a
variável de tratamento para esperar pela próxima tecla
      *****************************************************************
*************************************************************/

      char teclaAtual = varreduraTeclado();


      if (teclaAtual != -1 && teclaReconhecida == 0){
            teclaReconhecida = 1;
            tecla = teclaAtual;
            if (funcao == escolherValores){
                  if (digitosImpressao < 4 && tecla != '#'){
                        lcd_1.print(tecla);
                        impressao = impressao*10 + int(tecla) - 48;
                        digitosImpressao++;
                  }
            }
      }
      else if (teclaAtual == -1){
            teclaReconhecida = 0;
    }
}


//FUNÇÕES DATALOGGER
/*****************************************************************
*************************************************************
Nesta funções só é trocada a mensagem exibida no display LCD, sendo
que a função só será executada após a tecla de confirma
A variavel da função é atualizada dependendo da função
No caso da função de transferir pela os dados pela serial, como os
valores são adicionados por outra função, é resetada a tecla
manualmente, para esperar o usuário digitar a quantidade desejada
*****************************************************************
*************************************************************/
void funcaoReset(){
      lcd_1.clear();
      lcd_1.print("Apagar Memoria?");
      lcd_1.setCursor(0, 1);
      lcd_1.print("*/# - Nao/Sim");

      funcao = reset;
}

void funcaoStatus(){
      lcd_1.clear();
```

```
      lcd_1.print("Mostrar Status?");
      lcd_1.setCursor(0, 1);
      lcd_1.print("*/# - Nao/Sim");

      funcao = status;
}

void funcaoStart(){
      lcd_1.clear();
      lcd_1.print("Iniciar coleta?");
      lcd_1.setCursor(0, 1);
      lcd_1.print("*/# - Nao/Sim");

      funcao = start;
}

void funcaoStop(){
      lcd_1.clear();
      lcd_1.print("Terminar coleta?");
      lcd_1.setCursor(0, 1);
      lcd_1.print("*/# - Nao/Sim");

      funcao = stop;
}

void funcaoTransf(){
      lcd_1.clear();
      lcd_1.print("Transf. dados?");
      lcd_1.setCursor(0, 1);
      lcd_1.print("*/# - Nao/Sim");

      funcao = transferir;
      tecla = -1;
}

void cancela(){
      lcd_1.clear();
      lcd_1.print("Cancelado!");
      lcd_1.setCursor(0, 1);
      lcd_1.print("Escolha a funcao");

      digitosImpressao = 0;
      impressao = 0;
      funcao = semFuncao;
}


void funcaoImprimir(){
      /************************************************************
************************************************************
      Para que o programa continue rodando suas atividades
paralelamente ao envio dos dados pela serial não foi feito um loop,
      enviando somente um dado a cada vez que a função é chamada
      A variavel 'digitosImpressao' é reutilizada para contar quantos
valores já foram enviados
      Ao final da impressão os valores são zerados para serem
utilizados novamente em outra impressão, quando houver necessidade
      ************************************************************
************************************************************/
      if (digitosImpressao < impressao){
            float temp = lerEEPROM(digitosImpressao << 1);
```

```
            Serial.println(temp/100);

            digitosImpressao++;

            return;
      }

      digitosImpressao = 0;
      impressao = 0;
      funcao = semFuncao;
}

void confirma(){
      /*****************************************************************
*************************************************************
      Durante a função de confirmação que serão feitos os comandos, de
acordo com a sua função, sendo acrescido dois estados extras
      para a função de envio pela serial
      Em todas as funções as mensagens mostradas no display LCD serão
atualizadas conforme necessário

      Na função de reset será zerado as duas ultimas posições da
memória, que correspondem a quantidade de dados salvos, além de
      zerar a variavel que controla a quantidade de dados

      Na função de status é mostrado a quantidade de dados gravados e
a quantidade disponível para gravação

      Na terceira função, a coleta só será iniciada caso exista espaço
na memória, ativando uma flag para que outra função possa
      realizar a gravação

      A função de parar a gravação irá desativar a flag

      A função de impressão é dividida para esperar a quantidade
desejada do usuário e então o momento da impressão, onde só será
      impresso o menor valor entre a quantidade gravada e a quantidade
pedida pelo usuário, com o aviso caso o valor pedido
      ultrapasse a quantidade gravada
      Durante a transmissão é deixada a variável 'funcao' em 7 para
não aceitar outros comandos
      *****************************************************************
*************************************************************/
      switch (funcao){
            case 1:
                  escreverEEPROM(0x0000, 0x3FE);
                  quantOcupada = 0;

                  lcd_1.clear();
                  lcd_1.print("Memoria Apagada!");
                  lcd_1.setCursor(0, 1);
                  lcd_1.print("Disponivel: 1023");

                  funcao = semFuncao;
                  break;

            case 2:
                  lcd_1.clear();
                  lcd_1.print("Gravado:    ");
                  lcd_1.print(quantOcupada);
                  lcd_1.setCursor(0, 1);
```

```
        lcd_1.print("Disponivel: ");
        lcd_1.print(1023 - quantOcupada);

        funcao = semFuncao;
        break;

case 3:
        if (quantOcupada >= 1023) {
                lcd_1.clear();
                lcd_1.print("Memoria Cheia");
                lcd_1.setCursor(0, 1);
                lcd_1.print("Limpe a Memoria");

                funcao = semFuncao;
                break;
        }
        coletando = 1;

        lcd_1.clear();
        lcd_1.print("Coleta Iniciada!");
        lcd_1.setCursor(0, 1);
        lcd_1.print("Gravando Memoria");

        funcao = semFuncao;
        break;

case 4:
        coletando = 0;

        lcd_1.clear();
        lcd_1.print("Fim da coleta!");
        lcd_1.setCursor(0, 1);
        lcd_1.print("No DADOS: ");
        lcd_1.print(quantOcupada);

        funcao = semFuncao;
        break;

case 5:
        lcd_1.clear();
        lcd_1.print("Transf. dados:");
        lcd_1.setCursor(0, 1);
        lcd_1.print("Qntd dados: ");

        funcao = escolherValores;
        tecla = -1;
        break;

case 6:
        if (impressao > quantOcupada){
                impressao = quantOcupada;

                lcd_1.clear();
                lcd_1.print("Qnt > gravado");
                lcd_1.setCursor(0, 1);
                lcd_1.print("Imprimindo: ");
                lcd_1.print(impressao);
        }
        else {
                lcd_1.clear();
                lcd_1.print("Imprimindo: ");
```

```
                        lcd_1.print(impressao);
                  }
                  digitosImpressao = 0;

                  funcao = enviarValores;
                  break;
      }
}

void realizarFuncao(){
      /*************************************************************
************************************************************
      Esta função serve somente para direcionar o programa dependendo
da função escolhida pelo usuário, sendo que só é possivel
      escolher uma função caso nehuma outra esteja em execução e só é
possível confirmar ou cancelar durante uma função
      *************************************************************
************************************************************/
      if (funcao == 0){
            switch (tecla){
                  case '1':
                        funcaoReset();
                        break;
                  case '2':
                        funcaoStatus();
                        break;
                  case '3':
                        funcaoStart();
                        break;
                  case '4':
                        funcaoStop();
                        break;
                  case '5':
                        funcaoTransf();
                        break;
            }
      }
      else {
            if (tecla == '#'){
                  confirma();
            }
            else if (tecla == '*'){
                  cancela();
            }
      }
}


void converterTemperatura(int temp){
      /*************************************************************
************************************************************
      Os quatro digitos do valor da temperatura são separados em
variaveis separadas, para exibição no display de 7 segmentos
      *************************************************************
************************************************************/

      dezenaTemperatura = temp/1000;

      temp -= 1000*dezenaTemperatura;
      unidadeTemperatura = temp/100;
```

```
      temp -= 100*unidadeTemperatura;
      decimalTemperatura = temp/10;

      centesimalTemperatura = temp%10;
}

void medirTemperatura(){
      /*****************************************************************
*****************************************************
      O valor lido pelo pino analógico varia entre 0 (0V) e 1023 (5V),
logo para encontrar o valor da tensão na porta devemos
      multiplicar o valor lido por 5/1023
      Como a resolução do sensor de temperatura é de 10mV/°C, para
descobrir a temperatura devemos multiplicar a tensão por 100
      Para guardar e exibir o valor da temperatura com uma precisão de
centésimo de grau, é multiplicado o valor da temperatura
      novamente por 100 e então, para aumentar a velocidade das
contas, todas as constantes foram resumidas a uma só:
            ( A0 * 5 * 100 * 100 ) / 1023 = 48.8759

      A temperatura então é convertida para ser exibida nos displays
de 7 segmentos e, caso a função de coleta periódica esteja
      ativa, o valor é guardado na memória
      Caso a memória atinja sua ocupação máxima, a coleta é
finalizada, com uma mensagem sendo exibida no display LCD
      *****************************************************************
*****************************************************/


      temperatura = analogRead(A0) * 48.8759;

      converterTemperatura(temperatura);

      if (coletando){
            escreverEEPROM(temperatura, quantOcupada << 1);
            quantOcupada++;
            escreverEEPROM(quantOcupada, 0x3FE);

            if (quantOcupada >= 1023){
                  lcd_1.clear();
                  lcd_1.print("Memoria Cheia");
                  lcd_1.setCursor(0, 1);
                  lcd_1.print("Coleta Terminada");

                  coletando = 0;
                  funcao = semFuncao;
            }
      }
}


//INTERRUPÇÕES
ISR(TIMER0_COMPA_vect){
      /*****************************************************************
*****************************************************
      Interrupção do temporizador 0 - Acontece a cada 4ms

      Uma variavel é incrementada para medir o tempo entre cada
medição da temperatura
      *****************************************************************
*****************************************************/
```

```
	contadorTemperatura++;
}


//FUNÇÕES DE CONFIGURAÇÕES
void setupGPIO(){
	/*************************************************************
************************************************************
	As portas referentes ao teclado matricial são definidas como
saída (A1, A2, A3) ou entrada com pull-up (2, 3, 4, 5)
	As portas do display LCD não são alteradas aqui, pois são
alteradas dentro da biblioteca LiquidCrystal
	*************************************************************
************************************************************/
	DDRC &= 0xF0;
	PORTC |= 0x0E;

	DDRD |= 0x3C;
}

void setupInicial(){


	//Variaveis relacionadas a medição de temperatura
	temperatura = analogRead(A0) * 48.8759;
	contadorTemperatura = 0;
	coletando = 0;

	//Variaveis relacionadas aos displays de 7 segmentos
	converterTemperatura(temperatura);
		//dezenaTemperatura;
		//unidadeTemperatura;
		//decimalTemperatura;
		//centesimalTemperatura;
	digitos = 0;

	//Variaveis relacionadas ao uso da EEPROM
	quantOcupada = lerEEPROM(0x3FE);

	//Variaveis relacionadas a execução das funções
	funcao = semFuncao;

	//Variaveis relacionadas a impressão dos valores pela serial
	digitosImpressao = 0;
	impressao = 0;

	//Variaveis relacionadas ao teclado
	teclaReconhecida = 0;
	tecla = -1;

	//Variaveis relacionadas ao display LCD
	lcd_1.print("Bem Vindo!");
	lcd_1.setCursor(0, 1);
	lcd_1.print("Escolha a funcao");
}


void setupTimer(){
	/*************************************************************
************************************************************
	Configuracao do temporizador 0 (8 bits) para gerar interrupcoes
```

```
periodicas no modo Clear Timer on Compare Match (CTC)

      Relogio = 16e6
      Prescaler = 256
      Faixa = 250 (Contagem de 0 a OCR0A = 249)
      Intervalo entre interrupcoes: (Prescaler/Relogio)*Faixa =
(256/16e6)*(249+1) = 4 ms
      ************************************************************
************************************************************/

      /************************************************************
************************************************************
      TCCR0B - Timer/Counter Control Register B
```

FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00
0	0	-	-	0	1	0	0

```
      FOC0A: Force Output Compare A
      FOC0B: Force Output Compare B
      Desabilitados para compatibilidade com o modo CTC
      WGM02: Waveform Generation Mode
      Faz parte da configuração do modo CTC com TOP = OCR0A (0 1 0)
      CS02:0: Clock Select
      Prescaler definido em 256
      ************************************************************
************************************************************/
      TCCR0B = 0x04;

      /************************************************************
      TCCR0A - Timer/Counter Control Register A
```

COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00
0	0	0	0	-	-	1	0

```
      COM0A1:0: Compare Match Output A Mode
      COM0B1:0: Compare Match Output B Mode
      OC0A e OC0B desconectados, temporizador não gerará formas de
onda
      WGM01:0: Waveform Generation Mode
      Modo CTC com TOP = OCR0A (0 1 0)
      ************************************************************/
      TCCR0A = 0x02;

      //OCR0A - Output Compare Register A
      //Define a faixa de contagem
      OCR0A = 249;

      /************************************************************
************************************************************
      TIMSK0 - Timer/Counter Interrupt Mask Register
```

-	-	-	-	-	OCIE0B	OCIE0A	TOIE0
-	-	-	-	-	0	1	0

```
      OCIE0B: Timer/Counter Output Compare Match B Interrupt Enable
      Interrupção de comparação com o OCR0B desabilitada
      OCIE0A: Timer/Counter0 Output Compare Match A Interrupt Enable
      Interrupção de comparação com o OCR0A habilitada, para obter o
período de 25us
      TOIE0: Timer/Counter0 Overflow Interrupt Enable
      Interupção por overflow desabilitada
      ************************************************************
************************************************************/
      TIMSK0 = 0x02;

}



void setup() {

      /************************************************************
************************************************************
      Para o setup, foi modularizado diferentes outros setups para as
diferentes funções do programa:
      Timer   - Configurações do temporizador 0, utilizado para as
bases de tempo do programa
      GPIO  - Configuração das portas de entrada e saída, utilizadas
para as conexões externas

      Inicial - Configuração dos estados iniciais dos registradores e
variáveis usadas

      Foi utilizado a função cli() antes dos setups para garantir que
as interrupções estejam desabilitadas, evitando possíveis
      problemas durante a configuração, porém a inicialização das
variáveis foi deixada depois da ativação das interrupções pois a
      biblioteca wire.h utiliza as interrupções
      ************************************************************
************************************************************/

      cli();

      Serial.begin(9600);
      Wire.begin();
      lcd_1.begin(16, 2);

      setupTimer();
      setupGPIO();

      sei();

      setupInicial();
}



//FUNÇÃO PRINCIPAL
void loop () {
      if (contadorTemperatura >= 500){
            contadorTemperatura = 0;
            medirTemperatura();
      }
```

```
        mostrarDigitos();

        verificarTeclado();

        realizarFuncao();

        if (funcao == enviarValores){
              funcaoImprimir();
        }
}
```