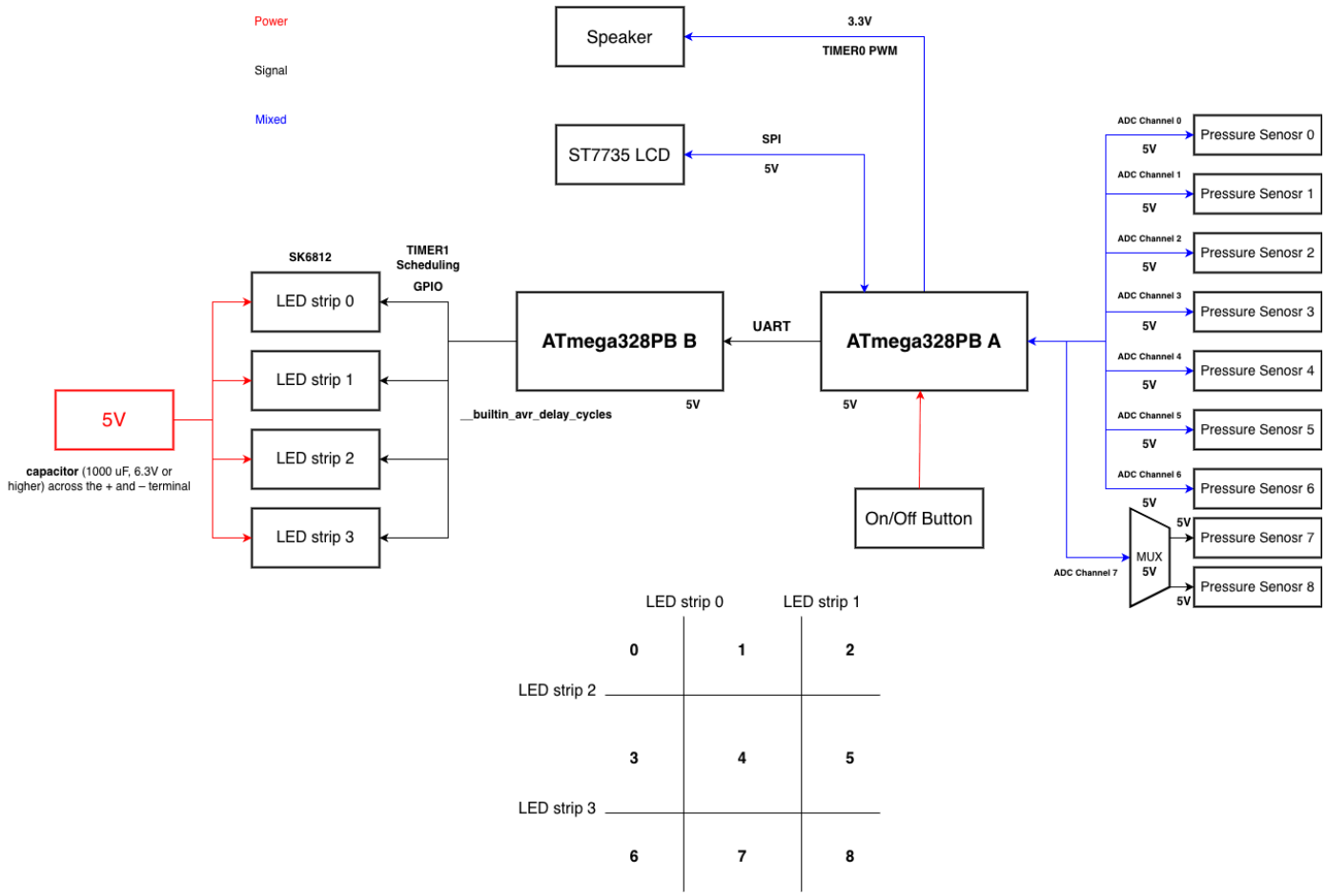
MVP Demo

A computer plays the song. At the same time, the essential notes from that song were already transformed into real frequencies, then mapped to Timer0 compare values (OCR0B), and saved in the ATmega firmware as fixed arrays. These stored notes are organized by pitch into nine frequency sets, matching nine zones on the carpet. Four LED strips visually mark a 3×3 grid, dividing the mat into nine blocks. As the song progresses, the firmware predicts which block corresponds to the next key note and makes the LED border around that block half-bright just before the note happens. If the player steps on the correct block at the expected moment, the MCU instantly outputs the same note's frequency as sound and turns that block's LED border fully bright. If the player does not step correctly, the MCU outputs no sound and the LEDs stay off. This closes the loop: the song runs normally, the firmware forecasts the next key note to preview the block, the footstep decides whether the note is triggered locally in real time, and the LEDs show either a half-bright preview or a full-bright hit result.



Hardware Implementation

1. **Pressure Sensing**

   A **74HC4051 analog multiplexer** selects among multiple pressure sensors.

   Three MCU pins (PC[5:4], PB4) drive the MUX address lines, and the selected signal is read through **ADC7** .

2. **Audio Output**

   A speaker is driven by **ATmega328PB A Timer0 PWM**.

   MIDI notes are converted to frequencies, and the timer's TOP value generates the corresponding tone.

3. **LED Visualization**

   Four **SK6812 LED strips** (60 LEDs each) are controlled using cycle-accurate RGBW timing.

   Each strip uses a dedicated MCU **GPIO** pin (PC0, PC1, PC2, PC3 on **ATmega328PB B**), enabling independent block lighting.

4. **LCD Display**

   A **1.8'' ST7735 TFT** display is connected via SPI.

   It shows text and simple graphics for system feedback (current score).

5. **Power**

   All components operate from a **5 V / 3.3 V supply** .

   LED strips share ground with the MCU and receive a separate 5 V rail due to higher current demand.

Explain your firmware implementation, including application logic and critical drivers you've written.

(a) Real-Time Scheduling (Timer1 Interrupts)

Timer1 on ATmega328PB A operates in CTC mode to generate a 1 ms interrupt. The ISR decreases the remaining duration of a musical note, and asserts sample_flag for periodic ADC sampling.

(b) Musical

A melody (173 MIDI notes) is played using:

- **MIDI → frequency mapping** via an exponential function
- Runtime conversion of **frequency to Timer0 TOP value**
- PWM generation on **OC0B** at 50% duty cycle

Runtime conversion of MIDI note to frequency (not storing precomputed frequencies) because the MIDI array is smaller than a frequency table, thus reducing SRAM usage. Frequencies are computed in real time instead of being hard-coded.

The part supports real-time pitch triggering via pressure input.

(c) Pressure Input

Signal chain : **5 V → 1 MΩ resistor → ADC input → Force-Sensitive Resistor 5 MΩ → ground**.

The pressure sensor is a force-sensitive resistor (5 MΩ when not pressed). Pressure lowers its resistance, decreasing its voltage.

The GPIO + MUX selects one sensor according to **the active note's block index**.

After **channel switching**, firmware inserts a short delay to ensure the MUX output and divider voltage are stable before ADC sampling.

ADC conversion is called by **software**.

The MCU compares the ADC reading to a **threshold** to detect a foot press and checks if the pressed sensor matches the active block.

A correct step adds +10 to score.

(d) SK6812 LED Strip Driver

The LED controller toggles GPIO pins (ATmega328PB B PORTC, PC0, PC1, PC2, PC3) using bit-level register manipulation macros that compile into single AVR instructions , achieving near one-clock-cycle execution time for deterministic GPIO switching.

LED bit timing is produced by **__builtin_avr_delay_cycles**() to match **SK6812 protocol** requirements under a 16 MHz clock.

A RGBW pixel frame is 32 bits per LED, transmitted serially for all LEDs in a selected strip segment.

To prevent hardware resource contention, LED driving is fully isolated on a dedicated UART-controlled sub-ATmega328PB. The hostATmega328PB sends LED update commands via UART, while retaining control of ADC channel selection, speaker PWM, and LCD SPI, ensuring stable timing for the MVP demo.

Block-to-strip mapping is generated by note_to_block() and led_set_block(), enabling LED visualization.

Implemented driver interface:

- `init_strips()` – configure data GPIOs.
- `clear_all_strips()` – clear all strips.

- `led_show_block(uint8_t block)` – solid color render for block `b`.
- `led_show_block_hint(uint8_t block)` – patterned/half-on hint render for block `b`.

The data transmission time (TH+TL=1.25μs±600ns):

T0H	0 code, high level time	0.3μs	±0.15μs
T1H	1 code, high level time	0.6μs	±0.15μs
T0L	0 code, low level time	0.9μs	±0.15μs
T1L	1 code, low level time	0.6μs	±0.15μs
Trst	Reset code，low level time	80μs	

(e) LCD Graphics Library

The ST7735 display uses a custom library supporting:

- Pixel, line, block, and circle drawing
- RGB565 conversion
- String rendering using an ASCII LUT
- Full-screen color fills
- Game UI functions (score updates)

(f) UART

The system is implemented across **two microcontroller boards** built on the **same hardware platform Microchip with the MCU model ATmega328PB** .

- **Board A (Main controller)** runs the **application state machine** that determines lighting modes (block vs hint display), generates command frames.
- **Board B (LED controller)** runs a **minimal loop-driven receiver** for lighting commands and controls LED strips.

A core application loop:

1. Decode incoming sensor or host command.
2. Decide LED mode.
3. Pack mode into a single byte frame.

4. Transmit frame via **UART1 to Board B**.

The lighting logic operates deterministically at **1 Hz block update**, alternating between led_show_block() and led_show_block_hint() based on mode.

DEMO

In the MVP demo, the device performs the following:

1. The system begins streaming the Mario melody.
2. For each note:
    - **Timer1 manages note duration.**
    - **The ADC reads the pressure value of the mapped block.**
    - **If the user steps on the correct block, the buzzer plays the active MIDI frequency.**
    - **LEDs on one or multiple strips illuminate the corresponding segment using RGBW control.**
3. The score increments in real time when correct steps are detected.

Block 4 LED demo:

https://drive.google.com/file/d/1B0y9c536ojgJNo_FRcyvFQ7g3qtpprv1/view?usp=drivesdk

(Alternate link) https://youtube.com/shorts/u9NZSA0S3M0?si=0TgmRJzvBosTRfMz

Block 7 LED demo:
https://drive.google.com/file/d/1sYWYKJYUE4qeJ1iFqWne7lrZtwDFRZ4v/view?usp=drivesdk

(Alternate link) https://youtube.com/shorts/9mWAL0HpEqE?si=zg2wSVhp8CC4X08K

MVP demo: https://drive.google.com/file/d/17nO7qO-LjWAxQxbXMBe6cqJf3eOCvlHs/view?usp=drivesdk

(Alternate link) https://youtu.be/cSHjhtIK4RI?si=b3RrLLuNMhZyoGL0

Software Requirements Specification (SRS)

We have achieved the following SRS elements:

- Real-time MIDI playback
- Real-time pressure sensing per block
- Dynamic LED block visualization
- Score accumulation and event detection

Data collection has been performed through UART logs and ADC sampling traces, showing reliable timing and stable pressure thresholds.

A computer plays the melody through a speaker as the reference audio stream. The essential tones from that song were pre-converted into real frequencies, then mapped to Timer0 PWM compare values (OCR0B), and stored in ATmega flash as fixed arrays. Runtime frequency computation is applied when stepping events occur, enabling local tone triggering without storing a large frequency table, which reduces SRAM usage. The firmware forecasts the upcoming target block before each key note and activates the associated LED border at half brightness ~0.5 ± 0.1 s earlier, a behavior validated using UART logs that record note index, block index, stepping timestamp, and hit/miss result. A correct step inside the valid timing window triggers real-time PWM sound generation at the same pitch as the upcoming song note, and activates full-brightness LED feedback. A missed or mistimed step produces no local sound and no LED activation. Score accumulation is updated deterministically in software after correct stepping, a behavior confirmed by UART log validation data. Gameplay logic closes the real-time loop: **song playback continues normally → block index for upcoming key note is predicted → footstep detection decides note triggering → LEDs render hint or hit result accordingly → score increments on valid hits.**

Hardware Requirements Specification (HRS)

Achieved HRS components:

- Fully functional pressure sensing hardware
- Stable 800 kHz SK6812 signaling on four strips
- Correct PWM audio generation
- Reliable MUX address line switching
- Verified power delivery to LED strips and MCU peripherals

- Mechanical platform (mat) for the step interface
- LCD mounting and front-panel design

Data collection includes logic analyzer and oscilloscope traces of the LED timing protocol and ADC conversion timing measurements.

The embedded system timing tick is generated using Timer1 in CTC mode at 1 ms, enabling periodic pressure sampling and note duration tracking at deterministic timing. Pressure signals are routed using 74HC4051 analog multiplexer address lines driven from MCU GPIO pins, and read through ADC7 after short stabilization delay post channel-switch, a behavior validated by stable ADC values. Four WS2812-class LED border strips are driven using cycle-accurate bit timing routines implemented with `__builtin_avr_delay_cycles` under 16 MHz clock, producing 800 kHz serial signaling within timing tolerance, validated with oscilloscope or logic timing captures. GPIO register toggle macros compile into atomic single AVR instructions, enabling deterministic LED data pin switching without instruction-level jitter. Audio tones are produced using Timer0 PWM on OC0B, capable of reproducing song pitches (200 Hz–5 kHz) with ≤1.0% frequency error, starting or stopping in <10 ms after hit/miss hardware comparison. LED strips operate from an isolated 5 V high-current rail sharing ground with MCU, preventing power noise from disturbing ADC or PWM timing. A dedicated LED control sub-board communicates via UART1 while the main board retains UART0 for `printf` debugging; this hardware partition prevents LED timing from being affected by concurrent interrupts during tone generation or sensor sampling. All major hardware paths required for the MVP are validated: **FSR divider chain → MUX zone selection → ADC7 foot detection → Timer0 PWM tone output → UART1 isolated LED command → dual-rail 5 V/3.3 V power with shared ground.**

The following items will complete the final system:

- sensor enclosure (need six more)
- Fully integrated LCD UI
- Final audio tuning and enhancement

Highest Risks

1. **LED timing jitter when system fully integrated**

   Additional load could jeopardize SK6812 timing accuracy.

2. **Host-Side Software (Java Application)**

   The Java application will:

   1. **Listen to serial data** sent by the MCU, where each message encodes the currently activated pressure block.
   2. **Highlight the corresponding tile** on the screen in real time, allowing the user to observe feedback during gameplay.
   3. Provide a foundation for later extensions, such as score display, timing accuracy visualization, or game-replay analysis.

De-Risking Plans

- Disable interrupts during LED transmission to guarantee timing stability.
- Use a dedicated timing window for LED updates to avoid ISR interference.

We would benefit from guidance on:

- **How to ensure UART0 printf debugging remains correct while UART1 is used for main–sub board communication**
- Best practices for **integrating** multiplexed pressure sensors into larger mechanical structures
- Recommendations for minimizing LED timing jitter in multi-interrupt embedded systems