Because there's no set structure for the lab notebook, we decided to treat ours as a general set of useful knowledge.

LAB 0 & 1:

I have changed stm32l476xx.h to include RCC_AHB2ENR value (4c)
GPIO_MODER was also added into stm321476xx.h (0x00)

USEFUL FUNCTIONS (X=some letter)
RCC->AHB2ENR |= RCC_AHB2ENR_GPIO**X**EN;
Enables the clock for **X** register.

GPIO**X**->ODR |= GPIO_ODR_ODR_**#**; //or
While used with red or green led on/off functions, allows the leds to be turned on. (Must correspond to correct GPIO port and pin #s).

GPIO**X**->ODR ^= GPIO_ODR_ODR_**#**; //xor
While used with red or green toggle functions, allows leds to be toggled. ((Must correspond to correct GPIO port and pin #s).

GPIO**X**->ODR &= ~(GPIO_ODR_ODR_**#)**; //bitclear
While used with red or green led on/off functions, allows the leds to be turned off. (Must correspond to correct GPIO port and pin #s).

HAL_GPIO_TogglePin(Port Name, Pin Name);
Allows for the corresponding LEDs to be toggled while using HAL functions. Port names and pin names correspond to their names in the CubeMX interface.

HAL_Delay(**#**);
Creates a delay while using HAL functions. Uses an integer as the input.

LAB 2:

From HW 3 (just to have more examples):
Configure green and red LEDS to:

- Be High Speed instead of Low Speed. ->OSPEEDR
- Have a Pull-Up resistor -> PUPDR

```
GPIOB->PUPDR &= ~(0x3<<(2*2)); //clear
GPIOB->PUPDR |= (0x1<<(2*2)); //set 0th bit
GPIOE->PUPDR &= ~(0x3<<(8*2)); //clear
GPIOE->PUPDR |= (0x1<<(8*2)); // set 0th bit
GPIOB->OSPEEDR |= (0x3<<(2*2)); // high speed
GPIOE->OSPEEDR |= (0x3<<(8*2)); //high speed
```

MODER:

00: Input mode
01: General purpose output mode
10: Alternate function mode
11: Analog mode (reset state)

ODR:

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODy:** Port output data bit (y = 0..15)
These bits can be read and written by software.

PUPDR:

00: No pull-up, pull-down
01: Pull-up
10: Pull-down
11: Reserved

OSPEEDR:

00: Low speed
01: Medium speed
10: Fast speed
11: High speed

LAB 3:

bold=brightness

Start Frame for LED: 0000 0000 0000 0000 0000 0000 0000 0000
LED Frame: 111**X XXXX** XXXX XXXX XXXX XXXX XXXX XXXX
End Frame: 1111 1111 1111 1111 1111 1111 1111 1111
To connect the LED strip physically, follow the pin numbers on the chip in a counterclockwise sequence around the board. Note that these numbers may not match the image provided in the handout.

LAB 4:

//This is where you write actions to be done after button presses. In this case, Green and Red LEDs are toggled:

```
void EXTI0_IRQHandler(void)
{
 /* USER CODE BEGIN EXTI0_IRQn 0 */

 /* USER CODE END EXTI0_IRQn 0 */
 HAL_GPIO_EXTI_IRQHandler(JOY_CENTER_Pin);
 /* USER CODE BEGIN EXTI0_IRQn 1 */
        HAL_GPIO_TogglePin(LD_R_GPIO_Port, LD_R_Pin); //GPIOB pin 2

 /* USER CODE END EXTI0_IRQn 1 */
}

/**
 * @brief This function handles EXTI line1 interrupt.
 */
void EXTI1_IRQHandler(void)
{
 /* USER CODE BEGIN EXTI1_IRQn 0 */

 /* USER CODE END EXTI1_IRQn 0 */
 HAL_GPIO_EXTI_IRQHandler(JOY_LEFT_Pin);
 /* USER CODE BEGIN EXTI1_IRQn 1 */
        HAL_GPIO_TogglePin(LD_G_GPIO_Port, LD_G_Pin); //GPIOE pin 8
 /* USER CODE END EXTI1_IRQn 1 */
}
```

Known input clock frequency divided by (prescaler * counter period) = goal freq
Hclk/(timx prescaler * timx counter period) = goal freq
If hclk = 40MHz and goal freq = 10hz, then prescaler * counter period must equal 4 million.

Tim2 uses APB1
Tim16 uses APB2
To find this info and more go to reference manual - (2) system and mem overview - bus matrix - ahb/apb bridges

For the LCD screen, check lab 4 for configuration. Everything is there. LCD IS ¼ DUTY CYCLE!

The NVIC handles interrupts so we don't have to worry about corruptions. If you don't use the NVIC, you will have to disable interrupts or create a lock/flag of some sort. Basically turn off

interrupts or mimic NVIC behavior.

LAB 5:

Data handled in interrupt functions managed by the NVIC won't ever get corrupted. For things in main, you need flags or you need to disable interrupts. This lab was really just on buffers, and writing a circular buffer doesn't really require notes. Do remember to stop writing data if the buffer is full though.

LAB 6:

The data seen from the remote is inverted. I just inverted it again, changing it back to its initial signal however this isn't necessary. You could just know it's inverted and search for 1s instead of 0s.

Timer code:

```
x = HAL_GPIO_ReadPin(IR_IN_GPIO_Port, IR_IN_Pin);
        if(flag == 0)
        {
                if(x == 0 || data == 1)
                {
                        data = 1;
                        if(x==0)
                        {
                                irdat[count] = 1;
                                count = (count +1) %SAMPLE_COUNT;
                        }
                        else
                        {
                                irdat[count]=0;
                                count= (count +1) %SAMPLE_COUNT;
                        }
                        if(count == SAMPLE_COUNT-1)
                        {
                                flag =1;
                                data =0;
                        }
                }
        }
```

Parse function:

```
uint32_t parseIRCode()
        {
  uint32_t num = 0;
                int count = 0;
                int arrayCount = 0;
                uint32_t mask = 0x80000000;
                /*skips header */
                while(irdat[arrayCount] !=0)
                {
                        arrayCount++;
                }
                while(irdat[arrayCount] !=1)
                {
                        arrayCount++;
                }
                /* skips header */
                while(arrayCount != SAMPLE_COUNT)
                {
                        if(irdat[arrayCount] ==0)
                        {
                                count++;
                        }
                        else
                        {
                                if(count < 8 && count !=0)
                                {
                                        mask/=2;
                                }
                                else if(count > 8 && count != 0)
                                {
                                        num |= mask;
                                        mask/=2;
                                }
                                count = 0;
                        }
                        arrayCount++;
                }
                return num;
        }
```

LAB 7:

We had a problem with part three in which our left click wasn't working correctly. We had to put our code in the "if (mouseInfo != NULL)" statement to make it work for the left click, but not the middle or right click. This is because the left click is the start of the array and is affected by movement changes as both data are sent around the same time with how Wolfe constructed his part of the code. Our change stops the code from getting two signals around the same time, and instead the data is updated one after the other when any mouseInfo change is detected. By putting the code in a section that activates if any mouseInfo changes, we solve for the left click issue without affecting right or middle click outputs.

MAIN.C CODE:

```
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_USB_HOST_Init();
 /* USER CODE BEGIN 2 */
        //DECLARE PRIVATE VARIABLES FOR USE
                int flags[3] = {0,0,0};
                int array[4] = {KBLUE, KRED, KPURPLE, KGREEN};
                int index =0;
                int toggle[3] = {0,0,0};
 /* USER CODE END 2 */

 /* Infinite loop */
 /* USER CODE BEGIN WHILE */
 while (1)
 {
                setDot(colors, NUM_LEDS, spotLocation, array[index]);
                send_array(colors);
                /* USER CODE END WHILE */
  MX_USB_HOST_Process();
```

```
  /* USER CODE BEGIN 3 */
  if (hUsbHostFS.gState == HOST_CLASS) {

#ifdef KBMOUSECOMBO
                        hUsbHostFS.device.current_interface = 1;
#endif
                    devType = USBH_HID_GetDeviceType(&hUsbHostFS);
                    if (devType == HID_MOUSE) {
                        HAL_GPIO_WritePin(LD_R_GPIO_Port, LD_R_Pin,GPIO_PIN_SET);
                    }   else {
                        HAL_GPIO_WritePin(LD_R_GPIO_Port, LD_R_Pin,GPIO_PIN_RESET);
                    }
                    if (devType == HID_KEYBOARD) {
                        HAL_GPIO_WritePin(LD_G_GPIO_Port, LD_G_Pin,GPIO_PIN_SET);
                    }   else {
                        HAL_GPIO_WritePin(LD_G_GPIO_Port, LD_G_Pin,GPIO_PIN_RESET);
                    }
                    if (devType == HID_MOUSE) {
                        mouseInfo = USBH_HID_GetMouseInfo(&hUsbHostFS);
                        if (mouseInfo != NULL) {
                            spotLocation = spotUpdate(spotLocation, fixData(mouseInfo->x));
                            //Code goes beyond specifications and allows toggle back to blue
                            //if you click the same button twice in a row. If you didn't want this,
                            //just delete toggle and all its instances.

                            //handles left click to red
                            if(mouseInfo->buttons[0]==1)
                            {
                                toggle[1] = 0;
                                toggle[2] = 0;
                                if (flags[0] ==0)
                                {
                                    flags[0]=1;
                                }
                            }
                            if(mouseInfo->buttons[0]==0 && flags[0] ==1)
                            {
                                index = 1;
                                flags[0] = 0;
                                toggle[0]++;
                            }

                            //handles right click to purple
                            if(mouseInfo->buttons[1]==1)
                            {
                                toggle[0] = 0;
                                toggle[2] = 0;
                                if (flags[1] ==0)
                                {
                                    flags[1]=1;
                                }
                            }
```

```
                                        if(mouseInfo->buttons[1]==0 && flags[1] ==1)
                                        {
                                                index = 2;
                                                flags[1] = 0;
                                                toggle[1]++;
                                        }

                                        //handles middle click to green
                                        if(mouseInfo->buttons[2]==1)
                                        {
                                                toggle[0] = 0;
                                                toggle[1] = 0;
                                                if (flags[2] ==0)
                                                {
                                                        flags[2]=1;
                                                }
                                        }
                                        if(mouseInfo->buttons[2]==0 && flags[2] ==1)
                                        {
                                                index = 3;
                                                flags[2] = 0;
                                                toggle[2]++;
                                        }

                                        //handles the toggle back to blue
                                        if(toggle[0] >=2 || toggle[1] >= 2 || toggle[2] >=2)
                                        {
                                                index = 0;
                                                toggle[0] = 0;
                                                toggle[1] = 0;
                                                toggle[2] = 0;
                                        }
                                }
                        }
                }
                else if (hUsbHostFS.gState == HOST_IDLE) {
                        HAL_GPIO_WritePin(LD_G_GPIO_Port, LD_G_Pin,GPIO_PIN_RESET);
                        HAL_GPIO_WritePin(LD_R_GPIO_Port, LD_R_Pin,GPIO_PIN_RESET);
                }
 }
 /* USER CODE END 3 */
}
```

LAB 8:

The FatFs - Generic FAT Filesystem Module is very useful and includes all the file functions we'd use.

Reads a text file and puts the contents into an array:

```
static void FS_FileOperations(void) {
        FRESULT res;                    /* FatFs function common result code */
 uint32_t bytesread;              /* File write/read counts */

         /* Register the file system object to the FatFs module */
 if(f_mount(&USBHFatFS, (TCHAR const*)USBHPath, 0) == FR_OK) {
                  /* Open the text file object with read access */
                  if(f_open(&MyFile, rfilename, FA_READ) == FR_OK) {
                           /* Read data from the text file */
                           res = f_read(&MyFile, rtext, sizeof(rtext)-1, &bytesread);
                           if((bytesread > 0) && (res == FR_OK)) {
                                    /* Close the open text file */
                                    f_close(&MyFile);
                                    /* Compare read data with the expected data */
                            HAL_GPIO_WritePin(LD_G_GPIO_Port, LD_G_Pin, GPIO_PIN_SET);
                            return;
                           }
                  }
        }
        HAL_GPIO_WritePin(LD_R_GPIO_Port, LD_R_Pin, GPIO_PIN_SET);
}
```

Reads all directories/files on a thumbdrive and prints them into a text file:

```
FRESULT scan_files( char* path) {
        FRESULT res;
        DIR dir;
        int i;
        static FILINFO fno;
        res = f_opendir(&dir, path);                  /* Open the directory */
  if (res == FR_OK) {
                           while (1) {
       res = f_readdir(&dir, &fno);              /* Read a directory item */
       if (res != FR_OK || fno.fname[0] == 0) break;  /* Break on error or end of dir */
       if (fno.fattrib & AM_DIR) {               /* It is a directory */
          i = strlen(path);
          sprintf(&path[i], "/%s", fno.fname);
          res = scan_files(path);                /* Enter the directory */
          if (res != FR_OK) break;
          path[i] = 0;
       }
                                                        else
                                                                 {                          /* It is a file. */
          sprintf(dirlist + strlen(dirlist), "%s/%s\n", path, fno.fname);
       }
                           }
    f_closedir(&dir);
```

```
  }
 return res;
}

FRESULT write_dirlist(char* path, char* list) {
        FIL OutFile;
        FRESULT res;
        uint32_t byteswritten;

        if (f_open(&OutFile, path, FA_WRITE | FA_CREATE_ALWAYS) == FR_OK)
        {
                res = f_write(&OutFile, list, strlen(list), &byteswritten);
                if (res == FR_OK)
                {
                        f_close(&OutFile);
                }
        }
 return res;
}


/**
 * @brief  Main routine for Mass Storage Class
 * @param  None
 * @retval None
 */
```

LAB 9 + EC:

Individual system examples:
IR Remote Code
LCD Display Code
LED Strip Code
Reading from USB Code

Lab 9:
This lab is really an amalgamation of all other labs. There is no specific useful code or diagrams that aren't in the drive (IR remote, FATFS docs, LCD display stuff, etc.) or in previous sections of this notebook. However, after doing the lab, it should be noted that working incrementally is extremely helpful. You should check the functionality of all your systems before you begin merging them together. Numerous groups had problems with the individual systems themselves and when the systems combined, so by checking to see if a system actually works before merging, you narrow down where the issue could be coming from. Timers are also super useful and the NVIC is goated. I used 3 different timers for lab 9 and 4 timers for the extra credit portion.

Extra Credit:
For the LED strip of the extra credit section, have it update 8 times a second (8Hz). This creates a good enough look. Without messing up or slowing down the other systems. For the multiple

mp3 files section, use the same everything and just place the fno.fname into a 2d array. For skipping songs, increment the row of the 2d array and take all the code from reset. Lastly for displaying the full song name, make sure to enable long file names in CubeMX.