ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

PCS3732 - Laboratório de Processadores

Teclado B.A.R.A.L.D.I

Integrantes:

Enzo Tassini das Neves	13682552
Gabriel Baraldi Souza	13680532
João Felipe Pereira Carvalho	11808189

15/Agosto/2025

Sumário

1. Introdução

Esse documento relata o desenvolvimento do projeto B.A.R.A.L.D.I, acrônimo para Bruno Approved Realtime Access to Low-level Digital Inputs. A proposta consistiu na criação de um sistema de entrada, baseado em teclado matricial, implementado em ambiente bare-metal sobre a plataforma Raspberry Pi. O sistema deveria ser capaz de realizar a leitura direta do estado das teclas por meio de varredura de linhas e colunas, aplicar um mecanismo de debounce para evitar leituras indevidas decorrentes do ruído mecânico e, posteriormente, transmitir os caracteres digitados a um terminal conectado via comunicação serial UART.

O projeto começou com a escolha de um teclado matricial 4x4 como protótipo inicial, em razão de sua simplicidade física e facilidade de compreensão lógica, e evoluiu até tentativas de integração com a membrana de um teclado comercial completo. Toda a lógica foi desenvolvida em linguagem C, com acesso direto aos registradores da Raspberry Pi, sem uso de sistema operacional ou bibliotecas de alto nível, o que proporcionou uma interação direta com o hardware.

2. Implementação inicial

A primeira semana de trabalho se concentrou na identificação dos pinos do teclado matricial, determinando quais correspondiam às linhas e quais às colunas. Foi produzida uma representação visual, presente na Figura 1, que passou a servir como referência para todas as etapas por vir.

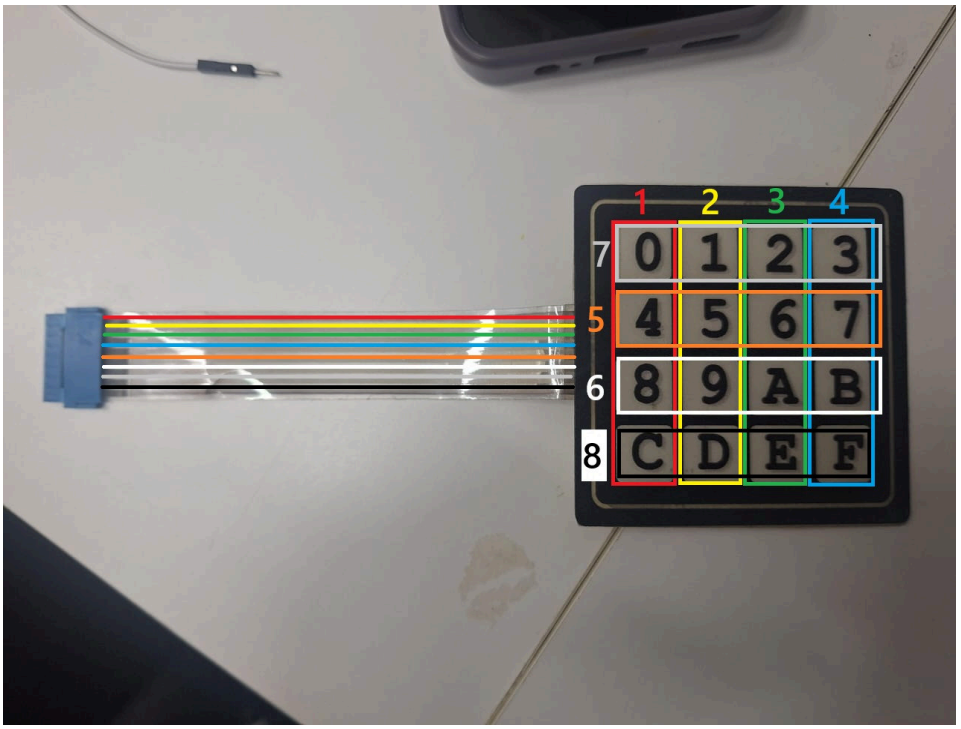

Figura 1: Diagrama de mapeamento do teclado.

Em seguida, as conexões do teclado foram associadas a GPIOs fixos da Raspberry Pi. A seleção foi feita levando em consideração a necessidade de evitar conflitos com as portas JTAG já presentes, e a configuração final foi a seguinte: colunas conectadas aos GPIOs 19, 13, 6 e 5, e linhas conectadas aos GPIOs 16, 20, 12 e 21, conforme representação visual da Figura 2.

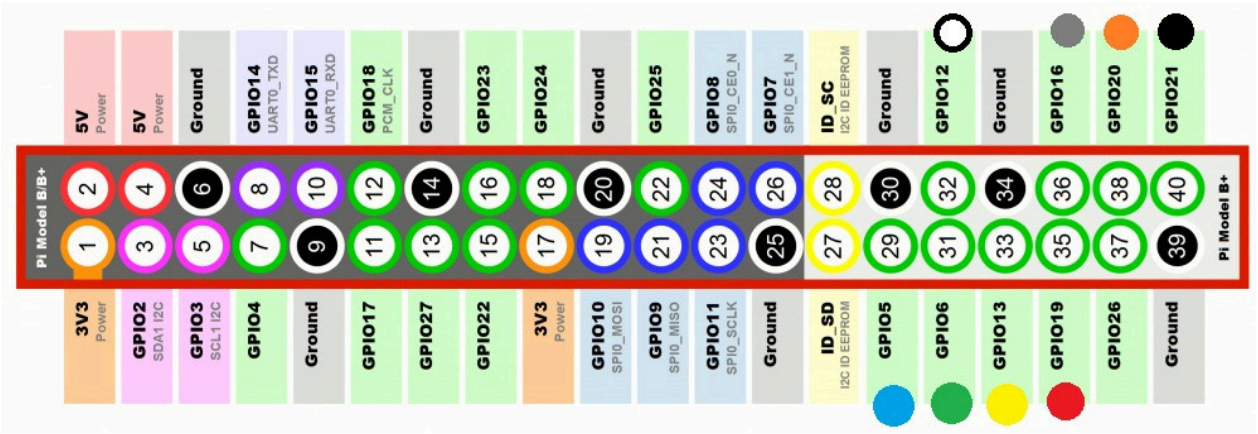

Figura 2: Diagrama de mapeamento das GPIOs.

A lógica de leitura adotada baseou-se na técnica de varredura por colunas, na qual uma única coluna é ativada por vez e as linhas correspondentes são lidas sequencialmente, detectando assim quais teclas estão pressionadas. Para reduzir leituras duplicadas provocadas pelo efeito físico de bouncing, foi implementado um primeiro mecanismo de debounce de funcionamento simples.

A implementação foi testada diretamente na Raspberry Pi, com auxílio do GDB para depuração em tempo real, possibilitando visualizar o comportamento da matriz durante o acionamento das teclas e avaliar o impacto do debounce nas leituras. A Figura 3 apresenta um dos testes com o GDB, que mostra uma implementação inicial funcional, com as teclas apertadas tendo seus respectivos valores registrados na variável “output”, por mais que ainda com um grande bouncing.

```
Breakpoint 1, trata_irq () at sched.c:247
247         if (bit_is_set(IRQ_REG(pending_basic), 0)) {
(gdb) c
Continuing.

Breakpoint 1, trata_irq () at sched.c:247
247         if (bit_is_set(IRQ_REG(pending_basic), 0)) {
(gdb) p output
$1 = {0 <repeats 84 times>, 1 <repeats 52 times>, 2 <repeats 74 times>, 3 <repeats 66 times>, 4 <repeats 66 times>,
  5 <repeats 64 times>, 6 <repeats 69 times>, 7 <repeats 68 times>, 8 <repeats 66 times>, 9 <repeats 63 times>,
  10 <repeats 66 times>, 11 <repeats 59 times>, 12 <repeats 56 times>, 13 <repeats 66 times>, 14 <repeats 54 times>,
  15 <repeats 24 times>, 0 <repeats 27 times>}
```

Figura 3: Teste inicial com o GDB.

Ao término dessa fase, foram definidos como próximos passos a implementação de comunicação UART para exibição em terminal, o aprimoramento do debounce e a criação de um buffer circular mais robusto para armazenamento de teclas.

3. Aperfeiçoamento do sistema

A segunda etapa de desenvolvimento foi marcada pela integração da comunicação serial UART, recurso essencial para permitir que os caracteres digitados no teclado físico fossem enviados e exibidos em tempo real em um terminal conectado à Raspberry Pi. Para isso, foi utilizada a UART0 da placa e um cabo USB-TTL conectado aos pinos GPIO correspondentes. Essa configuração foi implementada em um módulo dedicado, a partir de uma solução de colegas do grupo CrypGas, conforme a Figura 4. Nesse caso, a única função utilizada além de “uart_init” é “uart_send”, que envia um caractere para o terminal a um baudrate de 9600.

```
#include "uart.h"
#include <stdint.h>

#define UART0_BASE 0x3F201000

#define UART0_DR   (*(volatile uint32_t *)(UART0_BASE + 0x00))
#define UART0_FR   (*(volatile uint32_t *)(UART0_BASE + 0x18))
#define UART0_IBRD (*(volatile uint32_t *)(UART0_BASE + 0x24))
#define UART0_FBRD (*(volatile uint32_t *)(UART0_BASE + 0x28))
#define UART0_LCRH (*(volatile uint32_t *)(UART0_BASE + 0x2C))
#define UART0_CR   (*(volatile uint32_t *)(UART0_BASE + 0x30))
#define UART0_IMSC (*(volatile uint32_t *)(UART0_BASE + 0x38))
#define UART0_ICR  (*(volatile uint32_t *)(UART0_BASE + 0x44))


void uart_init(void) {
    UART0_CR = 0x00000000;

    UART0_ICR = 0x7FF;

    UART0_IBRD = 312;
    UART0_FBRD = 32;

    UART0_LCRH = (3 << 5);
    UART0_CR = 0x301;
}

void uart_send(char c){
    while(UART0_FR & (1 << 5));
    UART0_DR = c;
}

char uart_get(void){
    while (UART0_FR & (1 << 4));
    return (char)(UART0_DR & 0xFF);
}

void uart_puts(const char *str){
    while (*str)
        uart_send(*str++);
}
```

Figura 4: Módulo de comunicação UART.

Na mesma semana de desenvolvimento, a função de leitura do teclado “read_keyboard” foi adaptada para, além de identificar a tecla pressionada, realizar a transmissão do caractere correspondente ao terminal, imediatamente após sua validação. Para essa conversão de posição física (linha, coluna) para caracteres ASCII, foi criada uma função específica, como mostra a Figura 5. Nota-se que a conversão para números (0 a 9) e letras (A a F) é ligeiramente diferente.

```
char keyboard_position_conversion(int row, int col) {
    bool is_letter = (row == 3) || ((row == 2) && (col > 1));
    if (is_letter) {
        return (char) (row-2) + ((col-2) * ROWS) + 65;
    } else {
        return (char) row + (col * ROWS) + 48;
    }
}
```

Figura 5: Conversão para caracteres ASCII.

Ao mesmo tempo, o mecanismo de debounce foi aprimorado para um modelo progressivo, semelhante ao observado em teclados comerciais, nos quais a primeira letra é enviada praticamente de forma instantânea, após um pequeno atraso ocorre a repetição da segunda letra e, se a tecla permanecer pressionada, as repetições seguintes ocorrem em ritmo acelerado.

Esse comportamento foi implementado utilizando uma matriz de estado “already_read_matrix” associada a três parâmetros ajustáveis: o número de ciclos para a primeira detecção, o atraso para a segunda emissão e o número de ciclos reduzido para as repetições seguintes. A Figura 6 mostra a função “read_keyboard” adaptada com as novas funcionalidades.

```
void read_keyboard() {
    while (1) {
        for (int c = 0; c < COLS; c++) {
            write_io(out_pins[c], true);

            for(int r = 0; r < ROWS; r++) {
                if (read_io(in_pins[r])) {
                    if ((read_matrix[r][c] < FIRST_DEBOUNCE_CYCLES && !already_read_matrix[r][c])
                        || (read_matrix[r][c] < REMAINING_DEBOUNCE_CYCLES && already_read_matrix[r][c])
                    )
                        read_matrix[r][c]++;
                    else {
                        output[last_output++] = r * ROWS + c;

                        if (uart_enabled)
                            uart_send(keyboard_position_conversion(r, c));

                        if (!already_read_matrix[r][c]) {
                            read_matrix[r][c] = -SECOND_DEBOUNCE_DELAY;
                            already_read_matrix[r][c] = true;
                        } else{
                            read_matrix[r][c] = 0;
                        }

                        if (last_output >= OUTPUT_LENGTH) {
                            last_output = 0;
                        }
                    }
                } else {
                    already_read_matrix[r][c] = false;
                    read_matrix[r][c] = 0;
                }
            }

            write_io(out_pins[c], false);
        }
    }
}
```

Figura 6: Função “read_keyboard” atualizada.

Com essas melhorias, foi possível realizar testes no terminal serial, digitando diretamente no teclado físico e visualizando a resposta imediata na tela. A captura do terminal exibindo os caracteres transmitidos está representada na Figura 7.

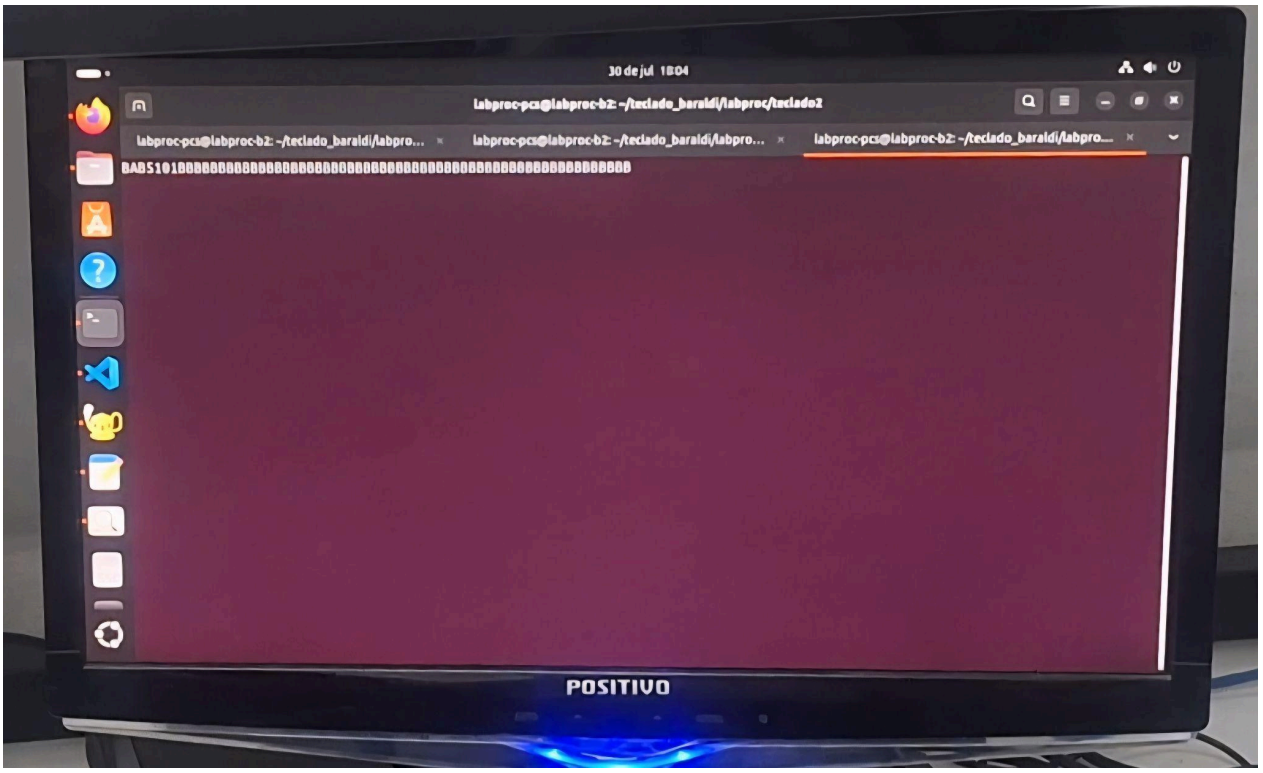

Figura 7: Exibição dos caracteres pressionados no terminal.

4. Tentativa de expansão

Na semana seguinte, houve uma tentativa de expansão do sistema, com o objetivo de suportar um teclado completo contendo todas as letras, números e símbolos. Para isso, foi realizada a desmontagem de um teclado de computador comercial, a fim de estudar e reaproveitar sua membrana interna, conectando-a diretamente aos GPIOs da Raspberry Pi e substituindo o conversor USB original.

A análise revelou que o teclado é composto por duas membranas sobrepostas, uma contendo oito trilhas condutoras e outra com dezoito. Essas trilhas, impressas sobre um filme plástico flexível e translúcido, formam a matriz de teclas, que fecha contato momentaneamente quando uma tecla é pressionada. A Figura 8 apresenta a visualização das membranas, conforme descritas. No teclado original, elas são conectadas a uma pequena placa controladora, que detecta o fechamento do contato, processa o sinal e o converte para o protocolo USB HID.

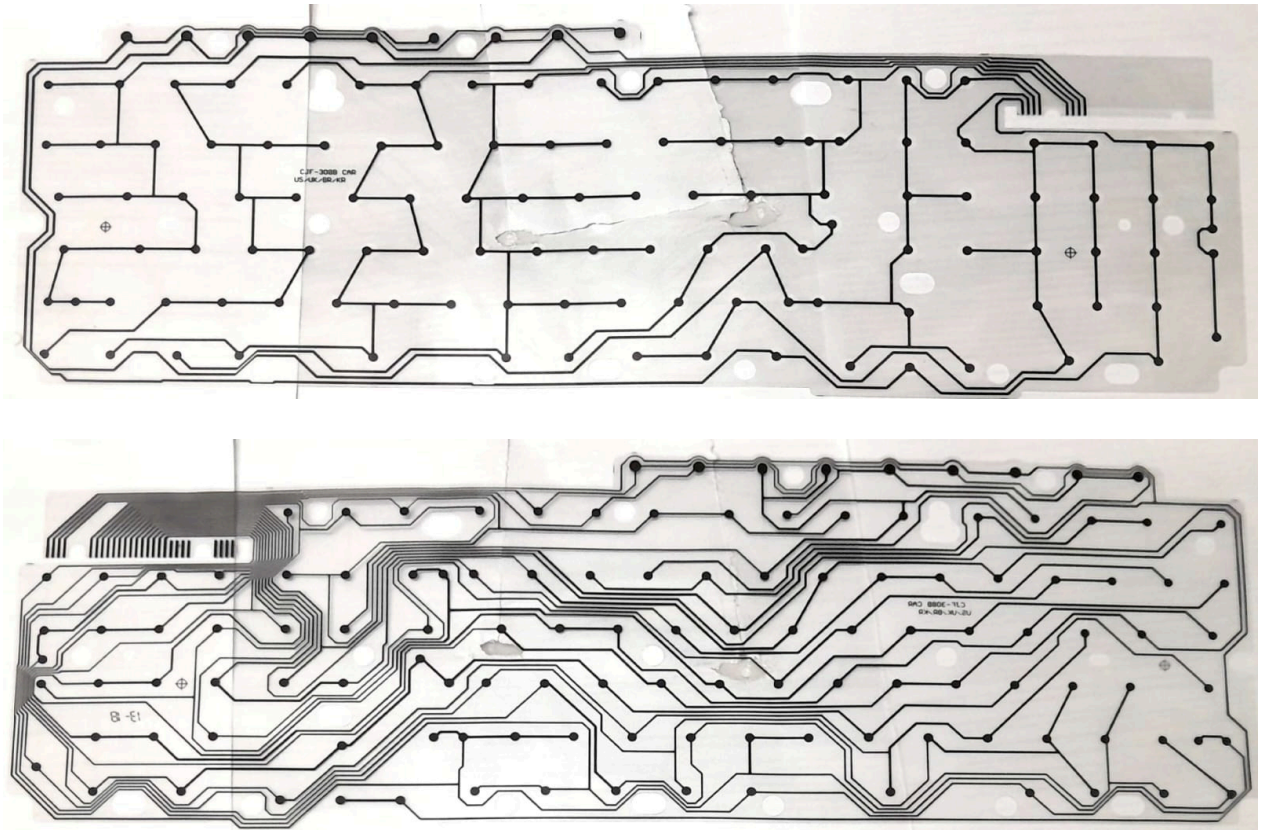

Figura 8: Membranas de um teclado comercial.

A proposta era substituir essa placa controladora por conexões diretas aos GPIOs e efetuar a leitura via software, mas surgiram problemas significativos. O espaçamento (pitch) entre as trilhas da membrana é de apenas 1 mm, enquanto os conectores e jumpers padrão utilizados possuem espaçamento de 2,54 mm, inviabilizando a conexão direta. Além disso, a soldagem em um pitch tão reduzido exigiria fios extremamente finos e ferramentas adequadas, com risco elevado de danificar a membrana durante o processo. A fabricação de uma PCB adaptadora, que converteria de 1 mm para 2,54 mm, foi considerada, mas o tempo necessário para projetar, fabricar e receber a placa ultrapassava o prazo disponível.

Foram também levadas em consideração alternativas como a aquisição de conectores FFC/FPC compatíveis, o reaproveitamento de cabos e adaptadores de outros dispositivos, e até improvisações com grafite para continuidade das trilhas, mas todas apresentaram problemas de confiabilidade, compatibilidade ou prazo. Diante dessas dificuldades, decidiu-se encerrar o projeto nessa fase, mantendo como entrega final o sistema funcional baseado no teclado matricial 4x4.

5. Conclusão

Apesar de não ter sido possível concluir a integração com o teclado completo, o projeto B.A.R.A.L.D.I. alcançou seus objetivos principais, entregando um sistema de leitura de teclado matricial totalmente funcional em ambiente bare-metal, com comunicação UART integrada e mecanismo de debounce progressivo. A experiência permitiu compreender não apenas a lógica de varredura de teclas e o controle por software, mas também os desafios mecânicos e elétricos associados à interface física com dispositivos de alta densidade.

Assim, embora o objetivo de expansão não tenha sido atingido, o projeto proporcionou um ganho significativo em conhecimento prático e em compreensão da interação direta entre hardware e software, cumprindo seu papel no contexto da disciplina.

6. Anexo

A implementação completa do código pode ser acessada através do seguinte link: https://github.com/BaraldsBR/labproc