Micro:bit LED Matrix Light Show Project Report

Project Objective

Our team set out to create an interactive LED matrix light show using the Micro:bit microcontroller. Unlike typical Micro:bit programming approaches that use MicroPython or Blocks, we challenged ourselves to work directly with ARM assembly language to gain precise hardware control and develop a deeper understanding of embedded systems programming. The primary goals were to create dynamic LED animations, implement responsive button controls, and demonstrate efficient low-level programming techniques.

Implementation Details

Hardware Configuration

We worked with the standard Micro:bit board containing:

- 5×5 LED matrix display
- Two programmable buttons (A and B)
- Nordic Semiconductor nRF51822 microcontroller

LED Matrix Working Mechanism

The LED matrix operates through a multiplexing technique rather than lighting all LEDs simultaneously. The system rapidly cycles through columns and rows at approximately 100Hz refresh rate, creating the visual illusion of a continuously lit display. We configured the following pin connections:

Column pins (current source):

- Column 1: Pin P0.28
- Column 2: Pin P0.11
- Column 3: Pin P0.31
- Column 4: Port 1, Pin 5
- Column 5: Pin P0.30

Row pins (current sink):

- Row 1: Pin P0.21

- Row 2: Pin P0.22
- Row 3: Pin P0.15
- Row 4: Pin P0.24
- Row 5: Pin P0.19

To illuminate a specific LED, we set its corresponding column pin HIGH (3.3V) and row pin LOW (0V), creating the necessary voltage difference for current flow. By quickly cycling through different combinations, we create various patterns and animations.

First Implementation: Triangle and Closing Animations

Our initial version featured two distinct animation patterns:

1. **Triangle Animation:** A dynamic triangle shape that appears to rotate and transform
2. **Closing Animation:** A pattern that converges from the edges toward the center

Button Implementation (Version 1)

We implemented button control with careful consideration of hardware characteristics. Button A triggers a switch between animation modes. Here's how the button detection works:

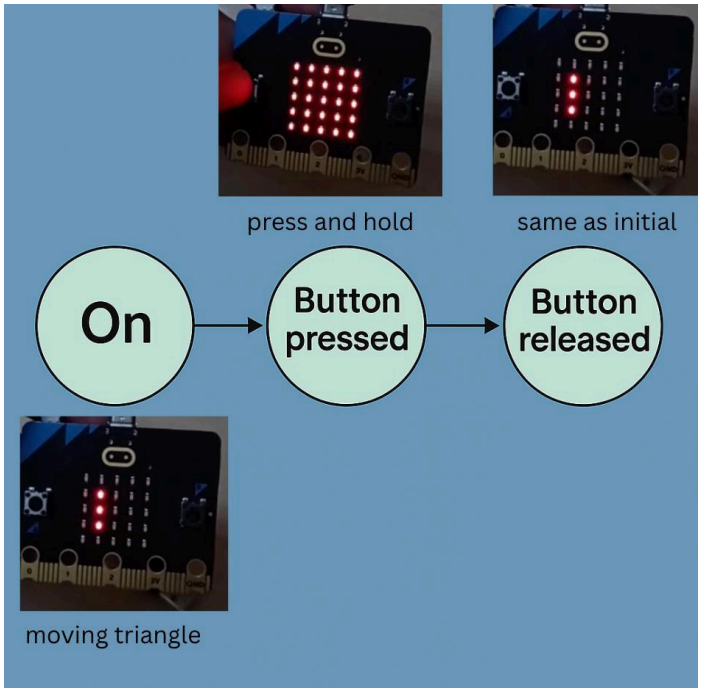
```
check_button:
    LDR r1, =BUTTON_A_ADDRESS  ; Load button A's GPIO address
    LDR r0, [r1]               ; Read current button state
    TST r0, #BUTTON_BIT_MASK   ; Test if button bit is set (pressed)
    BEQ button_pressed         ; If button is pressed (bit is 0),
jump to button handler
    B main_loop                ; Otherwise, continue main loop
```

To address button bouncing (rapid on/off switching during physical press), we implemented a simple debounce mechanism:

```
button_pressed:
    ; Short delay for debouncing
    MOV r0, #10000
    BL delay_loop

    ; Check if button is still pressed
    LDR r0, [r1]
    TST r0, #BUTTON_BIT_MASK
    BNE main_loop  ; Button released, go back

    ; Switch to second animation pattern
    B second_animation
```



Second Implementation: Enhanced Pattern Selector

Building on lessons from our first version, we developed a more sophisticated implementation featuring:

- A three-state pattern system
- Improved button interaction for pattern cycling
- Enhanced display patterns

Button Cycling Mechanism

To create natural button interaction, we implemented state tracking that registers a "click" only on the release of a button rather than the initial press:

```
check_button_B:
    LDR r1, =BUTTON_B_ADDRESS     ; Load button B address
    LDR r2, [r1]                  ; Read current state
    TST r2, #BUTTON_B_MASK        ; Test if pressed

    ; If button is pressed (bit is 0)
    BNE button_not_pressed

    ; Button is currently pressed, update flag
    MOV r3, #1                    ; r3 = button is pressed
    B continue_check
```

```
button_not_pressed:
    ; Button not pressed, check if it was pressed before
    CMP r3, #1                    ; Was button previously pressed?
    BNE continue_check            ; If not, continue

    ; Button was released! Advance pattern
    MOV r3, #0                    ; Clear button pressed flag
    ADD r4, r4, #1                ; Increment pattern counter
    CMP r4, #3                    ; Check if we've gone past pattern
3
    BNE continue_check
    MOV r4, #0                    ; Reset to pattern 0
```

Three Pattern States

Our implementation features three distinct display patterns:

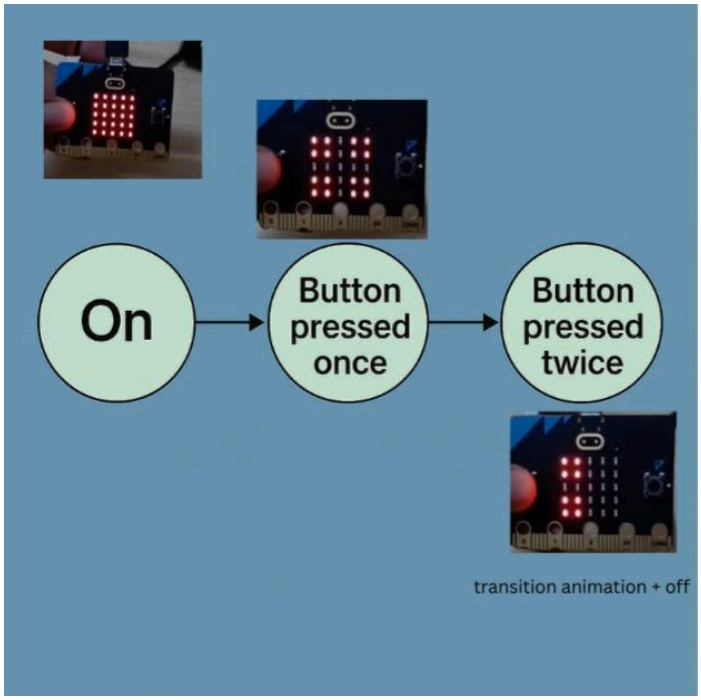
1. **Full Matrix:** All 25 LEDs illuminated
2. **Four Corners:** Four 2×2 squares positioned in each corner
3. **Two Left Squares:** Only the top-left and bottom-left 2×2 squares illuminated

Pattern rendering example:

```
display_full_matrix:
    MOV r5, #0x1F       ; Binary 11111 (all 5 rows)
    BL display_column_1
    MOV r5, #0x1F       ; All 5 rows again
    BL display_column_2
    ; Repeat for all 5 columns
```

Corner squares pattern:

```
display_corners:
    MOV r5, #0x11       ; Binary 10001 (rows 1 and 5)
    BL display_column_1
    MOV r5, #0x11       ; Same pattern
    BL display_column_5
    ; Set up other corners similarly
```



Challenges Faced and Solutions

1. Timing and Animation Smoothness

Challenge: Achieving smooth animations required precise timing control. Too rapid refreshing made patterns imperceptible, while too slow refreshing created visible flickering.

Solution: We developed customized delay functions calibrated through experimental testing:

```
short_delay:
    MOV r0, #5000   ; About 5ms
    B delay_loop

medium_delay:
    MOV r0, #20000  ; About 20ms
    B delay_loop

delay_loop:
    SUBS r0, r0, #1
    BNE delay_loop
    BX lr
```

Through iterative testing, we determined optimal delay values that produced visually appealing animations without perceptible flickering.

2. GPIO Configuration Complexity

Challenge: Understanding and correctly configuring GPIO pins for both the LED matrix and buttons required detailed hardware knowledge.

Solution: We consulted the nRF51822 datasheet extensively and created specialized GPIO configuration routines. For buttons, we enabled internal pull-up resistors to simplify the circuit and improve reliability:

```
configure_buttons:
    ; Set button pins as inputs with pull-ups
    LDR r0, =GPIO_CONFIG_REG
    LDR r1, =BUTTON_A_CONFIG
    STR r1, [r0, #BUTTON_A_OFFSET]
    LDR r1, =BUTTON_B_CONFIG
    STR r1, [r0, #BUTTON_B_OFFSET]
    BX lr
```

3. Debugging Without High-Level Tools

Challenge: Assembly language programming offers limited debugging capabilities compared to high-level languages.

Solution: We implemented visual debugging techniques using the LED matrix itself as a diagnostic display. Specific error patterns were programmed to indicate different program states or issues. Additionally, we established a rigorous step-by-step testing protocol for each function before integration.

4. Button Debouncing

Challenge: Physical button presses produce electrical noise that can register as multiple presses.

Solution: Our initial implementation used simple delay-based debouncing. For the second implementation, we developed a more sophisticated state tracking system that monitors both press and release events to provide a more reliable user experience.

Results

Our project successfully demonstrated:

1. **Precise Hardware Control:** Direct manipulation of GPIO pins for both input and output
2. **Efficient Multiplexing:** Proper LED matrix control through row-column scanning techniques
3. **Responsive User Interface:** Button interactions that feel natural and responsive
4. **Visual Effects:** Multiple animation patterns with smooth transitions
5. **Low-Level Optimization:** Efficient register usage and minimal instruction sequences

The final implementations showcase the power of assembly language programming for embedded systems, providing both educational value and practical demonstration of low-level hardware control.

Future Improvements

Short-Term Enhancements

1. **Accelerometer Integration:** Incorporate the Micro:bit's accelerometer to create motion-sensitive light patterns
2. **Sound Synchronization:** Add auditory feedback through the edge connector
3. **Brightness Control:** Implement PWM techniques to vary LED brightness levels
4. **Pattern Memory:** Store user-selected patterns in non-volatile memory

Long-Term Development Paths

1. **Advanced Animation Framework:** Create a comprehensive system for defining and transitioning between complex patterns
2. **Power Optimization:** Implement sleep modes and power-efficient scanning techniques to extend battery life
3. **Wireless Control:** Add Bluetooth communication to allow pattern selection from a smartphone application
4. **Interactive Games:** Develop simple games that use the LED matrix as both display and interaction medium

Conclusion

This project demonstrates effective low-level programming techniques on the Micro:bit platform. By working directly with ARM assembly language, our team gained valuable insights into embedded systems programming, hardware interfacing, and efficient resource utilization. The resulting LED matrix light show demonstrates both the technical capabilities of the Micro:bit and the power of assembly language for direct hardware control.

The developed system serves as an educational tool for understanding embedded programming concepts while also providing an engaging visual display. Our implementation approach

emphasizes the importance of hardware understanding, efficient programming practices, and interactive user experience design.

Team Members

1. P Jaya Raghunandhan Reddy - BT2024029
2. Pasham Godha - BT2024082
3. Varanasi Harshith Raj - BT2024177
4. Hasini Samudrala - BT2024113
5. Polareddy Harshavardhan Reddy - BT202406
6. Gangavarapu Jaswant - BT2024010