Lab 3 - LED Matrix

Goals

- Use timers and GPIO to control the LED matrix
- Display text on the LED matrix

Equipment

- Computer with build environment
- Micro:bit and USB cable

Documentation

- nRF52833 datasheet: https://infocenter.nordicsemi.com/pdf/nRF52833_PS_v1.3.pdf
- Microbit schematic: https://github.com/microbit-foundation/microbit-v2-hardware/blob/main/V2/MicroBit_V2.0.0_S_schematic.PDF
- Lecture slides are posted to the Canvas homepage

Lab Steps

1. Update your local repository

- `cd` into the base of your repo
- `git pull https://github.com/nu-ce346/nu-microbit-base.git`
- `git submodule update --init --recursive`
    - There may be no changes

- `cd software/apps/led_matrix/`
    - This lab will use the files in this directory. Your changes will be in `main.c`, `led_matrix.c`, and `led_matrix.h`

2. Control a single row of LEDs

- Initialize row and column pins and set default values for them in `led_matrix_init()`
    - To do so, you can use the nRF GPIO library
        - An example of using this library can be found in `apps/virtual_timers/main.c` and in `apps/app_timer_example/main.c`
        - You do not have to use this library, and may use your own instead if you prefer
    - Each LED row and column pin will need to be set as an output (10 total) using the nrf_gpio_pin_dir_set() function
        - You can refer to row pins as LED_ROW1, LED_ROW2, etc.
        - You can refer to column pins as LED_COL1, LED_COL2, etc.
        - These values are defined in `software/boards/microbit_v2/microbit_v2.h`
    - Rows and columns should default as cleared
        - You can control LEDs with `nrf_gpio_pin_clear()`, `nrf_gpio_pin_set()`, `nrf_gpio_pin_toggle()`, and `nrf_gpio_pin_write()`
    - If you want, you could create an array of row LEDs and an array of col LEDs and use that to get the pin numbers. Something like `uint32_t row_leds[] = {LED_ROW1, LED_ROW2, LED_ROW3, LED_ROW4, LED_ROW5};`
- Set initial values for the LEDs in `led_matrix_init()`
    - To activate an LED, ensure that its corresponding row pin is high and its corresponding column pin is low
    - To inactivate an LED within an active row, set its corresponding column pin to high
- Test your code to make sure that you understand how to choose which LEDs are active in a row
    - `led_matrix_init()` is already called in `main()` for you
- **No checkoff**: continue to the next step

3. Use an App Timer to modify LEDs in a single row

- Review the example App Timer code in `apps/app_timer_example/`
    - This example uses the nRF App Timer Library
        - You could use your own Virtual Timer library, but you're probably better off using the nRF App Timer library so you can practice using nRF libraries. Also so you don't run into any bugs.
    - `APP_TIMER_DEF()` is a macro that creates a global timer variable
        - This is how the library handles making multiple timers. Instead of using `malloc()`, it expects users to create each timer as a separate global variable
    - `app_timer_init()` initializes the timer library
    - `app_timer_create()` initializes a specific timer
        - Arguments are: a pointer to the timer variable, the timer mode (either APP_TIMER_MODE_SINGLE_SHOT or APP_TIMER_MODE_REPEATED), and a callback function
    - `app_timer_start()` starts a timer
        - Arguments are: the timer variable (not a pointer!), the number of ticks until expiration, and a `void*` pointer to pass to the callback function (usually just `NULL`)
        - The App Timer uses the RTC with a Prescaler of 0, so Period in seconds equals Ticks divided by 32768
- Initialize an App Timer and start it in `led_matrix_init()`
    - For now, you can just set it to fire once per second in APP_TIMER_MODE_REPEATED mode
- In the App Timer callback function, modify the active LEDs for the given row
    - Make sure the row pin is high, and set some column pins to high and some column pins to low so that some LEDs are active
- Test your code to make sure that you can actually modify the active LEDs. Change which row is active and which columns are active a few times to test your understanding
    - `led_matrix_init()` is already called in `main()` for you
- **No checkoff**: continue to the next step

4. Control arbitrary LEDs

- Keep track of the desired state for each LED in the matrix

  You can implement this any way you want to. However, here are some suggestions:

  - You probably want to keep your state as a global variable so multiple functions can interact with it

  - You can use any method you want to keep track of LED states
    - You could use a 2D matrix of boolean values, one for each LED. That would look something like `bool led_states[5][5] = {false};`
    - Or alternatively you could use a 1D matrix of row values, where each row corresponds to an 8-bit value (5 bits for LEDs and 3 bits of padding)
    - Or alternatively you could use a single `uint32_t` to contain the state of the LED matrix (25 bits for LEDs and 7 bits of padding)

- Each time the App Timer callback function triggers, modify a different row of the LED matrix.

  Only one row of the LED matrix can be active at a time, but by lighting up one row at a time and very quickly iterating through the rows, human perception will make it look as though all LEDs are active simultaneously.

  - You might need a global variable (or static function variable) to keep track of the current row being displayed

  - Inactivate the current row, then change the column pin states, then enable the next row
    - If you do this in a different order, it will briefly light the wrong LEDs

  - First get writing each row working at low speed, and then when you can see rows are changing correctly, increase the frequency in the next sub-step

  - `nrf_gpio_pin_write()` may be a useful function here

- Modify the App Timer to run fast enough to refresh the LED matrix imperceptibly

  - Each LED row needs to refresh at 100 Hz or higher in order to not flicker when viewed. Because only one row is active at a time and there are five rows, this means the App Timer should fire at least 500 times per second.

    - Don’t go below ~20 ticks for the interrupt period, or else your callback handler might take long enough that you will miss interrupts.

- Test your code to make sure you can write an arbitrary pattern to the LED matrix
    - An X pattern is a good test to ensure that everything is working
        - It would be impossible to draw without going row-by-row as we are doing
        - You can make this test part of `led_matrix_init()` so it runs immediately and automatically
    - Only the LEDs you selected should light up
        - If other LEDs are very very dimly lit up, you probably enabled the next row before modifying the columns
    - The refresh rate should be fast enough that you cannot detect the LEDs changing state
        - You might need to increase the configuration of the App Timer
- **No checkoff**: continue to the next step

5. Display a single character

- Create a function that takes in a character and displays it on the LED matrix

  This will need to access the font array in `font.c` in order to determine which LEDs to light for a given character. It contains LED active/inactive states for the first 128 ASCII characters including uppercase and lowercase letter, numbers, and various punctuation. The first 32 ASCII characters that are not representable are left blank. Here is a visualization of the font.

  The font array is a 2D array, indexed first by character (for any character from 0 to 127). The second dimension has 5 values, one corresponding to each row (1-5). The value at a combination of character index and row index is an 8-bit value, corresponding to the 5 LED states, padded with 3 zeros (in the most-significant bits).

  For example:

    `font[82][0]` corresponds to the first row for the letter `'R'` and has the value of 0x0F, which means the LEDs in column 1, 2, 3, and 4 should be active (and column 5 should be inactive).

    `font[82][1]` corresponds to the second row for the letter `'R'` and has the value of 0x11, which means the LEDs in column 1 and 5 should be active.

    Generally, 0x1F means all LED columns are active while 0x00 means all LED columns are inactive

  - The font array is already included in `led_matrix.c` via `font.h`

  - You will need to decode the 5 rows of the character and use them to modify your LED state (which will then lead to the display changing when the timer next fires)
    - Beware: the font row and bits are zero-indexed, while the LED matrix rows and columns are one-indexed

- Test your code to actually display the character

  - You will need to add the function to `led_matrix.h` in order to call it from `main()`

  - If everything from before is working, that should be sufficient to draw the character on the display
    - Be sure to test with characters that are not horizontally symmetrical and also with characters that are not vertically symmetrical

- **No checkoff**: continue to the next step

6. Display arbitrary strings

- Create a function that takes in a string and displays it on the LED matrix

  I'm going to leave the implementation here up to you. As long as the string is displayed in a readable fashion, you'll get credit for this lab.

  Some things to consider:

  - You should support the user modifying the text that is being displayed, likely by calling this function a second time
    - You could provide a callback function when the text is finished
    - You could turn this function into a blocking function that doesn't return until the text is complete
    - You could just return automatically after being configured

  - You likely need to keep an additional global variable or two (or possibly a struct) to contain information about the string currently being displayed and your offset within it

  - You almost certainly want a second timer to control when to move to the next character in the string

  - You can decide if you want to make the speed at which text is displayed publicly configurable
    - You could include the speed at which characters move as part of the public interface for the function if you want to
    - Or alternatively you could include the total duration in which to display the entire string
    - Or alternatively you could choose one default speed for moving between letters and not give the user any control

  - You can decide if you want to make the string repeat once it is complete
    - You could stop displaying anything once the string is completed
    - You could repeat the string automatically whenever it completes
    - You could include a boolean flag in the public interface to choose whether it repeats

  - You can decide how to move text on the display
    - Text could move like a marquee, which should be the most readable method, but also means handling part of two letters on the display simultaneously
    - Text could simply be written letter-by-letter, with only one displayed at a time

        - You could add even more complicated transforms, like fade-in or fade-out

- Make the display say “Hi CE346!”, then several seconds later write “It works!”
    - You will need to add the function to `led_matrix.h` in order to call it from `main()`
- **Checkoff**: demonstrate the working application to course staff
    - Also show them your code and walk through the interesting parts

Lab Checkoffs

You must be checked off by course staff to receive credit for this lab. This can be the instructor, TA, or PM during a Friday lab session or during office hours.

- Make the display say “Hi CE346!”, then several seconds later write “It works!”
- Show your code and walk through the interesting parts

Also, don’t forget to answer the lab questions assignment on Canvas.