UNIVERSIDADE ESTADUAL DE CAMPINAS – UNICAMP

FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO – FEEC

EA076 – Turma U

Laboratório de Sistemas Embarcados

PROJETO 2

Sistema para Acionamento e Controle de Ventilador de Teto

GUILHERME AUGUSTO AMORIM TERRELL **RA: 168899**

WELLTER MOMPEAN SOZIN **RA: 188625**

Prof.º ERIC ROHMER

Campinas

2022

SUMÁRIO

LISTA DE FIGURAS **3**

LISTA DE TABELAS **4**

1 INTRODUÇÃO **5**

2 MÓDULO BLUETOOTH **7**

3 MÓDULO DE CONTROLE DO MOTOR CC **8**

4 MÓDULO DE ESTIMATIVA DE VELOCIDADE **9**

5 MÓDULO DO DISPLAY LCD **10**

6 MÓDULO DOS DISPLAYS DE 7 SEGMENTOS **11**

7 LISTA DE COMPONENTES **11**

8 ESQUEMÁTICO DO HARDWARE **11**

9 VÍDEO DE APRESENTAÇÃO **14**

10 CÓDIGO COMENTADO **14**

LISTA DE FIGURAS

Figura 1: Módulos que compõem o sistema de acionamento e controle do ventilador de teto. . 5
Figura 2: Hardware implementado para o sistema de acionamento e controle de ventilador de teto. 7
Figura 3: Fluxograma para a identificação dos comandos e suas respectivas respostas. 8
Figura 4: Máquina de estados implementada para o controle do motor CC. 9
Figura 5: Fluxograma do processo implementado para estimativa da velocidade do motor. .. 10
Figura 6: Esquemático do hardware implementado (pt.1). 12
Figura 7: Esquemático do hardware implementado (pt.2). 13

LISTA DE TABELAS

Tabela 1: Funções disponíveis para controle do motor com seus respectivos comandos a serem enviados e respostas a serem obtidas. 6
Tabela 2: Componentes utilizados. 11

1 INTRODUÇÃO

Este projeto consiste no desenvolvimento e implementação de hardware e software para o acionamento e controle de um ventilador de teto.

O sistema é composto por diversos módulos: temos o módulo de controle de velocidade do motor (Ponte H); o de medição e estimativa da velocidade do motor (Encoder); o display LCD como mostrador da velocidade estimada; o módulo dos displays de 7 segmentos com o expansor de portas também para exibição da velocidade estimada; e o módulo bluetooth para comunicação. A Figura 1 exibe como esses módulos se interconectam ao Arduino Uno.

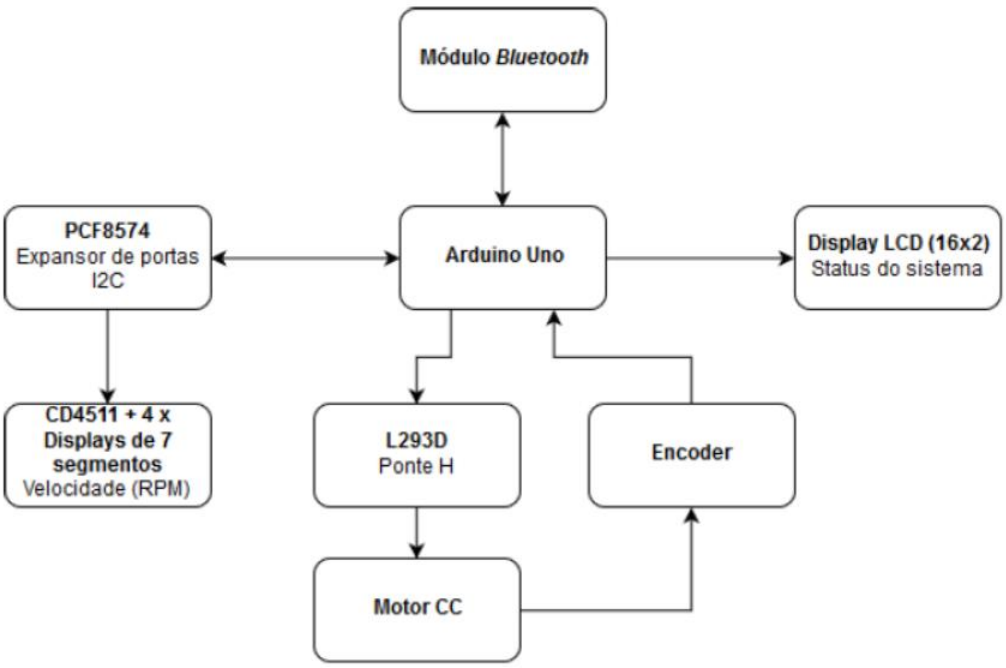

Figura 1: Módulos que compõem o sistema de acionamento e controle do ventilador de teto.

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

O motor do ventilador pode funcionar de dois modos, o modo de ventilação e o modo de exaustão, ou seja, cada um com um sentido de rotação. É possível controlar sua velocidade de rotação em qualquer um dos modos e pará-lo. Também pode-se alterar o modo de funcionamento a qualquer instante e, caso o motor esteja rodando em outro modo, ele irá parar para depois retomar sua rotação no modo solicitado na mesma velocidade que estava funcionando anteriormente. Por fim, também existe o comando para retornar a velocidade atual estimada do ventilador.

Todos os comandos para essas ações (funções) são enviados através de um aplicativo no celular conectado ao bluetooth do ventilador e, para cada comando enviado, também se obtém uma resposta. A Tabela 1 apresenta os comandos disponíveis e suas respectivas respostas.

Tabela 1: Funções disponíveis para controle do motor com seus respectivos comandos a serem enviados e respostas a serem obtidas.

FUNÇÃO	**COMANDO**	**RESPOSTA**
Ajuste de velocidade (%)	VEL xxx* (xxx entre 000 e 100)	OK VEL xxx%
Função ventilador	VENT*	OK VENT
Função exaustor	EXAUST*	OK EXAUST
Parar	PARA*	OK PARA
Retorna a estimativa (X) da velocidade atual do motor em RPM	RETVEL*	VEL: X RPM

Além do mais, caso seja enviado ao sistema um comando não contemplado na Tabela 1, o mesmo deve retornar uma mensagem específica para cada tipo de erro identificado de acordo com a Tabela 2.

TIPO DE ERRO	MENSAGEM DE RESPOSTA (SERIAL)
A string recebida não corresponde a nenhum comando válido	ERRO: COMANDO INEXISTENTE
O comando foi recebido sem parâmetro (para comandos que exigem parâmetros)	ERRO: PARÂMETRO AUSENTE
O parâmetro do comando não está na faixa correta de valores (para comandos que exigem parâmetros)	ERRO: PARÂMETRO INCORRETO

O hardware implementado contendo todos os módulos é exibido na Figura 2.

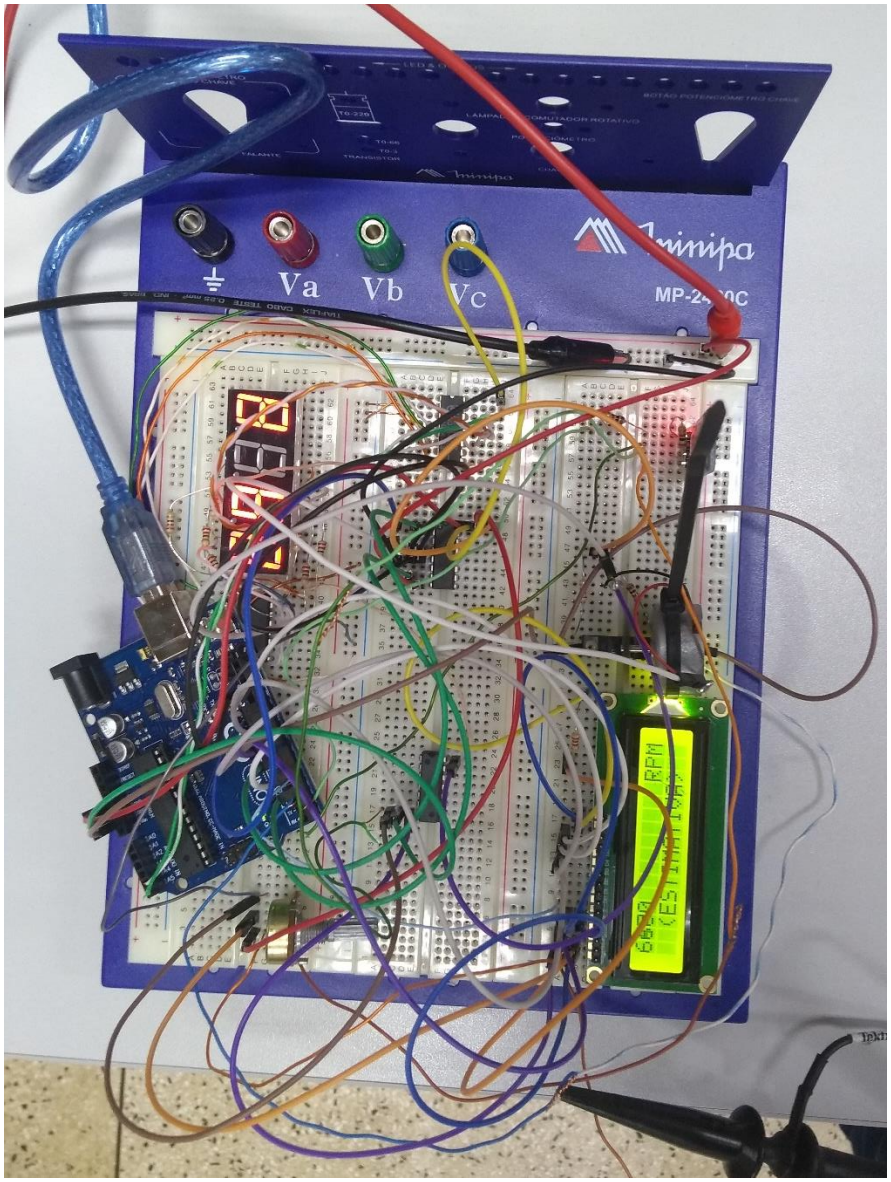

Figura 2: Hardware implementado para o sistema de acionamento e controle de ventilador de teto.

A seguir são apresentados de maneira detalhada cada módulo com explicações das suas funções, implementação e componentes.

2 MÓDULO BLUETOOTH

O módulo bluetooth é composto basicamente pelo componente Bluetooth HC-05 para realizar a comunicação serial entre o celular conectado e o Arduino Uno. Há também um divisor de tensão para compatibilizar seu terminal de recepção de dados RXD ao nível máximo de 3,3V.

Sua velocidade de comunicação serial com o Arduino (baud rate) foi configurada em 9600 bps e implementado um software no Arduino para leitura dos comandos recebidos através de duas funções: LerComando(), que tem o objetivo de ler caracteres recebidos e concatená-los de modo a formar uma única string; e a função decodComando(), que deve analisar essa string e dar o direcionamento, se a string corresponder a uma função válida com os devidos parâmetros (se houver) será exibido no terminal uma mensagem (cada uma das

funções válidas tem um padrão de mensagem), caso a string não corresponda a uma função válida ou faltar algum parâmetro será exibido no terminal uma mensagem com o tipo de erro. Não foi utilizado nenhum tipo de espera ativa, dessa forma a condição para concatenação dos caracteres é que o usuário digite algo no terminal.

A Figura 3 apresenta o fluxograma de funcionamento com a identificação dos comandos e as respectivas respostas.

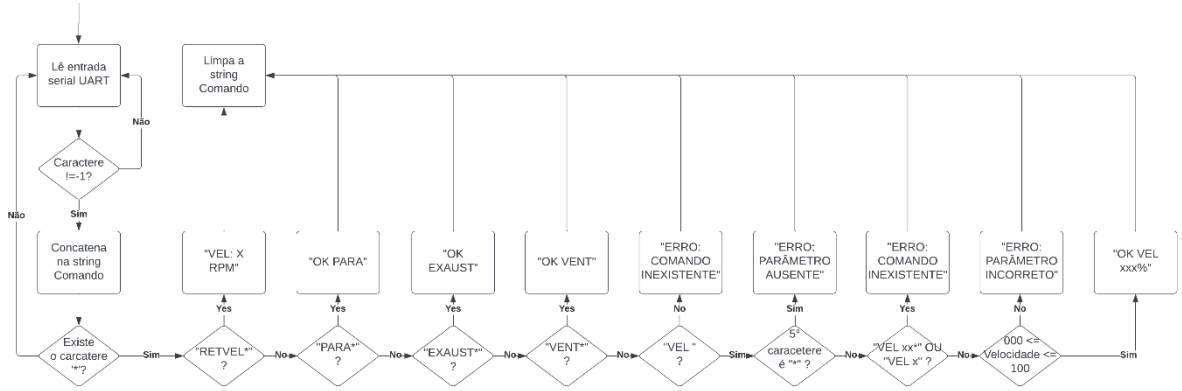

Figura 3: Fluxograma para a identificação dos comandos e suas respectivas respostas.

3 MÓDULO DE CONTROLE DO MOTOR CC

Esse módulo é composto pelo CI de ponte H L293D para ajustar a tensão média aplicada no motor CC mediante o parâmetro recebido pela interface bluetooth controlando assim, consequentemente, sua velocidade de rotação.

Para isso, esse controle do motor usa a técnica PWM (Pulse Width Modulation), a qual controla a tensão média aplicada ao motor pela variação do tempo em que ele fica ligado em ciclos liga-desliga.

Foi utilizada a interrupção do processador ATMega328P relativa ao gerador de onda quadrada PWM disponível no temporizador do microcontrolador que controla a largura (em tempo) do nível 1 dessa onda quadrada. Não foram utilizadas funções já prontas para PWM do Arduino como analogWrite(), por exemplo.

A função desenvolvida recebe o parâmetro numérico de entrada do comando “VEL xxx*” que pode variar entre 0 e 100, indicando o percentual do período da onda quadrada em que a mesma deve ficar em nível lógico 1, regulando-a.

Foram implementadas 3 funções que em conjunto controlam o sentido de rotação e a velocidade de giro do motor CC: maqEstadosMotor(), rotacaoMotor() e convertPercentPWM().

A função maqEstadosMotor() descreve através da implementação de uma máquina de estados o comportamento que deve ser executado pelo motor CC. Essa função trabalha com duas variáveis: estadoMotor, que indica o estado atual do motor CC na máquina de estados, e a variável comadoExecut, que recebida via comunicação serial. Abaixo temos o diagrama de estados e o pseudocódigo que representam essa função. Note que não existe caminho direto de 1 para 2 ou de 2 para 1 (ou seja, o motor obrigatoriamente deve ter velocidade de rotação igual a zero antes de inverter o sentido de rotação).

A Figura 4 mostra essa máquina de estados implementada.

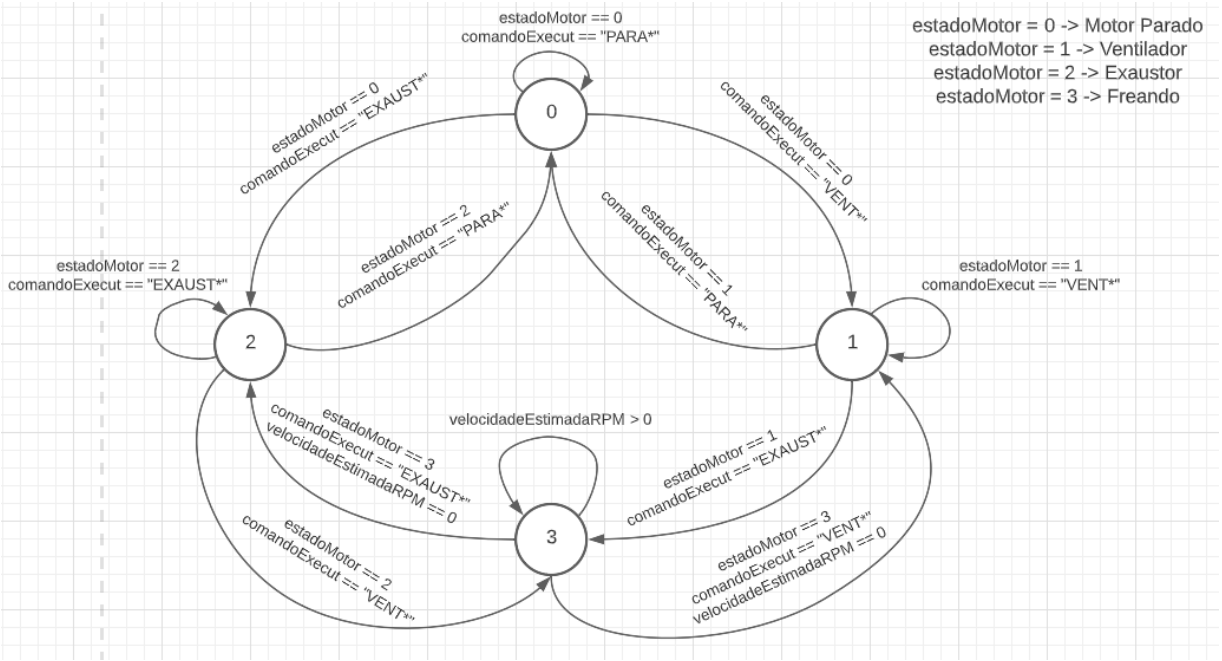

Figura 4: Máquina de estados implementada para o controle do motor CC.

4 MÓDULO DE ESTIMATIVA DE VELOCIDADE

O módulo de estimativa de velocidade é composto pelo codificador óptico PHCT202 e um inversor 7414 com entrada Schmitt Trigger. O codificador é acoplado à hélice do motor e os resistores de 1k Ω e 180Ω foram usados para polarizar o foto-transistor do codificador e limitar a corrente do seu LED, respectivamente.

O inversor foi inserido na saída do codificador para garantir que o sinal de saída do codificador seja digital e, portanto, lido corretamente pelo Arduino como nível alto ou baixo.

O codificador óptico foi conectado na porta 2 do Arduino UNO e a implementação do software se deu utilizando-se basicamente de três mecanismos. O mecanismo de interrupção por borda de subida do pino 2 para detectar toda vez em que a pá do motor passe pelo codificador óptico; um mecanismo de base de tempo implementado por um temporizador com interrupções periódicas de 8 ms; e a função propriamente dita de cálculo da estimativa de velocidade do motor em RPM.

Dessa forma, temos o seguinte funcionamento: a partir da interrupção configurada no pino 2 onde é ligado o codificador, realizamos a contagem de quantas vezes o codificador óptico identifica a passagem da pá, utilizando-se da base de tempo de 8 ms, a cada 200ms utilizamos essa quantidade para estimar a velocidade utilizando-se da seguinte equação:

$$VelocidadeEstimadaRPM = \frac{\left(\frac{60 \times 1000}{Nº\ de\ pás}\right)}{N \times 8} \times Nº\ de\ passagens\ das\ pás$$

Onde 1000/(N*8) representa a quantidade de vezes por segundo em que a velocidade do motor é verificada/estimada. Nesse caso, estamos utilizando N=25 e, portanto, estimamos a velocidade 5 vezes a cada segundo.

Essa equação utilizada, permite estimarmos a velocidade do motor relacionando a quantidade de vezes de passagem das pás pela quantidade total de pás que ele possui. Essa quantidade de passagens das pás foi observada num período e, portanto, precisamos dividir por esse período e, para sairmos da unidade de rotações por milissegundos, precisamos usar o fator 60*1000 para chegarmos na unidade de rotações por minuto (RPM).

A Figura 5 apresenta um fluxograma do funcionamento básico do software implementado.

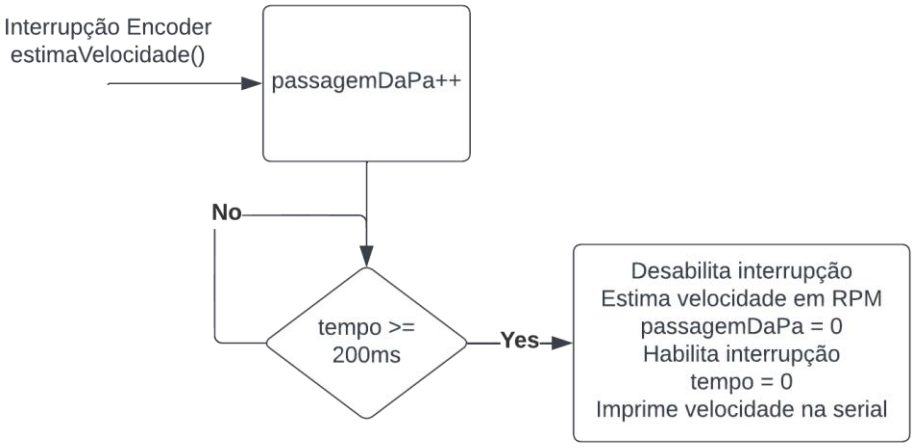

Figura 5: Fluxograma do processo implementado para estimativa da velocidade do motor.

A quantidade de vezes por segundo em que estimamos a velocidade não deve ser grande a ponto do tempo entre uma estimativa e outra ser menor que a base de tempo de 8 ms do código e não deve ser grande a ponto de não identificar nenhuma pá quando o motor estiver em baixas rotações. Desse modo, para permitir uma boa precisão, estamos estimando a velocidade 5 vezes por segundo usando um N=25.

5 MÓDULO DO DISPLAY LCD

Esse módulo é composto pelo LCD (Liquid Crystal Display) de 16 caracteres por 2 linhas (16x2) conectado ao Arduino UNO. Esse display é responsável por receber e exibir a informação da velocidade estimada em RPM do motor em tempo real

A 1ª linha contém “ROTAÇÃO:XXXX RPM” (até 4 dígitos, com as letras RPM sempre nas últimas posições); e, a 2ª linha: (ESTIMATIVA) (centralizada na linha).

Foi utilizada a biblioteca LiquidCrystal para interação da placa com o display e a configuração que emprega apenas 4 terminais de dados do LCD (D4, D5, D6 e D7) para reduzir o número de conexões.

6 MÓDULO DOS DISPLAYS DE 7 SEGMENTOS

Nesse módulo foi utilizado o protocolo I2C para realizar a comunicação entre o Arduino UNO e os displays de 7 segmentos. Esses displays, assim como o LCD, também mostram a velocidade estimada em RPM do motor em tempo real.

Para tal, também foi utilizado o expansor de portas PCF8574 conectado ao Arduino UNO e comunicando com o protocolo I2C. Essa comunicação, através apenas de dois terminais, envia ao expansor de portas um byte (8 bits) contendo em seus 4 dígitos mais significativos a indicação de qual display deve ser acionado e, nos seus 4 dígitos menos significativos, o valor a ser exibido nesse display.

Portanto, os 4 dígitos mais significativos da saída do expansor de portas são conectados diretamente ao display de quatro dígitos 5461AS para controlar qual dígito será ativado e os 4 dígitos menos significativos a um decodificador de display de 7 segmentos que converte esses 4 dígitos nos 7 sinais respectivos de cada segmento. Também foram acrescentados resistores de 180Ω em cada segmento para limitar a corrente dos LEDs

7 LISTA DE COMPONENTES

A Tabela 2 exibe a lista de componentes utilizados no hardware do sistema.

Tabela 2: Componentes utilizados.

COMPONENTE	QUANTIDADE
Arduino UNO	1
Fonte CC para alimentação externa	1
L293D (ponte H)	1
Motor CC	1
Resistor 1K Ohms	2
Resistor 2K Ohms	1
Resistor 180 Ohms	8
Display LCD 16 x 2	1
Potenciômetro 1K Ohm	1
PCF8574 (Expansor de portas I2C)	1
5461AS (Display 7 seg. de 4 dígitos)	1
Decodificador 7447	1
PHCT 202 (Codificador óptico)	1
HC-05 (Módulo Bluetooth)	1

8 ESQUEMÁTICO DO HARDWARE

A seguir é apresentado o esquemático do hardware implementado para esse sistema.

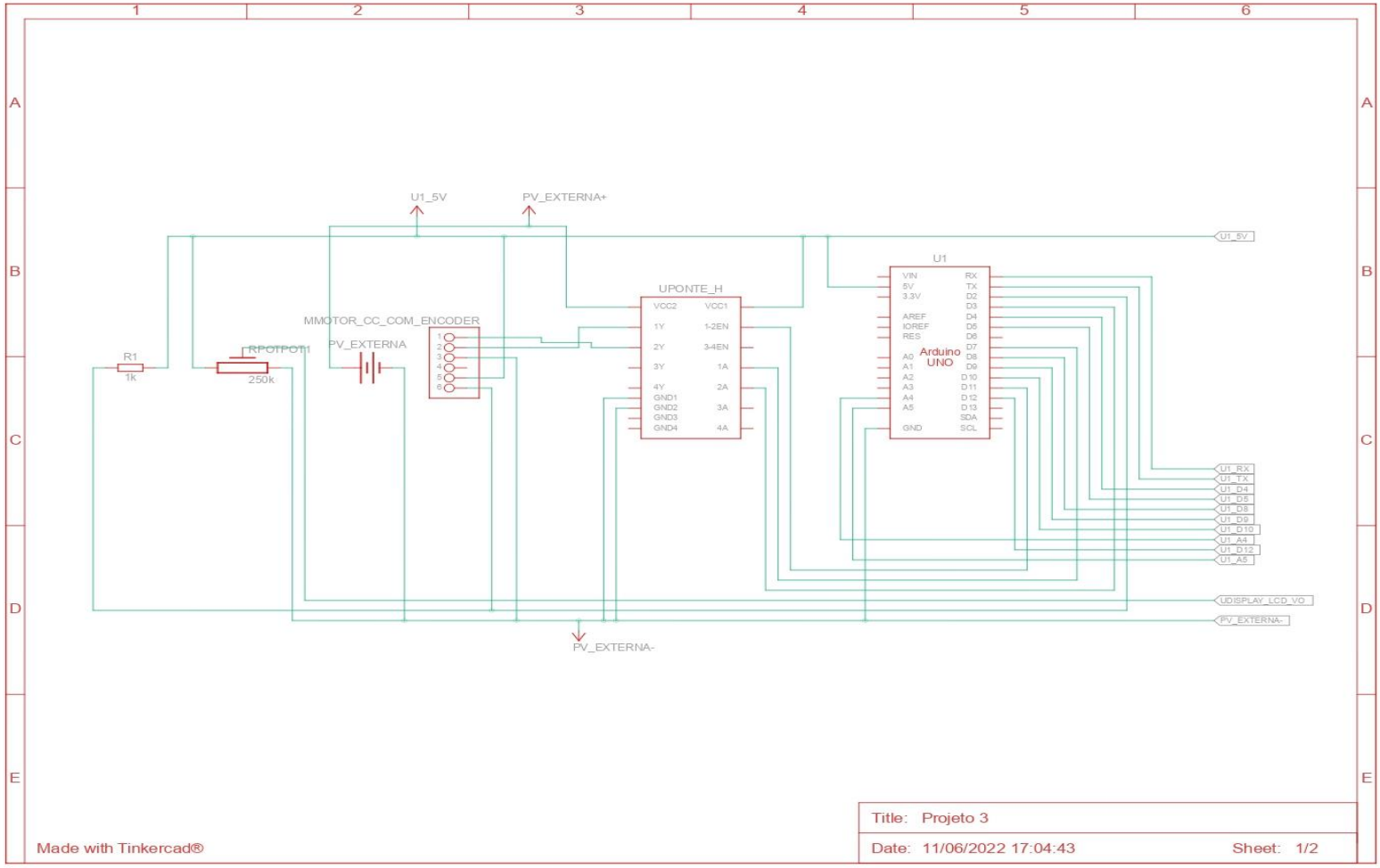

Figura 6: Esquemático do hardware implementado (pt.1).

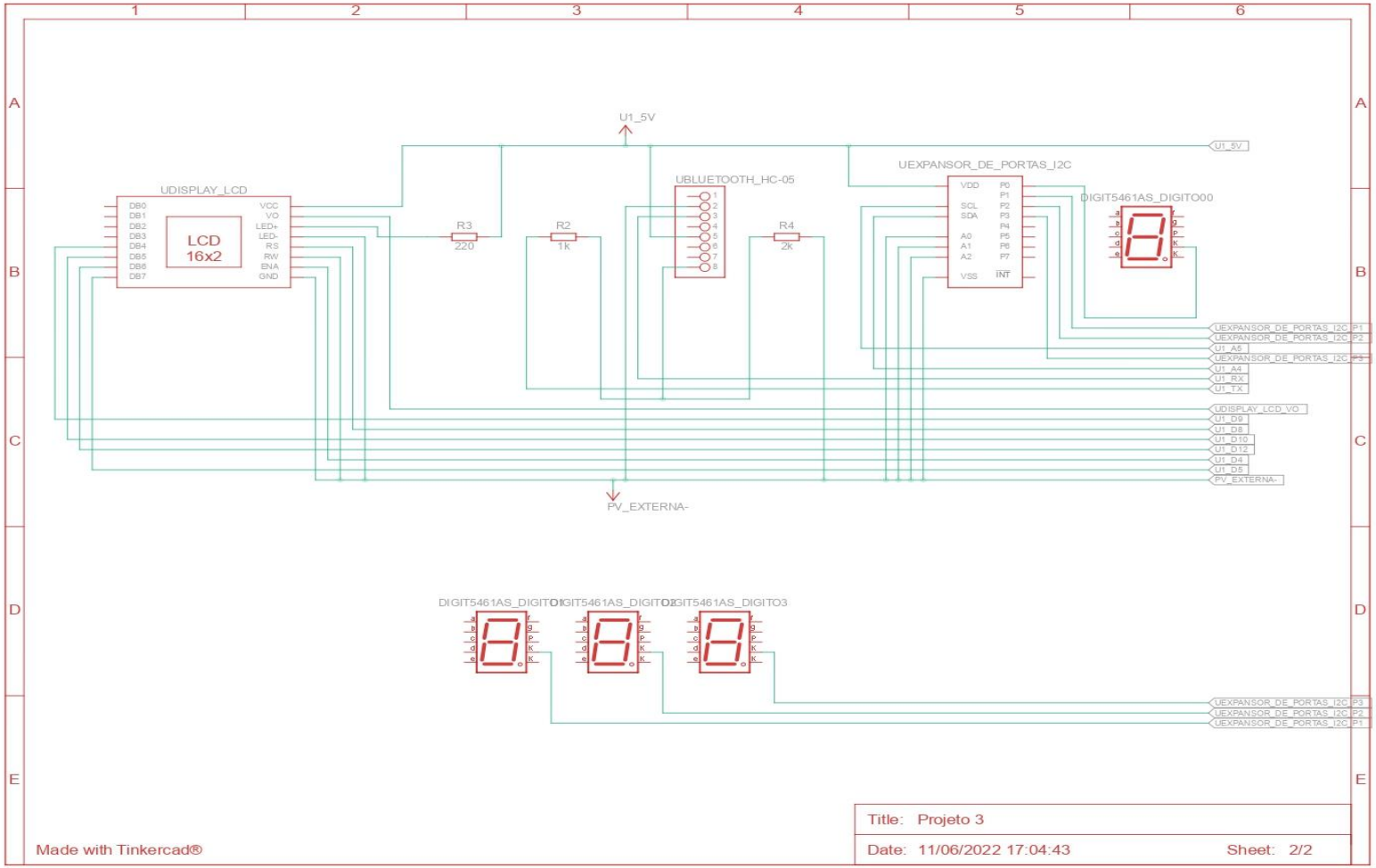

Figura 7: Esquemático do hardware implementado (pt.2).

9 VÍDEO DE APRESENTAÇÃO

Acesse o link abaixo para o vídeo de apresentação do funcionamento do sistema: https://youtu.be/U_MbJUPAZeM

10 CÓDIGO COMENTADO

```
*Projeto final*/
/*Welter Mompeam Sozin            RA: 188625*/
/*Guilherme Augusto Amorim Terrell          RA: 168899*/

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

/*Definição dos pinos*/
#define pinEntrada1PonteH 3
#define pinEntrada2PonteH 7
#define pinEncoder 2
#define valorMaximoPWM 254
#define pinRS 8
#define pinEnable 4
#define pinD4 9
#define pinD5 10
#define pinD6 12
#define pinD7 5
/*Pino de dados para comunicação I2C*/
#define pinSDA A4
/*Pino de clock para comunicação I2C*/
#define pinSCL A5
/*Pino 13 deve ter seu estado lógico alterado a cada chamada decomando da biblioteca Wire*/
#define inverteEstadoWire 13

/*Para fins de simução a velocidade começa em 100 e o estadoMotor começa em 1*/
int velocidadeMotorPWM = 0;
int estadoMotor = 0; /*0 -> motor parado; 1 -> ventilador; 2 -> exaustor; 3 -> freio*/
String comandoExecut = "";

/*Variáveis para estimar velocidade RPM*/
unsigned int pulsosPorVolta = 2; // Numero de pas/furos no encoder
unsigned int velocidadeEstimadaRPM = 0;
int counter = 0;
volatile unsigned long passagemDaPa = 0;
unsigned long verificaVelocidade = 25;
/*------------------------------*/

/*variáveis da parte 1*/
/*----------------------*/
char carac = '*';
String comando = "";
int velocidade = 0;
/*----------------------*/
```

```
/*variáveis da parte 3*/
/*LCD -> 2 linhas de 16 caracteres*/
LiquidCrystal lcd(pinRS, pinEnable, pinD4, pinD5, pinD6, pinD7);

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

/*Número de 1 byte a ser escrito nos 4 LED via comunicação I2C*/
byte numeroI2C = 0;
byte enderecoPCF8574P = 32;

/*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;

/*Função que converte o valor contido na string comando*/
/*em um valor dentro da escala PWM (0, 254)*/
void convertPercentPWM(){
  if(comandoExecut.substring(0,4) == "VEL "){
    velocidadeMotorPWM = (velocidade*valorMaximoPWM)/100;
    //Serial.println(velocidadeMotorPWM);
  }
}

/*Função que define o funcionamento do motor CC*/
/*comando = VENT* -> motor opera como ventilador (girar sentido horário)*/
/*comando = EXAUST* -> motor opera como exaustor (girar sentido anti-horário)*/
/*comando = PARA* -> motor parado*/

void maqEstadosMotor(){
  if(comandoExecut == "PARA*"){
    /*Motor parado*/
    estadoMotor = 0;
  }
  if((estadoMotor == 0) && (comandoExecut == "VENT*")){
    /*motor que estava parado deve operar como ventilador*/
    estadoMotor = 1;
    comandoExecut = "";
  }
  if((estadoMotor == 0) && (comandoExecut == "EXAUST*")){
    /*motor que estava parado deve operar como exaustor*/
    estadoMotor = 2;
    comandoExecut = "";
  }
  if((estadoMotor == 1) && (comandoExecut == "VENT*")){
    /*continuar operando como ventilador*/
    estadoMotor = 1;
    comandoExecut = "";
  }
  if((estadoMotor == 2) && (comandoExecut == "EXAUST*")){
    /*contnuar operando como exaustor*/
    estadoMotor = 2;
    comandoExecut = "";
  }
  if((estadoMotor == 1 || estadoMotor == 3) && (comandoExecut == "EXAUST*")){
```

```
    /*motor que estava operando como ventilador deve operar como exaustor*/
    /*Freando o motor*/
    estadoMotor = 3;
    /*Mudar sentido de rotação apenas apos parar de girar*/
    if(velocidadeEstimadaRPM == 0){
      estadoMotor = 2;
    }
    /*Caso não tenha parado deve continuar freando*/
    if(velocidadeEstimadaRPM > 0){
      estadoMotor = 3;
    }
  }
  if((estadoMotor == 2 || estadoMotor == 3) && (comandoExecut == "VENT*")){
    /*motor que estava operando como exaustor deve operar como ventilador*/
    /*Freando o motor*/
    estadoMotor = 3;
    /*Mudar sentido de rotação apenas apos parar de girar*/
    if(velocidadeEstimadaRPM == 0){
      estadoMotor = 1;
    }
    /*Caso não tenha parado deve continuar freando*/
    if(velocidadeEstimadaRPM > 0){
      estadoMotor = 3;
    }
  }
}

/*Função que habilita o sentido de rotação de acordo com o estado do motor*/
void rotacaoMotor(){
  if(estadoMotor == 0){
    /*motor desligado*/
    digitalWrite(pinEntrada1PonteH, LOW);
    digitalWrite(pinEntrada2PonteH, LOW);
    velocidadeMotorPWM = 0;/*zerar velocidade*/
  }
  if(estadoMotor == 1){
    /*Motor como ventilador*/
    digitalWrite(pinEntrada1PonteH, LOW);
    digitalWrite(pinEntrada2PonteH, HIGH);
    /*Deve receber parâmetro de velocidade do comando VEL XXX**/
  }
  if(estadoMotor == 2){
    /*Motor como exaustor*/
    digitalWrite(pinEntrada1PonteH, HIGH);
    digitalWrite(pinEntrada2PonteH, LOW);
    /*Deve receber parâmetro de velocidade do comando VEL XXX**/
  }
  if(estadoMotor == 3){
    /*Frear motor*/
    digitalWrite(pinEntrada1PonteH, HIGH);
    digitalWrite(pinEntrada2PonteH, HIGH);
    OCR2A = 0;
  }
}

/*Funções da parte 1*/
/*-------------------------------------------------------------*/
```

```
// Função: Lê caractere e concatena no comando
void LerComando()
{
  carac = Serial.read();      // Lê caractere
  if (carac!=-1 && carac!='\n' && carac!='\r'){        // Se for digitado algo, concatena no comando
    comando.concat(carac);
    comandoExecut = comando;
  }
}

// Função: Decodifica o comando
void DecodComando()
{
  if (comando.indexOf('*')!=-1){  // Verifica se o caractere de fim de comando está presente
    // Interpretação dos comandos
    if (comando=="RETVEL*"){
      Serial.println("VEL: X RPM");
    }
    else if (comando=="PARA*"){
      Serial.println("OK PARA");
    }
    else if (comando=="EXAUST*"){
      Serial.println("OK EXAUST");
    }
    else if (comando=="VENT*"){
      Serial.println("OK VENT");
    }
    else if (comando.substring(0,4)=="VEL "){
      // Comando = "VEL *"
      if (comando.substring(4,5)=="*"){
        Serial.println("ERRO: PARAMETRO AUSENTE");
        velocidade = -1;
      }
      // Comando = "VEL xx*" ou "VEL x*"
      else if (comando.substring(4).length()!=4){
        Serial.println("ERRO: COMANDO INEXISTENTE");
      }
      // Comando = "VEL xxx*"
      else{
        velocidade = comando.substring(4,7).toInt();
        if ((velocidade>=0) && (velocidade<=100)){
          Serial.println("OK VEL " + comando.substring(4,7) + "%");
        }
        else{
          Serial.println("ERRO: PARAMETRO INCORRETO");
        }
      }
    }
    else{
      Serial.println("ERRO: COMANDO INEXISTENTE");    // Comando inexistente
      Serial.println(comando);
    }
    comando="";                  // Reinicializa para receber um novo comando
  }
}

/*---------------------------------------------------------------*/
```

```
/*Funções para estimar velocidade*/

// Rotina de servico de interrupcao do temporizador 0
ISR(TIMER0_COMPA_vect){
  counter++;
  displaySelecionado++;
}

void configuracao_Timer0(){
  ////////////////////////////////////////////////////////////////////////////////////////////////////
  // Configuracao Temporizador 0 (8 bits) para gerar interrupcoes periodicas a cada 8ms no modo Clear Timer on
Compare Match (CTC)
  // Relogio = 16e6 Hz
  // Prescaler = 1024
  // Faixa = 125 (contagem de 0 a OCR0A = 124)
  // Intervalo entre interrupcoes: (Prescaler/Relogio)*Faixa = (64/16e6)*(124+1) = 0.008s

  // 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   1
  TCCR0B = 0x05;
  ////////////////////////////////////////////////////////////////////////////////////////////////////
}

// Rotina de servico de interrupcao do pinEncoder
void estimaVelocidade(){
  passagemDaPa++;
}

/*Configuração do sinal PWM no Timer2*/
void configuracao_Timer2(){
  ////////////////////////////////////////////////////////////////////////////////////////////////////
  // Configuracao Temporizador 2 (8 bits) para gerar sinal PWM no pino 11 e pino 3

  // TCCR2A – Timer/Counter Control Register A
  // COM2A1 COM2A0 COM2B1 COM2B0 – – WGM21 WGM20
  // 1     0     1     0         1     1
  DDRB = DDRB | (1<<3);/*Configura pino 11 do PORTB como saída*/
  TCCR2A = 0xA3;/*fast PWM*/

  /*OCR2A – Output Compare Register A*/
  OCR2A = velocidadeMotorPWM;
```

```
  // TCCR2B – Timer/Counter Control Register B
  // FOC0A FOC0B – – WGM02 CS02 CS01 CS0
  // 0    0       0    1   0   0
  TCCR2B = 0x05;/*prescaler*/
  TCNT2 = 0x00; /*Inicializar o Timer2*/
  ////////////////////////////////////////////////////////////////////////////////////////////////////////////
}


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

/*Funções da parte 3 (LCD)*/
void escreveLCD(unsigned int velocidade){
  if(String(velocidade).length() <= 4){
    lcd.setCursor(0,0);
    lcd.print(String(velocidade));
    lcd.setCursor(13,0);
    lcd.print("RPM");
    lcd.setCursor(2,1);
    lcd.print("(ESTIMATIVA)");
    //Serial.println(String(velocidadeEstimadaRPM));
  }
  else{
    lcd.setCursor(0,0);
    lcd.print("ERROR");
  }
}
/*------------------------*/

/*Funções parte I2C*/

unsigned int digito0 = 0;
unsigned int digito1 = 0;
unsigned int digito2 = 0;
unsigned int digito3 = 0;

void selecionaDisplay(){
  //Wire.beginTransmission(enderecoPCF8574P);
  if ((displaySelecionado)%4==0){
    digito0 = velocidadeEstimadaRPM%1000;
    Wire.write(display0+digito0);
  }
  else if ((displaySelecionado)%4==1){
    digito1 = (velocidadeEstimadaRPM-digito0*1000)%100;
    Wire.write(display1+digito1);
  }
  else if ((displaySelecionado)%4==2){
    digito2 = (velocidadeEstimadaRPM-digito0*1000-digito1*100)%10;
    Wire.write(display2+digito2);
  }
  else{
    digito3 = (velocidadeEstimadaRPM-digito0*1000-digito1*100-digito2*10);
    Wire.write(display3+digito3);
  }
  //Wire.endTransmission(enderecoPCF8574P);
}
```

```
void setup()
{
  pinMode(pinEncoder, INPUT);
  pinMode(pinEntrada1PonteH, OUTPUT);
  pinMode(pinEntrada2PonteH, OUTPUT);
  pinMode(inverteEstadoWire, OUTPUT);
  Serial.begin(9600);
  Wire.begin();/*Configura o barramento do I2C*/
  cli();
  configuracao_Timer0();
  sei();
  configuracao_Timer2();
  /*Habilita interrupcao do encoder*/
  attachInterrupt(digitalPinToInterrupt(pinEncoder), estimaVelocidade, RISING);
  /*Iniciar e limpar o LCD*/
  lcd.begin(16,2);
  lcd.clear();
}

void loop()
{
  //_delay_ms(1);
  LerComando();
  DecodComando();
  maqEstadosMotor();
  rotacaoMotor();
  convertPercentPWM();
  OCR2A = velocidadeMotorPWM;
  escreveLCD(velocidadeEstimadaRPM);

  // Escreve o dígito de RPM no display selecionado do mostrador
  //Serial.println(displaySelecionado);
  Wire.beginTransmission(enderecoPCF8574P);
  selecionaDisplay();
  Wire.endTransmission(enderecoPCF8574P);

  // Habilita a interrupcao do encoder em um tempo menor de 200ms
  if(counter < verificaVelocidade){
    attachInterrupt(digitalPinToInterrupt(pinEncoder), estimaVelocidade, RISING);
  }
  /* counter = 125 -> 1s */
  /* counter = 25 -> 0.2s */
  /* Durante 0.2s segundo incremento passagemDaPa */
  /* passageDaPa indica o número de voltas foram dadas em 0.2s */
  /* Multiplicar por 5 para estimar a rotação em 1s */
  if(counter >= verificaVelocidade){
    detachInterrupt(digitalPinToInterrupt(pinEncoder)); // Desabilita interrupcao para calcular a velocidade
    velocidadeEstimadaRPM = ((unsigned int)60*1000/pulsosPorVolta)/(verificaVelocidade*8) * passagemDaPa;
    passagemDaPa = 0;
    attachInterrupt(digitalPinToInterrupt(pinEncoder), estimaVelocidade, RISING); // Habilita interrupcoes apos
calculo da velocidade
    counter = 0;
    /*Para evitar que o último caractere de uma velocidade n+1 digitos*/
    /*Apareça em uma velocidade de n dígitos*/
    /*Devemos limpar o LCD*/
```

```
    lcd.clear();
  }
  //Serial.println(estadoMotor);
  //Serial.println(OCR2A);
}
```