RetroPM

Introducere

Proiectul constă în implementarea unui “PC” inspirat de cele old school pentru care interfața cu utilizatorul se rezuma la introducerea unor comenzi în terminal. O sa îl numesc RetroPM.

Prin acest proiect, mi-am propus sa replic un sistem ce poate reproduce într-un mod minimal gestionarea de fișiere și utilizatori.

Descriere generală

RetroPM se bazează în principal pe comunicarea dintre 3 microcontrollere cu scopuri diferite:

- ESP32: Se ocupa de afișarea pe 2 display-uri a input-ului si output-ului pentru utilizator (comenzile în timp ce se tastează, un prompt cu utilizatorul curent etc.) și cu conectarea la un server NTP pentru a afișa ora actuala pe display.
- STM32F103: Se ocupa de logica principala (FSM-ul) a sistemului. Acesta este responsabil de procesarea input-ului, de a decide dacă o comanda este invalida, ce privilegii are un user când când dorește sa modifice un fișier etc.
- Raspberry Pi Pico: Se ocupa de prelucrarea input-ului de la utilizator (tastele sunt procesate în caractere ASCII) pentru a le comunica către STM32.

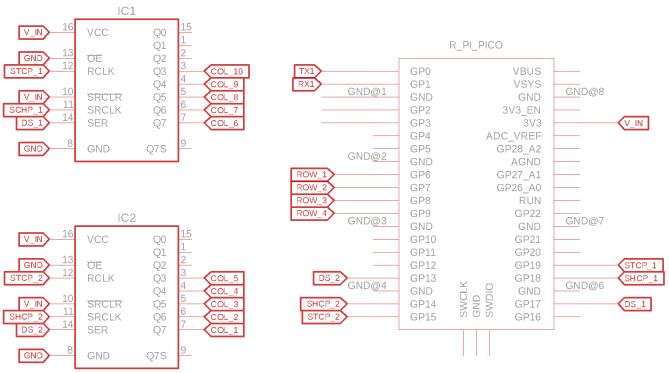
Hardware Design

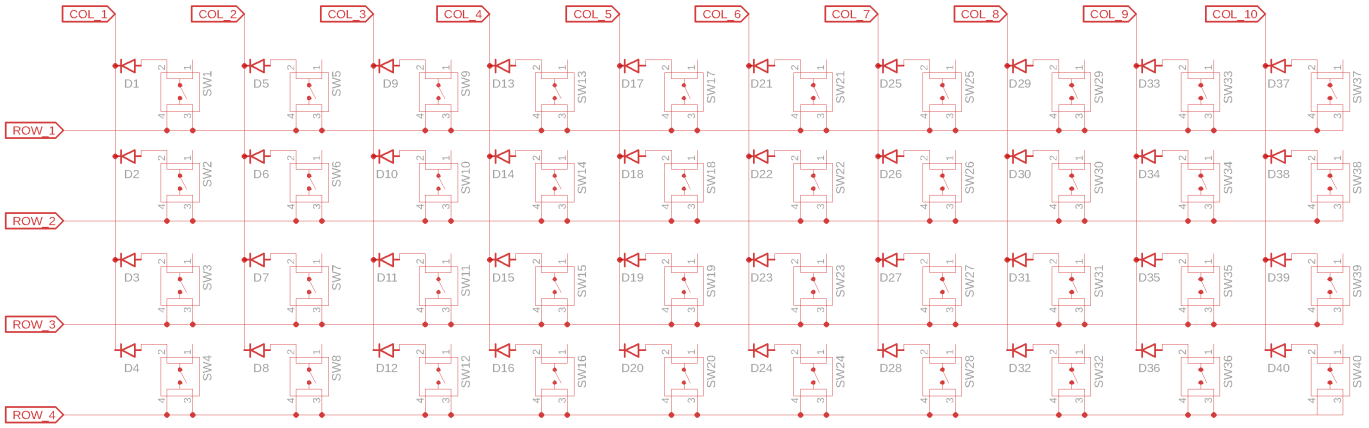
Lista componente:

- STM32F103
- Raspberry Pi Pico
- ESP32
- Modul microSD + microSD 32GB
- Shift registers
- Diode, Resistori
- Pushbuttons
- Memorie EEPROM
- Ecran OLED
- Ecran LCD 16×2
- Sursa alimentare

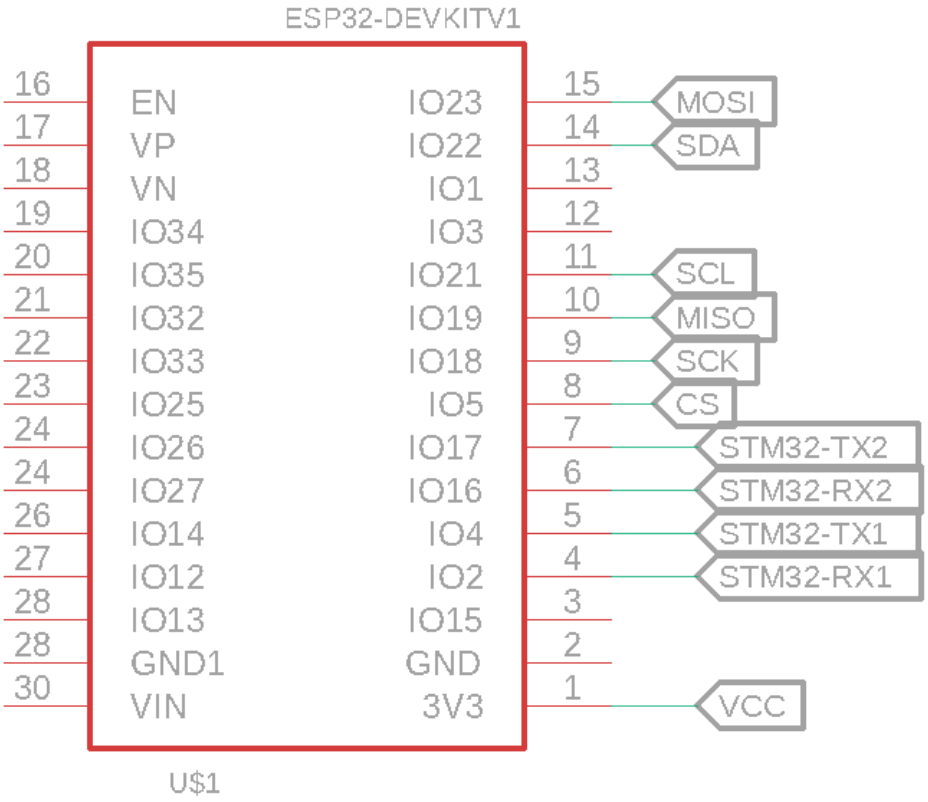
- Level shifter

Schema pentru Raspberry Pi Pico (partea de “tastatura”):





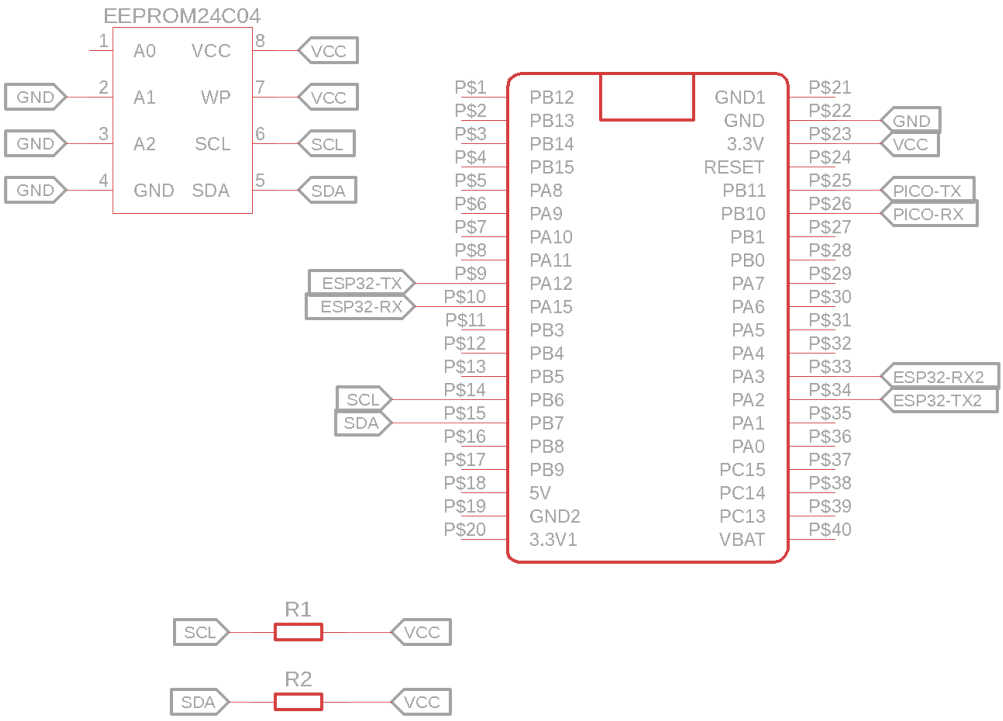
Schema pentru ESP32 (celelalte periferice: display, microSD):



(Notita: Am uitat sa pun in schema I2C-ul pentru ESP32, are rezistenta de pull-up ca STM32 si

foloseste un singur bus pentru ambele display-uri)

Schema pentru STM32 (microcontroller principal):



Software Design

Pentru STM32 și Raspeberry Pi Pico am folosit Rust, cu biblioteci de Hardware Layer Abstraction. Pentru ESP32, am folosit Arduino IDE și C++. Lista bibliotecilor folosite:

- C++
  - SPI.h, SD.h pentru modulul de microSD
  - Adafruit_GFX, Adafruit_SH110X, hd44780 pentru cele doua display-uri
  - Wire pentru I2C
  - HardwareSerial pentru UART
- Rust
  - rp2040_hal - pentru Pico
  - stm32f1xx_hal, eeprom34x (pentru a scrie/citi din IC-ul de memorie) - pentru STM32F103

Pentru partea de procesare de tastatura, a trebuit în primul rând sa implementez o funcție asemănătoare cu cea de shiftOut din Arduino.h (nu am găsit-o în vreu crate de Rust). Folosind 2 shift registeri ca output-uri (fiecare cu 5 coloane din matricea de butoane asignate), și 4 coloane ca input-uri pentru a reduce numărul de pini folosiți pe placa, am folosit o tehnica de matrix scanning pentru a decide ce buton a fost apăsat de utilizator. Astfel am putut sa scanez mai multe butoane în același ciclu de loop. (Lucrul acesta a fost folosit pentru a reproduce efectul de shift + lowercase = uppercase).

Code snippet: Prima parte din loop se ocupa cu scanarea și stocarea intr-o matrice de tip bool dacă tasta de la poziția i,j a fost apăsată.

```
        // Bits 9:5 for first shift register
        // Bits 4:0 for second shift register
        let mut bits = 0x0200u16; // 0000_0010_0000_0000

        // Mask for second shift register -> 0000_0000_0001_1111
        let mask = 0x001Fu16;

        for col in (0u8..10u8).rev() {
            // Byte to serial input for first shift register.
            let byte_higher = !((bits >> 5).to_be_bytes()[1]);

            // Byte to serial input for second shift register.
            let byte_lower = !((bits & mask).to_be_bytes()[1]);

            latch_pin_1.set_low().unwrap();
            latch_pin_2.set_low().unwrap();
            // Shift out value from register
            kb_lib::shift_out(&byte_lower, &mut data_pin_2, &mut clock_pin_2,
&mut delay, kb_lib::BitOrder::LSBFIRST);
            kb_lib::shift_out(&byte_higher, &mut data_pin_1, &mut
clock_pin_1, &mut delay, kb_lib::BitOrder::LSBFIRST);

            // Read from the input pins for scanning:
            let idx = if col == 9 { 9usize } else { 8 - col as usize };

            if rows0.is_low().unwrap() {
                keyState[0][idx] = true;
            } else {
                keyState[0][idx] = false;
            }

            if rows1.is_low().unwrap() {
                keyState[1][idx] = true;
            } else {
                keyState[1][idx] = false;
            }

            if rows2.is_low().unwrap() {
                if idx == 0 {
                    is_shift_pressed = true;
                } else {
                    keyState[2][idx] = true;
                }
            } else {
                keyState[2][idx] = false;
            }

            if rows3.is_low().unwrap() {
                keyState[3][idx] = true;
            } else {
                keyState[3][idx] = false;
```

```
            }

            bits >>= 1;
            latch_pin_1.set_high().unwrap();
            latch_pin_2.set_high().unwrap();
        }
```

A doua parte se ocupa cu comunicarea către STM32:

```
for i in 0..4 {
            for j in 0..10 {
                if keyState[i][j] {

                    if is_shift_pressed {
                        last_key_pressed = kb_lib::SHIFT_KEYS[i][j];
                        is_shift_pressed = !is_shift_pressed;
                    } else {
                        last_key_pressed = KEYS[i][j];
                    }
                    delay.delay_ms(100);
                    write!(uart,"{}", &last_key_pressed).unwrap();
                    delay.delay_ms(100);
                    break;
                }
            }
        }
```

Pentru partea de logica principala, atât în STM32, cât și în ESP32 am folosit logica unor FSM-uri.

ESP32 are FSM-uri separate pentru procesarea de SSID/Parola (pentru conectarea cu serverul NTP), procesarea comenzilor speciale pentru SD, rulării comenzilor:

```
enum State {
  MW_LOADING,
  MW_RUNNING,
  MW_RUNNING_SETUP
};

enum CredentialsState {
  CRED_NOT_PROV,
  CRED_PROV,
  CRED_NO_CHECK
};

enum CredentialsProvStatus {
    NONE,
    SSID,
    BOTH
};

enum NTPState {
```

```
  NTP_NOT_CONN,
  NTP_CONN
};

enum SDState {
    SD_PROC_B,
    SD_NO_PROC_B,
    SD_PROC_FN
};

enum SDCommand {
    SD_COMM_NONE,
    SD_COMM_WR,
    SD_COMM_RD,
    SD_COMM_AP,
    SD_COMM_CR,
    SD_COMM_LS,
    SD_COMM_RM
};
```

FSM-ul pentru STM32 se ocupa de inițializare, iar apoi intra într-un loop Procesare Comanda ↔ Afișare Status.

```
pub enum State {
    StateGetSsid,
    StateGetSsidPwd,
    StateInit,
    StateLoadCmd,
    StateDoneLoadCmd
}
```

Comunicarea dintre ESP32 și STM32 a fost cea mai “tricky” parte din software. Fiind doar 2 canale de comunicare seriale (unul prin care ESP32 trebuia sa proceseze o comanda legată de sistemul de fișiere, și unul pentru afișarea pe display a input-ului), și pentru ca a evita procesarea comenzilor la nivel de string de fiecare data (și pentru a mapa mai ușor la numărul de argumente așteptate), am folosit un byte de start pentru a determina tipul unei comenzi.

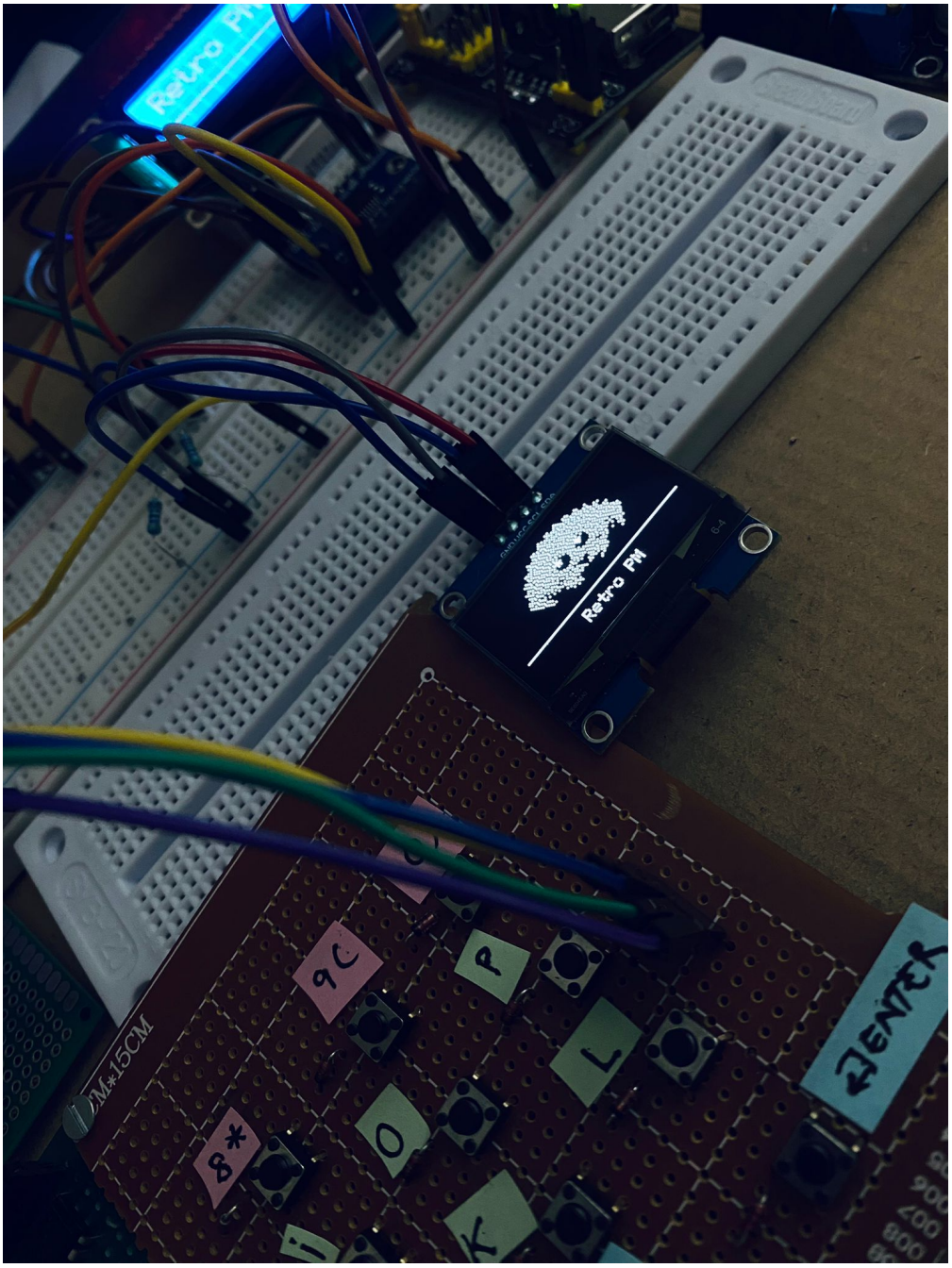
Code snippet ca exemplu:

```
match processed_cmd.cmd_type {
        CommandType::CmdLs => {
            first_byte = '5' as u8;
        }

        CommandType::CmdAp => {
            first_byte = '2' as u8;
        }

        CommandType::CmdRd => {
```

```
            first_byte = '1' as u8;
        }

        CommandType::CmdRm => {
            first_byte = '6' as u8;
        }

        CommandType::CmdWr => {
            first_byte = '3' as u8;
        }

        CommandType::CmdCr => {
            first_byte = '4' as u8;
        }
    }
```

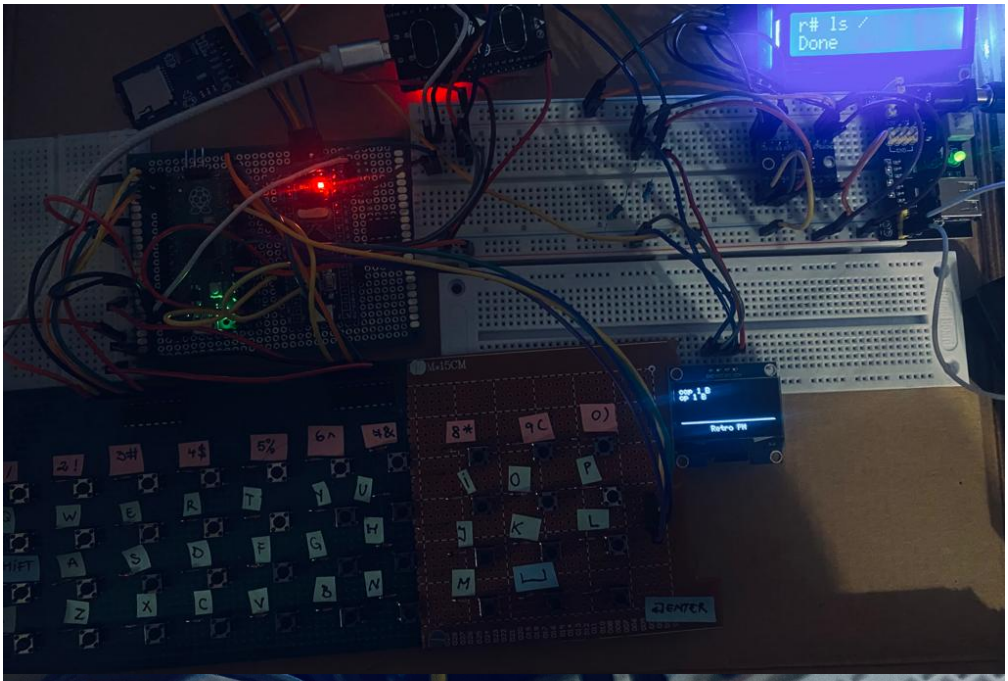
Rezultate Obţinute

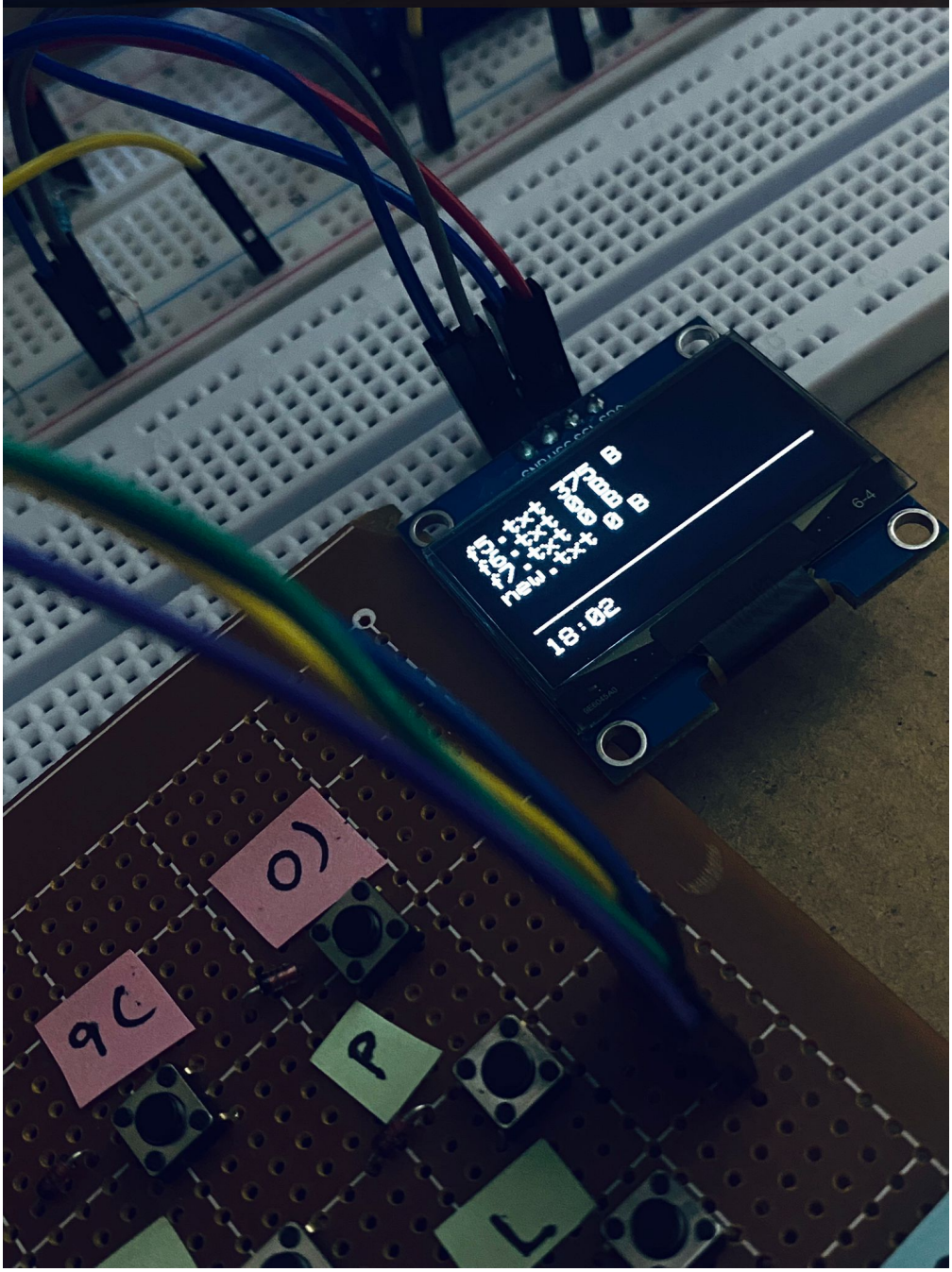
Ferris facandu-si prezenta simitita Ła power-up:



Cum arata circuitul final (utilizatorul logat default este root, se vede ca LCD-ul 16×2 este folosit pentru ca utilizatorul sa vadă ce tastează, iar display-ul OLED este folosit pentru informații despre sistemul de fișiere + afișarea orei).

Comanda ls care implicit afișează numele și dimensiunea fișierelor din /:


r# ls /
Done


18:02

Video: (just in case):

Concluzii

Mi-ar fi plăcut sa pot sa implementez alte funcționalități (de ex. mutarea cursorului printr-o secvența de caractere apăsate etc.), dar am subestimat dificultatea proiectului.

Concluzie finala: o experienta placuta (mai ales cand am făcut câteva scurturi XD).

Download

Download source code & .sch

Bibliografie/Resurse

- The Embedded Rust Book
- Matrix Scanning