Porta Eletrônica com Teclado Matricial, Display 7 Segmentos, PIR e Buzzer

Relatório de Projeto

Autor: Mainviel Abellard Christley

Conteúdo

1 Escopo do Projeto

1.1 Apresentação do Projeto

Este projeto consiste em um **cofre eletrônico** controlado por um microcontrolador (Raspberry Pi Pico), que utiliza um *teclado matricial 4x4* para entrada de senha, um *display de 7 segmentos multiplexado* para exibição de informação, um *sensor PIR* para detectar movimento, um *buzzer* para sinalização sonora e um led rgb *LEDs* (verde e vermelho) para indicar estados de acesso e alerta. O sistema tem como principal objetivo **proteger** um ambiente ou dispositivo, exigindo a digitação de uma senha após a detecção de movimento.

1.2 Título do Projeto

- **Título:** porta com alarme Eletrônico com Sensor de Movimento e Senha de 4 Dígitos para zonas sensiveis

1.3 Objetivos do Projeto

- Implementar um sistema de senha de 4 dígitos, cadastrada pelo usuário na inicialização.
- Utilizar um sensor PIR para detectar movimento e iniciar o processo de verificação de senha.
- Fornecer sinalização visual (LEDs) e sonora (buzzer) para indicar tentativas de acesso, erros e bloqueios.
- Exibir o tempo decorrido desde a detecção de movimento em um display de 7 segmentos.
- Bloquear o teclado caso a senha seja digitada incorretamente três vezes consecutivas.

1.4 Descrição do Funcionamento

1. **Cadastro da Senha:** Ao iniciar o sistema, o usuário cadastra uma senha de 4 dígitos.
2. **Aguardo de Movimento:** O sistema permanece em estado de espera até que o sensor PIR detecte movimento.
3. **Entrada de Senha:** Assim que há movimento, o LED verde acende e o usuário tem a chance de inserir a senha. O tempo decorrido desde a detecção de movimento é exibido no display.
4. **Validação da Senha:**
    - Caso a senha seja correta, o LED verde permanece aceso por alguns segundos e, em seguida, o sistema é liberado (porta aberta ...).
    - Caso a senha seja incorreta, o LED vermelho pisca e o sistema permite novas tentativas. Se houver três tentativas falhas, o teclado é bloqueado por 5 segundos.
5. **Sinal Sonoro e Visual:** Passados 30 segundos sem acesso, o LED vermelho é acionado e o buzzer emite um sinal com intensidade crescente para indicar possível intrusão.

1.5 Justificativa

A segurança eletrônica é um dos pilares de projetos de IoT e sistemas embarcados. Uma porta digital com senha e sensor de movimento agrega mais confiabilidade na proteção de bens e informações. O uso do teclado matricial reduz o custo e a complexidade em relação a outras soluções (por exemplo, RFID, biometria, etc.), tornando este projeto acessível e didático.

1.6 Originalidade

Foi realizada pesquisa sobre sistemas de controle de acesso utilizando senhas e detectores de movimento. Embora existam projetos similares, este projeto destaca-se pela combinação de **sensor PIR**, **teclado matricial**, **display 7 segmentos** e **buzzer PWM** com sinais variáveis, além de um **bloqueio automático** após três tentativas malsucedidas e sinalização sonora crescente.

2 Especificação do Hardware

2.1 Diagrama em Bloco

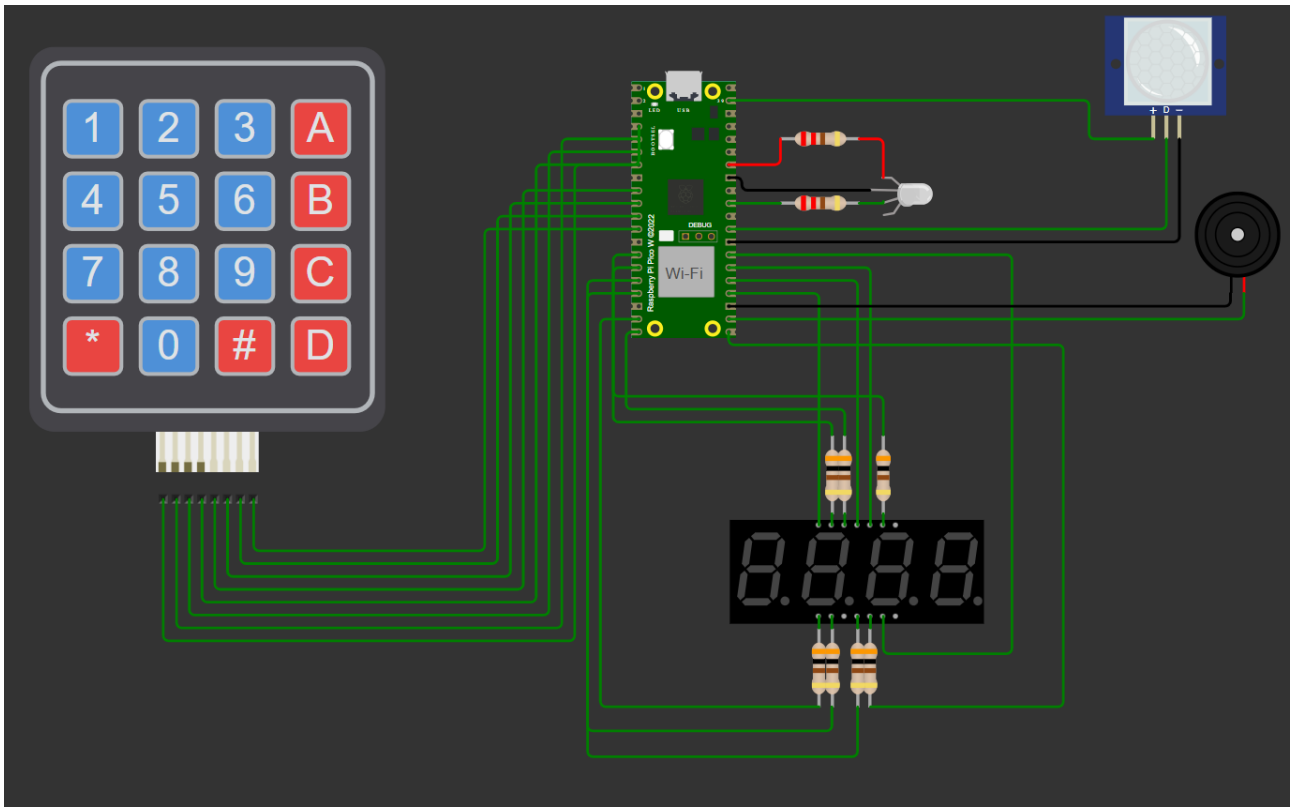


Figura 1: diagrama do sistema

2.2 Função de Cada Bloco

- **Teclado Matricial 4x4:** Entrada de dados numéricos (senha), controlado por pinos de linha e coluna.

- **Sensor PIR (Passive Infrared):** Detecta movimento gerando um sinal digital para o microcontrolador.
- **Buzzer PWM:** Emite sinais sonoros simples (beep rápido) ou modulados (beep crescente conforme o tempo que usamos nesse projeto).
- **LEDs:**
  - LED Verde: Indica acesso liberado ou detecção de movimento em progresso.
  - LED Vermelho: Indica tentativa de acesso incorreta, bloqueio ou sinal de alerta após 30 segundos.
- **Display 7 Segmentos (4 Dígitos):** Utilizado para exibir o tempo decorrido .

2.3 Configuração de Cada Bloco

- **Teclado Matricial:**
  - Linhas configuradas como saída digital.
  - Colunas configuradas como entrada digital com *pull-down*.
- **Sensor PIR:** Entrada digital com *pull-down*, gerando interrupção na borda de subida do sinal.
- **Buzzer (PWM):** Saída PWM com *duty cycle* e *frequência* configurados. Tensão de alimentação em 3,3 V.
- **LEDs:** Saída digital, ativação em nível lógico alto.
- **Display 7 Segmentos (Multiplexado):**
  - 7 pinos de segmentos (a-g) configurados como saída digital.
  - 4 pinos de seleção de dígitos (COM) configurados como saída digital.

2.4 Comandos e Registros Utilizados

No contexto do Raspberry Pi Pico (RP2040):

- **GPIO:** Uso das funções da `pico-sdk` como `gpio_init()`, `gpio_set_dir()`, `gpio_put()`, `gpio_get()`, etc.
- **PWM:** Configurado com `pwm_set_clkdiv()`, `pwm_set_wrap()`, `pwm_set_gpio_level()` e `pwm_set_enabled()`.
- **Interrupções:** Função de callback `gpio_set_irq_enabled_with_callback()` para tratar detecção de movimento.

2.5 Descrição da Pinagem Usada

- **Teclado Matricial (4 Linhas, 4 Colunas):**
  - Linhas: GPIO 2, 3, 4, 5
  - Colunas: GPIO 6, 7, 8, 9
- **Display 7 Segmentos:**
  - Segmentos (a-g): GPIO 10, 11, 12, 13, 14, 15, 16
  - Dígitos (COM): GPIO 18, 19, 20, 21
- **LED Verde:** GPIO 26
- **LED Vermelho:** GPIO 28
- **Sensor PIR:** GPIO 22
- **Buzzer (PWM):** GPIO 17

2.6 Circuito Completo do Hardware

O circuito completo inclui:

- **Microcontrolador (Raspberry Pi Pico)** conectado ao teclado, ao display, aos LEDs, ao sensor PIR e ao buzzer.
- **Alimentação** de 3,3 V fornecida pelo próprio Pico ou por fonte externa compatível.
- **Resistores** de adequação para o display de 7 segmentos (se necessário), dependendo do modelo usado.
- **Conexões do teclado** (Linhas como saída, Colunas como entrada com resistores *pull-down* internos).

3 Especificação do Firmware

3.1 Blocos Funcionais

- **Leitura do Teclado:** Varredura contínua (não-bloqueante) para detecção de teclas pressionadas.
- **Controle de Display:** Função para exibir números através de multiplexação rápida nos 4 dígitos.
- **Controle do Buzzer (PWM):** Configuração de frequência, *duty cycle* e intensidade do som.
- **Interrupção do PIR:** Rotina de *callback* que aciona a flag de movimento.
- **Validação da Senha:** Comparação de strings e contagem de tentativas erradas.
- **Rotina Principal (loop):** Controla a máquina de estados (aguardo de movimento, entrada de senha, bloqueio, etc.).

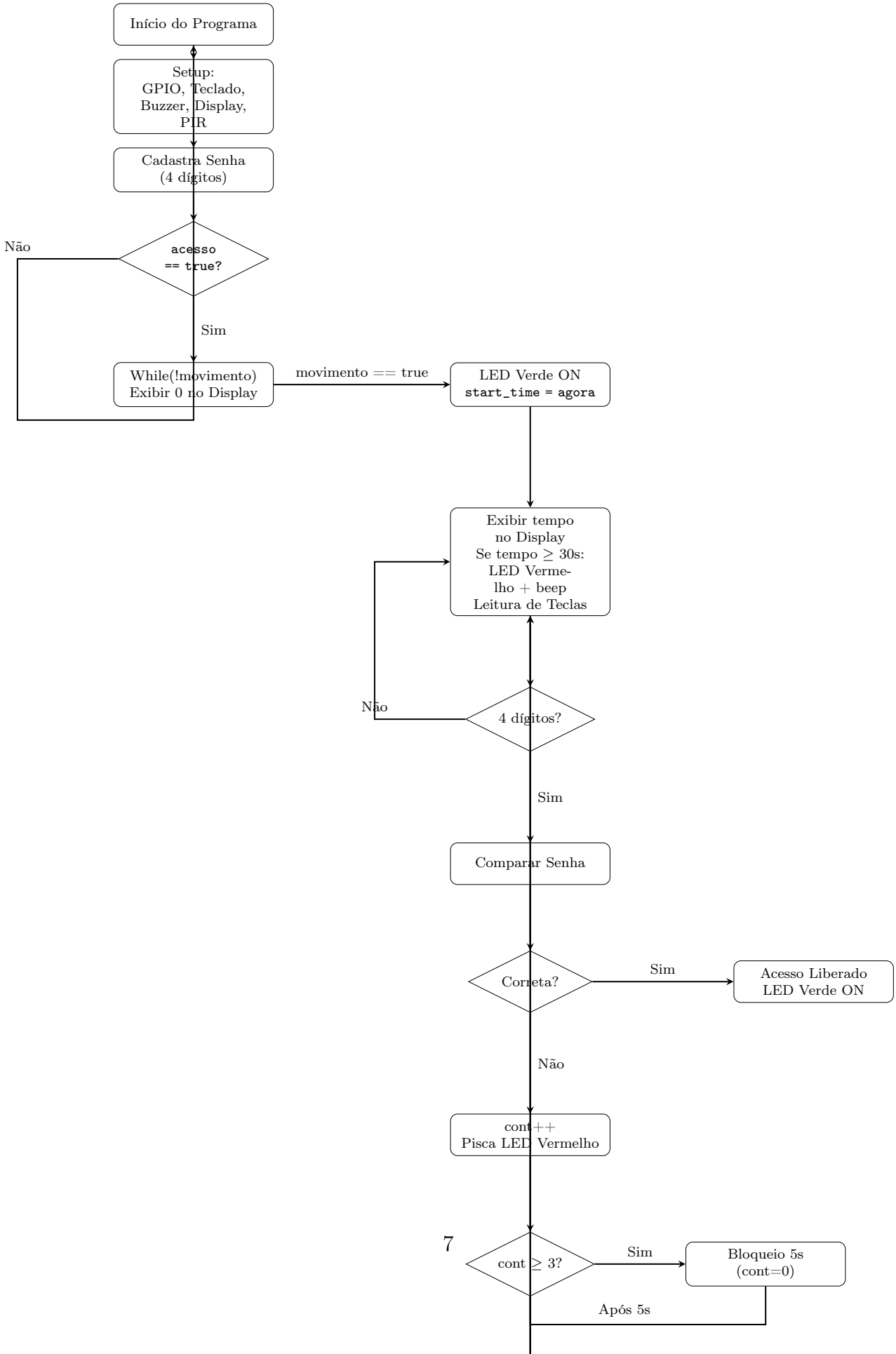
3.2 Descrição das Funcionalidades

- **Init Teclado:** Configura os pinos do teclado e prepara para leitura.
- **Leitura Teclado:** Retorna a tecla pressionada (se houver).
- **Compare Senhas:** Realiza comparação `strcmp()` entre a senha cadastrada e a senha digitada.
- **Setup:** Configura LEDs, display, GPIOs usados, e PWM do buzzer.
- **Show Digit / Display Number:** Responsáveis pela multiplexação do display.
- **Beep:** Emite som curto de confirmação para cada dígito pressionado.
- **Beep Crescente:** A cada 30s sem acerto de senha, a intensidade do buzzer aumenta.
- **Flag de Movimento (ISR):** Interrupção que sinaliza a mudança de estado da variável `movimento`.

3.3 Definição das Variáveis Principais

- `movimento (bool)`: Indica detecção de movimento pelo PIR.
- `password1[COMP+1]`: Armazena a senha cadastrada (4 dígitos + terminador).
- `acesso (bool)`: Mantém o sistema em operação (true até acertar a senha).
- `cont (int)`: Contador de tentativas erradas de senha.
- `duty_cycle (uint16_t)`: Valor de duty cycle para o buzzer.
- `slice_num (uint)`: Índice de configuração do PWM no RP2040.

3.4 Fluxograma do Sistema



3.5 Organização da Memória

Como se trata de um programa simples rodando em RAM (RP2040), não há divisão complexa. As variáveis globais (por exemplo, `movimento`, `password1`, etc.) permanecem na memória estática, enquanto as variáveis locais (ex: `input[]`) são alocadas na stack.

3.6 Protocolo de Comunicação

Este projeto não utiliza um protocolo de comunicação externo (como I2C, SPI ou UART) para o display ou o teclado, pois ambos são controlados diretamente via GPIO. Da mesma forma, o sensor PIR e o buzzer não utilizam protocolo de comunicação, apenas sinais digitais/PWM.

3.7 Formato do Pacote de Dados

Não se aplica, pois não há transmissão de dados em rede ou barramento serial. A senha é apenas manipulada internamente como string.

4 Execução do Projeto

4.1 Metodologia

1. **Pesquisa:** Estudo de exemplos de teclado matricial, uso de PWM no Pico, display multiplexado, sensor PIR.
2. **Escolha do Hardware:** Raspberry Pi Pico, teclado 4x4, display de 7 segmentos comum (cátodo compartilhado), sensor PIR, buzzer simples.
3. **Definição das Funcionalidades do Software:** Identificação de estados (aguardo de movimento, entrada de senha, bloqueio, etc.).
4. **Configuração da IDE:** Utilização do `CMake` e `pico-sdk` para compilar e gravar o firmware no RP2040.
5. **Programação na IDE:** Desenvolvimento do código em C, organização de funções, uso de bibliotecas da `pico-sdk`.
6. **Depuração:** Testes parciais (testar teclado, testar display, testar buzzer, testar PIR) e testes de integração.

4.2 Testes de Validação

- **Teste do Teclado:** Pressionar teclas e verificar se o caractere correspondente é reconhecido corretamente (exibindo no `printf`).
- **Teste do Display:** Enviar valores conhecidos (0, 1234, etc.) e verificar a exibição em cada dígito.
- **Teste do Buzzer:** Verificar se o *beep* ocorre ao pressionar teclas e se o tom aumenta a cada 30 segundos de espera.

- **Teste do PIR:** Movimentar-se diante do sensor para garantir que a interrupção é acionada e o LED verde acende.
- **Teste de Senha Correta/Errada:** Tentar senhas diferentes, validar a lógica de bloqueio após 3 erros consecutivos e a liberação com a senha correta.

4.3 Discussão dos Resultados

O sistema demonstrou boa estabilidade, respondendo prontamente à detecção de movimento e digitando a senha no teclado matricial. A multiplexação do display se mostrou eficiente para exibição do tempo, embora seja necessário ajustar os *delays* para evitar cintilação excessiva. O buzzer forneceu retorno auditivo claro, tanto nos beeps rápidos quanto no *beep* crescente após 30 segundos. O bloqueio de 5 segundos após três erros aumentou a segurança do sistema.

5 Referências

- Documentação oficial do Raspberry Pi Pico:
  `https://www.raspberrypi.org/documentation/pico-sdk`
- Exemplo de uso de teclado matricial em microcontroladores PIC e AVR, adaptado para RP2040.
- Datasheet do RP2040:
  `https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf`
- Documentação e exemplos práticos em sistemas embarcados vistos no curso:
  `WOLF, W. Computers as Components - Principles of Embedded Computing System Design. Morgan Kaufmann Publishers, San Francisco, 2ed., 2008.`

  `CUGNASCA, C. E. Projetos de Sistema Embarcados. Departamento de Engenharia de Computação e Sistemas Digitais. Escola Politécnica da USP, 02/2018.`

  `https://wokwi.com/projects/422808780374716417`

  `https://wokwi.com/projects/420803969862773761`

A Código-Fonte Completo em C com commentarios explicativos

```c
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/pwm.h"
#include <string.h>
#include <stdlib.h>

// ------------------ Defini  es Gerais ------------------
#define SET 1
#define RESET 0
#define COMP 4       // N mero de d gitos da senha

// ------------------ Pinos do Teclado Matricial 4x4 ------------------
const uint8_t Linha[] = {2, 3, 4, 5};
const uint8_t Coluna[] = {6, 7, 8, 9};

// Mapeamento das teclas em uma matriz 4x4
char teclas[4][4] = {
    {'1', '2', '3', 'A'},
    {'4', '5', '6', 'B'},
    {'7', '8', '9', 'C'},
    {'*', '0', '#', 'D'}
};

// ------------------ Display 7 segmentos (4 d gitos) ------------------
const uint8_t segment_pins[] = {10, 11, 12, 13, 14, 15, 16}; // Segmentos a-g
const uint8_t mux_display_pins[] = {18, 19, 20, 21};         // D gitos (COM)

// Mapeamento dos segmentos para os n meros (0-9)
const uint8_t display[10][7] = {
    {1, 1, 1, 1, 1, 1, 0}, // 0
    {0, 1, 1, 0, 0, 0, 0}, // 1
    {1, 1, 0, 1, 1, 0, 1}, // 2
    {1, 1, 1, 1, 0, 0, 1}, // 3
    {0, 1, 1, 0, 0, 1, 1}, // 4
    {1, 0, 1, 1, 0, 1, 1}, // 5
    {1, 0, 1, 1, 1, 1, 1}, // 6
    {1, 1, 1, 0, 0, 0, 0}, // 7
    {1, 1, 1, 1, 1, 1, 1}, // 8
    {1, 1, 1, 0, 0, 1, 1}  // 9
};

// ------------------ Pinos de LEDs ------------------
#define LED_VERDE 26
#define LED_VERMELHO 28

// ------------------ PIR e Buzzer (PWM) ------------------
#define PIR 22
#define BUZZER 17
```

```
// Par metros do PWM do buzzer
const uint16_t WRAP = 29860;
const float DIV = 16.0;
volatile uint16_t duty_cycle = 14930;  // Duty inicial
uint slice_num;                        // Canal do PWM

// ------------------ Flags e vari veis de estado ------------------
volatile bool movimento = false;    // Flag de detec  o de movimento
char password1[COMP + 1] = {0};     // Senha cadastrada (4 d gitos + '\0')
bool acesso = true;                 // Fica true at  senha correta
int cont = 0;                       // Contador de tentativas erradas

// ------------------ Prot tipos ------------------
void init_teclado(void);
char leitura_teclado(void);
bool compare_senhas(const char *password1, const char *password2);

void setup(void);
void show_digit(uint8_t digit_index, uint8_t digit);
void display_number(uint16_t number);

// PWM do Buzzer
void config_buzzer(void);
void beep_rapido(void);
void beep_crescente(uint32_t segundos);

// PIR
void config_gpio_pir(void);
void flag_movimento(uint gpio, uint32_t events);

// ------------------ Inicializar o teclado matricial ------------------
void init_teclado() {
    for (int i = 0; i < 4; i++) {
        gpio_init(Linha[i]);
        gpio_set_dir(Linha[i], GPIO_OUT);
        gpio_put(Linha[i], RESET);

        gpio_init(Coluna[i]);
        gpio_set_dir(Coluna[i], GPIO_IN);
        gpio_pull_down(Coluna[i]);
    }
}

// ------------------ Varredura n o-bloqueante do teclado
    ------------------
char leitura_teclado() {
    for (int row = 0; row < 4; row++) {
        gpio_put(Linha[row], SET);
        for (int col = 0; col < 4; col++) {
            if (gpio_get(Coluna[col])) {
                sleep_ms(50); // Debounce b sico
                while (gpio_get(Coluna[col])) {
                    tight_loop_contents();
                }
                gpio_put(Linha[row], RESET);
                return teclas[row][col];
```

```
            }
        }
        gpio_put(Linha[row], RESET);
    }
    return 0;
}

// ------------------ Comparar duas senhas ------------------
bool compare_senhas(const char *passwordA, const char *passwordB) {
    return strcmp(passwordA, passwordB) == 0;
}

// ------------------ Configurar LEDs e Display ------------------
void setup() {
    // LEDs
    gpio_init(LED_VERDE);
    gpio_set_dir(LED_VERDE, GPIO_OUT);
    gpio_put(LED_VERDE, 0);

    gpio_init(LED_VERMELHO);
    gpio_set_dir(LED_VERMELHO, GPIO_OUT);
    gpio_put(LED_VERMELHO, 0);

    // Segmentos do display
    for (int i = 0; i < 7; i++) {
        gpio_init(segment_pins[i]);
        gpio_set_dir(segment_pins[i], GPIO_OUT);
        gpio_put(segment_pins[i], 0);
    }

    // Pinos dos d gitos (mux)
    for (int i = 0; i < 4; i++) {
        gpio_init(mux_display_pins[i]);
        gpio_set_dir(mux_display_pins[i], GPIO_OUT);
        gpio_put(mux_display_pins[i], 1); // Desativados inicialmente
    }
}

// ------------------ Mostrar um n mero em um d gito espec fico
    ------------------
void show_digit(uint8_t digit_index, uint8_t digit) {
    // Desativa todos os d gitos
    for (int i = 0; i < 4; i++) {
        gpio_put(mux_display_pins[i], 1);
    }
    // Seta segmentos
    for (int i = 0; i < 7; i++) {
        gpio_put(segment_pins[i], display[digit][i]);
    }
    // Ativa d gito selecionado
    gpio_put(mux_display_pins[digit_index], 0);
}

// ------------------ Multiplexar n mero nos 4 d gitos do display
    ------------------
void display_number(uint16_t number) {
```

```
    // Extrai cada d gito
    uint8_t digits[4] = {
        (number / 1000) % 10, // Milhar
        (number / 100) % 10,  // Centena
        (number / 10) % 10,   // Dezena
        number % 10           // Unidade
    };
    // Multiplexa rapidamente pelos 4 d gitos
    for (int i = 0; i < 4; i++) {
        show_digit(i, digits[i]);
        sleep_us(300); // Tempo curto de exibi  o
    }
}

// ------------------ Configura  o e uso do PWM para o Buzzer
    ------------------
void config_buzzer() {
    gpio_set_function(BUZZER, GPIO_FUNC_PWM);
    slice_num = pwm_gpio_to_slice_num(BUZZER);

    pwm_set_clkdiv(slice_num, DIV);
    pwm_set_wrap(slice_num, WRAP);

    // Duty cycle inicial
    pwm_set_gpio_level(BUZZER, duty_cycle);
    pwm_set_enabled(slice_num, false);
}

// Beep simples de 300 ms
void beep_rapido() {
    pwm_set_gpio_level(BUZZER, duty_cycle);
    pwm_set_enabled(slice_num, true);
    sleep_ms(300);
    pwm_set_enabled(slice_num, false);
}

// Apita com intensidade crescente a cada 30s
void beep_crescente(uint32_t segundos) {
    uint32_t bloco30 = segundos / 30;
    if (bloco30 > 0) {
        uint16_t novoDuty = 14930 + (bloco30 * 2000);
        if (novoDuty > WRAP - 1) {
            novoDuty = WRAP - 1; // Limita
        }
        pwm_set_gpio_level(BUZZER, novoDuty);
        pwm_set_enabled(slice_num, true);
        sleep_ms(500);
        pwm_set_enabled(slice_num, false);

        // Volta ao duty base
        pwm_set_gpio_level(BUZZER, duty_cycle);
    }
}

// ------------------ Configura  o do PIR e rotina de interrup  o
    ------------------
```

```
void config_gpio_pir() {
    gpio_init(PIR);
    gpio_set_dir(PIR, GPIO_IN);
    gpio_pull_down(PIR);
    // Habilita interrup  o na subida
    gpio_set_irq_enabled_with_callback(PIR, GPIO_IRQ_EDGE_RISE, true, &
       flag_movimento);
}

// Callback que ativa a flag
void flag_movimento(uint gpio, uint32_t events) {
    movimento = true;
}

// ------------------ MAIN ------------------
int main() {
    stdio_init_all();
    setup();          // LEDs + Display
    init_teclado();   // Teclado matricial
    config_buzzer();  // PWM do Buzzer
    config_gpio_pir();// PIR

    // 1) Cadastra a senha inicialmente (bloqueante, s   ocorre 1x)
    printf("Cadastro da senha do cofre com 4 digitos!\n");
    {
        int index = 0;
        while (index < COMP) {
            char key = leitura_teclado();
            if (key != 0) {
                password1[index] = key;
                password1[index + 1] = '\0';
                printf("%c", key);
                index++;
                sleep_ms(200);
            }
        }
    }
    printf("\nSenha gravada!\n");

    while (acesso) {
        // 2) Espera movimento: display = 0, LEDs apagados
        while (!movimento && acesso) {
            for (int i = 0; i < 20; i++) {
                display_number(0);
                sleep_ms(5);
            }
        }

        // Assim que detectamos movimento, ligamos LED verde
        gpio_put(LED_VERDE, 1);
        gpio_put(LED_VERMELHO, 0);

        // Marca o tempo em que detectamos movimento
        uint32_t start_time = to_ms_since_boot(get_absolute_time()) / 1000;
           // seg
```

```
// Buffer para capturar a senha no loop principal
char input[COMP + 1] = {0};
int idx_senha = 0;

// 3) Fica num loop at  acertar a senha ou 3 erros (com bloqueio)
while (acesso) {
    // Calcula tempo decorrido
    uint32_t now_s = to_ms_since_boot(get_absolute_time()) / 1000;
    uint32_t elapsed = now_s - start_time;

    // 3.1) Mostra no display o tempo decorrido
    for (int i = 0; i < 10; i++) {
        display_number((elapsed > 9999) ? 9999 : elapsed);
        sleep_ms(100);
    }

    // 3.2) Se passaram 30s sem digitar nada, LED=vermelho, beep
       crescente
    if (elapsed >= 30) {
        gpio_put(LED_VERDE, 0);
        gpio_put(LED_VERMELHO, 1);
        beep_crescente(elapsed);
    }

    // 3.3) Verifica se foi pressionada alguma tecla
    char key = leitura_teclado();
    if (key != 0) {
        // Conforme o usu rio pressiona, preenche 'input'
        input[idx_senha] = key;
        input[idx_senha + 1] = '\0';
        printf("%c", key); // debug
        beep_rapido();     // beep a cada d gito
        idx_senha++;

        // Se completou 4 d gitos, validamos
        if (idx_senha >= COMP) {
            printf("\nSenha lida: %s\n", input);

            if (compare_senhas(password1, input)) {
                // Senha correta
                printf("Senha Correta! Acesso concedido.\n");
                gpio_put(LED_VERDE, 1);
                gpio_put(LED_VERMELHO, 0);
                sleep_ms(3000);
                gpio_put(LED_VERDE, 0);
                acesso = false; // Sai do while principal
                break;
            } else {
                // Senha incorreta
                printf("Senha Incorreta! Acesso negado.\n");
                cont++;

                // Pisca LED vermelho
                for (int i = 0; i < 3; i++) {
                    gpio_put(LED_VERMELHO, 1);
                    sleep_ms(200);
```

```
                        gpio_put(LED_VERMELHO, 0);
                        sleep_ms(200);
                    }

                    // Se errou 3 vezes, bloqueia por 5s
                    if (cont >= 3) {
                        printf("Cofre bloqueado por 5 segundos.\n");
                        gpio_put(LED_VERDE, 0);
                        gpio_put(LED_VERMELHO, 0);
                        sleep_ms(5000);
                        cont = 0; // Zera contagem
                    }

                    // Reseta buffer de entrada
                    memset(input, 0, sizeof(input));
                    idx_senha = 0;
                }
            }
        }
        // Tempo segue contando, n o zera start_time ap s tentativa.
    }
}
    return 0;
}
```