UNIVERSIDADE ESTADUAL DE CAMPINAS – UNICAMP

FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO – FEEC

EA076 – Turma U
Laboratório de Sistemas Embarcados

PROJETO 3

Gravador de Dados Ambientais (*datalogger*)

GUILHERME AUGUSTO AMORIM TERRELL **RA: 168899**
WELLTER MOMPEAN SOZIN **RA: 188625**
Prof.º ERIC ROHMER

Campinas
2022

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

Este projeto consiste no desenvolvimento e implementação de hardware e software de um *datalogger* com o intuito de ser um gravador de dados ambientais, nesse caso, de temperaturas do ambiente.

O sistema é composto por diferentes módulos: temos o teclado matricial que é o módulo utilizado pelo usuário para a inserção de comandos no sistema do *datalogger*; o de aquisição de dados de temperatura; o de gravação desses dados; o módulo para exibição do comportamento do sistema através de um display LCD; e, por fim, o módulo para exibição em tempo real das temperaturas medidas e coletadas. Conforme exibido no diagrama de blocos da Figura 1, todos esses módulos se conectam ao Arduino Uno onde é implementado o software para o controle e gerenciamento dos mesmos

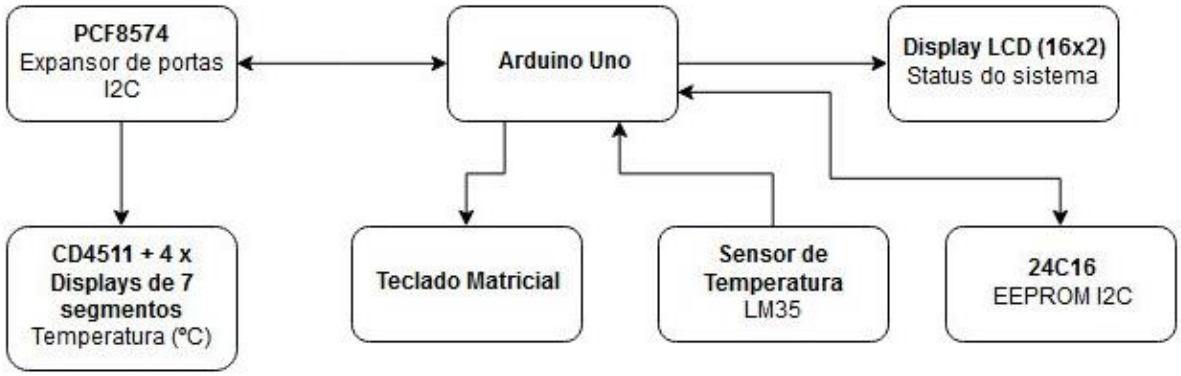


Figura 1: Módulos que compõem o sistema do *datalogger*.

Antes de entrarmos detalhadamente em cada um dos módulos, será apresentada uma visão geral do sistema e de como se dá seu funcionamento.

O software do *datalogger* conta com 5 comandos disponíveis que são requisitados e ativados através do teclado matricial de modo a ser possível o controle do sistema em campo sem a necessidade de um computador. Para requisitar um comando, o usuário deve digitar o número do mesmo seguido da tecla “#” para sua confirmação. No entanto, caso o usuário pressione uma tecla, mas deseja cancelar esse comando, basta pressionar “*”. O display LCD exibe o status do sistema em cada comando.

A Tabela 1 apresenta as teclas relativas a cada comando, sua função e a descrição da funcionalidade de cada um. Os dados coletados e armazenados no comando 3 são os valores de temperatura ambiente, com resolução de décimo de grau e com periodicidade de coleta de 2 segundos.

Tabela 1: Comandos disponíveis para controle do *datalogger*.

COMANDO	FUNÇÃO	DESCRIÇÃO
1	Reset	Apaga toda a memória, com aviso no display (zera o contador de medidas armazenadas)
2	Status	Mostra no display o número de dados gravados e o número de medições disponíveis
3	Inicia coleta periódica	Mostra mensagem no display
4	Finaliza coleta periódica	Mostra mensagem no display
5	Transferência de dados	Envia pela porta serial os dados coletados, mostra mensagem no display.

A seguir é feita uma descrição mais detalhada de cada um dos comandos disponíveis.

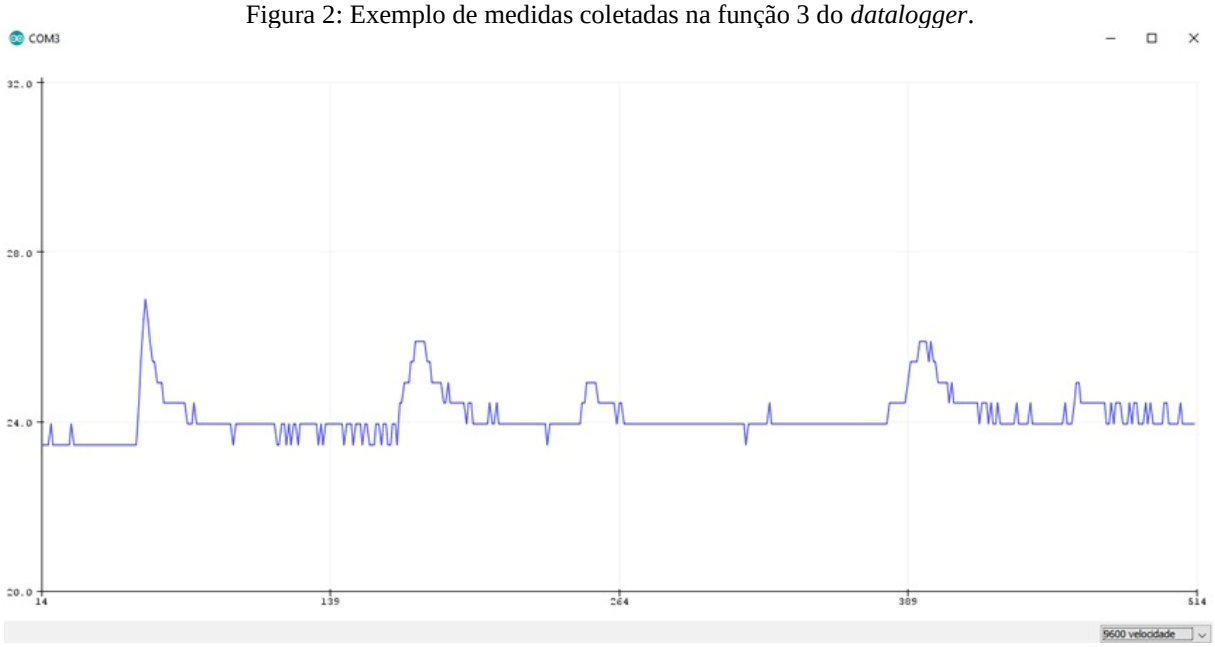
2 COMANDOS DISPONÍVEIS

O comando 1, que possui a função de Reset, apaga todos os dados de amostras armazenados na memória. Para tal, ao ser solicitado, ele zera o contador de dados armazenados e exibe no display LCD a mensagem “Reset”.

O comando 2 é utilizado para exibir o status do sistema, ou seja, a quantidade de memória ocupada e a quantidade de memória disponível para o armazenamento de novas amostras. Portanto, ao ser solicitado, ele exibe no display LCD na primeira linha o número de dados gravados, e na segunda linha o número de posições disponíveis para armazenamento.

O comando 3, ao ser requisitado, faz com que o sistema inicia a coleta dos dados de temperatura a cada 2 segundos os armazene na memória EEPROM. Nessa função, o sistema exibe no display LCD a mensagem “Coletando...”, nos displays de 7 segmentos o valor de temperatura medido em tempo real, e no plotter serial esses mesmos valores como exemplificado na Figura 2.

Figura 2: Exemplo de medidas coletadas na função 3 do *datalogger*.



O comando 4 é responsável por interromper a coleta de dados. Nessa função o sistema não realiza nenhuma atividade até que outro comando seja requisitado e exibe no display LCD a mensagem “Fim de coleta”.

O comando 5, deve transmitir as N ($1 \leq N \leq 1022$) medidas de temperatura mais antigas separadas por quebras de linha pela porta serial. Após a confirmação do comando (“5” seguido de “#”), deve aparecer no display uma mensagem solicitando o número de medidas a serem transferidas (“Qtd de Amostras:”). Nesse momento, o usuário deve digitar um valor entre 1 e 1022 e pressionar “#” em seguida para confirmar ou “*” para cancelar. As medidas transferidas podem ser visualizadas através do monitor serial ou do plotter serial e o display exibe a mensagem “Transferindo... N amostras”.

3 IMPLEMENTAÇÃO

A implementação do hardware pode ser observada na Figura 3 a seguir.

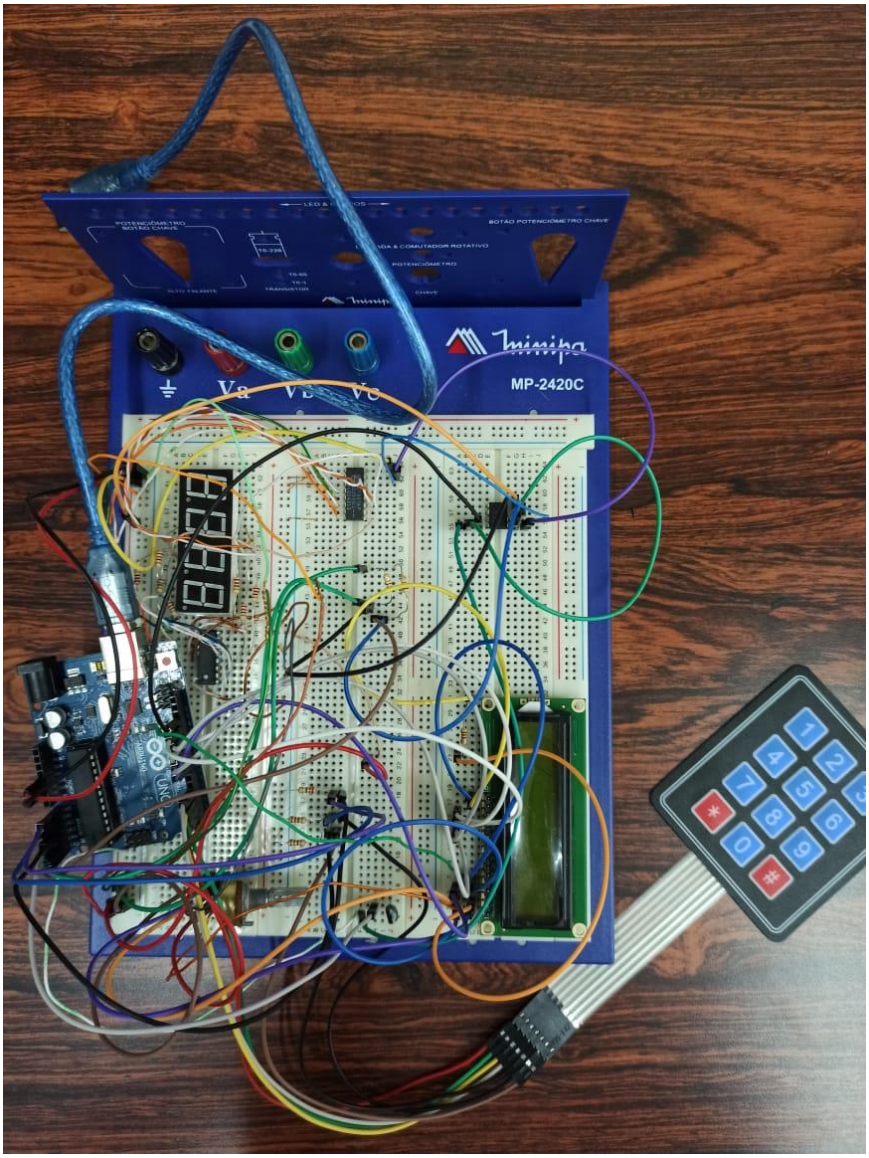


Figura 3: Hardware implementado para o *datalogger*.

O fluxograma da máquina de estados implementada no software é apresentado na Figura 4 descrevendo o mecanismo de coleta e apresentação dos dados.

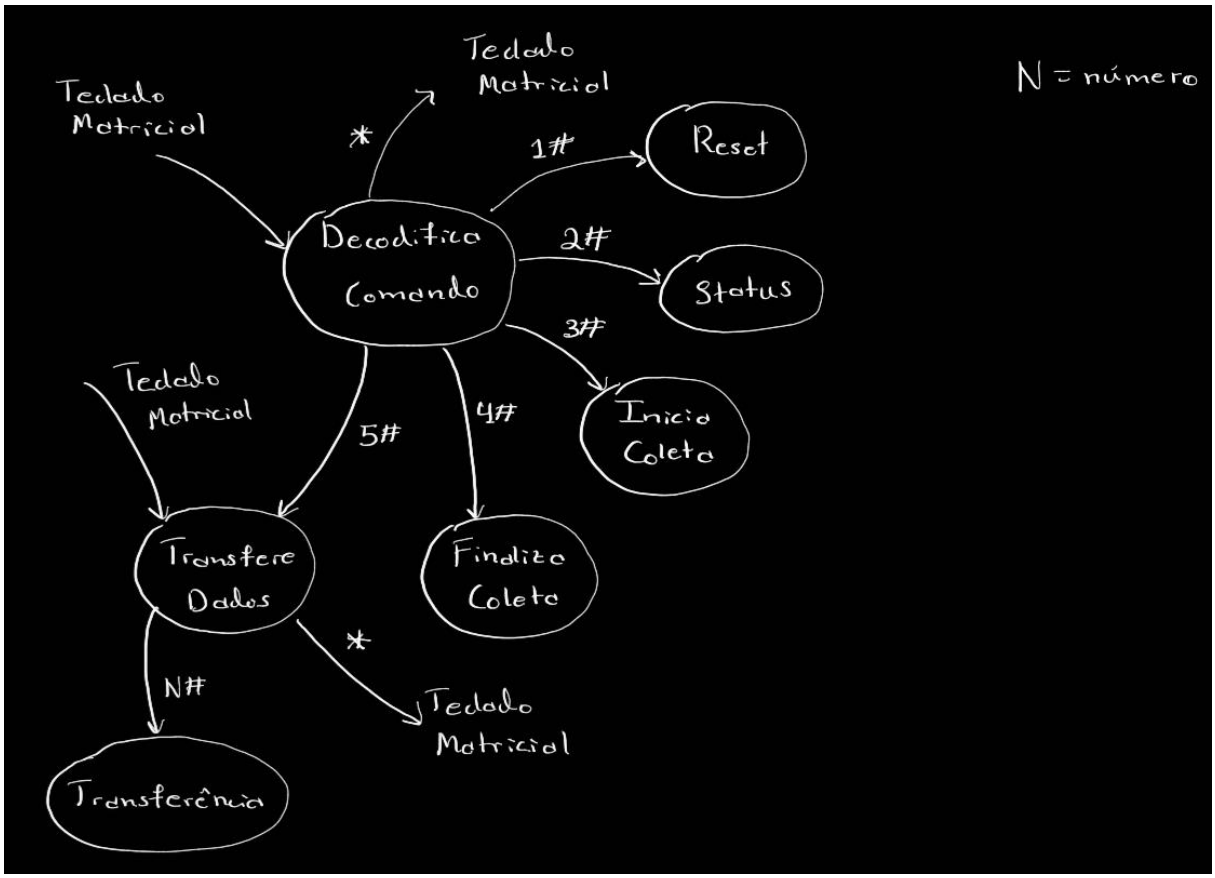


Figura 4: Máquina de estados implementada no software do *datalogger*.

4 ESQUEMÁTICO DO HARDWARE

O diagrama esquemático do hardware implementado pode ser analisado nas Figuras 5 e 6 que se seguem.

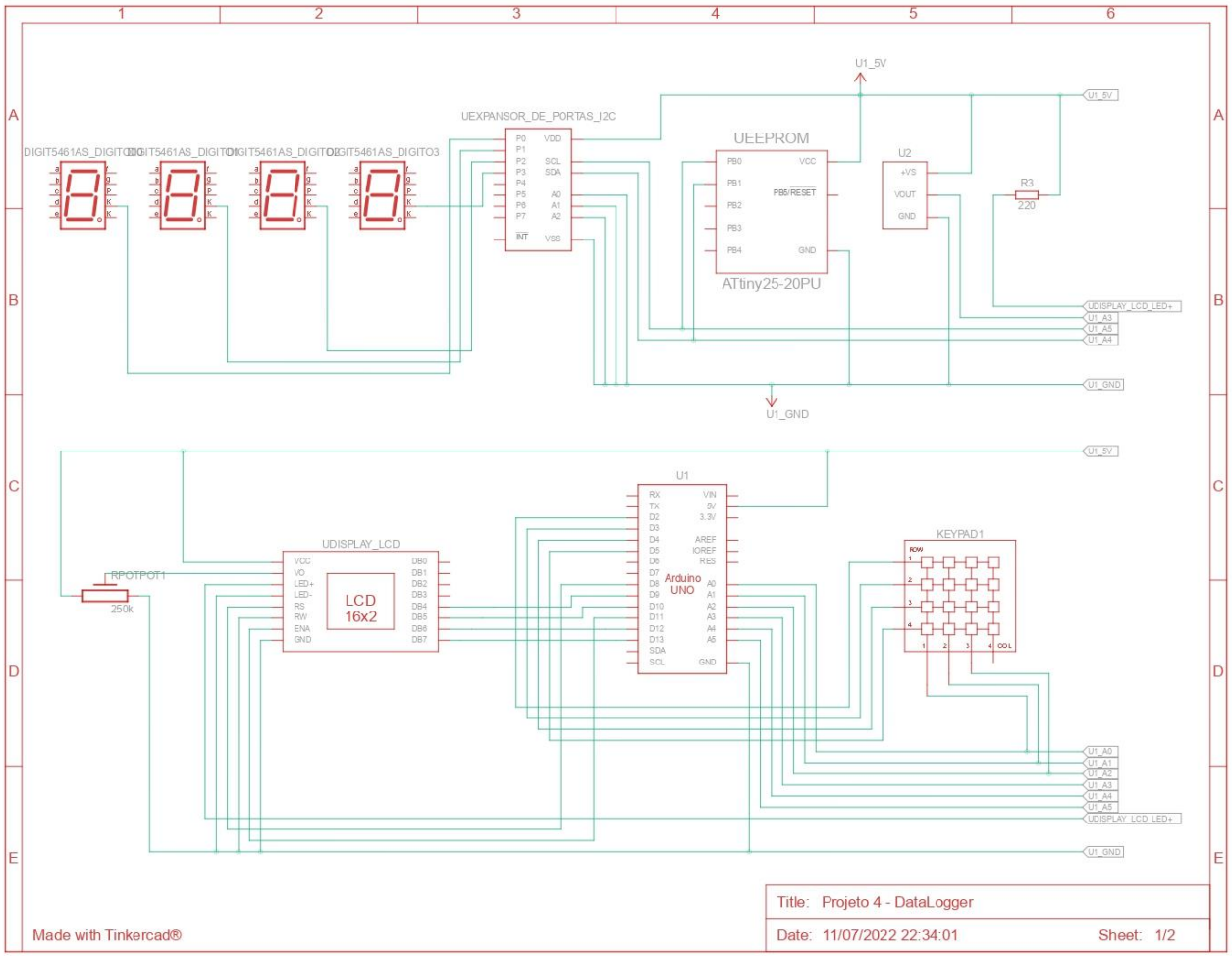


Figura 5: Esquemático do hardware implementado (pt.1).

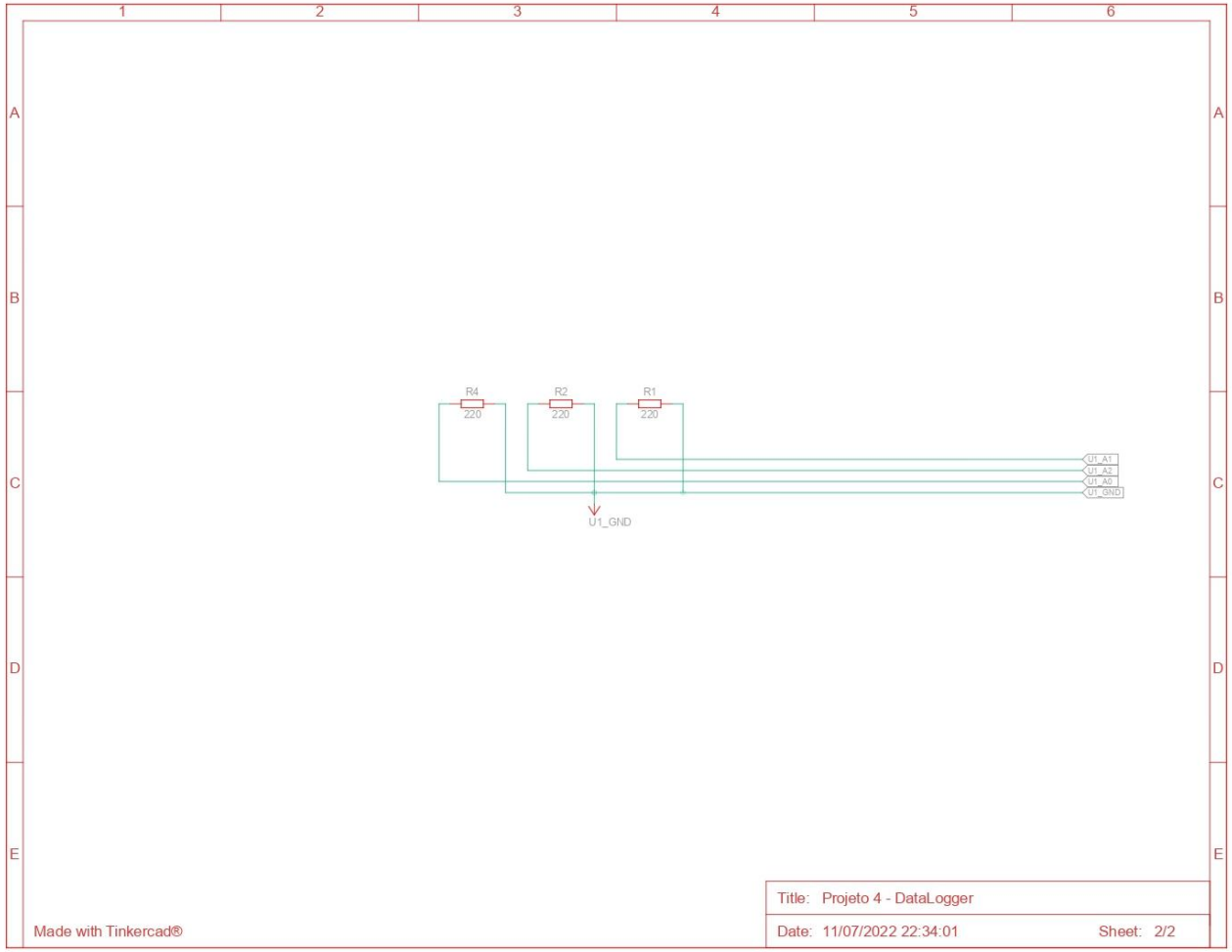


Figura 6: Esquemático do hardware implementado (pt.2).

5 VÍDEO DE APRESENTAÇÃO

Acesse o link abaixo para o vídeo de apresentação do funcionamento do sistema:
https://www.youtube.com/watch?v=k7OIxsWj35E

6 CÓDIGO COMENTADO

```
/*Projeto 3 -> DataLogger*/
/*Welter Mompeam Sozim          RA: 188625*/
/*Guilherme Augusto Amorim Terrell  RA: 168899*/

/*Importação de bibliotecas*/
#include <LiquidCrystal.h>
#include <Wire.h>

/*Definição dos pinos*/
#define pinRS 8
#define pinEnable 11
#define pinD4 9
#define pinD5 10
#define pinD6 12
#define pinD7 13
/*Pino de dados para comunicação I2C*/
#define pinSDA A4
/*Pino de clock para comunicação I2C*/
#define pinSCL A5
/*Pino sensor de temperatura LM35*/
#define pinLM35 A3

/*variáveis LCD*/
/*LCD -> 2 linhas de 16 caracteres*/
LiquidCrystal lcd(pinRS, pinEnable, pinD4, pinD5, pinD6, pinD7);

/*--------------------*/

/*Device address I2C*/
unsigned char numeroI2C = 0;
unsigned char enderecoPCF8574P = 0x20;/*Endereço do expandor de portas*/
unsigned char enderecoEEPROM = 0x50;/*Endereço da EEPROM*/

/*Variáveis EEPROM*/
/*EEPROM tem 8 paginas com 256 posiçoes de memória em cada página*/
/*Cada posição de memória tem capacidade de armazenar 1byte de informação*/
/*Portanto cada página da EEPROM tem uma capacidade de 256bytes de armazenamento*/
unsigned char paginaEEPROM = 0x00;
unsigned char posicaoMemoriaEEPROM = 0x00;
unsigned char penultimaPosicaoMemEEPROM = 0xfe;
unsigned char ultimaPosicaoMemEEPROM = 0xff;
unsigned int memOcupada = 0;
unsigned int memDisponivel = 0;

bool coletaFinalizada = 0;/*Flag que indica se a EEPROM está cheia. Se sim, a coleta de dados foi finalizada*/
```

```
/*Variáveis utilizadas na função de transferência de dados*/
bool podeLer = 0;/*Flag que indica que o usuário já passou a quantidade de leituras desejada*/
bool podeEscrever = 0;/*Flag que habilita novo ciclo de escrita*/
float tempTranfDados = 0;/*Armazenar os 2 bytes da temperatura que estava armazenada na EEPROM*/
unsigned char pagina = 0;/*iterar sobre a página da EEPROM durante a transferência de dados*/
unsigned char posicao = 0;/*iterar sobre a posicao da EEPROM durante a tranferencia de dados*/
unsigned char dadoMSB = 0;/*Armazenar o byte mais significativo*/
unsigned char dadoLSB = 0;/*Armazenar o byte menos significativo*/

/*Variáveis para selecionar qual display de 7 seg será acionado*/
unsigned int display0 = 112;
unsigned int display1 = 176;
unsigned int display2 = 208;
unsigned int display3 = 224;
unsigned char displaySelecionado = 0;

unsigned int counter = 0;
unsigned int delayEscrita = 0;/*Especifica o tempo que deve ser aguardado após escrever na EEPROM*/
unsigned int delayEntreMedicoes = 1000;/*Leitura do sensor de temperatura é feita de 2 em 2 segundos*/
unsigned int delayLimpaDisplay = 0;/*contador para limpar o conteúdo do display periodicamente*/
unsigned int delayDisplay7Seg4Dig = 0;/*Altera o valor do display 7Seg4Dig a cada 1 segundo*/

float valorAnalogLM35 = 0;/*Armazena o valor analogico do LM35 que varia entre 0 e 1023*/
float tensao = 0;/*Nível de tensão associado a leitura analógica do LM35*/
float temp = 0;/*Temperatura final do LM35*/
float temperaturaLida = 0; /*Temperatura lida pelo LM35 no pino analógico*/
unsigned int qdadeLeituras = 0;/*Quantidade de leituras solicitada pelo usuário a ser recuperada */
unsigned char tempMSB = 0;/*Armazenar os 8 bits mais significativos da variavel temp*/
unsigned char tempLSB = 0;/*Armazenar os 8 bits menos significativos da variável temp*/

/*--------------------------------*/
/*Variáveis teclados matricial*/
int count=0;
String LinhaColuna = ""; // Linha+Coluna pressionada
String tecla = ""; // Tecla pressionada
String comando = ""; // Variável onde as teclas serão concatenadas para formar o comando
String funcao = ""; // Variável para armazenar a função
bool habilitarVarredura = 0;/*habilita varredura do teclado matricial*/
int qtdAmostras = 0; // Quantidade de amostras a serem transferidas no comando 5

// Rotina de servico de interrupcao do temporizador 0
ISR(TIMER0_COMPA_vect) {
  counter++;
  delayEscrita++;
  if(delayEscrita >= 15){
    /*delay de 30ms após escrita*/
    podeEscrever = 1;
  }
  delayEntreMedicoes++;
  delayLimpaDisplay++;
  displaySelecionado++;
  count++;
  if(count >= 400){
    /*Varredura do teclado feita a cada 250ms*/
    varreduraTeclado();
    count = 0;/*reseta contador*/
  }
```

```
}

void configuracao_Timer0(){
 ////////////////////////////////////////////////////////////////////////////////////////////////////////
 // Configuracao Temporizador 0 (8 bits) para gerar interrupcoes periodicas a cada 2ms no modo Clear Timer on
Compare Match (CTC)
 // Relogio = 16e6 Hz
 // Prescaler = 256
 // Faixa = 125 (contagem de 0 a OCR0A = 124)
 // Intervalo entre interrupcoes: (Prescaler/Relogio)*Faixa = (256/16e6)*(124+1) = 0.002s

 // TCCR0A – Timer/Counter Control Register A
 // COM0A1 COM0A0 COM0B1 COM0B0 – – WGM01 WGM00
 // 0     0     0     0         1     0
 TCCR0A = 0x02;

 // OCR0A – Output Compare Register A
 OCR0A = 124;

 // TIMSK0 – Timer/Counter Interrupt Mask Register
 // – – – – – OCIE0B OCIE0A TOIE0
 // – – – – – 0     1     0
 TIMSK0 = 0x02;

 // TCCR0B – Timer/Counter Control Register B
 // FOC0A FOC0B – – WGM02 CS02 CS01 CS0
 // 0    0       0    1   0   0
 TCCR0B = 0x04;
 ////////////////////////////////////////////////////////////////////////////////////////////////////////
}

/*-----------------------------------------------------------------*/

/*Função LCD*/
void escreveLCD(String funcao) {
 if(delayLimpaDisplay >= 125){
  lcd.clear();
  delayLimpaDisplay = 0;
 }
 if(funcao == "Status") {
  lcd.setCursor(0, 0);
  lcd.print(String(memOcupada));
  lcd.setCursor(0, 1);
  lcd.print(String(memDisponivel));
 }
 if(funcao == "Reset") {
  lcd.setCursor(0, 0);
  lcd.print("Reset");
 }
 if(funcao == "Inicia coleta periodica"){
  lcd.setCursor(0, 0);
  lcd.print("Coletando...");
 }
 if(funcao == "Finaliza coleta periodica"){
  lcd.setCursor(0, 0);
  lcd.print("Fim de coleta");
 }
```

```
  if(funcao == "Transferencia de dados" && qtdAmostras == 0){
    lcd.setCursor(0, 0);
    lcd.print("Qtd de Amostras:");
  }
  if(funcao == "Transferencia de dados" && qtdAmostras != 0){
    lcd.setCursor(0, 0);
    lcd.print("Transferindo...");
    lcd.setCursor(0, 1);
    lcd.print(String(qtdAmostras) + " amostras");
  }
}

/*-----------------------------------------------------------------------------------------------------------------------------*/
/*Funções EEPROM*/

/*Função para escrever na EEPROM*/
void escreveEEPROM(unsigned char enderecoI2CEEPROM, unsigned char pageEEPROM, unsigned char
posMemEEPROM, unsigned char dadoParaEscreverNaEEPROM) {
  /*Iniciar transmissão*/
  /*endereço da EEPROM: 1 0 1 0 A10 A9 A8 R/W */
  /*Para escrever na EEPROM: R/W = 0x00*/
  if(podeEscrever == 1){
    Wire.beginTransmission(enderecoI2CEEPROM | pageEEPROM);/*Seleciona o device address da EEPROM de
acordo com o protocolo I2C*/
    Wire.write(posMemEEPROM);/*Seleciona a posição da memória na página*/
    Wire.write(dadoParaEscreverNaEEPROM);/*Escrever o dado na EEPROM*/
    Wire.endTransmission();
    delayEscrita = 0;
    podeEscrever = 0;
  }
}

/*Função que atualiza a quantidade de posições ocupadas*/
/*As últimas duas posições da EEPROM são destinadas a isso*/
void atualizaEEPROM(){
  escreveEEPROM(enderecoEEPROM, 7, penultimaPosicaoMemEEPROM, paginaEEPROM);
  escreveEEPROM(enderecoEEPROM, 7, ultimaPosicaoMemEEPROM, posicaoMemoriaEEPROM);
}

void coletaDados(){
  /*Leitura da temperatura*/
  temperaturaLida = leTemp();
  Serial.println(temperaturaLida);
  tempMSB = (temperaturaLida*100) / 100;/*byte mais significante da temperatura*/
  tempLSB = (temp*100) - 100*tempMSB;/*byte menos significante da temperatura*/
  /*Bloco de escrita de dados*/
  /*escrita de dados só ocorre caso não haja pedido de transferência de dados por parte do usuário*/
  if(((paginaEEPROM << 8) + posicaoMemoriaEEPROM) <= 2045){
    if(posicaoMemoriaEEPROM % 2 == 0){
      escreveEEPROM(enderecoEEPROM, paginaEEPROM, posicaoMemoriaEEPROM, tempMSB);
    }
    else{
      escreveEEPROM(enderecoEEPROM, paginaEEPROM, posicaoMemoriaEEPROM, tempLSB);
    }
    if(posicaoMemoriaEEPROM == 255){
      /*todas as posições daquela página já foram lidas*/
      paginaEEPROM++;
```

```
    posicaoMemoriaEEPROM = 0;
   }
   else{
    posicaoMemoriaEEPROM++;
   }
   /*Atualizar as duas últimas posições da EEPROM com as posições ocupadas*/
   atualizaEEPROM();
  }
  else{
   /*Memória cheia*/
   /*As duas últimas posições da EEPROM devem armazenar a quandidade de posições ocupadas*/
   atualizaEEPROM();
   coletaFinalizada = 1;/*Coleta de dados encerrada*/
  }
}

/*Função para ler a EEPROM*/
unsigned char leEEPROM(unsigned char enderecoI2CEEPROM, unsigned char pageEEPROM, unsigned char
posMemEEPROM) {
 /*Iniciar transmissão*/
 /*endereço da EEPROM: 1 0 1 0 A10 A9 A8 R/W */
 /*Para ler a EEPROM: R/W = 0x01*/
 unsigned char dadoLidoNaEEPROM = 0;
 Wire.beginTransmission(enderecoI2CEEPROM | pageEEPROM);/*Seleciona o device address da EEPROM de
acordo com o protocolo I2C*/
 Wire.write(posMemEEPROM);/*Seleciona a posição da memória na página*/
 Wire.endTransmission();
 Wire.requestFrom(enderecoEEPROM | paginaEEPROM, 1);/*deviceAddress e quanidade de bytes que
desejamos ler*/
 if (Wire.available()) {
  dadoLidoNaEEPROM = Wire.read();
 }
 return dadoLidoNaEEPROM;
}

float leTemp(){
 valorAnalogLM35 = float(analogRead(pinLM35));/*Armazena o valor analogico do LM35 que varia entre 0 e
1023*/
 tensao = (valorAnalogLM35*5)/1023;/*Nível de tensão associado a leitura analógica do LM35*/
 temp = tensao/0.01;/*Temperatura final do LM35*/
 //Serial.println(temp);
 return temp;
}

/*Função que reseta a EEPROM*/
/*Reset -> Escrever zero nos últimos dois bytes da memória*/
void resetEEPROM(){
 paginaEEPROM = 0;
 posicaoMemoriaEEPROM = 0;
 escreveEEPROM(enderecoEEPROM, 7, penultimaPosicaoMemEEPROM, paginaEEPROM);
 escreveEEPROM(enderecoEEPROM, 7, ultimaPosicaoMemEEPROM, posicaoMemoriaEEPROM);
}

/*Função que escreve no LCD quantos bytes de memória estão ocupados e quantos estão disponíveis*/
void statusEEPROM(){
 memOcupada = ((paginaEEPROM << 8) + posicaoMemoriaEEPROM);
 memDisponivel = 2048 - memOcupada;
```

```
}

/*Função que lê os n dados solicitados pelo usuário*/
/*1 <= n <= 1022*/
void transfDadosEEPROM(unsigned int qdadeDados){
  bool dadosDeSobra = 0;
  if((paginaEEPROM << 8) + posicaoMemoriaEEPROM >= qdadeDados){
    /*tem mais dados disponíveis do que foi solicitado*/
    dadosDeSobra = 1;
  }
  else{
    dadosDeSobra = 0;
  }
  /*Bloco executado quanto tem mais dados do que o solicitado*/
  /*-----------------------------------------------------------------------------------------------------------------------*/
  if(dadosDeSobra == 1){
    if(qdadeDados <= 1022){
      if(((pagina << 8) + posicao) <= qdadeDados - 1){
        if(posicao % 2 == 0){
          /*O byte mais significativo ocupa posições pares da mamória*/
          dadoMSB = leEEPROM(enderecoEEPROM, pagina, posicao);
        }
        if(posicao % 2 == 1){
          /*O byte menos significativo ocupa posições ímpares da memória*/
          dadoLSB = leEEPROM(enderecoEEPROM, pagina, posicao);
        }
        if(posicao == 255){
          pagina++;
          posicao = 0;
        }
        else{
          posicao++;
        }
      tempTranfDados = dadoMSB + 0.01*dadoLSB;
      Serial.println(tempTranfDados);
      }
      else{
        /*Todos os dados disponíveis foram transferidos*/
        /*Deve-se zerar as flags´e as variáveis de iteração para habilitar nova coleta*/
        pagina = 0;
        posicao = 0;
        qtdAmostras = 0;
      }
    }
    else{
      /*Se a quantidade de leituras solicitada for maior que 1020*/
      /*Deve-se retornar no máximo 1020*/
      if(((pagina << 8) + posicao) <= 1021){
        if(posicao % 2 == 0){
          dadoMSB = leEEPROM(enderecoEEPROM, pagina, posicao);
        }
        if(posicao % 2 == 1){
          dadoLSB = leEEPROM(enderecoEEPROM, pagina, posicao);
        }
        if(posicao == 255){
          pagina++;
          posicao = 0;
```

```
      }
      else{
       posicao++;
      }
      tempTranfDados = dadoMSB + 0.01*dadoLSB;
      Serial.println(tempTranfDados);
    }
     else{
      /*Todos os dados disponíveis foram transferidos*/
      /*Deve-se zerar as flags´e as variáveis de iteração para habilitar nova coleta*/
     pagina = 0;
     posicao = 0;
     qtdAmostras = 0;
    }
   }
  }
  else{
   /*bloco executado quando for solicitado mais dados do que tem disponível*/
   if(qdadeDados <= 1022){
    if(((pagina << 8) + posicao) <= qdadeDados - 1 && ((pagina << 8) + posicao <= (paginaEEPROM << 8) +
posicaoMemoriaEEPROM)){
      if(posicao % 2 == 0){
       /*O byte mais significativo ocupa posições pares da mamória*/
       dadoMSB = leEEPROM(enderecoEEPROM, pagina, posicao);
      }
      if(posicao % 2 == 1){
       /*O byte menos significativo ocupa posições ímpares da memória*/
       dadoLSB = leEEPROM(enderecoEEPROM, pagina, posicao);
      }
      if(posicao == 255){
       pagina++;
       posicao = 0;
      }
      else{
       posicao++;
      }
    tempTranfDados = dadoMSB + 0.01*dadoLSB;
    Serial.println(tempTranfDados);
    }
    else{
     /*Todos os dados disponíveis foram transferidos*/
     /*Deve-se zerar as flags´e as variáveis de iteração para habilitar nova coleta*/
     pagina = 0;
     posicao = 0;
     qtdAmostras = 0;
    }
   }
   else{
    /*Se a quantidade de leituras solicitada for maior que 1020*/
    /*Deve-se retornar no máximo 1020*/
    if(((pagina << 8) + posicao) <= 1021 && ((pagina << 8) + posicao <= (paginaEEPROM << 8) +
posicaoMemoriaEEPROM)){
      if(posicao % 2 == 0){
       dadoMSB = leEEPROM(enderecoEEPROM, pagina, posicao);
      }
      if(posicao % 2 == 1){
       dadoLSB = leEEPROM(enderecoEEPROM, pagina, posicao);
```

```
      }
      if(posicao == 255){
        pagina++;
        posicao = 0;
      }
      else{
        posicao++;
      }
      tempTranfDados = dadoMSB + 0.01*dadoLSB;
      Serial.println(tempTranfDados);
    }
     else{
      /*Todos os dados disponíveis foram transferidos*/
      /*Deve-se zerar as flags´e as variáveis de iteração para habilitar nova coleta*/
     pagina = 0;
     posicao = 0;
     qtdAmostras = 0;
     }
   }
  }
}

/*-------------------------------------------------------------------------------------------------------------------------------------------------------------*/
/*Funções teclado matricial*/
// Função para leitura do teclado matricial
void LeituraTeclado(){
  for (int ti = 2; ti<6; ti++){
    //Alterna o estado dos pinos das linhas
    digitalWrite(2, LOW);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
    digitalWrite(5, LOW);
    digitalWrite(ti, HIGH);
    //Verifica se alguma tecla da coluna 1 foi pressionada
    if ((digitalRead(A0)==HIGH)){
      LinhaColuna = ti-1;
      LinhaColuna.concat('1');
    }
    //Verifica se alguma tecla da coluna 2 foi pressionada
    if ((digitalRead(A1)==HIGH)){
      LinhaColuna = ti-1;
      LinhaColuna.concat('2');
    }
    //Verifica se alguma tecla da coluna 3 foi pressionada
    if ((digitalRead(A2)==HIGH)){
      LinhaColuna = ti-1;
      LinhaColuna.concat('3');
    }
  }
}


// Função para decodificar LinhaColuna para a respectiva tecla
void Decodifica_LinhaColuna(){
  if (LinhaColuna=="11"){
    tecla='1';
    LinhaColuna="";
```

```
  }
  else if (LinhaColuna=="12"){
    tecla='2';
    LinhaColuna="";
  }
  else if (LinhaColuna=="13"){
    tecla='3';
    LinhaColuna="";
  }
  else if (LinhaColuna=="21"){
    tecla='4';
    LinhaColuna="";
  }
  else if (LinhaColuna=="22"){
    tecla='5';
    LinhaColuna="";
  }
  else if (LinhaColuna=="23"){
    tecla='6';
    LinhaColuna="";
  }
  else if (LinhaColuna=="31"){
    tecla='7';
    LinhaColuna="";
  }
  else if (LinhaColuna=="32"){
    tecla='8';
    LinhaColuna="";
  }
  else if (LinhaColuna=="33"){
    tecla='9';
    LinhaColuna="";
  }
  else if (LinhaColuna=="41"){
    tecla='*';
    LinhaColuna="";
  }
  else if (LinhaColuna=="42"){
    tecla='0';
    LinhaColuna="";
  }
  else if (LinhaColuna=="43"){
    tecla='#';
    LinhaColuna="";
  }
}


// Função para decodificar o comando
void Decodifica_Comando(){

  if (comando.endsWith("#")){ // Comando confirmado
    Serial.println("Confirma comando");
    // remove o caractere "#" do final do comando
    comando.remove(comando.length()-1);
    if (funcao!="Transferencia de dados" && comando=="1"){
      funcao = "Reset";
```

```
    qtdAmostras = 0;
   }
   else if (funcao!="Transferencia de dados" && comando=="2"){
    funcao = "Status";
    qtdAmostras = 0;
   }
   else if (funcao!="Transferencia de dados" && comando=="3"){
    funcao = "Inicia coleta periodica";
    qtdAmostras = 0;
   }
   else if (funcao!="Transferencia de dados" && comando=="4"){
    funcao = "Finaliza coleta periodica";
    qtdAmostras = 0;
   }
   else if (funcao!="Transferencia de dados" && comando=="5"){
    funcao = "Transferencia de dados";
    qtdAmostras = 0;
   }
   else if (funcao=="Transferencia de dados"){
    qtdAmostras = comando.toInt();
   }
   comando="";
  }
  else if (comando.endsWith("*")){ // Comando cancelado
   funcao="";
   Serial.println("Comando cancelado!");
   comando="";
   qtdAmostras = 0;
  }
}

/*Função executada dentro da rotina de interupção por timer*/
/*Responsável por fazer a varredura do teclado matricial*/
void varreduraTeclado(){
  // Realiza a leitura do teclado matricial a cada 140ms
  // e decodifica a tecla pressionada através da variável LinhaColuna

  LeituraTeclado();
  Decodifica_LinhaColuna();

  // Concatena a tecla pressionada à variável comando para ser decodificado posteriormente
  if (tecla!=""){
   comando.concat(tecla);
  }

  // Reinicializa tecla e o contador da interrupção
  tecla="";

  // Decodifica o comando para obter a funcao
  Decodifica_Comando();
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*Função para calcular o valor de cada dígito, escolher o dígito do display de 7 segmentos a ser exibido e
escrever nele*/
/*-------------------------------------------------------------------------------------------------------------------*/
void selecionaDisplay(){
```

```
  int dadoDisplay0 = temperaturaLida / 10;
  int dadoDisplay1 = (100*temperaturaLida - 1000*dadoDisplay0) / 100;
  int dadoDisplay2 = (100*temperaturaLida - 1000*dadoDisplay0 - 100*dadoDisplay1) / 10;
  int dadoDisplay3 = 100*temperaturaLida - 1000*dadoDisplay0 - 100*dadoDisplay1 - 10*dadoDisplay2;
  if ((displaySelecionado)%4==0){
    escreveDisplay7Seg(enderecoPCF8574P, display0, dadoDisplay0);
  }
  else if ((displaySelecionado)%4==1){
    escreveDisplay7Seg(enderecoPCF8574P, display1, dadoDisplay1);
  }
  else if ((displaySelecionado)%4==2){
    escreveDisplay7Seg(enderecoPCF8574P, display2, dadoDisplay2);
  }
  else{
    escreveDisplay7Seg(enderecoPCF8574P, display3, dadoDisplay3);
  }
}
/*-----------------------------------------------------------------------------------------------------------------------*/
/*-----------------------------------------------------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------------------------------------------------*/
/*Função para escrever no display de 7 segmentos de 4 dígitos*/
/*-----------------------------------------------------------------------------------------------------------------------*/
void escreveDisplay7Seg(unsigned char enderecoI2CDisplay7Seg, unsigned int displayEscolhido, unsigned int
dado){
  /*Iniciar transmissão*/
  /*endereço da EEPROM: 1 0 1 0 A10 A9 A8 R/W */
  /*Para escrever na EEPROM: R/W = 0x00*/
  Wire.beginTransmission(enderecoI2CDisplay7Seg);/*Seleciona o device address do display de acordo com o
protocolo I2C*/
  Wire.write(displayEscolhido + dado);/*Seleciona o display*/
  Wire.endTransmission();
}
/*-----------------------------------------------------------------------------------------------------------------------*/
/*-----------------------------------------------------------------------------------------------------------------------*/
/*Máquina de estados para as 5 funções*/
/*
flagFuncao = 1 -> reset
flagFuncao = 2 -> status
flagFuncao = 3 -> coleta periódica de dados
flagFuncao = 4 -> fim da coleta periódica
flagFuncao = 5 -> tranferencia de dados
*/
void maqEstados(String funcao){
  if(funcao == "Reset"){
    resetEEPROM();
    coletaFinalizada == 0;/*Habilita novo ciclo de coleta de dados após resetar a EEPROM*/
  }
  if(funcao == "Status"){
    statusEEPROM();
  }
  if(funcao == "Inicia coleta periodica"){
    if(delayEntreMedicoes >= 1000){
      coletaDados();
      delayEntreMedicoes = 0;
    }
    if(coletaFinalizada == 1){
```

```
      /*Para de ler a temperatura*/
      /*Para de escrever no EEPROM*/
      funcao == "Finaliza coleta periódica";
    }
  }
  if(funcao=="Transferencia de dados"){
    if(qtdAmostras>=1 && qtdAmostras<=1022){
      transfDadosEEPROM(qtdAmostras);
    }
    else{
      qtdAmostras=0;
    }
  }
}

/*-------------------------------------------------------------------------------------------------------------------------*/
void setup(){
  pinMode(pinLM35, INPUT);/*Pino termômetro*/
  //Pinos 1, 2, 3 e 4 do teclado matricial - Linhas
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);

  //Pinos 5, 6 e 7 do teclado matricial - Colunas
  pinMode(A0, INPUT);
  pinMode(A1, INPUT);
  pinMode(A2, INPUT);
  Serial.begin(9600);
  Wire.begin();/*Configura o barramento do I2C*/
  cli();
  configuracao_Timer0();
  sei();
  /*Iniciar e limpar o LCD*/
  lcd.begin(16, 2);
  lcd.clear();
}

/*-------------------------------------------------------------------------------------------------------------------------*/
void loop(){

  /*Varredura do teclado matricial*/
  //if(habilitarVarredura == 1){
  //  varreduraTeclado();
  //  habilitarVarredura = 0;
  //}

  maqEstados(funcao);
  escreveLCD(funcao);

  /*Escrever a temperatura no display 7seg4dig*/
  /*A variável temperatura lida é atualizada na função coletaDados() a cada 2 segundos*/
  selecionaDisplay();

}
```