SANTA CLARA UNIVERSITY
Electrical and Computer Engineering Department

Real-Time Embedded Systems - ECEN 121
Lab 3

DYLAN THORNBURG & KAI HOSHIDE

Using the LED Strip

Andrew Wolfe

Prelab:

1) You will need to turn a GPIO pin on and off. From your last lab, provide the code that was used to turn a GPIO pin on and the code that was used to turn a GPIO pin off.

```
HAL_GPIO_TogglePin(LD_R_GPIO_Port, LD_R_Pin); // on
HAL_Delay(500);
HAL_GPIO_TogglePin(LD_R_GPIO_Port, LD_R_Pin); // off
HAL_Delay(500);
```

Note: The delays are not needed but it's realistic to have them as you won't be able to observe the lights turning off and on without them. They were also a part of my lab code.

2) Review the data sheet for the SK9822. Provide the 32-bit pattern for light purple and the 32-bit pattern for yellow. Provide the numbers in hexadecimal form. Explain how you calculated them.
Use e4 to get 111 and moderate luminance, then in pattern bgr, 00ffff is yellow and light purple would be E3C3CB (used google for purple).
Light purple:0xe4e3c3cb
Yellow:0xe400ffff

3) Identify where in the manuals the HAL_Delay() function API is documented.
It is found in "Description of STM32L4/L4+ HAL and low-layer drivers - User manual" on page 33/2719.

It states:
"HAL_Delay(). this function implements a delay (expressed in milliseconds) using the SysTick timer. Care must be taken when using HAL_Delay() since this function provides an accurate delay (expressed in milliseconds) based on a variable incremented in SysTick ISR. This means that if HAL_Delay() is called from a peripheral ISR, then the SysTick interrupt must have highest priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR will be blocked."

Lab Procedure:

You will use the STM32cubeMX development environment to write some code related to the LED strip you have in your lab kit. The basic steps are as follows:

- Connect up the hardware using the LED strip, power connector, power supply, proto-boards, level-converter chip, and wires.
- Use STM32cubeMX to configure a project using GPIO pins E.13 and E.15 as outputs. •

Write SPI code to send 32-bit messages to the LED strip as serial signals.

- Create data structures for the LED programming
- Use that code and those data structures to send programming messages to the LED strip •

Do some interesting manipulations to your data to animate the LED strip.

Flashing lights in this lab can be bright and can be disturbing. If you are epileptic or have any other sensitivity to flashing lights, please inform me in advance. If you experience any discomfort during the lab, unplug everything and let us know. Keeping the brightness under 25% helps. (Also, we can't see anything brighter than that over video)

ALL VIDEOS ATTACHED SEPARATELY

Step 1: Connect up the hardware

The LED strip you have is from Pololu. In particular - https://www.pololu.com/product/3089.

A company called DONGGUANG OPSCO OPTOELECTRONICS CO., LTD makes a chip called the SK9822. It includes red/green/blue LEDs and a digital controller for those LEDs that uses the SPI protocol. Other companies mount these onto flex circuits and place them in a reasonably waterproof clear tube – generally 5M worth using this chip. Pololu cuts this into strips and solders a connector and power wires on them and provides some directions and software – but not the software you need .

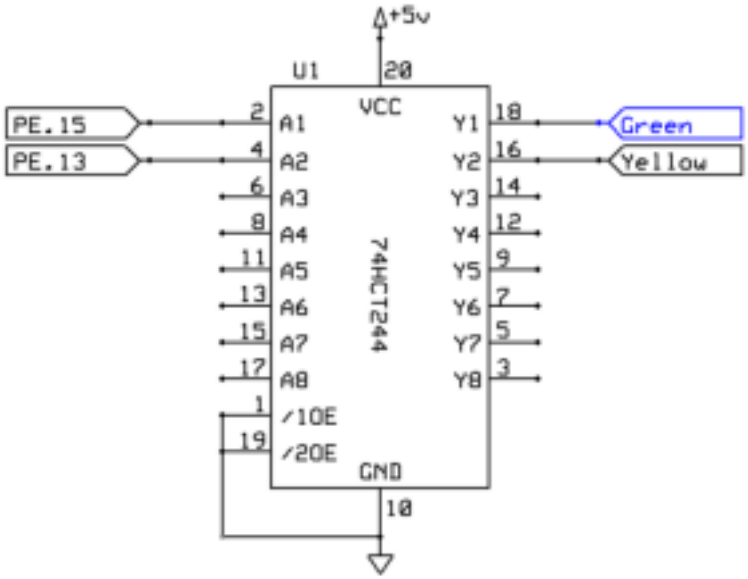
I have provided you with the datasheet for the SK9822 in the Lab Handouts/LED Strip Files directory. Review it and use it to develop this project. I have also copied the directions from Pololu. They recommend a small change to the protocol. The protocol in the SK9822 datasheet only works for up to 64 LEDs. Either one probably works for this lab.

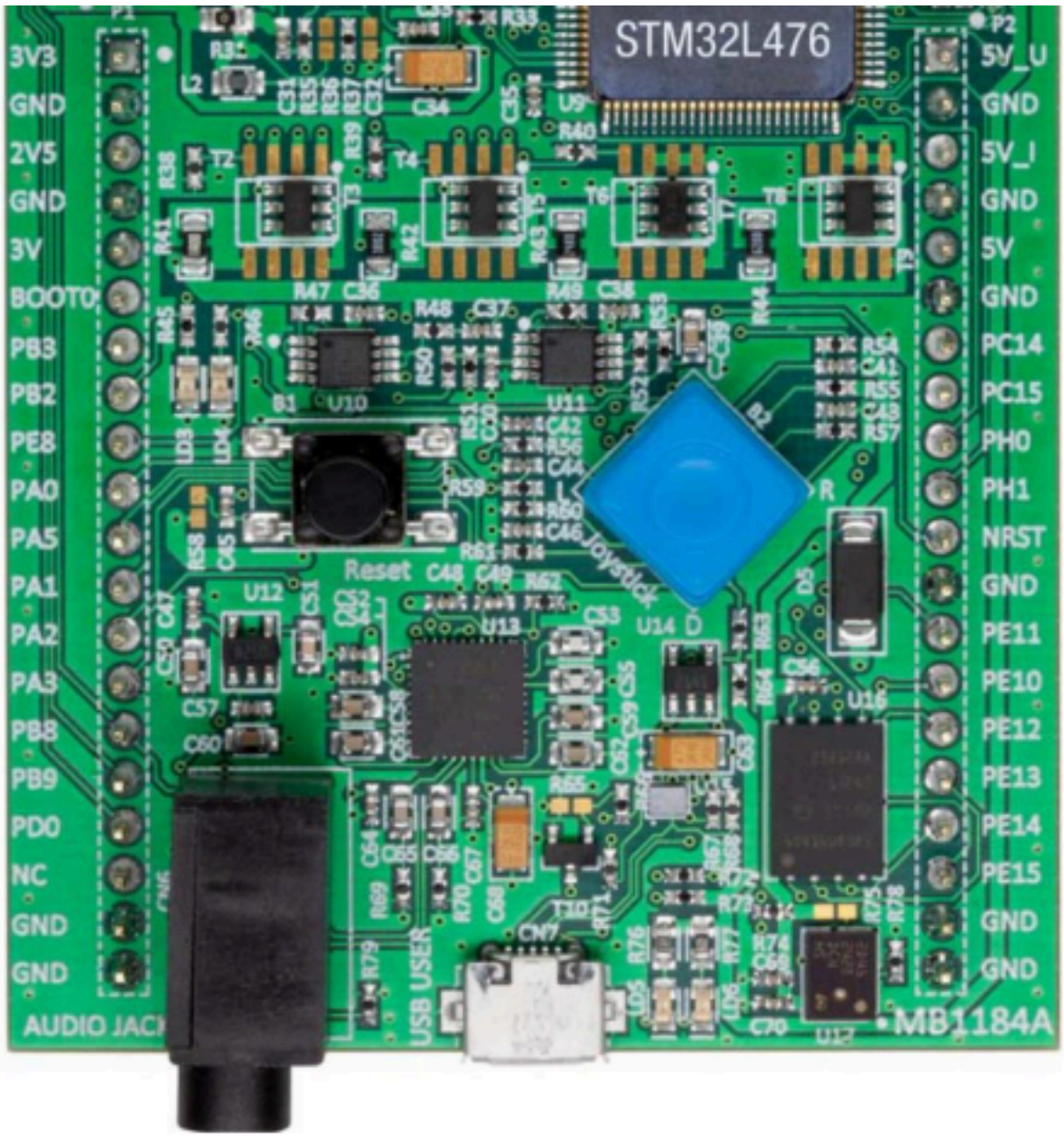
The connector on the LED with 4 male pins, near the power supply connector, is for the data connections. You can use male-female wires to connect the proto-board to the connector on the LED strip. They may already be installed. You can push the female end over the connector pins on the LED. The female wire connections are not really square, so they only all fit in one orientation. Connect yellow, green, and black wires to the matching colors on the LED strip.

We are going to use 2 GPIO pins to create an SPI output port. Pin E.13 will be the SPI clock. Pin E.15 will be the SPI Data Out. These will drive the LED chips. The LED will be powered by 5V. The GPIO signals should be configured as push-pull outputs even though this will only provide 3.3V. You need to convert these signals to 5V with the converter chip. Use the proto board and wires.

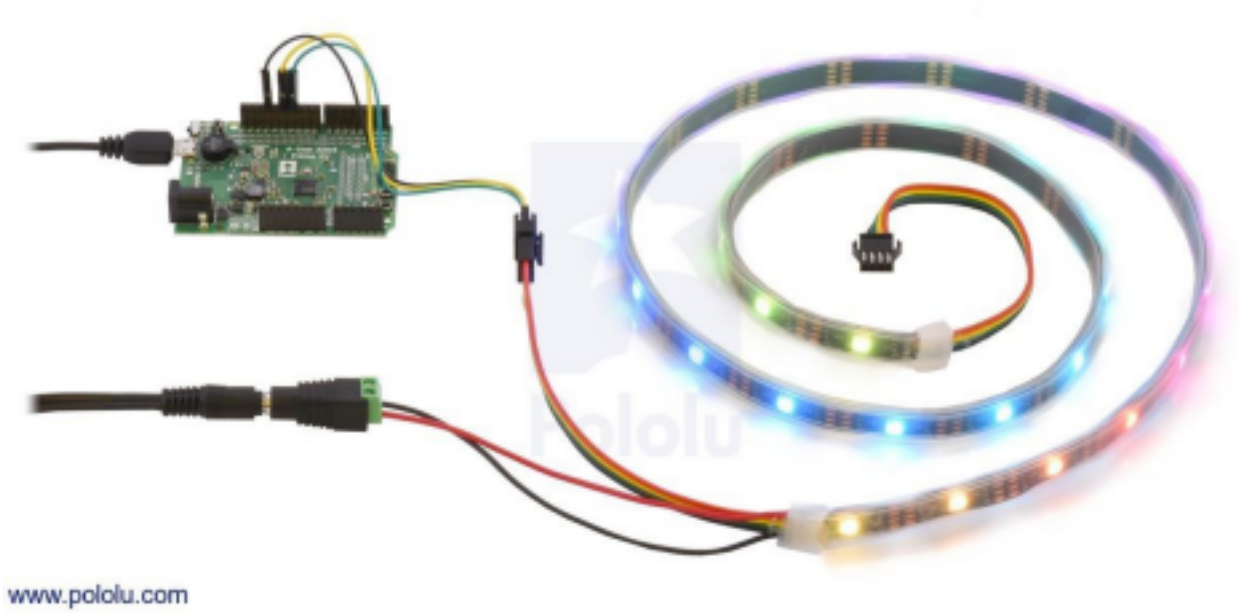
A schematic drawing is provided here. Review the 74HCT244 data sheet carefully and make sure you know which pins on the chip correspond to which pin numbers. Make sure you can identify pin 1 properly. There should be a 74HCT244 chip in your proto board straddling the center valley. Connect DISCO board pins PE.13 and PE.15 to pins 2 and 4 of the 74HCT244 chip as shown. Connect ground on the DISCO board to pins 1, 19, and 10 on the 74HCT244 chip. Connect 5V on the DISCO board to pin 20 on the 74HCT244 chip.

Connect the green and yellow LED wires to the 74HCT244 chip as shown. Connect the black wire from your LED strip to ground. The remaining pins can be unconnected.


+5v
U1
20
VCC
PE.15
PE.13
2
A1
4
A2
6
A3
8
A4
11
A5
13
A6
15
A7
17
A8
1
/1OE
19
/2OE
GND
10
74HCT244
Y1
18
Y2
16
Y3
14
Y4
12
Y5
9
Y6
7
Y7
5
Y8
3
Green
Yellow



Do not connect the red wire on the LED strip to your Discovery board or proto-board. The picture, below, has a different board and no level converter– but the idea is the same.



Step 2: Create an MX project

You need to create an MX project that is configured for the LED strip. I suggest starting from the scaled down MX project file that you created for MX Third Project. Create a new directory for your first LED project (maybe called First LED Project) and copy MX project file that you created for MX Third Project into that directory. Rename it to match the directory name. Follow my directions for copying an MX project.

Open that MX project file.

You need to configure port pins E.13 and E.15 to be output pins for a software-managed SPI port. These pins should be unconfigured to start and thus shown in yellow. (If not, you need to unconfigure them first.)

You can click on port pin PE13 in the chip diagram and select GPIO_Output. Do the same for port pin PE15. Then in the left-hand menu, select the GPIO block for configuration. This will open a GPIO configuration section. Select PE13. You can now configure it. Select Low for the GPIO output Level, Output Push Pull for the GPIO mode, No pull-up and no pull-down, Medium output speed, and set the User Label to SCK.

Next configure pin PE15. Select Low for the GPIO output Level, Output Push Pull for the GPIO mode, No pull-up and no pull-down, Medium output speed, and set the User Label to SDO.

Finally, go to the clock configuration and set HCLK to 80 MHz. You can then save this project, generate code, open the project, and quit STM32codeMX.

Find the comment in main.c:

```
 /*Configure GPIO pins : SCK_Pin SDO_Pin */
```

Copy the 5 lines of code that follow this comment into your lab report and explain in the lab report what each one does.

```
 GPIO_InitStruct.Pin = SCK_Pin|SDO_Pin;
```
//sets the GPIO Initialization Struct (GIS) pin value as both SCK and SDO hence the "or" bitwise operator.

```
 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
```
//sets the GIS mode as push pull

```
 GPIO_InitStruct.Pull = GPIO_NOPULL;
```
//sets the GIS pull down as no pull

```
 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_MEDIUM;
```
// sets the GIS speed to medium

```
 HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);
```
//calls the function that configures everything using the struct we just modified and the GPIOx we are using (in this case E)

//Everything here we configured in CubeMX beforehand

Step 3: Write a 32-bit SPI output routine

You need to write a routine to send out a 32-bit unsigned value out of the software-defined SPI port. The bits are sent out most significant bit (MSB) first. The procedure is as follows:

- Place the value of the MSB (bit 31) on the SPI data pin (SDO).
- Set the SPI clock pin (SCK) high
- Set the SPI clock pin low
- Repeat these 3 steps for the other 31 bits in order.

I have provided the comments and the function shell that I used. You must write the actual code.

```
/*****************************
/ send 32 bits out the SPI port - MSB first
 send out the 32 bits of c
 sclk starts low and ends low
 SCK_Pin is the SPI clock pin; SCK_GPIO_Port is the port  SDO_Pin
is the SPI data pin; SDO_GPIO_Port is the port */
```

```
void spi32(unsigned int c) {
 }
```

When you write subroutines like this in the MX environment, you must either:

1. Place them in the /* Private user code ----------*/ section in main.c
2. Place them in another C file you added to the project and provide a function prototype in the proper location. Your new C file may need to include main.h.

Step 4: Create messages and send them to the LED strip.

The protocol for the SK9822 works as follows:

- You are communicating with the first chip. A header word alerts the chip that data is coming.

- The first chip grabs the next 32 bits as its new lighting command.
- The remaining bits are passed on to the next chip. It will grab the next 32 bits as its lighting command and pass on the rest, etc.
- Each set of passed-on bits is delayed by ½ clock cycle. Because of this you need to add on some extra data at the end. The SK9822 recommends 0xFFFFFFFF. Sending 0xFFFFFFFF twice after the lighting commands also worked fine for me. Pololu has an alternate recommendation.

The basic structure of messages is shown in the diagram from the datasheet below:

Each message is made up of 32-bit words. A message is a header, one or more LED frames, and 1 or 2

end frames. The header is a 32-bit word 0x00000000. Each end frame is a 32-bit word 0xFFFFFFFF. Use a 2-word end frame whenever you are lighting more than 64 LEDs. When setting only a small number of LEDs, this still may give you some extra lights. You should probably use the exact technique shown in the Pololu documentation for small LED strings – but you can try 32-bits of all 0 and that should work.

Each LED frame specifies the color for one LED. The LEDs work like a shift register. If you send 4 LED frames, the LED 4 down from the wires gets the first one and so forth.

An LED frame has the 3 MSB bits set. The next 5 bits are the brightness. Keep the brightness below 30% until you have everything working well. Then 8 bits each for Blue, Green, and Red.

For the remaining project steps, you must create any data structures you need and any algorithms to prepare and send messages to the LED strip.

For this step – send a message with one LED frame that sets one LED to red. If some additional LEDs light up (maybe white) that means that the protocol you are using is not the best one. You can try to fix that, or you can just move on to working with the full LED strip.

Step 5: Model the LED as a display

The most common way of managing any modern display is to use a memory buffer called a frame buffer that includes a data representation of the image to display. In the case, for example, of your PC display, there is a 2D array of pixel color values held in memory. Periodically, that array is transmitted to the display device. There can be multiple such buffers with one active at any given time.

For the LED display, you need a 1D frame buffer. In the case of my Pololu display that would be 60 elements each of which is a 32-bit color value according to the LED Frame format above.

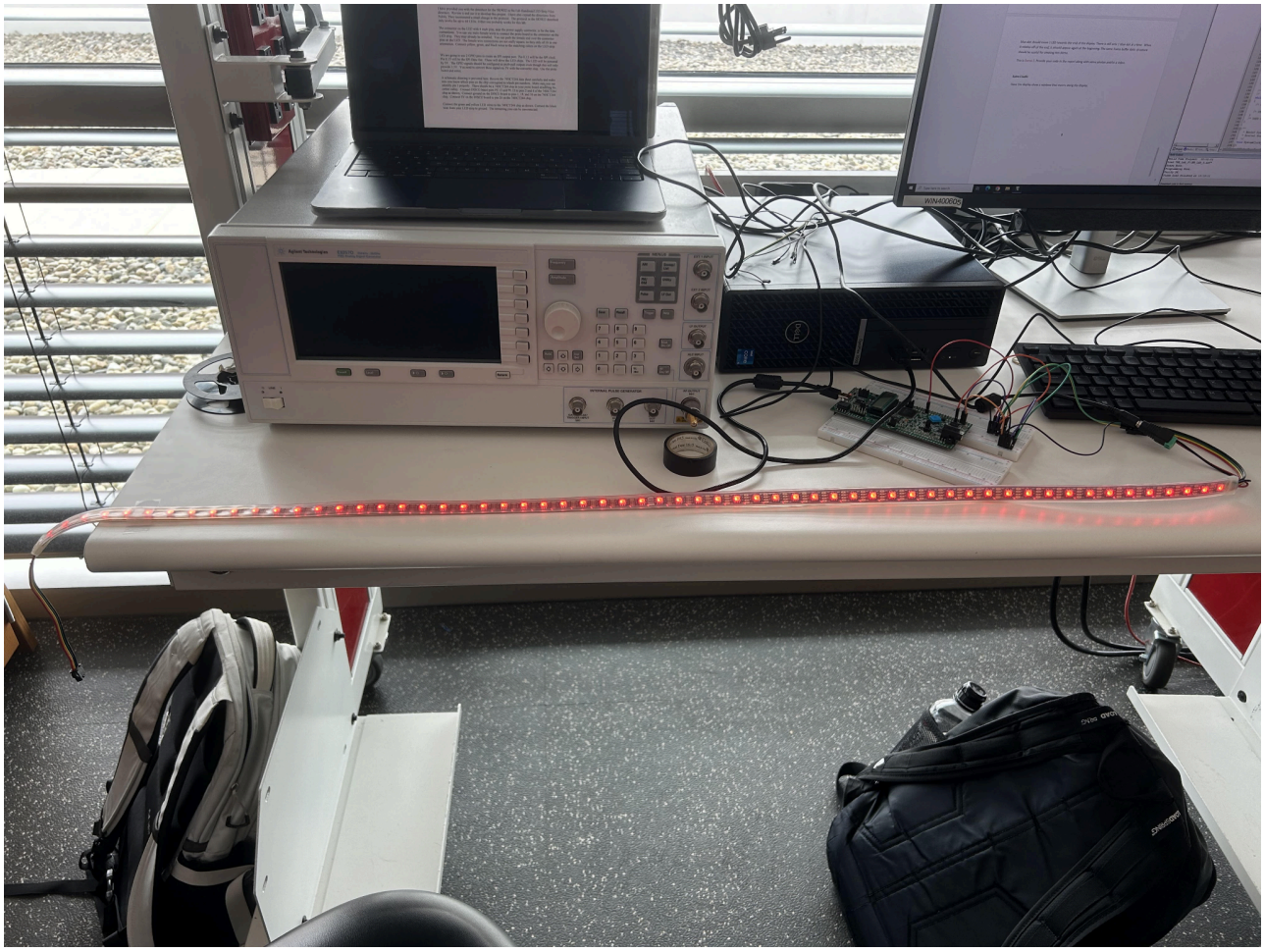
The LED display remembers what is sent to it as long as there is power, so I can either:

- Periodically refresh the LED display
- Refresh it specifically when I want it to change

For this step:

1. Create a data structure to act as your frame buffer.
2. Initialize it to all one color
3. Create a send_array() routine that sends the header, the frame buffer data, and the end frames. It should use spi32(). Call this routine from main() to set the LED's to all one color. Demo this as Demo 1. Provide your code in the report along with a photo.

PHOTO:



DEMO 1 CODE:

```
/* USER CODE BEGIN 0 */
/*****************************
/ send 32 bits out the SPI port - MSB first
 send out the 32 bits of c
 sclk starts low and ends low
 SCK_Pin is the SPI clock pin;
 SCK_GPIO_Port is the port
 SDO_Pin is the SPI data pin;
 SDO_GPIO_Port is the port */
void spi32(unsigned int c)
{
            int i;
  for (i = 0; i < 32; ++i) {
     if (c & 0x80000000)
                                        {
         HAL_GPIO_WritePin(SDO_GPIO_Port, SDO_Pin, GPIO_PIN_SET);
                                        }
       else
                                        {
```

```
        HAL_GPIO_WritePin(SDO_GPIO_Port, SDO_Pin, GPIO_PIN_RESET);
                                        }
    HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_SET);
                                        HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_RESET);
    c <<= 1;
                }
}
void send_array(uint32_t header, uint32_t color, uint32_t end)
{
        int i;
        spi32(header);
        for(i=0; i < 60; i++)
        {
                spi32(color);
        }
        spi32(end);
}
/* USER CODE END 0 */

/**
 * @brief  The application entry point.
 * @retval int
 */
int main(void)
{
 /* USER CODE BEGIN 1 */
 /* USER CODE END 1 */

 /* MCU Configuration--------------------------------------------------------*/

 /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
 HAL_Init();

 /* USER CODE BEGIN Init */

 /* USER CODE END Init */

 /* Configure the system clock */
 SystemClock_Config();

 /* USER CODE BEGIN SysInit */

 /* USER CODE END SysInit */

 /* Initialize all configured peripherals */
 MX_GPIO_Init();
```

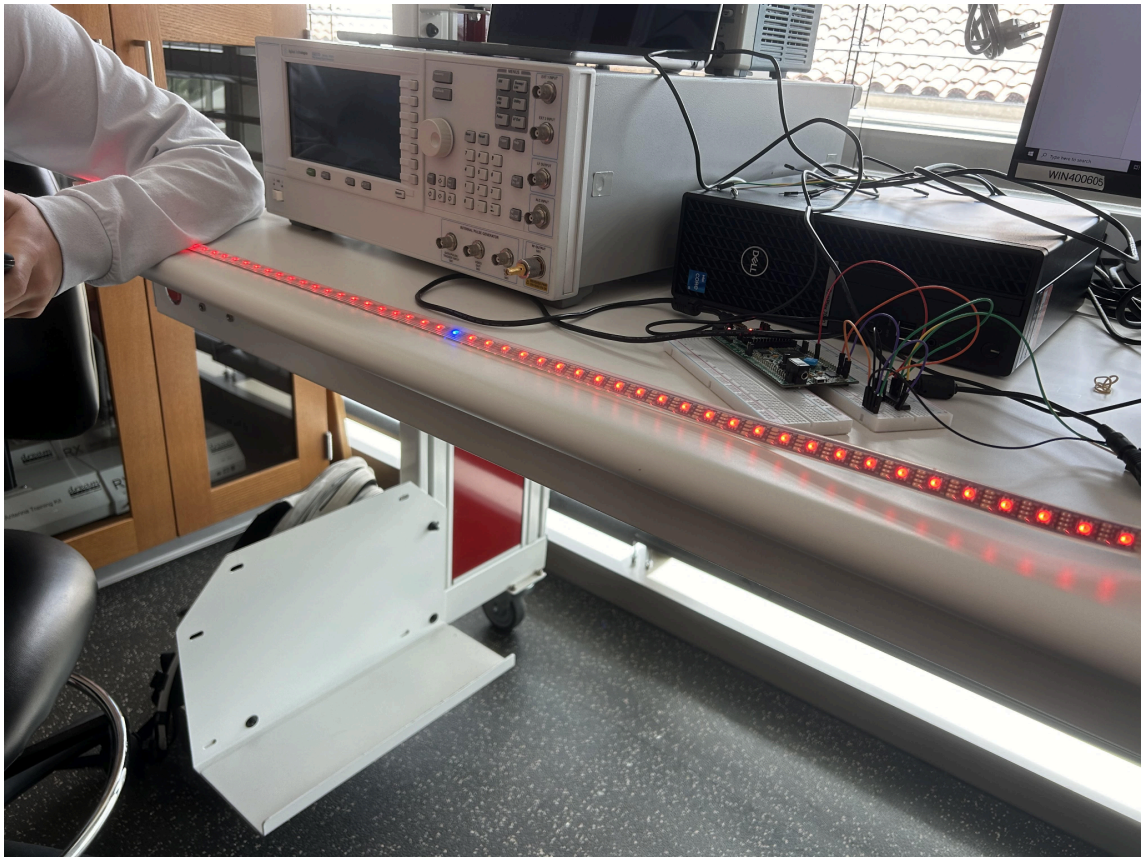
```
 MX_RTC_Init();
 /* USER CODE BEGIN 2 */
          uint32_t start = 0x00000000;
          uint32_t end = 0xffffffff;
          uint32_t red = 0xe40000ff;
          uint32_t blue = 0xe4ff0000;
          uint32_t green = 0xe400ff00;
          send_array(start, red ,end);
 /* USER CODE END 2 */

 /* Infinite loop */
 /* USER CODE BEGIN WHILE */
 while (1)
 {
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
 }
 /* USER CODE END 3 */
}
```

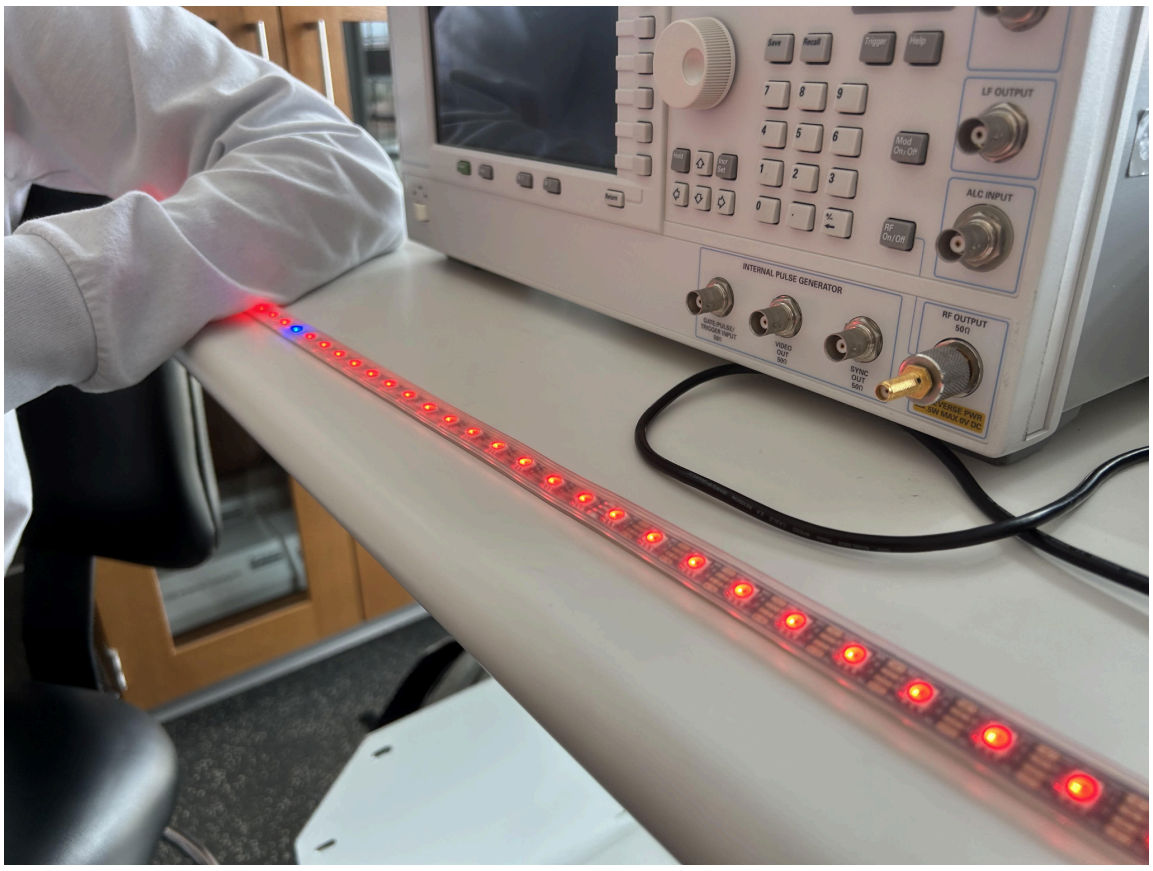
Step 6: Animate the LED as a display

Copy your project using one of the techniques previously explained. Call the new project LED display 2.

You are going to program your LED to show a single blue dot in the middle. Then every 250ms or so, the blue dot should move 1 LED towards the end of the display. There is still only 1 blue dot at a time. When it rotates off of the end, it should appear again at the beginning. The same frame buffer data structure should be useful for creating this demo.

This is Demo 2. Provide your code in the report along with some photos and/or a video.
PHOTO(s):


WIN400605


Save
Recall
Trigger
Help
LF OUTPUT
Mod On/Off
ALC INPUT
RF On/Off
INTERVAL PULSE GENERATOR
RF OUTPUT 50Ω
VIDEO OUT 50Ω
SYNC OUT 50Ω

DEMO 2 CODE:

```
/* USER CODE BEGIN 0 */
/*****************************
/ send 32 bits out the SPI port - MSB first
 send out the 32 bits of c
 sclk starts low and ends low
 SCK_Pin is the SPI clock pin;
 SCK_GPIO_Port is the port
 SDO_Pin is the SPI data pin;
 SDO_GPIO_Port is the port */
int globalCount = 30;
void spi32(unsigned int c)
{
        int i;
  for (i = 0; i < 32; ++i) {
    if (c & 0x80000000)
                                        {
      HAL_GPIO_WritePin(SDO_GPIO_Port, SDO_Pin, GPIO_PIN_SET);
                                        }
     else
                                        {
      HAL_GPIO_WritePin(SDO_GPIO_Port, SDO_Pin, GPIO_PIN_RESET);
                                        }
     HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_SET);
                                        HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_RESET);
     c <<= 1;
                }
}
void send_array(uint32_t header, uint32_t color, uint32_t end)
{
        int i;
        spi32(header);
        for(i=0; i < 60; i++)
        {
                spi32(color);
        }
        spi32(end);
}
void bluemove (uint32_t header, uint32_t color, uint32_t end)
{
        uint32_t blue = 0xe4ff0000;
        int i;
        spi32(header);
        for(i=0; i < 60; i++)
        {
                if (i==globalCount)
                {
                        spi32(blue);
                }
                else
                {
                        spi32(color);
                }
```

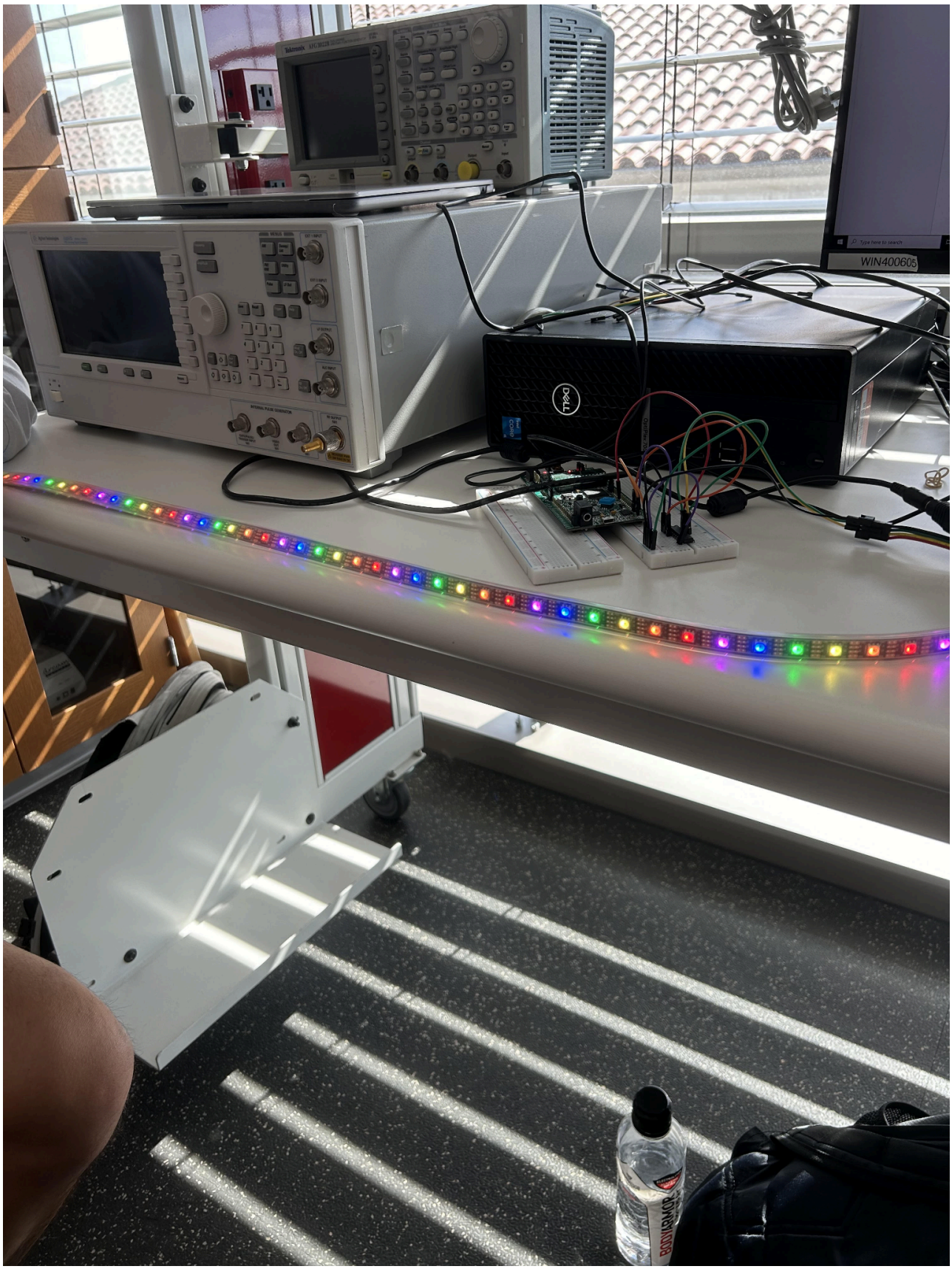
```
		}
		spi32(end);
		globalCount++;
		if(globalCount > 60)
		{
			globalCount=0; //
		}

}
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_RTC_Init();
  /* USER CODE BEGIN 2 */
		uint32_t start = 0x00000000;
		uint32_t end = 0xffffffff;
		uint32_t red = 0xe40000ff;
		uint32_t blue = 0xe4ff0000;
		uint32_t green = 0xe400ff00;
		uint32_t black = 0xe0000000;
		//send_array(start, red ,end);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
			bluemove(start, black ,end); //other color that blue moves through is the middle argument
			HAL_Delay(250);
```

```
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Extra Credit:

Have the display show a rainbow that moves along the display.

EC PHOTO:


WIN400605
DELL
BODYARMOR

EXTRA CREDIT CODE:

```
/* USER CODE BEGIN 0 */
/*****************************
/ send 32 bits out the SPI port - MSB first
 send out the 32 bits of c
 sclk starts low and ends low
 SCK_Pin is the SPI clock pin;
 SCK_GPIO_Port is the port
 SDO_Pin is the SPI data pin;
 SDO_GPIO_Port is the port */
int globalCount = 0;
uint32_t start = 0x00000000;
uint32_t end = 0xffffffff;
uint32_t red = 0xe40000ff;
uint32_t blue = 0xe4ff0000;
uint32_t green = 0xe400ff00;
uint32_t yellow = 0xe400a8ff;
uint32_t orange = 0xe40024ff;
uint32_t purple = 0xe4cf00ff;
uint32_t rainbow[6] = {0xe40000ff, 0xe40024ff, 0xe400a8ff, 0xe400ff00, 0xe4ff0000, 0xe4cf00ff};
uint32_t black = 0xe0000000;
void spi32(unsigned int c)
{
            int i;
  for (i = 0; i < 32; ++i) {
     if (c & 0x80000000)
                                          {
         HAL_GPIO_WritePin(SDO_GPIO_Port, SDO_Pin, GPIO_PIN_SET);
                                          }
       else
                                          {
         HAL_GPIO_WritePin(SDO_GPIO_Port, SDO_Pin, GPIO_PIN_RESET);
                                          }
       HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_SET);
                                          HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_RESET);
       c <<= 1;
                    }
}
void send_array(uint32_t header, uint32_t color, uint32_t end)
{
            int i;
            spi32(header);
            for(i=0; i < 60; i++)
            {
                        spi32(color);
            }
            spi32(end);
}
void bluemove (uint32_t header, uint32_t color, uint32_t end)
{
            uint32_t blue = 0xe4ff0000;
            int i;
            spi32(header);
            for(i=0; i < 60; i++)
```

```
		{
			if (i==globalCount)
			{
				spi32(blue);
			}
			else
			{
				spi32(color);
			}

		}
		spi32(end);
		globalCount++;
		if(globalCount > 60)
		{
			globalCount=0; //
		}

}
void rainbowMoveRtoL()
{
		int i;
		spi32(start);
		for(i=0; i < 60; i++)
		{
			int x = (globalCount + i) % 6;
			spi32(rainbow[x]);
		}
		spi32(end);
		globalCount+=5;
		if(globalCount ==30)
		{
			globalCount = 0;
		}
}
void rainbowMoveLtoR()
{
		int i;
		spi32(start);
		for(i=0; i < 60; i++)
		{
			int x = (globalCount + i) % 6;
			spi32(rainbow[x]);
		}
		spi32(end);
		globalCount++;
		if(globalCount ==6)
		{
			globalCount = 0;
		}
}
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
```

```
 * @retval int
 */
int main(void)
{
 /* USER CODE BEGIN 1 */
 /* USER CODE END 1 */

 /* MCU Configuration--------------------------------------------------------*/

 /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
 HAL_Init();

 /* USER CODE BEGIN Init */

 /* USER CODE END Init */

 /* Configure the system clock */
 SystemClock_Config();

 /* USER CODE BEGIN SysInit */

 /* USER CODE END SysInit */

 /* Initialize all configured peripherals */
 MX_GPIO_Init();
 MX_RTC_Init();
 /* USER CODE BEGIN 2 */
          uint32_t start = 0x00000000;
          uint32_t end = 0xffffffff;
          uint32_t red = 0xe40000ff;
          uint32_t blue = 0xe4ff0000;
          uint32_t green = 0xe400ff00;
          uint32_t black = 0xe0000000;
          //send_array(start, red ,end);
 /* USER CODE END 2 */

 /* Infinite loop */
 /* USER CODE BEGIN WHILE */
 while (1)
 {
                    //changes direction after 60 iterations endlessly
                    int i;
                    for(i=0; i < 60;i++)
                    {
                              rainbowMoveRtoL();
                              HAL_Delay(100);
                    }
                    for(i=0; i < 60;i++)
                    {
                              rainbowMoveLtoR();
                              HAL_Delay(100);
                    }

  /* USER CODE END WHILE */
```

```
  /* USER CODE BEGIN 3 */
 }
 /* USER CODE END 3 */
}
```