1. Introduction

The purpose of this lab is to explore various aspects of synchronous serial communication with a suitable peripheral, in this case, an LED matrix display.

2. Prerequisites

- StmCubeIDE is installed on your machine.
- STM32 Microcontroller
- SPI LED Light Array

3. Activities

- Connect your matrix display to your CE Dev board. Per the prelab assignment, I suggest SPI2 using pins on port B. You will need to enable alternate functions only for MOSI and SCK. You can use the designated pin for NSS if you wish, but since the NSS function relies on a pull-up resistor to pull high, the LEDs on the CE Dev Board will interfere with that functionality. Therefore, you must use regular GPIO to operate the CS signal on the matrix display.

- Implement an API to control the LED matrix. There are a number of ways you could interact with the display and the consensus was to implement a framebuffer approach with setpixel/clearpixel functions. You will need to have a block of memory to store each LEDs state. This state will be written to the display only when requested (calling matrix_update()) which means you can turn on or off multiple pixels in between calls to update. Note that the state of all of the LEDs can be contained in a single uint64_t variable. However, it might be easier to store in an array of eight uint8_t variables. Your choice. In any case, this buffer will need to be visible to setpixel, clearpixel and update functions. Be sure to make file scope!

The following public functions are suggested:

- MatrixInit(): sets up GPIO pins, alternate functions, enables clock, etc. Some configuration can be done on SPI2 in CR1 and/or CR2. Similar to the LCD, after setting up pins and peripherals, you may need to write to the matrix itself to take it out of standby mode, set the scan limit, etc.

- MatrixSetPixel(x, y): sets the selected LED to be on by setting the appropriate bit in the framebuffer. Will not take effect until matrix_update() is called.

- MatrixClearPixel(x,y): sets the selected LED to be off by clearing the appropriate bit in the framebuffer. Will not take effect until matrix_update() is called.

- MatrixClear(): sets all LEDs to be off. Will not take effect until matrix_update() is called.

- MatrixUpdate(): updates the LED matrix by writing the frame buffer out to display.

You will likely benefit from private helper function(s) as well, such as a generic "WriteSpi" function that will write a single frame to the SPI port. Your MatrixUpdate() function would then call this helper function 8 times to update the display.

Create a suitable demonstration of your API. You may choose to integrate into an existing application or a standalone demonstration. Your demonstration should be some sort of animation or active display. Some ideas:

- Bounce a "ball" around the display.
- Create a "starfield" display where LEDs turn on in the center and travel to the outside of the display.
- Create a graphic equalizer effect to your play a tune lab.
- Draw a moving spiral.
- Your idea… Be creative.

4. Hints

- The SPE bit in CR1 must be set at the start of a transaction and cleared at the end. If NSS is being used, it will become active when SPE is set and inactive when SPE is cleared at the end of the transaction. Since we cannot really use the native NSS function, you can achieve a similar behavior by simply using GPIO and bringing the GPIO pin wired to CS on the LED matrix low immediately prior to setting SPE, etc. To enable this mode the SSM, SSI, and MSTR bits in CR1 should all be set to '1'.

- There are a couple of SPI configurations that will work for us in terms of full duplex / bidirectional etc. The default configuration is for full duplex, input on MISO. Even though we are not connecting to MISO, this will still work fine.

- Be sure to follow the "recipe" for communication in the reference manual. Be sure to wait for the TXE flag to be set and then the BUSY flag to be cleared before ending communication.

- Analog Discovery? Of course! Use it to observe SS, SCK, and MOSI. You can use the scope feature, but the logic analyzer will actually decode and show the raw data you are sending.

- Want to use interrupts? Quite optional here but it would allow a frame to be updated "automatically" either on a timed basis or triggered by a call to MatrixUpdate().

5. Deliverables

- All code should include a header block with:
    - Your name
    - Course number and section
    - Assignment name
    - File name
    - List of any dependencies.
    - Description
- Print out you document code in the following order:
    - spi(.c/.h)
    - matrix(.c/.h)
    - other relevant API
    - main.c
- Print in Light mode with Line numbers, filenames, and time and date using CubeIDE or Notepad++
- Staple together packet in the top left, in order, with rubric cover sheet.
- Final packet and demonstration due by the beginning of Week13 Lab.