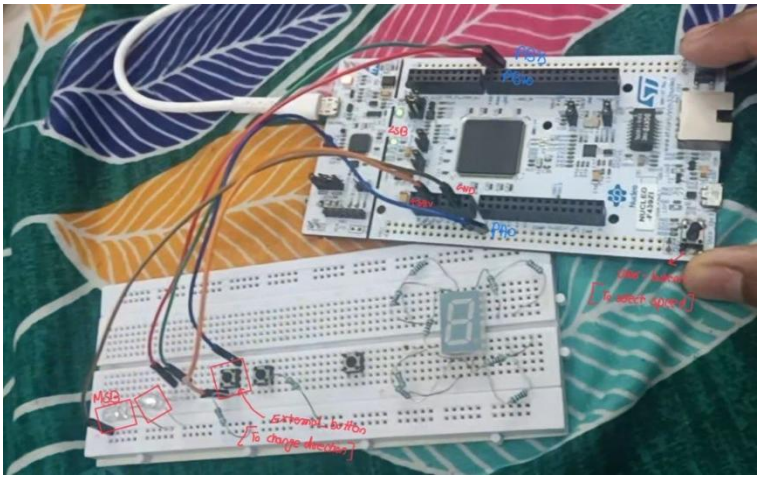
Lab Assignment 4

Mohit Maurya (22110145)

Parag Sarvoday Sahu (22110179)

Q1: Interrupt-based LED chase pattern

Overview:



An image of the circuit

Here, we have implemented a light sweep pattern.

The variable external_button is used to change the direction of the sweep. We have used 4 different speeds for chasing patterns 4 Hz, 2 Hz, 1 Hz, and 0.5Hz. The user_button is used to increase or decrease the speed. The speed array contains the value of ARR (Autoreload Register), which is set during the runtime whenever the user presses the user_button. We have used the TIM2 timer which has a natural clock frequency of 84 MHz.

The value of the prescaler is 8400-1, and the value of ARR depends on the speed array.

```
/* USER CODE BEGIN PV */
int counter=1;
int external_button=0;  // 0 = lsb->msb    1-> msb->lsb   // PA0
int user_button=0;      // PB13  user button
int speed[4]  ={2500,5000,10000,20000};
int i=0;
int previoustime;
int currenttime;
int debouncetime=200;

/* USER CODE END PV */
```

Notice the TIM2

```
/* Private function prototypes -----------------------------------------
----*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ETH_Init(void);
static void MX_USART3_UART_Init(void);
static void MX_USB_OTG_FS_PCD_Init(void);
static void MX_TIM2_Init(void);
```

```
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ETH_Init();
  MX_USART3_UART_Init();
  MX_USB_OTG_FS_PCD_Init();
  MX_TIM2_Init();
```

Initialization of timer 2

```
  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim2);
  /* USER CODE END 2 */
```

External button callback

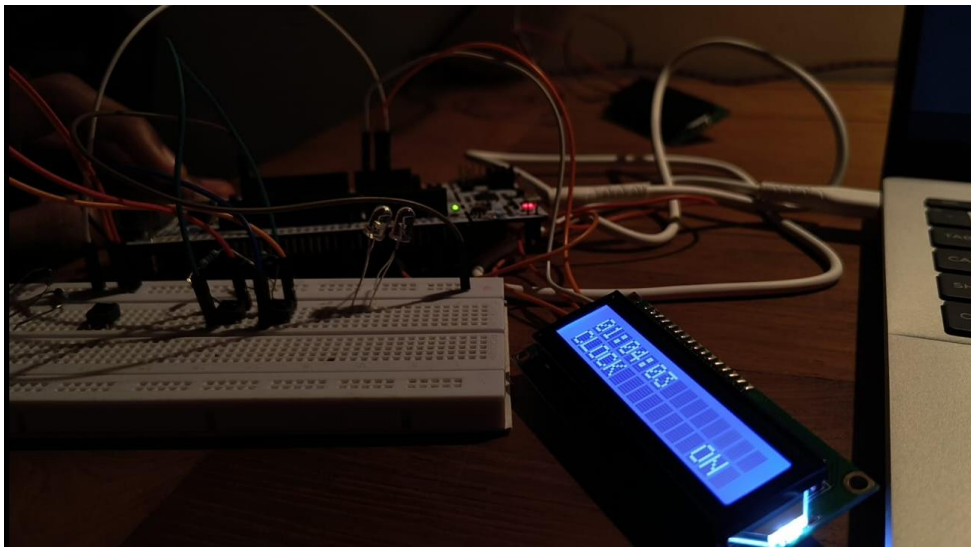
```
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(GPIO_Pin);
  /* NOTE: This function Should not be modified, when the callback is
needed,
           the HAL_GPIO_EXTI_Callback could be implemented in the user
file
   */
  // PIN 15 FOR increment BUTTON
  // PIN 10 FOR select_BUTTON
  currenttime = HAL_GetTick();

  // user button to change speed
  if (GPIO_Pin == GPIO_PIN_13 && (currenttime-previoustime>debouncetime)
){
                        i=(i+1)%4;
                        __HAL_TIM_SET_AUTORELOAD(&htim2, speed[i]-1);
                          TIM2->CNT = 0;                // Reset counter
                          TIM2->EGR |= TIM_EGR_UG;      // Force update
                          TIM2->CR1 |= TIM_CR1_CEN;     // Restart Timer
```

```
                        previoustime=currenttime;
        }
  // external button to change the pattern
  if (GPIO_Pin == GPIO_PIN_0 && (currenttime-previoustime>debouncetime) ){
         external_button=!external_button;
          if (!external_button){
              counter=counter<<1;
          }
          if (external_button==1) {
              counter=counter>>1;
          }
         previoustime=currenttime;
        }
 }
```

Timer interrupt callback

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(htim);

  if (htim->Instance==TIM2){

              HAL_GPIO_WritePin(GPIOB, LD1_Pin, (counter&1));
              HAL_GPIO_WritePin(GPIOB, LD2_Pin, (counter&2));
              HAL_GPIO_WritePin(GPIOB, LD3_Pin, (counter&4));
              HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, (counter&8));
              HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, (counter&16));
          if (!external_button){
              counter=counter<<1;
          }
          if (external_button==1) {
              counter=counter>>1;
          }

              if (counter==0){
                      counter=16;
              }
              if (counter==32){
                      counter=1;
              }
  }
}

/* USER CODE END 4 */
```

Q2: Implementation of a digital clock with an alarm function using LCD display



An image of the digital clock

To use the LCD, the LCD driver files must be copied to the "Inc" (C header file) and "Src" (C file) folders of the current folder. The I2C functionality (I2C1 in our case) must also be turned on in the ".ioc" file to enable communication of the STM32 board with the LCD display. One must also make sure to keep the right slave address of the LCD display for I2C communication; it was "0x4E" in our case.

To make the clock tick, we used the timer interrupt functionality, which provided an interrupt every second. This interrupt was used to increment the seconds on the clock and, hence, the minutes and hours. The Prescalar (PSC) and AutoReload Register (ARR) were chosen in the following way:

$$Prescalar = 8400 - 1$$

$$ARR = 10000 - 1$$

$$Natural\ clock\ frequency\ of\ TIM2 = 84\ MHz$$

The timer interrupt function counts to 10000 with each clock pulse and then provides an interrupt. Hence, the interrupt occurs every:

$$\frac{8400}{84\ MHz} * 10^4 = 1\ second$$

Including the LCD driver header file in the main program.

```
/* USER CODE BEGIN Includes */
#include "i2c-lcd.h"
/* USER CODE END Includes */
```

Defining the relevant variables.

```
/* USER CODE BEGIN PV */
int mode = 0; //0 – time set mode; 1 - normal mode; 2 - alarm set mode
int cursor_loc = 0; //0 - set seconds; 1 - minutes; 2 - hrs
int tim_sec = 0;
int alarm_sec = 0;
int tim_min = 0;
int alarm_min = 0;
int tim_hrs = 0;
int alarm_hrs = 0;
int alarm_on_off = 0;
int debounce_time = 200;
int current_time = 0;
int previous_time = 0;
int alarm_detected = 0;
char time_array[9];
/* USER CODE END PV */
```

Function to display clock on the LCD display.

```
/* USER CODE BEGIN 0 */

void display_clock(void){

	//a function to indicate that time==alarm time when alarm_on_off ==
1

	if (mode!=2){
	lcd_put_cur(0, 0);
	sprintf(time_array, "%02d:%02d:%02d" , tim_hrs , tim_min, tim_sec);
	lcd_send_string(time_array);
	}
	if (mode==0){
		lcd_put_cur(1, 0);
		lcd_send_string("SET TIME  ");
	}
	if (mode==1 && alarm_detected==0){
		lcd_put_cur(1, 0);
		lcd_send_string("CLOCK    ");
		HAL_GPIO_WritePin(GPIOB, LD1_Pin, 0);
		HAL_GPIO_WritePin(GPIOB, LD2_Pin, 0);
		HAL_GPIO_WritePin(GPIOB, LD3_Pin, 0);
	}
	if (mode==2){
		lcd_put_cur(0, 0);
		sprintf(time_array, "%02d:%02d:%02d",alarm_hrs , alarm_min,
alarm_sec);
		lcd_send_string(time_array);
		lcd_put_cur(1, 0);
		lcd_send_string("SET ALARM");
```

```
        }
        if (alarm_on_off){
                lcd_put_cur(1, 13);
                lcd_send_string("ON ");
        }
        if (!alarm_on_off){
                lcd_put_cur(1, 13);
                lcd_send_string("OFF");
        }
        if (alarm_detected>0){
                lcd_put_cur(1, 0);
                lcd_send_string("ALARM!     ");
        }

}

/* USER CODE END 0 */
```

Initialization of the timer (TIM2) and I2C communication (I2C1).

```
/* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_I2C1_Init();
  MX_TIM2_Init();
```

Starting the timer, initializing and clearing the LCD display.

```
  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim2);
  lcd_init();
  lcd_clear();
  /* USER CODE END 2 */
```

Calling the display_clock function inside the while loop.

```
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
          display_clock();
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
```

Utilizing the interrupt provided by TIM2.

```
/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(htim);

  /* NOTE : This function should not be modified, when the callback is
needed,
            the HAL_TIM_PeriodElapsedCallback could be implemented in the
user file
   */
  if(alarm_on_off){
        if( (tim_sec == alarm_sec && tim_min == alarm_min && tim_hrs ==
alarm_hrs) | (alarm_detected>0)){
                alarm_detected = (alarm_detected +1)%30;
                     HAL_GPIO_TogglePin(GPIOB, LD1_Pin);
                     HAL_GPIO_TogglePin(GPIOB, LD2_Pin);
                     HAL_GPIO_TogglePin(GPIOB, LD3_Pin);

        }
  }

  if (htim->Instance==TIM2){
        if(1){
                tim_sec++;
                if(tim_sec == 60){
                        tim_min++;
                        tim_sec = 0;
                }
                if(tim_min == 60){
                        tim_hrs = (tim_hrs + 1)%24;
                        tim_min = 0;
                }
        }

   }

}
```

Handling the interrupt provided by the buttons.

```
//PIN15 for set button (cycle between setting cursor location to seconds,
minutes, hrs and alarm ON or OFF)
//PIN10 for increment button

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
 /* Prevent unused argument(s) compilation warning */
 UNUSED(GPIO_Pin);
```

```
 /* NOTE: This function Should not be modified, when the callback is
needed,
          the HAL_GPIO_EXTI_Callback could be implemented in the user file
  */
 current_time = HAL_GetTick();

 if(alarm_detected > 0){
        if(((GPIO_Pin == GPIO_PIN_13) |  (GPIO_Pin == GPIO_PIN_15) |
(GPIO_Pin == GPIO_PIN_10)) && (current_time - previous_time >
debounce_time) )
        alarm_detected = 0;
        previous_time = current_time;
 }

 if(GPIO_Pin == GPIO_PIN_13 && current_time - previous_time >
debounce_time){

        mode = (mode + 1)%3;
        cursor_loc = 0;
        previous_time = current_time;
 }

 if(GPIO_Pin == GPIO_PIN_15 && mode!=1 && current_time - previous_time >
debounce_time){
        if(mode == 0){
                cursor_loc = (cursor_loc+1)%3;
        }
        if(mode == 2){
                cursor_loc = (cursor_loc+1)%4;
        }
        previous_time = current_time;
 }

 if(GPIO_Pin == GPIO_PIN_10 && mode!=1 && current_time - previous_time >
debounce_time){
        if(mode == 0){
                if(cursor_loc == 0){
                        tim_sec = 0;
                }
                if(cursor_loc == 1){
                        tim_min = (tim_min+1)%60;
                }
                if(cursor_loc == 2){
                        tim_hrs = (tim_hrs+1)%24;
                }
        }
        if(mode == 2){
                if(cursor_loc == 0){
                                        alarm_sec = (alarm_sec+1)%60;
                                }
                if(cursor_loc == 1){
                        alarm_min = (alarm_min+1)%60;
                }
                if(cursor_loc == 2){
```

```
                        alarm_hrs = (alarm_hrs+1)%24;
                }
                if(cursor_loc == 3){
                        alarm_on_off = !alarm_on_off;
                }
        }
        previous_time = current_time;
 }


}
/* USER CODE END 4 */
```