portable_mp3_player.zip

Portable MP3 Player

Introduction

The project consists of creating a **portable MP3 Player** capable of playing audio files from a microSD card and displaying current track information on an OLED screen. Music is listened to through headphones connected to the auxiliary output.

- **Goal:** Creating a compact and autonomous device for audio playback with a simple button-based control interface.
- **Core Idea:** Using a dedicated module (DFPlayer) for audio decoding, leaving the microcontroller the task of managing the user interface and control logic.
- **Utility:** For me, it represents an opportunity to learn the integration of communication protocols (I2C, Serial) and power management (Li-Po battery). It is also a way to listen to music using a device I created myself. For others, maybe a cool toy :) that is an example of an embedded system balanced between dedicated hardware resources and software control.

General Description

The project integrates the following hardware modules:

- **Arduino Nano (ATmega328P):** The “brain” of the system that processes button inputs and commands the peripherals.
- **DFPlayer Mini:** The module responsible for reading files from the microSD card and playing the sound.
- **OLED Screen (I2C):** Displays the player status and the menu.
- **Power System:** 3.7V Li-Po battery with a TP4056 charging module and protection.

Block design:

Interaction: The user presses the buttons (Input), the Arduino processes the command (Software Logic), sends instructions via UART to the DFPlayer (Sound Output), and updates the screen via I2C (Visual Output).

Hardware Design

Hypothesis

“We believe that using a dedicated hardware decoding module (DFPlayer) will improve audio system performance because it offloads the ATmega328P microcontroller from the intensive task of processing the MP3 data stream, allowing for a smooth visual interface on the OLED screen.”

Components

- 1x Arduino Nano (Atmega328P)
- 1x DFPlayer Mini Module
- 1x OLED Module (I2C)
- 1x TP4056 Module (Li-Po Charger)
- 1x 3.7V 1200mAh Li-Po Battery
- 1x 3.5mm Audio Jack Module
- 3x Push buttons (Next/Vol+, Play/Pause, Prev/Vol-)
- 1x Toggle Switch (On/Off)
- 1x 1kOhm Resistor (for UART RX line protection)
- 1x Prototype PCB board & Jumping cables

Schematics

The electrical schematic was designed in KiCad, ensuring a modular and portable design.

Hardware Details:

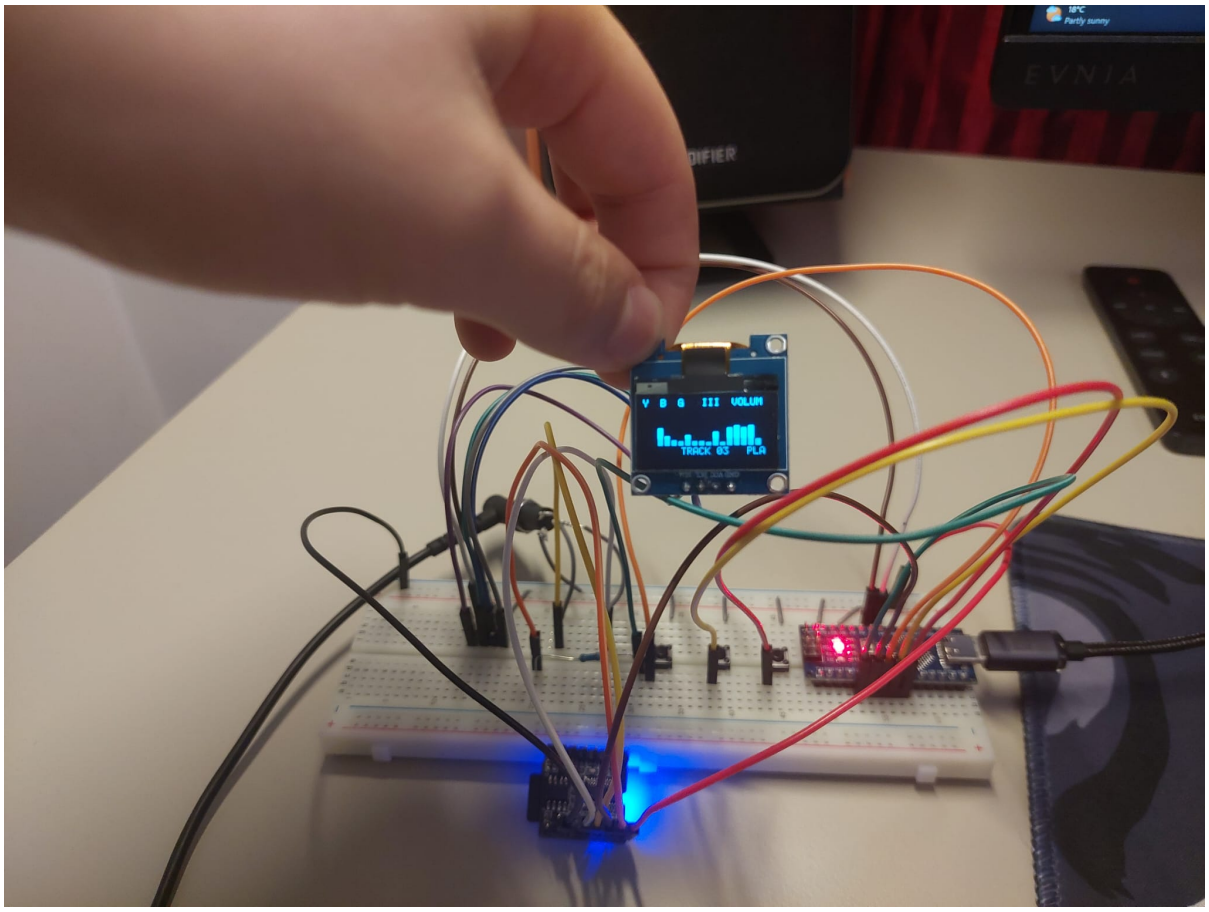
- **Power:** Managed by the TP4056 module. The battery provides ~3.7V, which powers the Arduino via

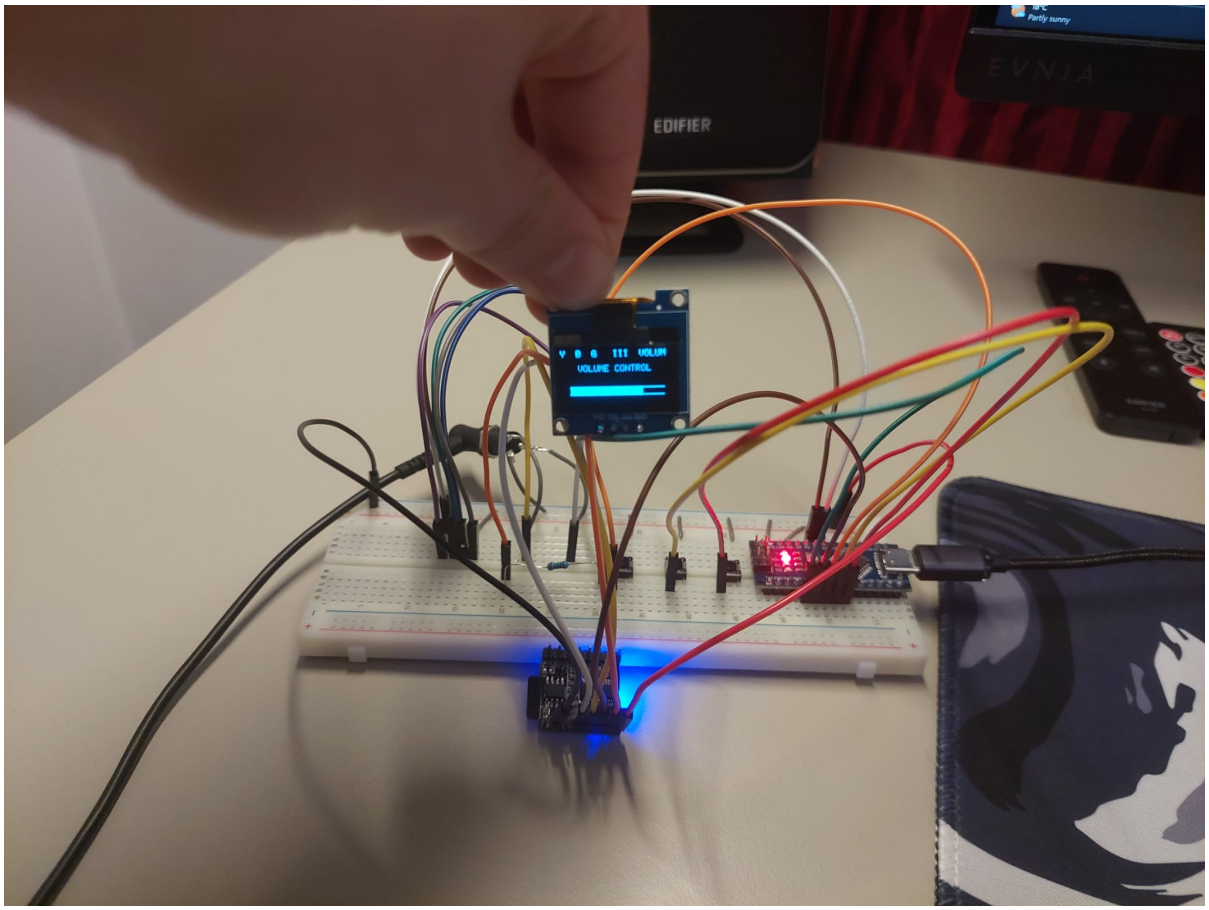
the VIN pin.
- **Audio:** DAC_R and DAC_L pins from DFPlayer are connected to the Audio Jack for stereo output.
- **Protection:** A 1kOhm resistor is placed on the UART line between Nano (TX) and DFPlayer (RX) to prevent noise and protect the 3.3V logic of the player.

Pin Mapping

Component	Arduino Pin	Connection Type
OLED Display	A4 (SDA), A5 (SCL)	I2C (Hardware TWI)
DFPlayer Mini	D8 (TX), D9 (RX)	UART (9600 bps)
Buttons	D2, D3, D4	Digital Input (Interrupts)
Audio Jack	DAC_R, DAC_L, GND	Analog Audio
Power Switch	VIN	Power Control

Exemplu de functionare:





Software Design

Description of the application code (firmware):

- **Development Environment:** AVR / VS Code with PlatformIO.
- **Libraries and 3rd-party sources:**
  - Using the standard `avr/io.h` library for register access.
  - Custom implementation for I2C (OLED) and UART (DFPlayer) protocols.
  - `util/delay.h` for timing.
- **Algorithms and Structures:**
  - **State Machine:** To manage player states (Play, Pause, Stop, Next).
  - **UART Protocol:** Implementation of `UART_init` and `UART_transmit` functions to control the DFPlayer.
  - **I2C Protocol (TWI):** Managing the OLED screen by manipulating the TWI (Two Wire Interface) registers of the ATmega328P.

I have included concepts from the following laboratories:

- **Laboratory 3 (Interrupts):** Used for control buttons (External Interrupts) to ensure minimal

response time.
- **Laboratory 6 (UART):** Asynchronous communication with the audio module (9600 bps).
- **Laboratory 7 (I2C/TWI):** Control of the OLED screen via the I2C protocol, using the hardware TWI module of the microcontroller.

Software Design

Overview

The firmware is written entirely in bare-metal C with no external Arduino libraries. All peripheral control is done through direct manipulation of ATmega328P hardware registers (TWBR, TWCR, TWDR, DDRD, PORTD, PIND). The project is built with PlatformIO using the AVR-GCC toolchain.

Key metrics:

- ~300 lines of non-trivial, original code (`src/main.cpp`)
- RAM usage: 54.8% (1123 / 2048 bytes)
- Flash usage: 11.4% (3502 / 30720 bytes)

Laboratory Concepts Applied

- **Laboratory 0 (GPIO):** Buttons on PD4/PD5/PD6 configured as digital inputs with internal pull-ups via DDRD/PORTD registers. State is polled each loop iteration directly from the PIND register. DFPlayer communication pins (PD2, PD3) configured as output/input via the SOFT_DDR register.
- **Laboratory 1 (UART):** Software UART (bit-banging at 9600 baud, 104µs bit period) implemented from scratch for bidirectional communication with the DFPlayer Mini. TX sends 10-byte command packets; RX reads 10-byte response packets with a timeout. No hardware USART registers are used — the serial protocol is reproduced entirely in software.
- **Laboratory 6 (I2C/TWI):** Hardware TWI module used to drive the SSD1306 OLED display. Direct register access: TWBR=72 (100kHz clock), TWCR for START/STOP/ACK control, TWDR for data bytes. A full 1024-byte framebuffer is flushed to the display every frame via 32-byte I2C transactions.

Pin Mapping (Firmware)

Component	Arduino Pin	Register	Type
OLED Display	A4 (SDA), A5 (SCL)	TWBR,TWCR,TWDR	I2C Hardware TWI
DFPlayer RX	D2 (PD2)	PORTD bit 2	Software UART TX (bit-bang)
DFPlayer TX	D3 (PD3)	PIND bit 3	Software UART RX (bit-bang)
Button PREV/VOL-	D4 (PD4)	PIND bit 4	Digital Input, pull-up

Button PLAY/PAUSE	D5 (PD5)	PIND bit 5	Digital Input, pull-up
Button NEXT/VOL+	D6 (PD6)	PIND bit 6	Digital Input, pull-up
Audio Jack	DAC_R, DAC_L	—	Analog (DFPlayer direct)

State Machine

The main loop (~110ms/iteration) drives a three-state UI machine:

State	ui_mode	Enter trigger	Exit trigger
Track Mode	0	Default / PLAY in vol / timeout	—
Volume Mode	1	Long-press PREV or NEXT (>880ms)	Short PLAY or 5s inactivity
Track Select	2	Long-press PLAY	Short/long PLAY confirms track

In each state the OLED shows different content. Transitions are triggered exclusively by button events or a countdown timer (vol_timer), with no hardware interrupts — the entire logic is polling-based inside the main while(1) loop.

Software UART — DFPlayer Communication

The DFPlayer Mini communicates at 9600 baud using 10-byte packets:

```
0x7E  0xFF  0x06  CMD  0x00  P1  P2  CHK_H  CHK_L  0xEF
```

A bit-bang TX function drives the PD2 line manually, respecting the 104µs bit period:

```
static void suart_tx_byte(uint8_t b) {
    SOFT_PORT &= ~(1 << SOFT_TX_PIN);  // start bit
    _delay_us(104);
    for (uint8_t i = 0; i < 8; i++) {
        if (b & 0x01) SOFT_PORT |=  (1 << SOFT_TX_PIN);
        else          SOFT_PORT &= ~(1 << SOFT_TX_PIN);
        _delay_us(104);
        b >>= 1;
    }
    SOFT_PORT |= (1 << SOFT_TX_PIN);   // stop bit
    _delay_us(104);
}
```

A matching RX function reads PD3 with a timeout, used at startup to query the total number of tracks on the SD card (command 0x48). If the DFPlayer does not respond, the firmware falls back to a compile-time constant (MAX_TRACKS_FALLBACK = 6).

Commands used at runtime:

Command	Hex	Description

Reset	0x0C	Hardware reset on startup
Volume	0x06	Set volume level (0–30)
Play	0x03	Play specific track by number
Pause	0x0E	Pause playback
Resume	0x0D	Resume playback
Query	0x48	Read total file count from SD

I2C / OLED Display System

A 1024-byte framebuffer (`oled_buf[1024]`) mirrors the 128×64 display in RAM (8 pages × 128 columns, 1 bit per pixel). Each frame follows this sequence:

1. `memset(oled_buf, 0, 1024)` — clear the buffer
2. `draw_ui()` — render current UI state into the buffer
3. `oled_flush()` — push the full buffer to the OLED via I2C

Two rendering primitives exist with an important constraint:

- `oled_char(col, page, c)` — writes a 5×8 font glyph using byte assignment (=)
- `draw_pixel(x, y, color)` — sets/clears one pixel using bitwise OR (`|=`)

`oled_char` uses = which overwrites the entire page byte, erasing any pixels set by `draw_pixel` in the same column. The screen layout is designed so that each page is used exclusively by text OR by pixels, never both simultaneously.

The font is a 42-entry 5×8 bitmap array stored in PROGMEM (Flash), covering digits 0–9, letters A–Z, and special glyphs for `:`, `-`, %, ▶ (\\x01), □ (\\x02).

Big Character Rendering

To make the track number and volume level visually prominent, a `draw_big_char()` function scales every 5×8 glyph to 10×16 pixels by replacing each source pixel with a 2×2 block:

```
static void draw_big_char(uint8_t x, uint8_t y_top, char c) {
    // resolve font index for c ...
    for (uint8_t col = 0; col < 5; col++) {
        uint8_t bits = pgm_read_byte(&font5x8[idx][col]);
        for (uint8_t row = 0; row < 8; row++) {
            if (bits & (1 << row)) {
                draw_pixel(x + col*2,     y_top + row*2,     1);
                draw_pixel(x + col*2 + 1, y_top + row*2,     1);
                draw_pixel(x + col*2,     y_top + row*2 + 1, 1);
                draw_pixel(x + col*2 + 1, y_top + row*2 + 1, 1);
            }
        }
```

```
    }
}
```

In track mode this renders 01 (icon + two digits) centered in the top 16px. In volume mode it renders the current level 05 centered in the same area.

Spectrum Bar Animation

15 bars are animated each frame using a smooth target-following algorithm:

- 20% chance per frame: pick a new random target height for that bar (3–28px)
- Attack: bar grows +2px/frame toward its target
- Decay: bar shrinks −1px/frame toward its target
- On pause: all bars and targets are instantly set to 0 → flat baseline line

This produces a natural-looking equalizer effect without any real audio analysis.

Button Logic

Each button uses a hold counter incremented every ~110ms loop iteration:

- **Short press** (released with counter 1–8): change track / adjust volume / play-pause
- **Long press** (counter exceeds 8, ~880ms): open volume menu (PREV/NEXT) or track select menu (PLAY)

A `prev_long` / `next_long` flag is set on long-press activation to prevent the short-press action from also firing when the button is released.

Volume is stored as an integer 0–10. The DFPlayer always receives `vol_level × 3` (mapping 0–10 → 0–30, which is the DFPlayer's native 0–30 range).

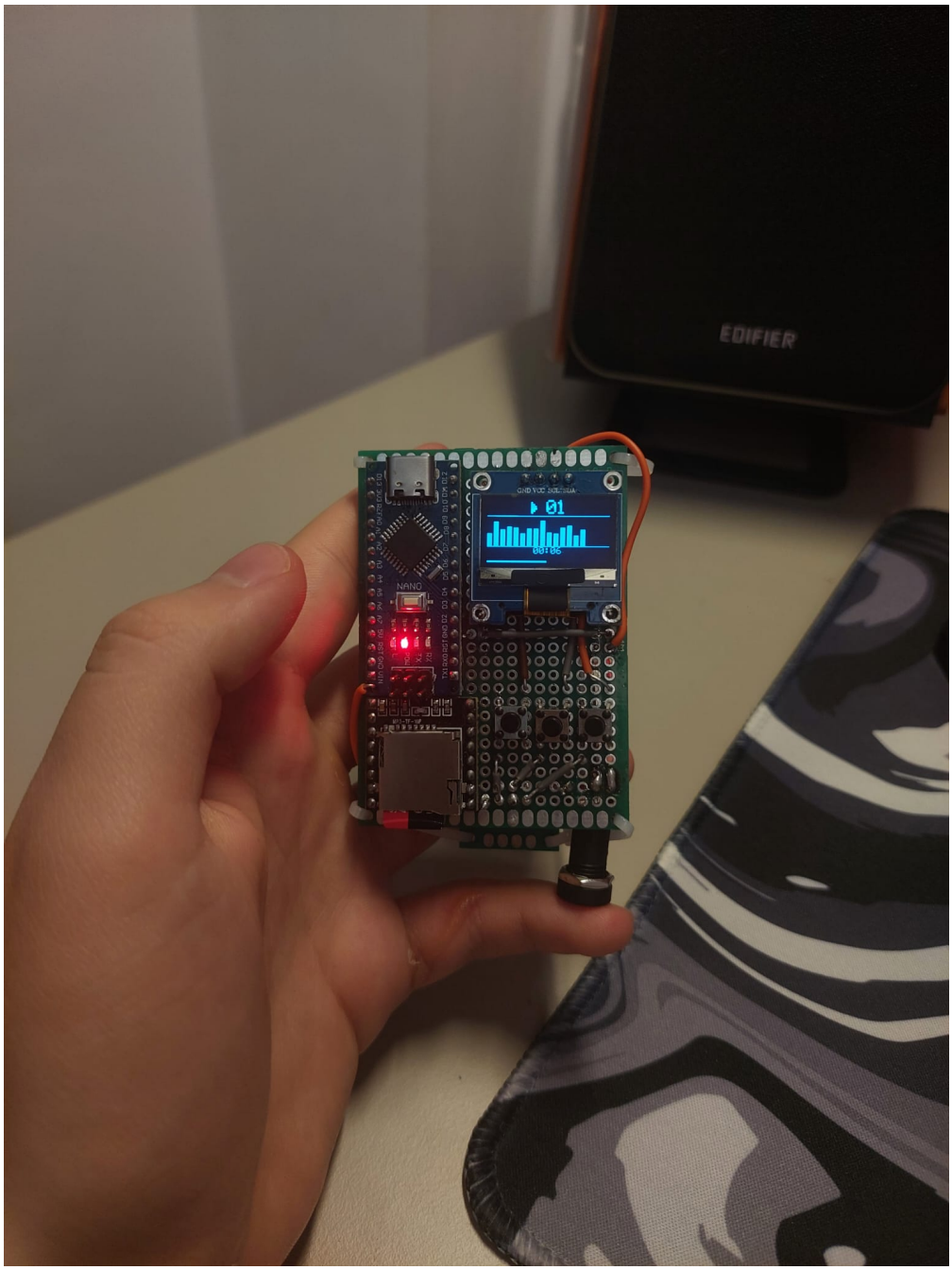
Screen Layout

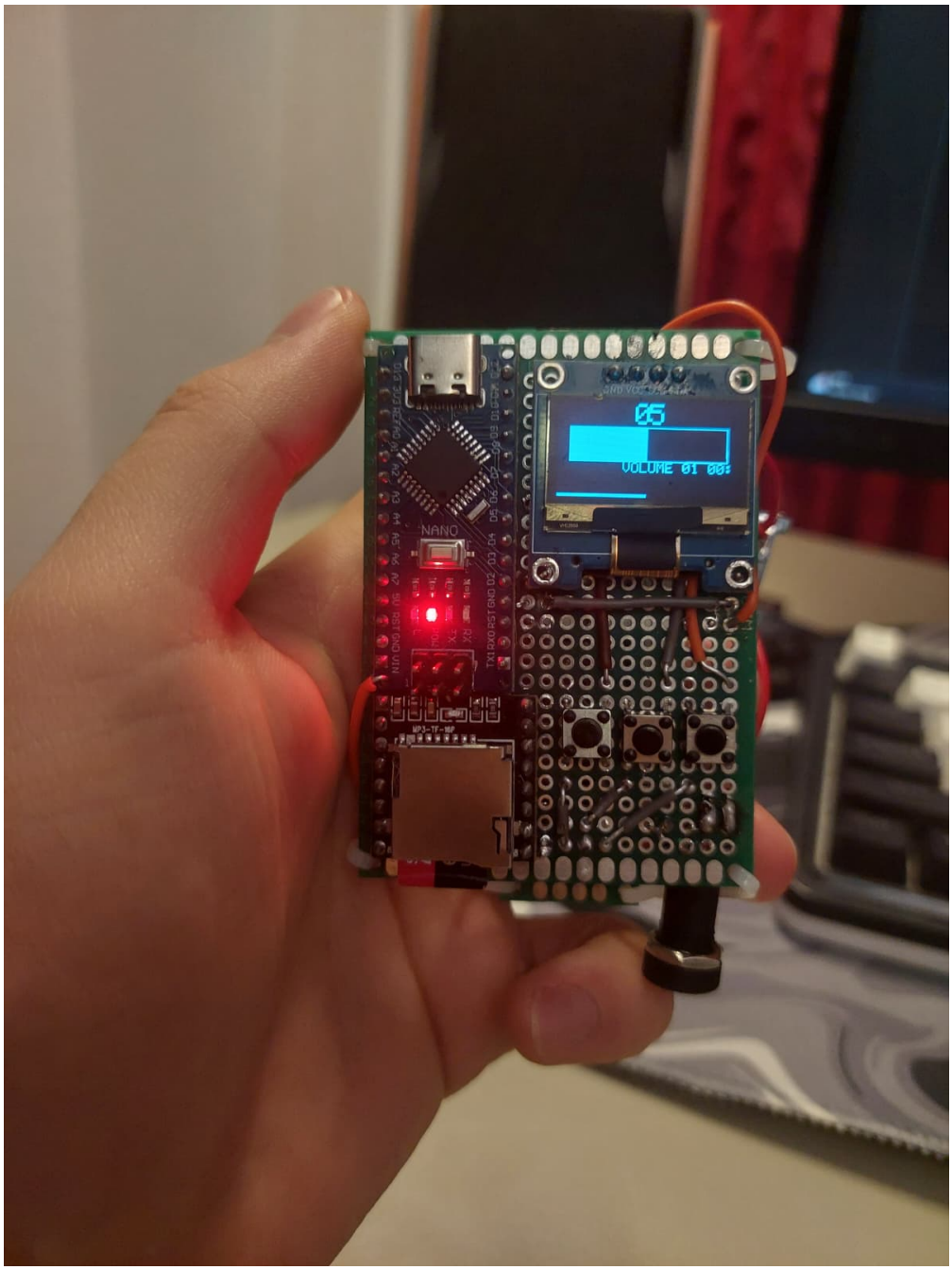
```
Track Mode:
  y=0–15  (pages 0–1):  /▶  01    ← 16px big characters (draw_pixel only)
  y=16:                 ────────    ← separator line
  y=17–47 (pages 2–5):  spectrum    ← 15 bars × 5px wide, 7px spacing
  y=47:                 ────────    ← baseline
  y=48–55 (page 6):     00:23       ← elapsed time (oled_char only)
  y=62–63 (page 7):     ▓▓▓▓▓▓▓     ← 2px volume fill bar (draw_pixel only)

Volume Mode:
  y=0–15  (pages 0–1):  05          ← big vol number (draw_pixel only)
```

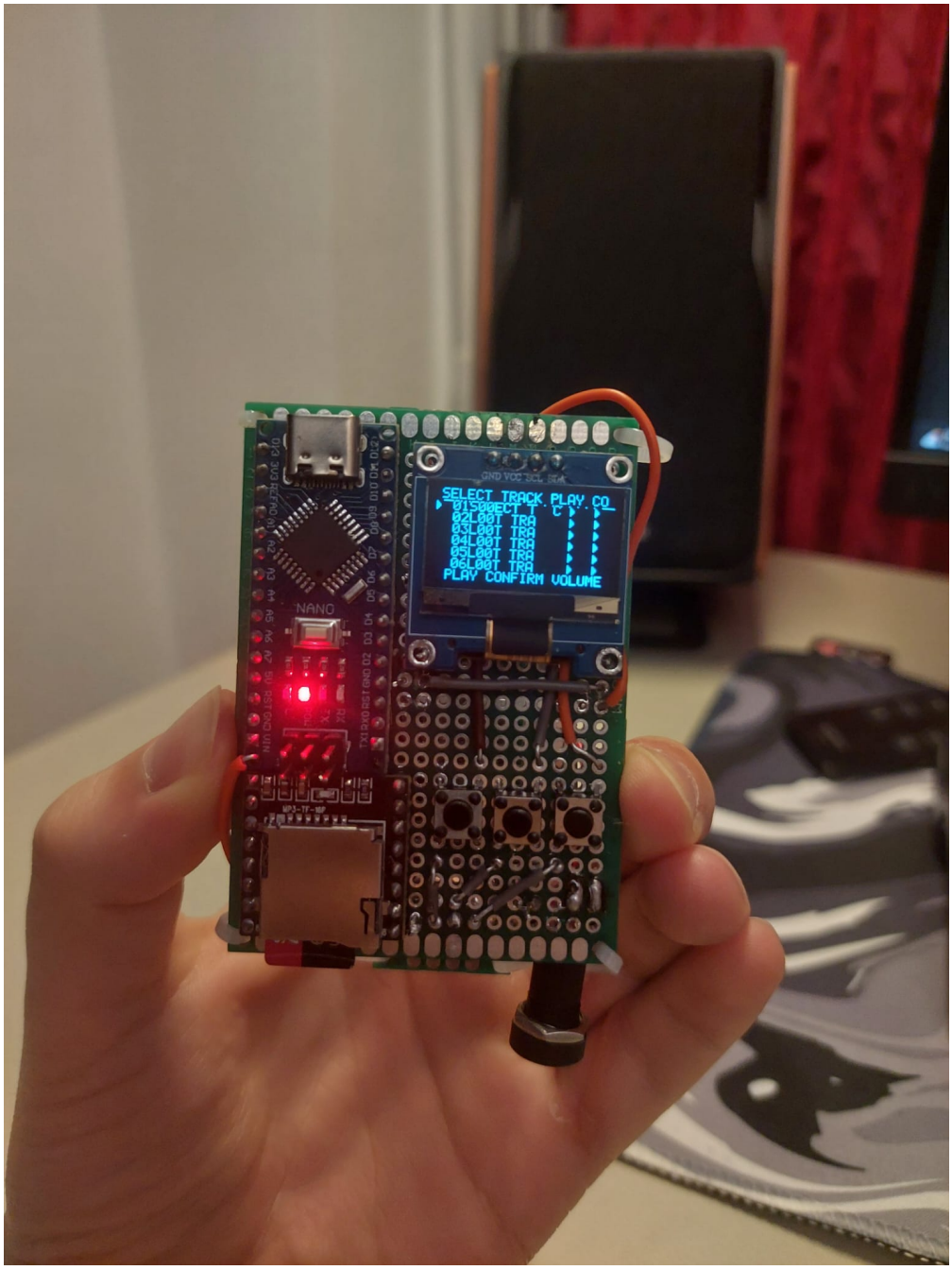
```
  y=16–39 (pages 2–4):  [▓▓▓▓▓▓▓▓▓▓]   ← horizontal fill bar with border
  y=40–47 (page 5):     VOLUME         ← label (oled_char only)
  y=62–63 (page 7):     ▓▓▓▓▓▓▓▓       ← same 2px volume indicator

Track Select:
  page 0:    SELECT TRACK
  pages 1–6: ▶ 01 / 02 / ... / 06  ← cursor + track number list
  page 7:    PLAY=CONFIRM
```

Results

The final product is a functional, standalone portable MP3 player built around the Arduino Nano and DFPlayer Mini, with a custom OLED user interface written entirely without external libraries.




05
NANO



Functional features achieved:

- Playback of MP3 files from a microSD card with Play / Pause / Next / Previous control
- 3-mode user interface (Track Mode, Volume Mode, Track Select) managed by a state machine
- OLED display showing a 16px scaled track number and play/pause icon, a spectrum bar animation, elapsed time, and a volume indicator bar
- Volume control (0–10 steps) via long-press + short-press button interaction
- Track select menu listing all available tracks (auto-detected from SD card at startup)
- Spectrum animation that instantly flattens to a straight line on pause
- Portable power supply: 3.7V Li-Po battery with TP4056 charging and protection circuit
- Stereo audio output via 3.5mm jack

Performance measurements:

- Main loop iteration: ~110ms (10ms active delay + ~100ms I2C framebuffer flush)
- Firmware RAM usage: 54.8% (1123 / 2048 bytes) — stable, no stack overflow observed

- Firmware Flash usage: 11.4% (3502 / 30720 bytes) — significant headroom remaining
- Button response latency: max 110ms (one loop iteration) — imperceptible in practice
- Volume range: 0–30 on DFPlayer (mapped from 0–10 user-facing steps)

Hypothesis validation:
The initial hypothesis was confirmed. Offloading audio decoding to the DFPlayer Mini allowed the ATmega328P to dedicate its full cycle budget to UI rendering and button polling, resulting in a smooth 9 FPS display refresh with no audio glitches.

Conclusions

The project successfully demonstrates a balanced embedded system where dedicated hardware (DFPlayer) handles the computationally expensive task (MP3 decoding) while the microcontroller focuses on user interaction and visual feedback.

What worked well:

- The bare-metal approach (no libraries) gave full control over timing and memory,

which was essential given the 2KB RAM constraint of the ATmega328P.

- The software UART bit-bang implementation proved reliable for DFPlayer communication

and also enabled reading the track count response at startup.

- The framebuffer rendering approach (draw everything into RAM, flush once) eliminated

partial-update artifacts on the OLED.

- The hold-counter button logic (short press vs. long press on the same button) reduced

the required number of physical buttons from 5 to 3 without sacrificing usability.

Challenges encountered:

- The `oled_char` (uses =) vs. `draw_pixel` (uses |=) page conflict required

careful layout planning to avoid text erasing pixel graphics in the same page byte.

- DFPlayer communication is sensitive to timing — the soft UART bit period (104µs)

had to be exact, and any I2C activity during transmission caused corruption.

- Power stability: the Li-Po voltage drop under load initially caused resets;

resolved by adding a decoupling capacitor near the Arduino VIN pin.

Future improvements:

- Replace the simulated spectrum with a real VU meter using the ADC and a signal

tap from the audio line.

- Add a progress bar using DFPlayer command 0x28 (query current playback position).
- Design a custom PCB to reduce size and improve reliability over the prototype board.

Journal

- **May 3**: Created the Wiki documentation page and the Block Diagram. Defined project scope and selected components.
- **May 10**: Hardware milestone: developed the full system schematic in KiCad including power management (TP4056) and audio routing. Verified I2C and UART pin mapping on breadboard. Confirmed stable communication between Arduino, OLED, and DFPlayer.
- **May 15**: Implemented bare-metal I2C driver (TWI registers) and basic OLED framebuffer rendering. First text displayed on screen.
- **May 18**: Implemented software UART TX for DFPlayer control. First successful track playback triggered from firmware. Volume and play/pause commands verified.
- **May 20**: Added software UART RX and `dfp_query_tracks()` to auto-detect SD card track count at startup. Verified fallback behavior when DFPlayer does not respond.
- **May 22**: Implemented spectrum bar animation with smooth attack/decay algorithm. Added instant flatten-to-line behavior on pause.
- **May 24**: Designed and implemented the 3-mode state machine (Track / Volume / Track Select). Rewrote button logic to support short press vs. long press on the same button using hold counters.
- **May 26**: Added `draw_big_char()` function for 2× scaled 16px characters. Redesigned the OLED layout: big icon + track number in top 16px, spectrum in the middle, time and volume bar at the bottom.
- **May 27**: Final firmware build verified: RAM 54.8%, Flash 11.4%, no compilation errors. Assembled final prototype on PCB board. Completed Wiki documentation.

Bibliography/Resources

- ATmega328P Datasheet
- DFPlayer Mini Documentation
- PM Laboratories (UART, I2C, Timers)
- SSD1306 OLED Tutorials
- Project Schematic: proiect_pm_v2.pdf
- Project Code : portable_mp3_player.zip

Export to PDF