ID: Daniel Costa 112185

Projeto 02 de Sistemas Embarcados: simulação de exaustão automática

São José dos Campos - Brasil

2019

ID: Daniel Costa 112185

Projeto 02 de Sistemas Embarcados: simulação de exaustão automática

Relatório apresentado à Universidade Federal de São Paulo como parte dos requisitos para aprovação na disciplina de Sistemas Embarcados.

Docente: Prof. Dr. Sérgio Ronaldo Barros dos Santos

Universidade Federal de São Paulo - UNIFESP

Instituto de Ciência e Tecnologia - Campus São José dos Campos

São José dos Campos - Brasil

2019

Resumo

O projeto em questão teve como visa a implementação de um sistema de exaustão simulado, controlado por microcontroladores presentes presentes nas placas *Arduino Mega* e *Kit PICGenios PIC18F*.

Palavras-chaves: Arduíno. Embarcados. Automação. Residência.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Com a evolução da tecnologia, muitas tarefas são deixadas de serem realizadas por humanos, sendo passadas para uma tecnologia capaz de automatizar processos ou, ao menos, facilitar esse. A residência não foge desse padrão, considerando tempos prévios aos anos 2000, pode-se dizer que, para a época, a aparição do controle remoto em televisões tenha seguido a linha de uma automação.

A automação residencial pode oferecer diversos benefícios desde que aplicada de maneira correta. Diversos dispositivos podem ser empregados para reduzir consumo de energia, baseado na luminosidade por exemplo, melhorar a segurança, como sensores de movimento, e, talvez a maior motivação para a procura de uma automação residencial, o menor gasto e preocupação com diversas atividades rotineiras, agora feitas por dispositivos.

2 Objetivos

2.1 Geral

O objetivo desse projeto visa a simulação, utilizando uma placa Arduíno, composta por um microcontrolador ATmega 2560, e um *Kit* de desenvolvimento PICGenios PIC18F, contendo um microcontrolador PIC18F4520, ambas juntos há IDEs específicas, de uma automação de exaustão.

2.2 Específico

- Compreender os dispositivos à serem usados.
- Implementar o algoritmo.
- Simular o funcionamento.

3 Fundamentação Teórica

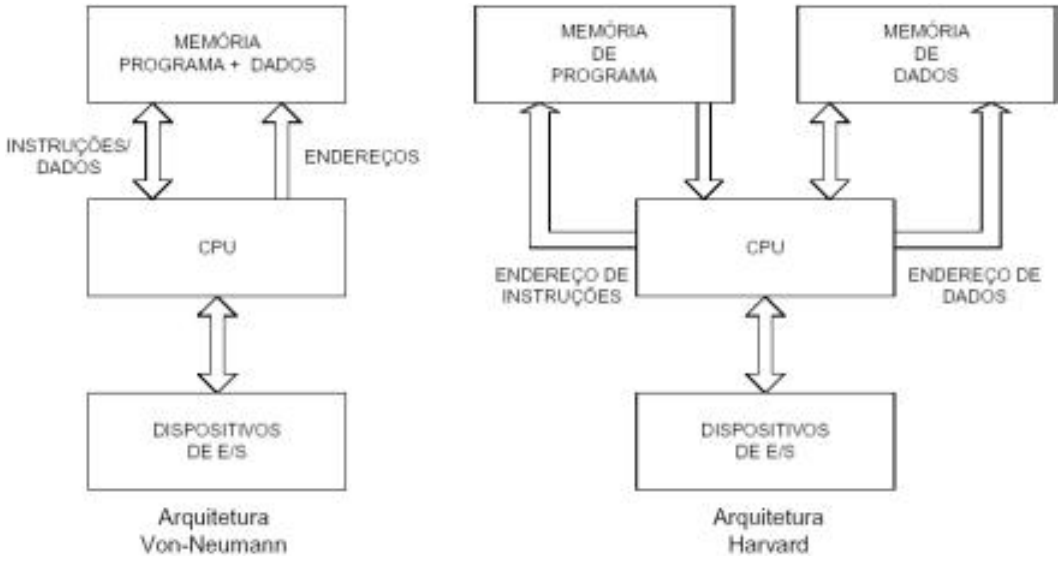
Alguns dispositivos, mais comuns, não serão relatados, como resistores, leds, botões, cabos, entre outros.

3.1 Modelos de arquitetura

Como principais modelos de estudo, temos as arquiteturas de Von Neumann e de Harvard. A primeira trata-se de uma arquitetura baseada em uma organização com uma única unidade de memória, tanto para dados quanto para instruções, além de possuir a unidade lógica e aritmética e uma unidade de controle. Já a arquitetura de Harvard diferencia-se da de Von Neumann na questão da memória, já que divide a memória em duas, uma para dados e outra para instruções.

A unidade de controle tem como função controlar os outros componentes, por meio de sinais que geram bloqueios ou não a um determinado sinal. A unidade lógica e aritmética é responsável por realizar as operações de lógica e aritmética, normalmente controlada por meio de sinais vindos da unidade de controle. Tem-se ainda os módulos de entrada e saída.

Figura 1 – Arquiteturas Von Neumann e Harvard



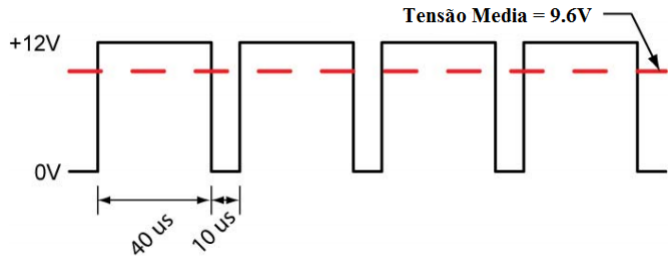
Fonte: Os poderosos microcontroladores AVR (IFSC)

Há ainda um outro modelo de arquitetura, conhecido como arquitetura Harvard modificada, essa arquitetura apresenta-se assim como uma Harvard, porém, sua memória

programa, ou de instrução, pode ser utilizada como uma memória de dados caso necessário. Tal arquitetura é a utilizada nas placas Arduíno.

3.2 PWM *Pulse Width Modulation*

Figura 2 – PWM



Fonte: MecaWeb - PWM

A Modulação por Largura de Pulso dá-se através do tempo de acionamento de de tensão durante um ciclo de *clock*, ou seja, de acordo com o tempo em que o sinal se manteve em valor alto, um valor de tensão será aplicado na saída do circuito. Como exemplo, caso um pulse mantenha se alto por 50% do tempo, o valor aplicado de saída será de 0.5 da tensão de entrada. O *Duty Cycle* nada mais é do que a conversão, do tempo em que o pulso manteve-se em valor alto, para porcentagem.

3.3 Placa Arduíno composta por ATmega2560

3.3.1 Microcontrolador e pinos

As placas Arduíno apresentam microcontroladores da família AVR, mais especificamente, megaAVR, apresentando como características uma memória flash de 8kB à 256kB, 4kB de EEPROM e 8kB de SRAM, entre 23 e 86 pinos de E/S, comparadores e canais PWM, ADC, I2C, USART e SPI.

Os Arduínos que apresentam o microcontrolador ATmega2560, possuem 54 pinos de entradas e saídas digitais (DPIO), sendo que 15 podem operar como pinos de modulação de largura de pulso (PWM), 16 pinos analógicos, além de pinos para comunicações Serial, *Serial peripheral interface protocol* (SPI) e *I2C*. Os pinos para realizar comunicação podem ser emulados em pinos não definidos para esse popósito com bibliotecas. Apresenta ainda pinos para alimentação.

3.4 Comunicação Serial

A comunicação Serial é utilizada para comunicação entre a placa Arduino e outros dispositivos, como o computador, os pinos são descritos como RX e TX na placa. A comunicação Serial feita pelo Arduino Mega é do tipo USART (Universal Synchronous/Asynchronous Receiver/Transmitter). A transmissão dos dados ocorre com a entrada de dados paralelos, os quais são convertidos em bits através de uma onda quadrada, tendo seu limite marcado por bits de inicialização e de término de dados.

3.5 KIT PICGenios PIC18F

O KIT PICGenios PIC18F é composto por inúmeros periféricos (leds, botões, ventoinha e outros) controlados pelo microcontrolador PIC18F4520.

3.5.1 PIC18F4520

O microcontrolador PIC18F4520 possui uma arquitetura do tipo Harvard e um conjunto de instruções do tipo RISC. Além disso apresenta 40 pinos, 83 instruções, 35 pinos de entrada e saída, 20 interrupções. Memória FLASH de 32k, SRAM de 2k e EEPROM de 256 bytes, cristal de 40 MHz, 2 canais comparadores/PWM, SPI, I2C, USART. 13 canais ADC (10 bits) e 4 temporizadores/contadores (TIMER0, TIMER1, TIMER2, TIMER3), sendo de 8 bits e 16 bits.

3.5.1.1 Conversor analógico digital

No microcontrolador há 13 canais A/D de 10 bits contendo 3 registradores para as devidas configurações (ADCONx).

O registrador ADCON0 possui 8 bits, sendo 6 para configuração. Da direta para esquerda tem-se:

- ADON: liga (quando 1) o conversor
- GO_DONE: conversão está sendo realizada (quando 1).
- CHSx: 4 bits de configuração do canal AD

O registrador ADCON1 possui 8 bits, sendo 6 para configuração. Da direta para esquerda tem-se:

- PCFGx: configura o A/D
- VCGF0: configura tensão de referência
- VCGF1: configura tensão de referência

O registrador ADCON2 possui 8 bits, sendo 7 para configuração. Da direta para esquerda tem-se:

- ADCSx: 3 bits para seleção do clock do conversor
- ACQTx: 3 bits para seleção do tempo de aquisição.
- Bit 6: nulo
- ADFM: formatação do resultado da conversão

3.5.1.2 Temporizadores e contadores

O microcontrolador em questão possui 4 temporizadores/contadores sendo o TIMER0 de 8 ou 16 bits (contagem de até 255 ou 65535) e o restante, TIMER1, TIMER2 e TIMER3, de 16 bits. São responsáveis por realizar contagens de eventos e temporizar. Como o microcontrolador possui um clock de 8MHz e cada instrução leva 4 ciclos para completar, então temos um contador a uma frequência máxima de máximo de 2MHz de saída, ou 0,5 microssegundo de ciclo de máquina. Os valores iniciais são armazenados em TMRLx e TMRHx (parte baixa e parte alto do valor em hexadecimal). Tem-se ainda um prescalar reponsável por dividir a frequência.

Considerando apenas o timer 0 (usado no projeto), temos o registrador T0CON, responsável pela configuração do timer 0. Tal registrador apresenta 8 bits sendo eles, em ordem de configuração dos bits da esquerda para a direta:

- TMR0ON: ativação (quando 1) do timer 0
- T08BIT: contagem com 16 bits (quando 0) e contagem de 8 bits (quando 1)
- T0CS: clock externo (quando 1) ou 2MHz (quando 0)
- T0SE: seleciona se a contagem ocorre em borda de subida ou descida
- PSA: ativa (0) ou desativa (1) o prescalar
- T0PSx: 3 bits responsáveis pela definição do valor do prescalar, sendo o valor uma potência de 2 variando de 2 (000) até 256 (111).

3.5.1.3 Interrupção

O registrador INTCON controla as interrupções do microcontrolador. Tal registrador apresenta 8 bits sendo eles, em ordem de configuração dos bits da esquerda para a direta:

- GIE/GIEH: ativação (quando 0) da chave geral de interrupções
- GIEL: ativação (quando 1) das interrupções dos periféricos

- TMR0IE: habilita (quando 1) a interrupção do TIMER0 quando há estouro de contagem
- RBIE: Ativa as interrupções de mudança de estado (RB4,RB5,RB6,RB7)
- TMR0IF: sinaliza (quando 1) estoura na contagem do timer 0
- RBIF: sinaliza interrupção de RB (RB4,RB5,RB6,RB7).

3.6 Dispositivos periféricos

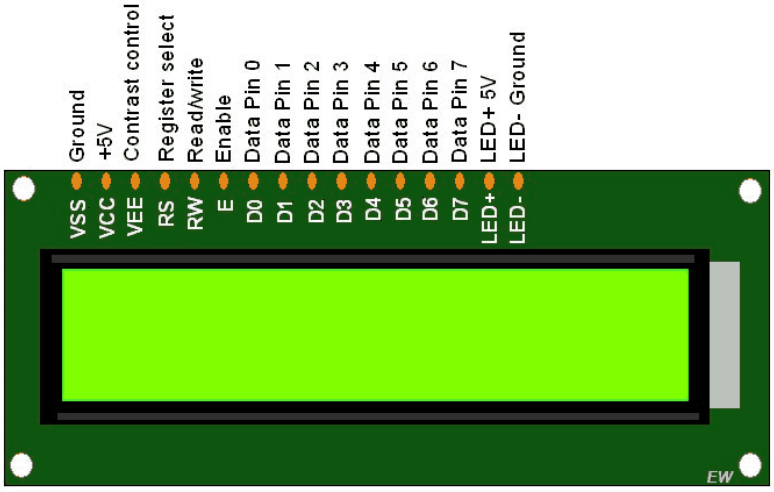
No projeto foram usados os seguintes componentes contido na placa de desenvolvimento PICGenios PIC18F:

- *Display* LCD 16x2
- *Buzzer*
- Sensor de temperatura LM35
- Ventoinha
- Aquecedor (resistência)

Junto à placa Arduino Mega foi utilizado um Módulo Bluetooth HC-05.

3.6.1 *Display LCD*

Figura 3 – LCD 16x2



Fonte: LCD 16x2 custom character display using LPC2148

O *display* LCD possui várias matrizes de pontos a fim de gerar um símbolo. Uma memória ROM interna armazena caracteres alfabéticos, fazendo a conversão dos dados

enviados em bits para um caractere específico. Possuem um RAM de armazenamento de dados de 80 bytes, sendo capaz de armazenar 80 caracteres de dado.

Com relação a pinagem mostrada na Figura 3, os pinos de D0, mais significativo, até D7 são os pinos de dados, que podem operar em 4 bits (D4 até D7) ou 8 bits (D0 até D7). O pino RS informa se o lcd está em modo de dado ou instrução, nível alto e baixo, respectivamente. O RW indica se haverá leitura ou escrita, nível alto e baixo, respectivamente. O pino E ativa ou desativa a transmissão de dados, nível alto e baixo, respectivamente. Os demais pinos são de alimentação e controle da luz de fundo e contraste.

3.6.2 Módulo Bluetooth HC-05

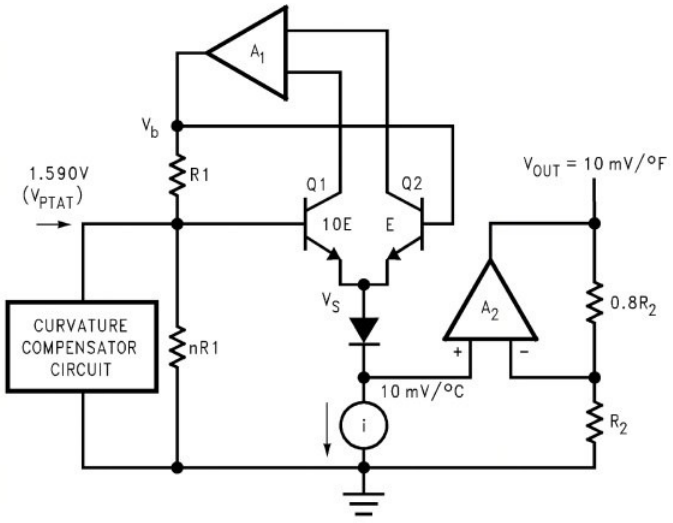
Figura 4 – Módulo Bluetooth HC-05

Fonte: RG Retail HC05 Bluetooth Module for Arduino

O Módulo Bluetooth utiliza comunicação Serial, utilizando os pinos RX e TX. O pino EN coloca em modo de dados ou de comando AT. TX e RX, transmitem e recebem dados respectivamente. O pino STATE, serve apenas para mostrar o funcionamento do dispositivo, é também conectado ao led de estado presente.

3.6.3 Sensor de temperatura LM35

Figura 5 – Circuito LM35

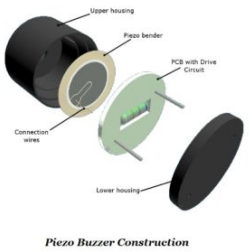


Fonte: How does LM35 work

O sensor LM35 visa medir a temperatura de um objeto. Tem como funcionamento base a passagem de tensão pelo diodo, com maiores tensões (temperaturas mais altas) há queda de tensão entro os terminais receptor e emissor de transistor, com a amplificação do sinal tem-se o valor analógico de saída proporcional à temperatura.

3.6.4 Buzzer

Figura 6 – Buzzer



Fonte: Buzzer

Um *buzzer* trata-se de um material piezo elétrico (cerâmica ou cristal) o qual deforma-se de acordo com uma tensão aplicada a este, essas deformações continuas geram ondas a níveis sonoros.

3.6.5 Ventoinha

Através de um canal PWM é possível controlar a velocidade do giro da ventoinha

3.6.6 Aquecedor (*Heater*)

O Aquecedor nada mais é do que uma resistência dissipadora de temperatura.

4 Desenvolvimento

Inicialmente houve a busca por informações, basicamente presente no Capítulo 3. Com isso, testes individuais foram feitos, utilizando diversos algoritmos e bibliotecas, até ao algoritmo que será mostrado a seguir.

No caso do projeto, a ventoinha simula uma exaustão de acordo com a temperatura capturada pelo sensor LM35, a obtenção da temperatura dá-se através do calculo 100*(tensão de entrada)/(1023), considerando um conversor A/D de 10 bits. Para valores acima de 35 graus a ventoinha gira em potência média, controlada pelo PWM, e acima de 40 em velocidade máxima, porém a ativação é feita em intervalos de 10 segundos e com duração de 10 segundos. Para controlar e observar a temperatura foi utilizado a resistência para elevar a temperatura e o display LCD para observar o valor em graus Celsius capturada pelo sensor.

À placa Arduino Mega foi conectado o módulo Bluetooth, o qual será responsável por enviar informações para um smartphone contendo ao valor da temperatura e o estado atual da ventoinha. Como os periféricos do parágrafo acima estão todos no kit PICGenios, houve-se a a necessidade de realizar uma comunicação Serial entre o PIC18F e o Arduino. A seguir é mostrado os códigos utilizados para o funcionamento do projeto.

PIC18F A implementação do código para o PIC começa com a configuração do display LCD. Além disso variáveis para armazenar o valor, já convertido, lido do sensor de temperatura (temp), armazenar o valor da temperatura em uma string (temper[15]) e realizar contagem (cont).

Em seguida na a função interrupt(), já pré estabelecida para interrupções, há a verificação do estouro do timer 0, e frente a um estouro, o incremento de *cont* e reinicialização do Timer 0.

Já na função main() há a configuração do conversor AD seguindo o subseção 3.5.1.1, configuração do canal UART1 (fará comunicação com o arduino), configuração inicial do display LCD, configuração da resistência e do *buzzer*, configuração do PWM e configuração do Timer 0. O Timer 0 e as interrupções está configurado para que o estouro ocorra a cada 1 segundo seguindo as informações passados na subseção 3.5.1.2 e subseção 3.5.1.3.

No laço infinito while(), a o acionamento da resistência (será ativada de fato somente via ativação das chaves no kit), a obtenção da temperatura já convertida e transferida a um vetor de caracteres. Na condicionais seguintes, há a adição de um byte ao vetor para saber se o valor de *cont* é maior que 0 ou não a fim de controlar as mensagens enviadas por Bluetooth posteriormente. Em seguida há o envio via UART1 do vetor

temper contendo a temperatura e o byte de controle. Logo após, há as configurações para mostrar a temperatura no display LCD.

As condicionais seguintes tratam a respeito do funcionamento da ventoinha. Caso a temperatura seja maior do que 35 graus Celsius e a variável *cont* seja maior que 0 (tal variável é incrementada a cada estouro do temporizador 0) há o acionamento em potência média, e caso a temperatura seja superior a 40 a ventoinha é ligada em potência máxima. Caso o a temperatura seja menor que 35 ou *cont* seja maior que 10, o valor de *cont* é colocado em -10 para que há a pausa de 10 segundos da ventoinha, já que essa funciona apenas se *cont* for maior que 0.

Na condicional final há o acionamento do *buzzer* caso a temperatuara seja maior que 40 graus Celsius.

```
1  // --- Liga??es entre PIC e LCD ---
2  sbit LCD_RS at RE2_bit;   // PINO 2 DO PORTD LIGADO AO RS DO DISPLAY
3  sbit LCD_EN at RE1_bit;   // PINO 3 DO PORTD LIGADO AO EN DO DISPLAY
4  sbit LCD_D7 at RD7_bit;  // PINO 7 DO PORTD LIGADO AO D7 DO DISPLAY
5  sbit LCD_D6 at RD6_bit;  // PINO 6 DO PORTD LIGADO AO D6 DO DISPLAY
6  sbit LCD_D5 at RD5_bit;  // PINO 5 DO PORTD LIGADO AO D5 DO DISPLAY
7  sbit LCD_D4 at RD4_bit;  // PINO 4 DO PORTD LIGADO AO D4 DO DISPLAY
8
9  // Selecionando dire??o de fluxo de dados dos pinos utilizados para a comunica??o com
       display LCD
10 sbit LCD_RS_Direction at TRISE2_bit;  // SETA DIRE??O DO FLUXO DE DADOS DO PINO 2 DO
       PORTD
11 sbit LCD_EN_Direction at TRISE1_bit;  // SETA DIRE??O DO FLUXO DE DADOS DO PINO 3 DO
       PORTD
12 sbit LCD_D7_Direction at TRISD7_bit;  // SETA DIRE??O DO FLUXO DE DADOS DO PINO 7 DO
       PORTD
13 sbit LCD_D6_Direction at TRISD6_bit;  // SETA DIRE??O DO FLUXO DE DADOS DO PINO 6 DO
       PORTD
14 sbit LCD_D5_Direction at TRISD5_bit;  // SETA DIRE??O DO FLUXO DE DADOS DO PINO 5 DO
       PORTD
15 sbit LCD_D4_Direction at TRISD4_bit;  // SETA DIRE??O DO FLUXO DE DADOS DO PINO 4 DO
       PORTD
16
17 char temper[15]; //temperatura em string
18 unsigned int temp=0; //valor da temperatura j   convertido
19 int cont=0; //vari vel para realizar contagem
20 int k=0;
21
22 void interrupt(){               // Palavra "interrupt" sempre em minuscula.
23     if(INTCON.TMR0IF==1){       // Incrementa somente quando existir o overflow do timer
           0.
24         TMR0H = 0xC2 ;
25         TMR0L = 0xF7;
26         INTCON.TMR0IF = 0;       // Limpa o flag de estouro do timer0 para uma nova
               contagem de tempo.
27         cont++;
28         k=0;
29     }
30 }
31
32 void main(){
33    int valorAD;
```

```
34    ADCON0 = 0b00001011;
35    ADCON1 = 0b11001100;
36    ADCON2 = 0b10111110;
37
38    UART1_Init(9600);  // Utiliza bibliotecas do compilador para configura  o o Baud
          rate.
39
40    PORTA=255;
41    TRISA=255;
42
43    TRISC.RC5=0;
44    TRISC.RC1=0;
45    PORTC.RC1=1;
46    TRISE.RE0=1;
47
48     OSCCON =  0X70;                  // Set internal oscillator to 8MHz
49
50     Lcd_Init();                      // Initialize LCD module
51     Lcd_Cmd(_LCD_CURSOR_OFF);        // cursor off
52     Lcd_Cmd(_LCD_CLEAR);             // clear LCD
53     lcd_out(1, 3, "Temperature:");
54
55    //PORTC.RC5=1;
56
57    //PWM1_Init(5000); //5kHz
58    PWM1_Start();
59
60
61    T0CON.T08BIT=0;
62    T0CON.T0CS=0;
63    T0CON.T0SE=0;
64    T0CON.PSA=0;
65    T0CON.T0PS2=1;
66    T0CON.T0PS1=1;
67    T0CON.T0PS0=0;
68    TMR0H = 0xC2;            // Carrega o valor alto do n?mero 57723.
69    TMR0L = 0xF7;            // Carrega o valor baixo do numero 57723.
70    INTCON.TMR0IE = 1;       // Habilita interrup??o do timer0.
71    INTCON.TMR0IF = 0;       // Apaga flag de estouro do timer0, pois ? fundamental para a
          sinaliza??o do estouro.
72    T0CON.TMR0ON=1;
73    INTCON.GIE = 1;
74    INTCON.PEIE = 1;
75
76    while(1){  // SELECIONE A VARIAVEL DE CONTROLE (CONTROL) DECLARADA ACIMA.
77
78       PORTC.RC5=1; //resist ncia
79
80       temp = ADC_Read(2)*0.489; //temp em celsius500/1023)
81       inttostr(temp,temper); //convert temp to string
82       if(cont<0) temper[6]='0'; //byte to control text on arduino
83       else temper[6]='1';
84       temper[14]='k'; //end char
85
86       UART1_Write_Text(temper); //send temper through UART1
87       Delay_ms(50);
88
89       lcd_out(1,1,"TEMPERATURE="); //show on display
90       lcd_out(1,15, "   ");
91       lcd_out(1,13, Ltrim(temper));
```

```
92        lcd_out(1,15, "  ");
93
94        if(temp>35 && cont<10){ //control fan
95          if(cont<0)PWM1_Set_Duty(0);
96          else if (temp<40)PWM1_Set_Duty(511);
97          else (PWM1_Set_Duty(1023));
98        }
99        else{
100         cont=-10; //10 seconds off
101         PWM1_Set_Duty(0);
102       }
103       //buzzer
104       if(temp>40 && cont%2==0){
105         PORTC.RC1=0;
106       }
107       else{
108         PORTC.RC1=1;
109       }
110     }
111 }
```

4.1 Arduino

No código passado ao Arduino, há a configuração do Serial de 0 à 2, sendo o 1 o que fará comunicação com o PIC18F e o Serial 2 onde haverá a transmissão dos dados. Na função loop() no primeiro bloco há a verificação do canal Serial1 e o recebimento do texto contido nesse até o caractere 'k', previamente adicionado no código passado ao PIC18F, contendo a temperatura e um byte de controle. O primeiro digito da temperatura é armazenado em a, o segundo em b e o byte de controle em c, sendo devidamente atualizados sempre que for um caractere com símbolo entre 0 e 9 da tabela ASCII.

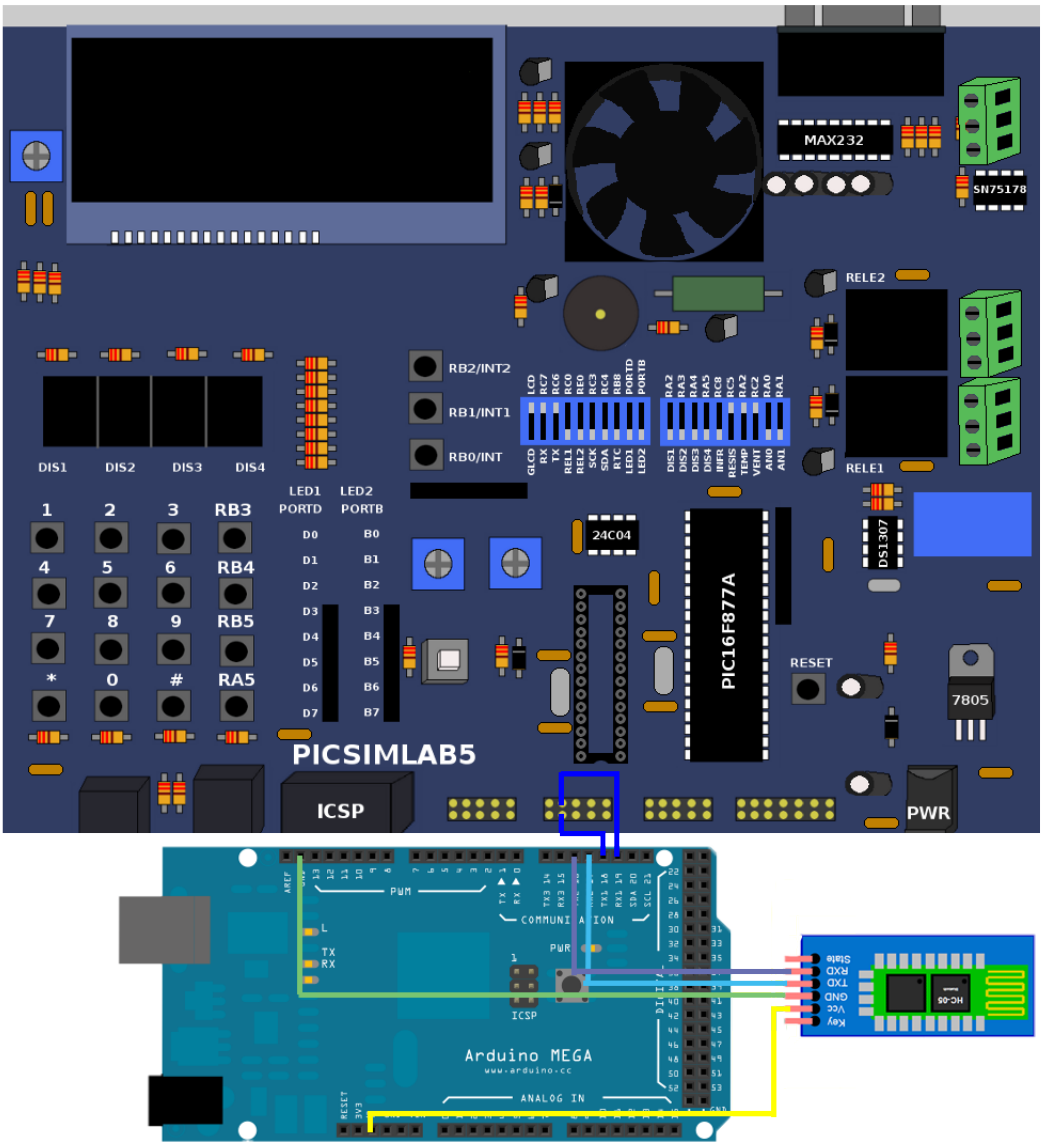
No bloco seguinte, há o envio do estado atual da temperatura e da ventoinha. A temperatura é passada simplesmente enviando os caracteres armazenados em a e b. Já o estado da ventoinha é passado através do byte de controle e dos caracteres a e b. Caso o byte de controle seja '1', significa que a variável de contagem de estouros no PIC18F está entre 0 e 10 possibilitando o acionamento da ventoinha, caso contrário a ventoinha está desligada. Supondo o byte de controle com caractere igual a '1', há a verificação dos caracteres contidos em a e b, caso a concatenação de ambos forme um valor abaixo de 35, é enviado que a ventoinha está desligada, caso esteja entre 35 e 39 é enviado que a ventoinha está em potência média e caso o caractere contido em a seja maior ou igual ao '4', sendo então a temperatura maior ou igual a 40 graus, há o envio de que a ventoinha está em potência máxima. O envio ocorre sempre que o usuário envia o caractere 'f' pelo celular.

```
1 String temp;
2 char k;
3 char a;
4 char b;
5 char c;
```

```
6
7  void setup() {
8  Serial.begin(9600);
9  Serial1.begin(9600);
10 Serial2.begin(9600);
11 temp[14]='\0';
12 }
13
14 void loop(){
15
16   if(Serial1.available()){
17    Serial.println("HI");
18    //receive string from Serial1 until char 'k'
19    temp=Serial1.readStringUntil('k');
20
21    Serial.println(temp);
22    Serial.print(temp[4]);
23    Serial.print(temp[5]);
24    //control to trash strings
25    if(temp[4]>=48 && temp[4]<=57)
26       a=temp[4];
27    if(temp[5]>=48 && temp[5]<=57)
28       b=temp[5];
29    if(temp[6]>=48 && temp[6]<=57)
30       c=temp[6];
31    Serial.print(a);
32    Serial.print(b);
33    Serial.println();
34   }
35
36   if(Serial2.available()){
37     if((char)Serial2.read()=='f'){
38       Serial2.println();
39       Serial.print("######");
40       if(c=='1'){
41         Serial.print("HIIII");
42         if(a>=52) Serial2.print(" Exaustao maxima ativada ");
43         else if(a<51 || (a==51 && b<53)) Serial2.print(" Exaustao desligada ");
44       else Serial2.print(" Exaustao media ativada ");
45       }
46       else Serial2.print(" Exaustao desativada ");
47       //send via Serial2 (bluetooth)
48       Serial2.print(a);
49       Serial2.print(b);
50       Serial2.print(" c ");
51     }
52   }
53
54
55 }
```

4.2 Circuito Final

Figura 7 – Circuito final



Fonte: O Autor

5 Resultados Obtidos e Discussões

Os resultados obtidos foram aqueles esperados, sem problemas maiores. Sempre que requisitado pelo usuário em um smartphone, há o envio das condições atuais de temperatura e da ventoinha.

Um único ponto que poderia gerar problemas é o envio, já que o usuário precisa solicitar duas vezes o envio das condições para que ocorra o envio correto. Isso ocorre devido ao fato do *buffer*, que recebe o texto, somente ser atualizado quando há a solicitação do usuário. Assim primeiro ocorre a atualização e em seguida o dado correto estará disponível para visualização.

Além disso, inicialmente foi trabalhoso a sincronização dos dados de forma a não afetar a performance, inúmeras vezes o recebimento de dados era travada devido ao grande número de caracteres seguidos lidos, travando o sistema.

6 Considerações Finais

Algumas poucas complicações de sincronização geraram pequenas dificuldade na obtenção de dados. Porém isso levou a um aprendizado sobre as plataformas utilizadas no desenvolvimento, além da necessidade de dedicação a um estudo um pouco mais aprofundado para finalização do projeto. Com isso, certamente conhecimento foi agregado, podendo ser importante para o futuro.

Referências