Júlia Wotzasek Pereira

Desenvolvimento e Descrição de um Alarme Residencial utilizando os Microcontroladores *PIC* e *Arduino*

São José dos Campos - Brasil

Junho de 2019

Júlia Wotzasek Pereira

Desenvolvimento e Descrição de um Alarme Residencial utilizando os Microcontroladores *PIC* e *Arduino*

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

Junho de 2019

Sumário

1 Descrição do Projeto

1.1 Descrição

A segurança residencial em um país de taxas de criminalidade crescentes se faz de importância ímpar. Seja por meio de grades de proteção, cacos de vidro, ou mesmo avisos de "cachorro bravo", os métodos para garantir a segurança são vários.

Em um contexto de cada vez mais incentivos a projetos de automatização de casas, com a popularização dos conceitos de *IoT (Intelligence of Things)* ou Inteligência das coisas, desenvolver um projeto de alarme residencial com a utilização de microcontroladores é um passo natural para quem visa tornar-se engenheiro na atual conjuntura nacional.

Como um projeto que visa facilitar a vida dos usuários deste, a ideia do funcionamento é bem simples: O sistema de segurança conta com um sensor de presença por infravermelho que sempre está verificando se há alguém no seu perímetro, mas que só emite alarme sonoro (*buzzer*) caso o usuário tenha ativado o alarme.

O usuário possui três possibilidades de ação nesse projeto, escolhendo a opção do menu por um teclado matricial de membrana 4×4, visualizando suas ações por um *display LCD* de 16×2:

- **Ligar o alarme (*tecla A*):** Quando o usuário pede para que o alarme seja ligado, o sistema pede para que ele digite a senha de 4 dígitos (senha esta iniciada pelo sistema como 1234).

  Se o usuário digita corretamente a senha, o sistema acende um conjunto de *leds* que vão se apagando conforme vai se extinguindo o tempo que o usuário tem para se afastar do alarme antes que este apite. Vale destacar que esse tempo é configurável por um potenciômetro, podendo demorar mais ou menos de acordo com o desejo do usuário.

- **Desligar o alarme (*tecla B*):** Quando o usuário pede para que o alarme seja desligado, o sistema também requisita a senha de 4 dígitos.

  Se o usuário digita corretamente a senha, o alarme é desligado imediatamente.

- **Mudar a senha (*tecla C*):** Como vem como uma senha atribuída de fábrica, é interessante que o usuário possa alterar a senha do sistema. Se assim o desejar,quando apertar o botão C, se o alarme estiver desligado, o sistema requisitará a senha atual e, se esta for verificada, receberá a nova senha digitada como senha definitiva, avisando que a senha foi alterada no *display LCD*. Se o sistema estiver ligado, o sistema

emitirá a resposta de que a senha não pode ser alterada pelo fato de o sistema estar ligado.

Para a implementação, foi utilizado o microcontrolador *PIC18F4520* no *kit PICGenios* da *Microgenios* e o *Arduino Mega2560*, visando a integração do sensor de presença PIR e do teclado matricial e de um *buzzer* ao *kit PICGenios* via *Arduino.* Do *kit*, utilizou-se o *display LCD*, os *leds*, um potenciômetro, um *timer*, o *buzzer* e a o *PORTC* para comunicação com o *Arduino* e com a resposta digital do sensor.

1.2 Objetivos

1.2.1 Objetivos Gerais

Por objetivo geral desse projeto tem-se assimilar os conceitos básicos de sistemas embarcados por meio da definição, desenvolvimento e implementação de um projeto utilizando em conjunto o *Arduino Mega 2560* e o microcontrolador *PIC18F4520* no *kit PICGenios* da *Microgenios*.

1.2.2 Objetivos Específicos

Por objetivos específicos temos:

- Integração da plataforma *PIC Microgenios* com o *kit Arduino* usando comunicação *USART*.
- Utilização dos seguintes recursos do microcontrolador *PIC 18F4520*:
  - Comunicação serial *USART*
  - *Timer*
  - Interrupção do *Timer*
- Utilização dos seguintes periféricos da plataforma *PIC Microgenios*:
  - *Display LCD*
  - Potenciômetro
  - *Leds*
  - *Buzzer*
- Utilização dos seguintes recursos do microcontrolador *Arduino Mega 2560*:
  - Módulo de comunicação *USART*

  - GPIO (Leitura do sensor PIR)

- Utilização dos seguintes periféricos conectados ao *kit Arduino*:
  - Módulo PIR
  - Teclado matricial de membrana 4×4
  - *Buzzer*

2 Funcionamento do Sistema

2.1 *Hardware*

2.1.1 Descrição dos Componentes

2.1.1.1 Teclado Matricial 4×4 de Membrana

O teclado matricial 4×4 de membrana consiste em 16 teclas *push-buttons* do tipo membrana dispostos na configuração da figura 1. Como podemos observar, botões de uma mesma linha são conectados entre si, correspondendo aos canais de 1 a 4. Da mesma maneira, botões de uma mesma coluna são conectados entre si, correspondendo aos canais de 5 a 8.

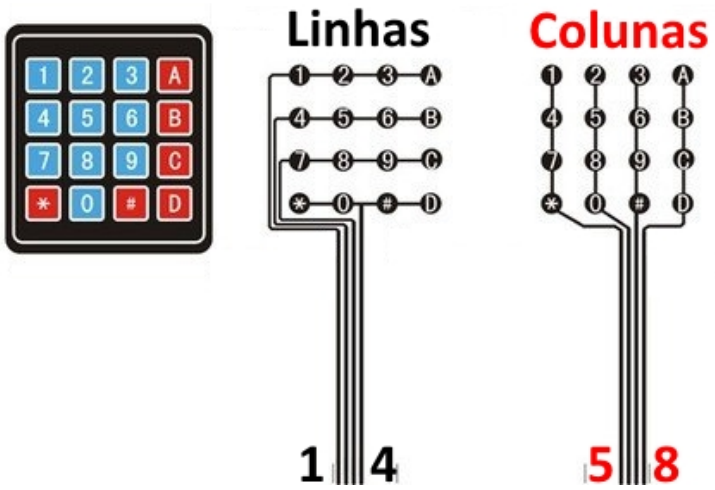

Figura 1 – Teclado Matricial de Membrana. Fonte: https://www.filipeflop.com/blog/teclado-matricial-4x4-arduino/

Para os botões do teclado, 4 estados possíveis são considerados:

1. **IDLE:** Esse é o estado de livre, sem valor. É o estado para quando o botão não está pressionado.
2. **PRESSED:** Esse é o estado para quando o botão é apertado.
3. **RELEASED:** Esse é o estado para quando o botão é solto.
4. **HOLD:** Esse é o estado para quando o botão é segurado pressionado. É interessante destacar de antemão que o que faz considerar que é nesse estado que o botão se encontra depende de quanto tempo se define como tempo de *holding*. Na biblioteca *keypad.h* esse *debounce* é feito com a função millis().

Destacamos estes estados do teclado pois eles são bastante relevantes para a explicação de como ocorre a leitura de qual botão foi apertado. O processo de leitura ocorre por varredura. A ideia do processo é a seguinte: Pelas quatro colunas de entrada, o Arduino envia um sinal digital alto ($HIGH$). Se nenhuma das teclas estiver pressionada, o sinal das quatro linhas estará baixo (LOW).

Suponha agora que apertemos, por exemplo, a tecla 8. Nesse instante, a corrente fluirá entre a coluna 2 e a linha 3, fazendo o sinal digital da coluna 2 ir para LOW. Essa mudança de estado - $(HIGH, LOW) \rightarrow (LOW, LOW)$ - faz com que se comece a fazer a varredura nas linhas, enviando um sinal digital alto ($HIGH$) linha por linha. Ainda no exemplo em que apertamos a tecla 8, quando o sinal alto é enviado para as linhas 1 e 2, nenhuma mudança de estado é alterada pois, como podemos notar em 2, os caminhos não estão conectados. Porém, quando é enviado um sinal alto para a linha 3, o sinal da coluna 2 retorna para $HIGH$ e consegue, dessa forma, determinar que o botão apertado fora o 8.

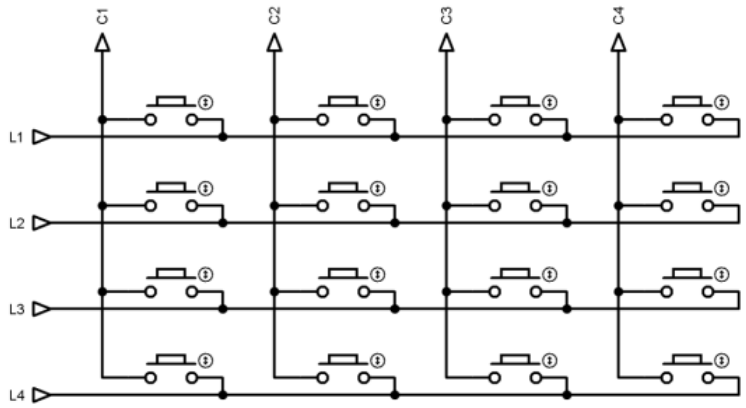

Figura 2 – Circuito de multiplexação do teclado matricial. Fonte: https://portal.vidadesilicio.com.br/teclado-matricial-e-multiplexacao/

2.1.1.2 Sensor de Presença PIR

Um sensor PIR é um sensor piroelétrico, isto é, é um sensor capaz de detectar níveis de irradiação infravermelha emitidas pelo corpo humano. Como podemos encontrar explicado em (1), quando alguém passa na frente desse sensor, a pessoa passará na zona de detecção do primeiro elemento piroelétrico e depois na zona de detecção do segundo elemento. Ao passar pelo primeiro sensor, gera um pulso de tensão de saída, e passando pelo segundo gera um pulso de tensão de sinal contrário, como podemos ver na figura abaixo:

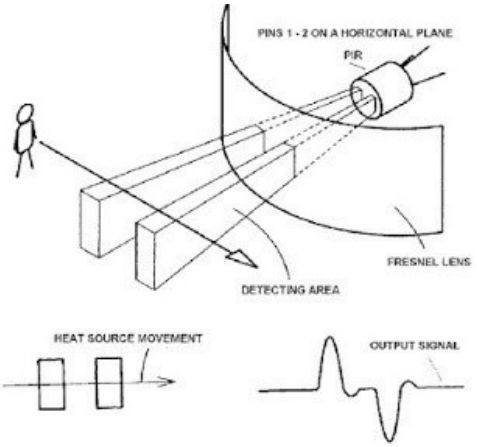

Figura 3 – Esquema do sensor de presença PIR. Fonte: https://portal.vidadesilicio.com.br/sensor-de-presenca-hc-sr501/

O sensor utilizado possui um domo branco sobre ele, que é uma lente de *Fresnel* para aumentar o alcance do sensor. Além disso, possui dois potenciômetros para controlar a sensibilidade e a permanência em nível alto, além de um *jumper* para selecionar se a resposta deve ser periódica ou contínua.

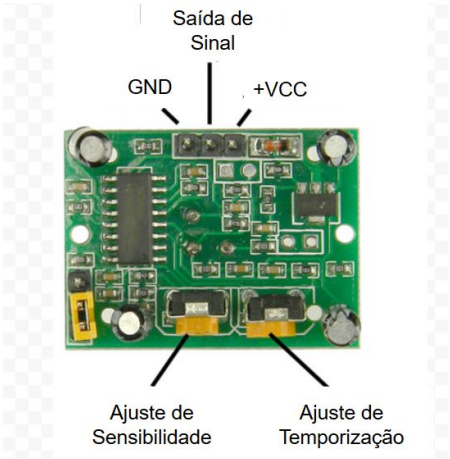

Figura 4 – Sensor de presença PIR. Fonte: https://portal.vidadesilicio.com.br/sensor-de-presenca-hc-sr501/

2.1.1.3 Potenciômetro

O potenciômetro é um componente eletrônico que controla o fluxo de corrente elétrica que passa por ele, sendo essa limitação ajustada manualmente. O potenciômetro costuma possuir três terminais, para o controle da resistência.

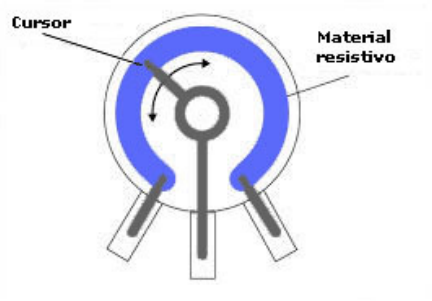

Figura 5 – Esquema de um potenciômetro

Fonte: http://www.comofazerascoisas.com.br/potenciometro-o-que-e-para-que-serve-e-como-funciona.html

2.1.1.4 *Display* LCD

O *display* LCD utilizado possui alimentação de $5V$, 1 pino para *backlight*, 6 pinos de dados. O *display* em questão possui 16×2. Cada célula do *display é de* 8×5 *bits*.

2.1.1.5 *Buzzer*

O *buzzer* é um transdutor piezoelétrico, isto é, um componente que de acordo com a tensão recebida gera uma tensão mecânica. Quando enviamos essa tensão com certa frequência, fazemos com que ele vibre, o que gera sons. o componente utilizado é um alimento ativo, possuindo oscilador interno, como visto em (2).

Na figura 6, notamos que devemos conectar uma tensão (usamos $5V$, que é a da placa arduino) no V da figura. O outro conector do *buzzer* é colocado para chavear o transistor. Daí, quando conduz o *buzzer* emite som, e quando não conduz o *buzzer* não emite som.

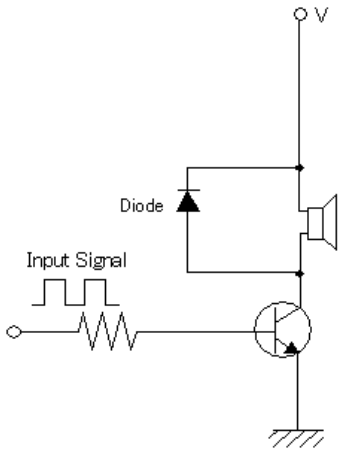

Figura 6 – Sistema elétrico de um *buzzer* Fonte: https://www.murata.com/en-us/support/faqs/products/sound/sounder/char/sch0007

2.1.1.6 *Led*

Um *led*, ou *Light Emissor Diode*, é um diodo emissor de luz, isto é, um componente eletrônico feito de germânio ou silício, que conduz apenas corrente no sentido da polarização, onde o catodo deve ser positivo e o anodo negativo. O Led, em particular é um diodo que, quando passa corrente, emite luz, como explicado em (3).

O *led* é um componente bipolar, e a forma como está polarizado determina o sentido que a corrente pode passar, como está indicado na figura 7.

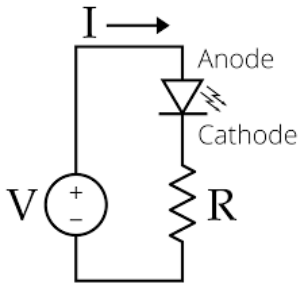

Figura 7 – Esquemático elétrico de um *led*

Existem vários tipos de *led*. Dentre eles, destacamos:

- ***led* difuso comum:** Em *leds* desse tipo, a luz é espalhada por toda a cápsula de plástico, buscando distribuição uniforme desta.
- ***led* de alto brilho:** A potência luminosa de *leds* de alto brilho é bastante superior a dos *leds* difusos, aumentando a luminosidade deste.

2.1.2 Integração do *hardware*

2.1.2.1 Conexão dos periféricos embarcados na plataforma *PIC Microgenios*

Alguns dos periféricos utilizados se encontram no *kit PIC Microgenios*:

- ***Display LCD*:** O *Display LCD* é conectado ao microcontrolador *PIC* utilzando-se da *PORTD* do *kit*.
- ***Buzzer*:** O *Buzzer* é conectado ao microcontrolador *PIC* utilizando-se do registrador PORTC, no *bit* RC1.
- **Potenciômetro:** O potenciômetro é conectado a porta analógico ANAL0. Vale destacar que a conversão A/D é feita pelo conversor ADC do microcontrolador.
- ***LEDs*:** Os 8 *LEDs* utilizados são referentes aos 8 *bits* do registrador PORTB do microcontrolador.

2.1.2.2 Conexão *buzzer* - Arduino

Para a conexão do *buzzer*, colocamos a polaridade negativa no GND e a positiva em uma entrada analógica, no caso a porta 10.

2.1.2.3 Conexão teclado matricial - Arduino

Para conexão do teclado matricial, utilizamos as entradas de 44 até 53, como sinais digitais para a leitura do teclado como já explicado anteriormente.

2.1.2.4 Conexão sensor PIR - Arduino - PIC

O sensor PIR foi conectado a porta 13 do Arduino, e aos pinos GND e $5V$. Como sua leitura é digital, sua saída no pino 2 é enviada para ser lida na PORTC.RC3 do *kit PIC*.

2.1.2.5 Conexão Arduino - PIC

A conexão feita foi a comunicação serial $USART$. Para isso, conectamos os pinos $RX1$ e $TX1$ do *Arduino* às portas $RC6$ e $RC7$, respectivamente.

2.1.3 Esquemático do Circuito

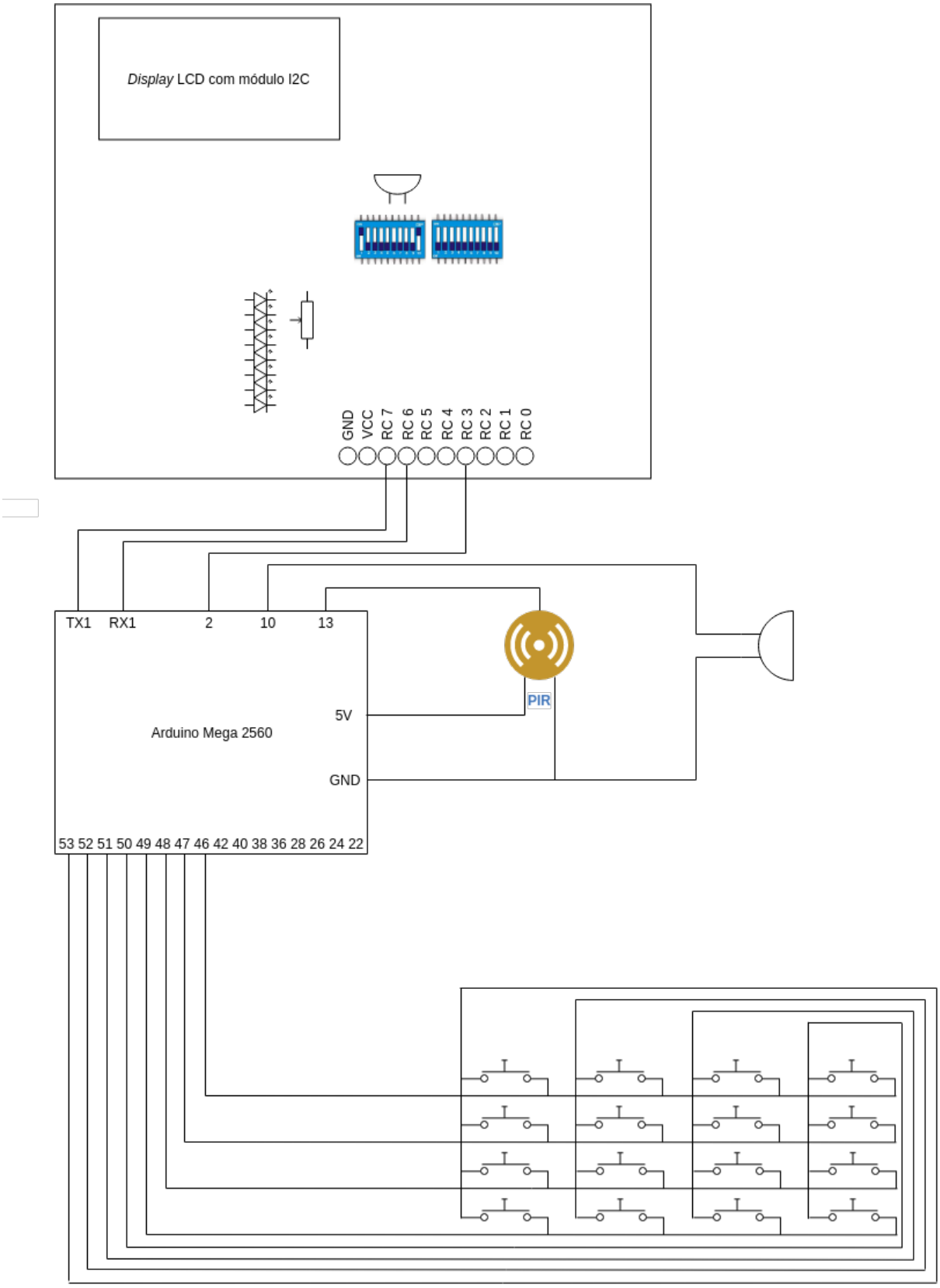

Figura 8 – Esquemático elétrico do projeto

2.2 *Software*

Por critério de organização do presente relatório, iremos apresentar parte a parte do código, abordando primeiro o código utilizado no *Arduino Mega 2560* e depois abordaremos o código utilizado no *PIC 18F4520*, dividindo essa parte entre *setup* e a lógica do alarme.

2.2.1 *Software* no *Arduino Mega 2560*

Como explicamos na descrição do *hardware*, no *Arduino* precisamos ler o teclado matricial, enviando os dados via comunicação serial *USART*, e ler os dados do sensor PIR, enviando-os digitalmente pela porta 2.

Para isso, utilizamos a biblioteca *Keypad.h*. A biblioteca **Keypad.h** é para a utilização do teclado matricial. Na descrição sobre o teclado matricial discutimos que a leitura das teclas pressionadas é feita por varredura. Essa biblioteca esconde esse método na sua função **getKey()**. Ela testa se houve mudança de estado e se houve alguma atividade no teclado pela função **getKeys()**, que é a função que verifica se mais de um botão foi apertado, e com isso varre o teclado para a obtenção do que foi teclado, como precisávamos. Nesse projeto, só utilizamos a função getKey() e o construtor do teclado, no qual passamos o tamanho do teclado e quais os dígitos desse teclado.

2.2.1.1 Bibliotecas Utilizadas

```
1 /* Bibliotecas utilizadas */
2 #include <Keypad.h>
```

Como já discutimos antes, importamos a biblioteca Keypad para esse projeto.

2.2.1.2 Definição da Pinagem

```
1 /* Definicao do teclado */
2 const byte linhas = 4; //4 linhas
3 const byte colunas = 4; //4 colunas
4 byte pinoslinhas[linhas] = {46,47,48,49}; //pinos utilizados nas
     linhas
5 byte pinoscolunas[colunas] = {50,51,52,53}; //pinos utilizados
     nas colunas
6
7 /* Pinagem do buzzer*/
8 const int buzzer = 10; // buzzer conectado no pino 10
9
10 /* Pinagem do PIR e da saida digital */
11 const int PIR = 13; // sensor PIR no pino 13
12 const int out = 2;
```

Essa parte do código serve para apenas definir quais portas do Arduino serão utilizadas para cada uma das entradas/saídas necessárias.

2.2.1.3 Variáveis

```
1 //inicializando o teclado
2 Keypad teclado = Keypad( makeKeymap(matrizteclado), pinoslinhas,
     pinoscolunas, linhas, colunas );
3
4 int tom; // variavel para apitar o buzzer quando apertar o botao
```

Nesse trecho estamos apenas construindo variáveis. Vale destacar que utilizamos o construtor da biblioteca Keypad para utilizá-lo.

2.2.1.4 *Setup*

```
1  void setup() {
2    Serial1.begin(9600);
3    pinMode(buzzer,OUTPUT);
4    pinMode(PIR,INPUT);
5    pinMode(out,OUTPUT);
6 }
```

O **setup** do programa é bem simples também. A ele cabe apenas indicar que o *buzzer* e *out* são sinais de saída, estabelecer o PIR como leitura e estabelecer a comunicação serial *USART* com taxa de transmissão de 9600.

2.2.2 Leitura de Dígito

```
1  int leDigito(){
2    char digito = teclado.getKey();
3      while(!digito){
4        digito = teclado.getKey();
5      }
6    int d = (int)digito;
7    tom = 420 + 100 * (d - 48);
8    digitalWrite(buzzer,HIGH);
9    delayMicroseconds(tom);
10   delay(100);
11   digitalWrite(buzzer,LOW);
12   delayMicroseconds(tom);
13   delay(100);
14   return d - 48;
15 }
```

Para a leitura do dígito do teclado matricial, por intermédio das funções da biblioteca Keypad.h, utilizamos a função getKey(), sendo teclado o nome da variável que criamos no cabeçalho para o tipo Keypad. Para que o *buzzer* apite toda vez que um botão for apertado, quando o digito recebe algo e sai do laço while, mandamos um sinal alto para o *buzzer*, com a função de fazê-lo apitar.

2.2.3 Laço principal

```
1 void loop(){
2     int d,p;
3     d = leDigito();
4     p = digitalRead(PIR);
5     digitalWrite(out,p);
6     Serial1.Write(d);
7     Serial1.write('E');
8 }
```

Como mostramos na função *leDigito()*, o programa fica preso nela até que algum botão seja pressionado. Isso acarreta que o valor enviado pela comunicação serial não seria atualizado, gerando um problema sério para o PIC resolver de quando entender o dado enviado como um novo dígito ou não. Dessa maneira, optamos por enviar logo após o dígito apertado a saída 'E', para que no PIC se entenda que 'E' é um sinal de "vazio"de tecla apertada.

2.2.4 *Software* no *PIC 18F4520*

2.2.4.1 Fluxograma Geral do Alarme

Mostramos nessa seção o fluxograma geral projeto de alarme, para depois percorrermos cada parte dele explicando, junto ao código, o seu funcionamento.

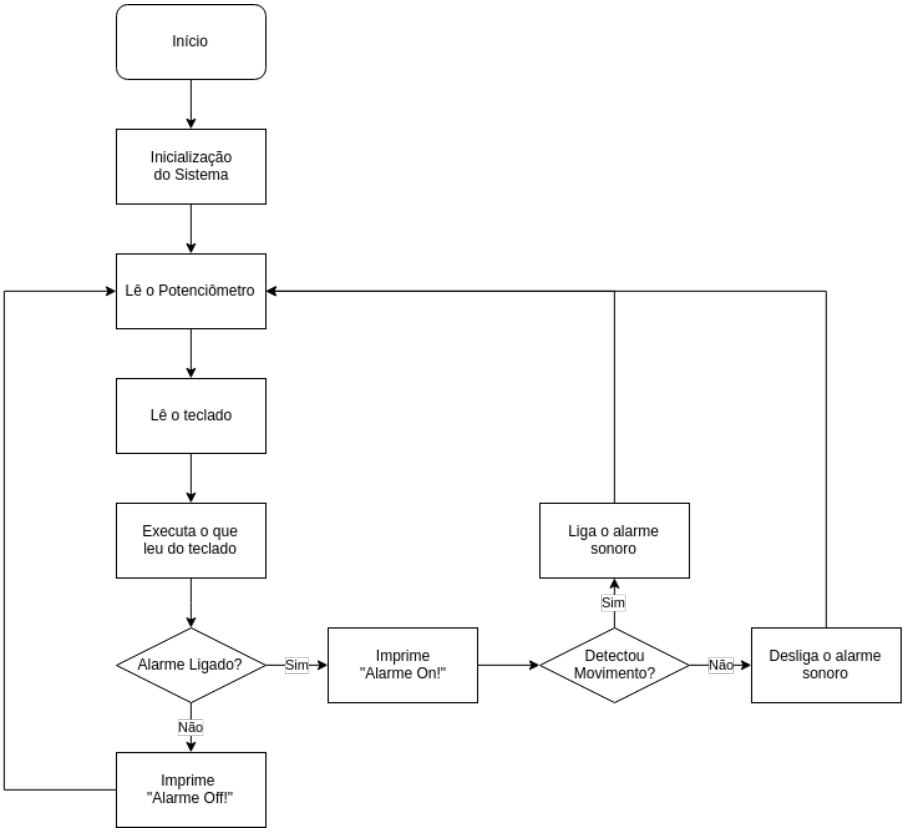

Figura 9 – Fluxograma do Alarme Residencial

2.2.4.2 Inicialização do sistema

O projeto no *PIC 18F4520* exige que definamos especificamente os registradores utilizados, sendo mais baixo nível em relação a programação com o *Arduino*. Dessa maneira, o *setup* do projeto é bastante extenso e, por isso, vamos explicar separadamente o *setup* para cada um dos periféricos e recursos utilizados.

- ***Setup* do LCD**: Para o *setup* do LCD, precisamos definir o fluxo de dados logo no início do programa. Destacamos que utilizamos o PORTD para poder utilizar o *display de LCD*. Temos:

```
1       /* Ligacoes entre o PIC e o 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 direcao do fluxo de dados dos pinos
      utilizados para comunicacao com o display */
10  sbit LCD_RS_Direction at TRISE2_bit;  // SETA DIRECAO DO
       FLUXO DE DADOS DO PINO 2 DO PORTD
11  sbit LCD_EN_Direction at TRISE1_bit;  // SETA DIRECAO DO
       FLUXO DE DADOS DO PINO 3 DO PORTD
12  sbit LCD_D7_Direction at TRISD7_bit;  // SETA DIRECAO DO
       FLUXO DE DADOS DO PINO 7 DO PORTD
13  sbit LCD_D6_Direction at TRISD6_bit;  // SETA DIRECAO DO
       FLUXO DE DADOS DO PINO 6 DO PORTD
14  sbit LCD_D5_Direction at TRISD5_bit;  // SETA DIRECAO DO
       FLUXO DE DADOS DO PINO 5 DO PORTD
15  sbit LCD_D4_Direction at TRISD4_bit;  // SETA DIRECAO DO
       FLUXO DE DADOS DO PINO 4 DO PORTD
```

Além disso, precisamos inicializar o LCD, com a definição dos pinos do PORTB como digitais e chamar as funções de inicialização do LCD:

```
1   /* Funcao de setup do LCD */
2   void setup_LCD()
3   {
4      ADCON1   = 0x0E;//Configura os pinos do PORTB como
          digitais, e RA0 (PORTA) como analogico
5      Lcd_Init();//Inicializa modulo LCD
6      Lcd_Cmd(_LCD_CURSOR_OFF);//Apaga cursor
7      Lcd_Cmd(_LCD_CLEAR);//Limpa display
8   }
```

- ***Setup* do PIR**: Para inicializar o sensor de presença PIR, como estamos recebendo o seu sinal digital pela PORTC.RC3, tudo o que precisamos fazer é definir o fluxo de dados pelo registrador TRISC, definindo que o pino RC3 deve ser de entrada de dados e todos os outros de saída.

```
1   /* Funcao de setup do sensor de presenca PIR */
2    void setup_PIR()
3    {
4       TRISC = 0b00001000; // Configura como entrada
5    }
```

- ***Setup* do *TIMER0***: Para utilizar o *timer* do PIC, precisamos inicializar os regitradores T0CON, TMR0H, TMR0L e INTCON. Temos:

```
  void setup_TIMER()
{
   ADCON1 = 0x0F; // Configura todos os pinos A/D como I/O
   TRISD = 0; // Define todos os pinos Do PORTD como saida.
   ucContador=0; // Inicializa a variavel com o valor 0.
   // Configuracao do Timer0.
   // Cristal de 8Mhz, ciclo de maquina: 8MHz / 4 = 2Mhz -->
      1/2Mhz = 0,5us.
   T0CON.T0CS = 0; // Timer0 operando como temporizador.
   T0CON.PSA = 0;  // Prescaler ativado.
   T0CON.T0PS2 = 1; // Define prescaler 1:256.
   T0CON.T0PS1 = 1; // Define prescaler 1:256.
   T0CON.T0PS0 = 1; // Define prescaler 1:256.
   T0CON.T08BIT = 0; // Define contagem no modo 16 bits.
   // Valor para 1 segundo.
   TMR0H = 0xE1;// Carrega o valor alto do numero 57723.
   TMR0L = 0x7B;// Carrega o valor baixo do numero 57723.
   INTCON.TMR0IE = 1; // Habilita interrupcao do timer0.
   INTCON.TMR0IF = 0; // Apaga flag de estouro do timer0,
      pois ? fundamental para a sinalizacao do estouro.
   T0CON.TMR0ON = 0; // Liga timer0.
   INTCON.GIE = 1; // Habilita as interrupcoes nao-mascaradas
      .
   INTCON.PEIE = 1; // Habilita as interrupcoes dos
      perifericos.
}
```

- ***Setup* do potenciômetro:** Sabemos que o potenciômetro tem por resposta um valor analógico. Dessa maneira, precisamos configurar os registradores ADCON, TRISA e PORTA para isso:

```
  /* Funcao de setup do potenciometro */
   void setup_Potenciometro()
   {
      TRISA = 0xFF; // todos os pinos de escrita
      PORTA = 0xFF; // todas as entradas altas
      ADCON0 = 0b00000001;
      ADCON1 = 0b11001110;
      ADCON2 = 0b10111111;
   }
```

- ***Setup* dos *LEDs*:** Como explicamos na descrição do projeto, os *LEDs* da PORTB devem acender quando o usuário acertar a senha para ligar o alarme e ir decrementando de acordo com o tempo definido no potenciômetro. Dessa maneira, devemos começar com todos os pinos como saída (definido em TRISB) e todos com saída apagada (definido em PORTB).

```
1  /* Funcao de setup dos LEDs */
2   void setup_LEDs()
3   {
4      TRISB =  0x00; // configura os pinos de B como saida
5      PORTB =  0x00; // configura todas as saidas como zero
6   }
```

- ***Setup* da comunicação serial USART:** O estabelecimento da comunicação é bem simples, pois utilizamos bibliotecas do configuração.

```
1  /* Funcao de setup da comunicacao USART */
2   void setup_USART()
3   {
4      UART1_Init(9600); // Utiliza bibliotecas do compilador
          para configuracao do Baud rate
5   }
```

2.2.4.3 Leitura do Potenciômetro

Para a leitura do potenciômetro, precisamos receber o valor analógico da leitura e transformá-lo em digital para poder ser utilizado no programa. Recebemos o valor com a função do sistema, e depois fazemos com que fique no intervalo de 0 a 5.

```
1 /* Funcao para leitura do potenciometro */
2    void read_Potenciometro()
3    {
4      Delay_10us;
5      pot1 = ADC_Get_Sample(0);
6      pot1 = pot1 * 5 / 1023;
7      acende_leds(pot1);
8    }
```

2.2.4.4 Leitura do Teclado

A varredura do teclado de membrana ficou sob responsabilidade do *Arduino*, como explicamos quando discutímos o código do *Arduino*. Dessa maneira, no *PIC* precisamos apenas receber o que está sendo enviado pela comunicação serial.

```
/* Funcao para ler o teclado do arduino */
   void read_keypad()
   {
     // Lcd_Out(1,1,"Lendo       ");
      if(UART1_Data_Ready()){  // Verifica se um dado foi
         recebido no buffer
         Delay_ms(50);
         ucRead = UART1_Read(); // Le o dado recebido do buffer
         Delay_ms(50);
       }
   }
```

2.2.4.5 Execução da tecla lida

Essa extensa função é a principal em termos de execução das funcionalidades do alarme, como ligá-lo, desligá-lo e trocar a senha. Como não podemos permanecer nessa função até que alguma tecla seja apertada (pois precisamos toda hora verificar se houve movimentação próxima ao sensor). Dessa maneira, a lógica dessa função não é exatamente um laço isolado, e se trata apenas de testes lógicos *if-else*. Como o código é bem extenso, fora deixado no apêndice A. O fluxograma explica bem a lógica da função na figura 10. Observe que começamos com um *switch-case* sobre a tecla lida do teclado. Temos:

- **ucRead == 'A':** Caso seja solicitado ao sistema que ligue o alarme, se o sistema ainda não estiver tentando ler uma senha "uiLeSenha == 0", então reinicializamos o valor do contador de dígitos, trocamos o valor da *flag* de ler senha para 1, bem como a *flag* de ligar o alarme. Por fim, imprimimos "Senha:"e encerramos a função.

- **ucRead == 'B':** Caso seja solicitado ao sistema que desligue o alarme, se o sistema ainda não estiver tentando ler uma senha, então alteramos as *flags* para que o sistema passe a ler uma senha, mande desligar o alarme e reinicie o contador de dígitos. Após isso, ele imprime "Senha:"para esperar que o usuário a digite.

- **ucRead == 'C':** Caso seja solicitado ao sistema a alteração da senha, temos duas possibilidades: se o alarme estiver ligado, deve-se imprimir "Alarme Ligado! Acesso Negado!", pois não faz sentido poder alterar a senha com o alarme funcionando; caso o alarme esteja desligado, se a leitura de senha estiver desativada, alteramos as *flags* para que o sistema leia a senha, mande mudar a senha e reinicialize o contador de dígitos. Por fim, imprime "Senha:", esperando que o usuário digite a senha original. Encerra-se a função.

- ***Default*:** O *Default* foi utilizado para a leitura da senha, atualização da senha e execução dos procedimentos de atualização do estado do sistema.

Discutimos quando falamos do código executado no *Arduino* da necessidade de "limpar"o dado enviado pela comunicação serial, para que o *PIC* tivesse facilidade em identificar o que era ou não um novo dígito. Resolvemos isso enviando a letra 'E'. Dessa maneira, só executamos o código de *Default* se a letra apertada não for a letra 'E'.

Enquanto o contador de dígitos não zera, o número apertado é alocado no vetor de senha digitada, decrementando o contador. Se o contador não estiver zerado, a função se encerra. Se estiver zerado, então temos uma sequência de 3 *ifs*:

- **uiSenhaDeveMudar == 1?:** Se esse teste der verdadeiro, significa que o usuário solicitara a troca da senha e já digitou a senha original, de modo que a senha digitada dessa vez era a senha nova que deve ser armazenada. Nisso se atualiza as *flags* para que não se leia mais senha, e que não mude mais a senha.

- **uiSenhaDeveMudar == 0?:** Se o teste anterior deu falso, significa que devemos verificar a senha. Se a senha não for correta, deve-se imprimir "Senha Incorreta"e encerrar a função. Se então a senha for verificada, temos:

  * **uiLigarAlarme == 1?:** Se for verdadeiro, devemos iniciar o *TIMER0* e ligá-lo, não ligando ainda o alarme, pois isso só deve ser ativado na função de *interrupt*, como veremos em seguida.

  * **uiDesligarAlarme == 1?:** Se for verdadeiro, devemos desligar o alarme, o timer e zerar o contador.

  * **uiMudaSenha == 1?:** Se for verdadeiro, significa que o usuário solicitou a mudança de senha e acabou de digitar a senha original, que foi verificada. Dessa maneira, é necessário que configuremos as *flags* para que entremos no caso *uiSenhaDeveMudar* que descrevemos anteriormente nessa função. Fazemos então com que as *flags* de leitura da senha e de mudança de senha fiquem ativos, além de reinicializarmos o contador de dígitos e desligar o *TIMER0*. Deve por fim imprimir "Senha:", para esperar que o usuário digite a senha nova.

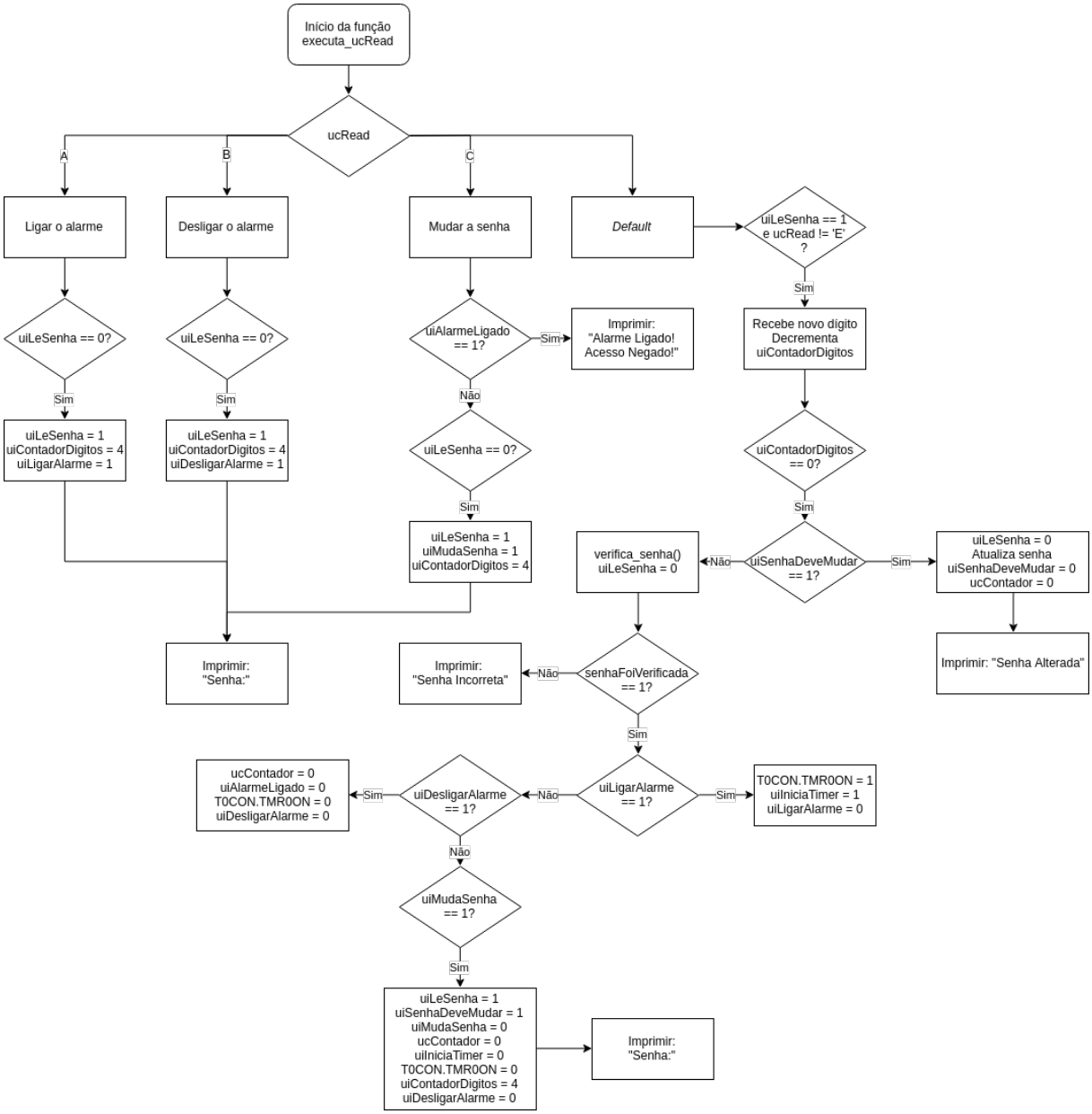

Figura 10 – Fluxograma do Alarme Residencial

2.2.4.6 Verificação da senha

A função para verificação de senha, subrotina chamada em *execute_ucRead*, é uma função de comparação da senha digitada pelo usuário com a senha salva pelo usuário. Temos o seu código:

```
1  /* Funcao para verificar se a senha esta correta */
2     void verifica_senha()
3     {
4        int i; // contador
5        for(i = 0, uiSenhaFoiVerificada = 1;i < NDIGITOS; i++){
```

```
6           if(uiSenhaDigitada[i] != uiSenha[i]){
7               uiSenhaFoiVerificada = 0;
8           }
9       }
10  }
```

2.2.4.7 Execução do *TIMER0*

Quando ativamos o *TIMER0* ao acertar a senha para ligar o alarme, o *TIMER0* inicia a contagem e, de acordo com os estouros do *TIMER0*, vai apagando os *LEDs* da PORTB. Destacamos que o número de estouros considerados antes de apagar todos os *LEDs* é ditado pelo valor que está no potenciômetro, como podemos ver no seguinte trecho de código:

```
1  /* Funcao para interrupcao via timer */
2     void interrupt(){
3        // Incrementa Contador.
4        if(INTCON.TMR0IF==1){       // Incrementa somente quando
            existir o overflow do timer 0.
5           // Recarrega o timer0.
6           TMR0H = 0xE1 ;          // Carrega o valor alto do
               numero 57723.
7           TMR0L = 0x7B;           // Carrega o valor baixo do
               numero 57723.
8
9           if(uiIniciaTimer == 1){
10             //ucContador = 0;
11
12             if(uiContadorNovo == 5 * pot1){
13                uiAlarmeLigado = 1;
14                uiContadorNovo = 0;
15                uiIniciaTimer = 0;
16                uiPrimeiraVez = 1;
17                uiMovimento = 1;
18             }
19             uiContadorNovo++;
20          }
21          INTCON.TMR0IF = 0;      // Limpa o flag de estouro do
               timer0 para uma nova contagem de tempo.
22          uiValorB = ucContador;
23          ucContador++;           // Esse operador aritmetico(++)
               realiza o mesmo que variavel = variavel + 1.
```

```
24        }
25    }
```

Além disso, temos o código para fazer com que os *LEDs* corretos acendam:

```
1  /* Funcao para acender os leds */
2     void acende_leds(int valor)
3     {
4        unsigned int faixa;
5
6        if(uiIniciaTimer == 1){
7           faixa = (255 - ucContador * (255/(pot1 * 5)))/31;
8           if(faixa == 1) PORTB = 0b10000000;
9           if(faixa == 2) PORTB = 0b11000000;
10          if(faixa == 3) PORTB = 0b11100000;
11          if(faixa == 4) PORTB = 0b11110000;
12          if(faixa == 5) PORTB = 0b11111000;
13          if(faixa == 6) PORTB = 0b11111100;
14          if(faixa == 7) PORTB = 0b11111110;
15          if(faixa == 8) PORTB = 0b11111111;
16       }else{
17          PORTB = 0x00;
18       }
19    }
```

2.2.4.8 Função Principal

Integrando todo o código explicando e garantindo a correta execução do *fluxo*. Destaque para o teste de se $PORTC.RC3 == 0$ para verificar se o sinal digital do sensor PIR está indicando que há presença ativando os sensores. Temos o código:

```
1  /* Funcao principal */
2    void main() {
3       setup_LCD();
4       setup_PIR();
5       setup_Potenciometro();
6       setup_LEDs();
7       setup_USART();
8       setup_TIMER();
9
10      uiAlarmeLigado = 0;
11
12      while(1)
13      {
```

```
14          read_Potenciometro();
15          read_keypad();
16          execute_ucRead();
17          delay_ms(500);
18          if(uiAlarmeLigado == 1){
19            if(uiPrimeiraVez == 1){
20              Lcd_Cmd(_LCD_CURSOR_OFF);//Apaga cursor
21              Lcd_Cmd(_LCD_CLEAR);//Limpa display
22              uiPrimeiraVez = 0;
23            }
24            Lcd_Out(1,4,"Alarme On!");
25            if(PORTC.RC3 == 0 ){
26               PORTC.RC1 = 1;
27               delay_ms(100);
28            }else{
29               PORTC.RC1 = 0;
30               delay_ms(100);
31            }
32         }else{
33             Lcd_Out(1,3,"Alarme Off!");
34               PORTC.RC1 = 1;
35               delay_ms(100);
36         }
37      }
38  }
```

3 Componentes do Projeto

3.1 Lista de componentes utilizados

- 1 placa Arduino Mega 2560;
- 1 teclado matricial 4×4 de membrana;
- 1 *buzzer* de $5V$ passivo;
- 8 *jumpers* macho-fêmea de $200mm$;
- 1 sensor de presença PIR;
- kit PIC Microgenios.

Referências

1 SENSOR PIR. 2019. Disponível em: <https://portal.vidadesilicio.com.br/sensor-de-presenca-hc-sr501/>. Citado na página 6.

2 USANDO o buzzer com Arduino – Transdutor piezo elétrico. 2019. Disponível em: <https://portal.vidadesilicio.com.br/usando-o-buzzer-com-arduino-transdutor-piezo-eletrico/>. Citado na página 8.

3 O que é um LED? 2019. Disponível em: <https://www.mundodaeletrica.com.br/o-que-e-um-led/>. Citado na página 9.

Apêndices

APÊNDICE A – execute_ucRead.v

```
1  /* Funcao para realizar a atividade de acordo com o botao
       apertado */
2     void execute_ucRead()
3     {
4       int i;
5         switch(ucRead){
6            case 'A': // Botao para ligar o alarme
7               if(uiLeSenha == 0){
8                  uiLeSenha = 1;
9                  uiContadorDigitos = NDIGITOS;
10                 uiLigarAlarme = 1;
11                 Lcd_Cmd(_LCD_CURSOR_OFF);//Apaga cursor
12                 Lcd_Cmd(_LCD_CLEAR);//Limpa display
13                 Lcd_Out(2,1,"Senha: ");
14              }
15              break;
16           case 'B': // Botao para desligar o alarme
17              if(uiLeSenha == 0){
18                 uiLeSenha = 1;
19                 uiContadorDigitos = NDIGITOS;
20                 uiDesligarAlarme = 1;
21                 uiIniciaTimer = 0;
22                 Lcd_Cmd(_LCD_CURSOR_OFF);//Apaga cursor
23                 Lcd_Cmd(_LCD_CLEAR);//Limpa display
24                 Lcd_Out(2,1,"Senha: ");
25              }
26              break;
27           case 'C': // Botao para mudar a senha
28              if(uiAlarmeLigado == 1){
29                 Lcd_Cmd(_LCD_CURSOR_OFF);//Apaga cursor
30                 Lcd_Cmd(_LCD_CLEAR);//Limpa display
31                 Lcd_Out(1,2,"Alarme Ligado!");
32                 Lcd_Out(2,2,"Acesso Negado!");
33                 delay_ms(5000);
34                 Lcd_Cmd(_LCD_CURSOR_OFF);//Apaga cursor
35                 Lcd_Cmd(_LCD_CLEAR);//Limpa display
36              } else {
37                 if(uiLeSenha == 0){
```

```
38                    uiLeSenha = 1;
39                    uiMudaSenha = 1;
40                    uiContadorDigitos = NDIGITOS;
41                    Lcd_Cmd(_LCD_CURSOR_OFF);//Apaga cursor
42                    Lcd_Cmd(_LCD_CLEAR);//Limpa display
43                    Lcd_Out(2,1,"Senha: ");
44                 }
45              }
46
47              break;
48           default: // Leitura dos numeros
49              if(uiLeSenha == 1 && !(ucRead == 69)){
50                 uiSenhaDigitada[NDIGITOS - uiContadorDigitos] = (
                      int)ucRead - 48; // arruma ucRead para int
51                 Lcd_Out(2,7 + (NDIGITOS - uiContadorDigitos),"*")
                      ;
52                 uiContadorDigitos--;
53                 if(uiContadorDigitos == 0){
54                    if(uiSenhaDeveMudar == 1){
55                       uiLeSenha = 0;
56                       for(i = 0;i < NDIGITOS;i++){
57                          uiSenha[i] = uiSenhaDigitada[i];
58                       }
59                       uiSenhaDeveMudar = 0;
60                       ucContador = 0;
61                       Lcd_Cmd(_LCD_CURSOR_OFF);//Apaga cursor
62                       Lcd_Cmd(_LCD_CLEAR);//Limpa display
63                       Lcd_Out(1,6,"Senha");
64                       Lcd_Out(2,4,"Alterada");
65                       delay_ms(1000);
66                       Lcd_Cmd(_LCD_CURSOR_OFF);//Apaga cursor
67                       Lcd_Cmd(_LCD_CLEAR);//Limpa display
68                    }else{
69                      verifica_senha();
70                      uiLeSenha = 0;
71                      if(uiSenhaFoiVerificada == 1){
72                          Lcd_Cmd(_LCD_CURSOR_OFF);//Apaga cursor
73                          Lcd_Cmd(_LCD_CLEAR);//Limpa display
74                         if(uiLigarAlarme == 1)
75                         {
76                            T0CON.TMR0ON = 1;
77                            uiIniciaTimer = 1;
```

```
78                              uiLigarAlarme = 0;
79                          }
80                          if(uiDesligarAlarme == 1){
81                              ucContador = 0;
82                              uiAlarmeLigado = 0;
83                               T0CON.TMR0ON = 0;
84                              uiDesligarAlarme = 0;
85                          }
86                          if(uiMudaSenha == 1){
87                              uiLeSenha = 1;
88                              uiSenhaDeveMudar = 1;
89                              uiMudaSenha = 0;
90                              ucContador = 0;
91                              uiIniciaTimer = 0;
92                              T0CON.TMR0ON = 0;
93                              uiContadorDigitos = NDIGITOS;
94                              uiDesligarAlarme = 0;
95                              Lcd_Cmd(_LCD_CURSOR_OFF);//Apaga
                                    cursor
96                              Lcd_Cmd(_LCD_CLEAR);//Limpa display
97                              Lcd_Out(2,1,"Senha: ");
98                          }
99                      }else{
100                          Lcd_Cmd(_LCD_CURSOR_OFF);//Apaga cursor
101                          Lcd_Cmd(_LCD_CLEAR);//Limpa display
102                          Lcd_Out(1,6,"Senha");
103                          Lcd_Out(2,4,"Incorreta");
104                          delay_ms(1000);
105                          Lcd_Cmd(_LCD_CURSOR_OFF);//Apaga cursor
106                          Lcd_Cmd(_LCD_CLEAR);//Limpa display
107                      }
108                  }
109              }
110          }
111      }
112  }
```