Practical 3

Mattew Rothenburg
EEE3096S
University of Cape Town
South Africa
RTHMAT003

September 23, 2024

Abstract - **This report covers the implementation of practical 3 on the STM32 development board. A LED was blinked at a certain frequency using timer 6, this frequency could be switched between two value using a push-button. The push-button made use of hardware interrupts in order to switch frequencies. The ADC was used in order to read a potentiometer, and a PWM signal relative to the potentiometer was sent to a LED in order to dim it. Various value were written to the EEPROM using SPI and were then repeatedly fetched and displayed to the LCD.**

1 Introduction

There are several goals of this practical. The first is too blink an LED with a variable frequency, this frequency can be switched between two values through use of a push button. The second goal is too dim a second LED in accordance with a potentiometer. The final goal of this practical is too communicate with the on-board EEPROM over SPI, this is done too store and load values. These value are then to be displayed on the LCD

2 Method

2.1 Software Implementation

In order to blink the on board LED at a certain frequency TIM6 was used. The ARR and prescaler were both configured to result in an interrupt being triggered at the frequency of 2Hz. During the interrupt the LED is toggled, thus causing the LED to flash with a frequency of 2Hz.

In order to change the frequency of the pulsing LED a push button was used. This push-button was configured in interrupt mode. Upon the push button being pressed the interrupt service routine would be called, during this service routine TIM6 was paused its prescaler was changed to result in a frequency of 1 Hz and then restarted. This cause the pulsing LEDs frequency to change as well. Upon multiple presses the frequncy would toggle between 1 Hz and 2 Hz. The push button was also debounced using HAL getTick() to stop interrupts caused by component tolerances.

The potentiometer was read through the ADC. TIM3 was setup to send a PWM signal to a LED. This PWM signal dims the LED in relation to the potentiometers reading. The STM32 has a 12 bit ADC while TIM3 is setup in 16 bit mode. This causes issues as the potentiometer readings range from 0 - 4095 and TIM3 using values ranging from 0 - 4799. To solve this a formula was used map potentiomter to TIM3.

$$PWM = \frac{ADC \times 47999}{4096}$$

Using the on board EEPROM allows for value to be retained even between power cycles. Values were stored and retrieved through the use of the SPI communication protocol. Several different binary values were stored on the EEPROM. These values were then fetched one by one, converted to decimal numbers and then displayed to the LCD screen. This was done continuously, producing a pattern of displayed number on the screen.

3 Conclusion

This practical successfully demonstrated the use of various peripherals on the STM32 development board, including timers, ADC, PWM, and SPI communication. Each of the requirements laid out were completed successfully.

```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdio.h>
#include "stm32f0xx.h"
#include <lcd_stm32f0.c>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

// Definitions for SPI usage
#define MEM_SIZE 8192 // bytes
#define WREN 0b00000110 // enable writing
#define WRDI 0b00000100 // disable writing
#define RDSR 0b00000101 // read status register
#define WRSR 0b00000001 // write status register
#define READ 0b00000011
#define WRITE 0b00000010
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc;

TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim6;
TIM_HandleTypeDef htim16;

/* USER CODE BEGIN PV */

// TODO: Define input variables
uint8_t LED7_Freq = 0;
uint8_t debounceDelay = 100;
int lastTick = 0;
int currentTick = 0;
uint16_t currentAddress = 0;

uint8_t bitVals[6] = {
  0b10101010,
  0b01010101,
  0b11001100,
  0b00110011,
  0b11110000,
  0b00001111
```

```
};


/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM16_Init(void);
static void MX_TIM6_Init(void);
/* USER CODE BEGIN PFP */
void EXTI0_1_IRQHandler(void);
void TIM16_IRQHandler(void);
void writeLCD(char *char_in);

// ADC functions
uint32_t pollADC(void);
uint32_t ADCtoCCR(uint32_t adc_val);

// SPI functions
static void init_spi(void);
static void write_to_address(uint16_t address, uint8_t data);
static uint8_t read_from_address(uint16_t address);
static void spi_delay(uint32_t delay_in_us);
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 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 */
  init_spi();
  MX_GPIO_Init();
  MX_ADC_Init();
  MX_TIM3_Init();
  MX_TIM16_Init();
  MX_TIM6_Init();
  /* USER CODE BEGIN 2 */

  // Initialise LCD
  init_LCD();
  lcd_command(TWOLINE_MODE);
  lcd_putstring("EEPROM byte:");

  // Start timers
  HAL_TIM_Base_Start_IT(&htim6);
  HAL_TIM_Base_Start_IT(&htim16);

```

```
  // PWM setup
  uint32_t CCR = 0;
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3); // Start PWM on TIM3 Channel 3

  // TODO: Write all bytes to EEPROM using "write_to_address"
  uint16_t startAddress = 0;
  for(int i = 0; i < 6; i++){
    write_to_address(startAddress + i, bitVals[i]);
  }

  /* USER CODE END 2 */

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

  // TODO: Poll ADC
  uint32_t adcVal = pollADC();
  // TODO: Get CRR
  uint32_t brightness = ADCtoCCR(adcVal);

  // Update PWM value
  __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_3, brightness);

    /* USER CODE END WHILE */

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

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  LL_FLASH_SetLatency(LL_FLASH_LATENCY_0);
  while(LL_FLASH_GetLatency() != LL_FLASH_LATENCY_0)
  {
  }
  LL_RCC_HSI_Enable();

   /* Wait till HSI is ready */
  while(LL_RCC_HSI_IsReady() != 1)
  {

  }
  LL_RCC_HSI_SetCalibTrimming(16);
  LL_RCC_HSI14_Enable();

   /* Wait till HSI14 is ready */
  while(LL_RCC_HSI14_IsReady() != 1)
  {

  }
  LL_RCC_HSI14_SetCalibTrimming(16);
  LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
  LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
  LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_HSI);

   /* Wait till System clock is ready */
  while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_HSI)
  {

  }
  LL_SetSystemCoreClock(8000000);

   /* Update the time base */
  if (HAL_InitTick (TICK_INT_PRIORITY) != HAL_OK)
  {
    Error_Handler();
  }
  LL_RCC_HSI14_EnableADCControl();
```

```
}

/**
  * @brief ADC Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC_Init(void)
{

  /* USER CODE BEGIN ADC_Init 0 */
  /* USER CODE END ADC_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC_Init 1 */

  /* USER CODE END ADC_Init 1 */

  /** Configure the global features of the ADC (Clock, Resolution, Data Alignment
      and number of conversion)
  */
  hadc.Instance = ADC1;
  hadc.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
  hadc.Init.Resolution = ADC_RESOLUTION_12B;
  hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc.Init.ScanConvMode = ADC_SCAN_DIRECTION_FORWARD;
  hadc.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  hadc.Init.LowPowerAutoWait = DISABLE;
  hadc.Init.LowPowerAutoPowerOff = DISABLE;
  hadc.Init.ContinuousConvMode = DISABLE;
  hadc.Init.DiscontinuousConvMode = DISABLE;
  hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc.Init.DMAContinuousRequests = DISABLE;
  hadc.Init.Overrun = ADC_OVR_DATA_PRESERVED;
  if (HAL_ADC_Init(&hadc) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure for the selected ADC regular channel to be converted.
  */
  sConfig.Channel = ADC_CHANNEL_6;
  sConfig.Rank = ADC_RANK_CHANNEL_NUMBER;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC_Init 2 */
  ADC1->CR |= ADC_CR_ADCAL;
  while(ADC1->CR & ADC_CR_ADCAL);     // Calibrate the ADC
  ADC1->CR |= (1 << 0);           // Enable ADC
  while((ADC1->ISR & (1 << 0)) == 0);   // Wait for ADC ready
  /* USER CODE END ADC_Init 2 */

}

/**
  * @brief TIM3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM3_Init(void)
{

  /* USER CODE BEGIN TIM3_Init 0 */

  /* USER CODE END TIM3_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

```

```
  /* USER CODE BEGIN TIM3_Init 1 */

  /* USER CODE END TIM3_Init 1 */
  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 0;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 47999;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM3_Init 2 */

  /* USER CODE END TIM3_Init 2 */
  HAL_TIM_MspPostInit(&htim3);

}

/**
  * @brief TIM6 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM6_Init(void)
{

  /* USER CODE BEGIN TIM6_Init 0 */

  /* USER CODE END TIM6_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM6_Init 1 */

  /* USER CODE END TIM6_Init 1 */
  htim6.Instance = TIM6;
  htim6.Init.Prescaler = 8000-1;
  htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim6.Init.Period = 500-1;
  htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
```

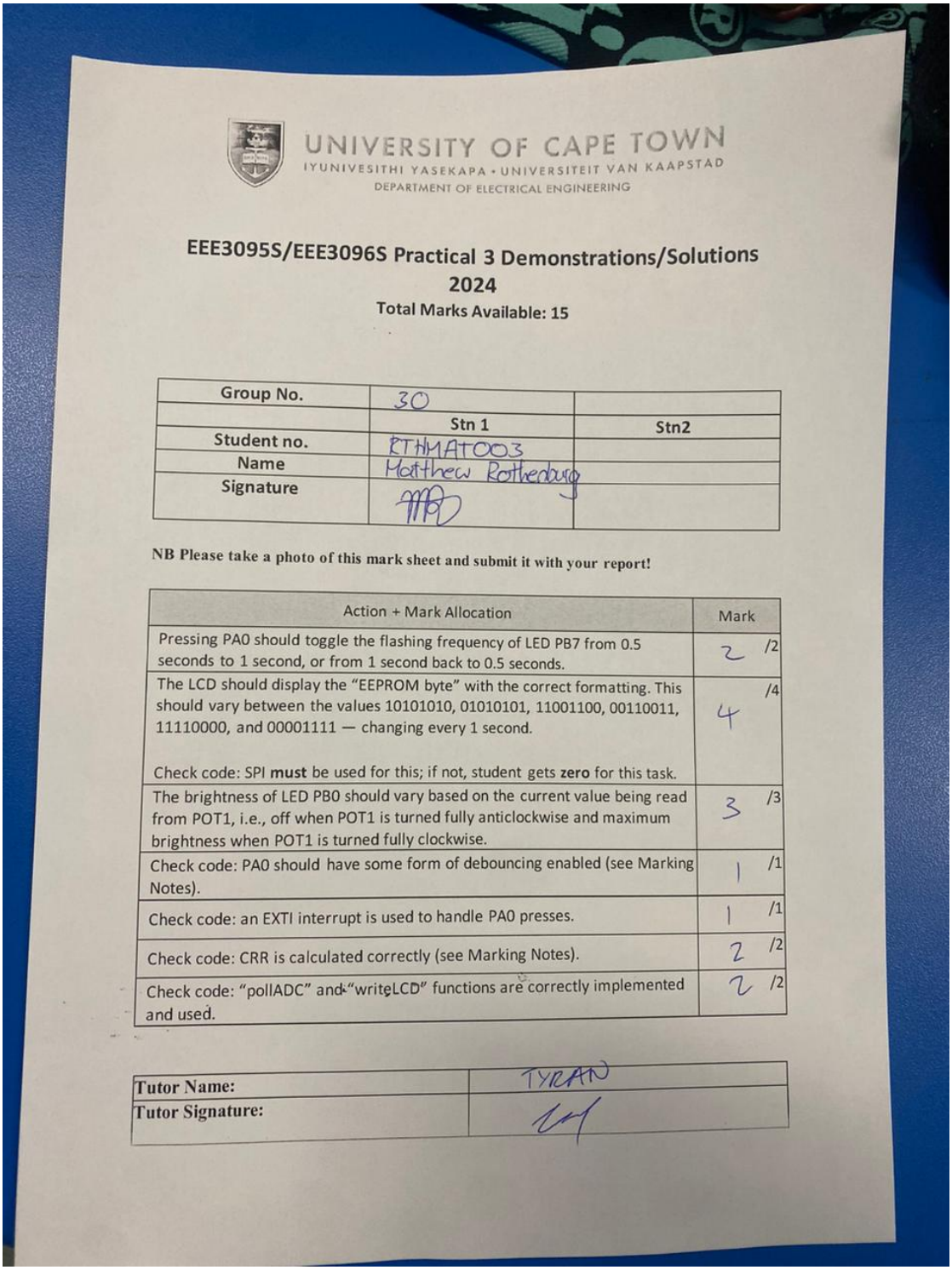
```
  /* USER CODE BEGIN TIM6_Init 2 */
  NVIC_EnableIRQ(TIM6_IRQn);
  /* USER CODE END TIM6_Init 2 */

}

/**
  * @brief TIM16 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM16_Init(void)
{

  /* USER CODE BEGIN TIM16_Init 0 */

  /* USER CODE END TIM16_Init 0 */

  /* USER CODE BEGIN TIM16_Init 1 */

  /* USER CODE END TIM16_Init 1 */
  htim16.Instance = TIM16;
  htim16.Init.Prescaler = 8000-1;
  htim16.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim16.Init.Period = 1000-1;
  htim16.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim16.Init.RepetitionCounter = 0;
  htim16.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_Base_Init(&htim16) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM16_Init 2 */
  NVIC_EnableIRQ(TIM16_IRQn);
  /* USER CODE END TIM16_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  LL_EXTI_InitTypeDef EXTI_InitStruct = {0};
  LL_GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOF);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);

  /**/
  LL_GPIO_ResetOutputPin(LED7_GPIO_Port, LED7_Pin);

  /**/
  LL_SYSCFG_SetEXTISource(LL_SYSCFG_EXTI_PORTA, LL_SYSCFG_EXTI_LINE0);

  /**/
  LL_GPIO_SetPinPull(Button0_GPIO_Port, Button0_Pin, LL_GPIO_PULL_UP);

  /**/
  LL_GPIO_SetPinMode(Button0_GPIO_Port, Button0_Pin, LL_GPIO_MODE_INPUT);

  /**/
  EXTI_InitStruct.Line_0_31 = LL_EXTI_LINE_0;
  EXTI_InitStruct.LineCommand = ENABLE;
  EXTI_InitStruct.Mode = LL_EXTI_MODE_IT;
  EXTI_InitStruct.Trigger = LL_EXTI_TRIGGER_RISING;
  LL_EXTI_Init(&EXTI_InitStruct);

  /**/
```

```
  GPIO_InitStruct.Pin = LED7_Pin;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  LL_GPIO_Init(LED7_GPIO_Port, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
  HAL_NVIC_SetPriority(EXTI0_1_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI0_1_IRQn);
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
void EXTI0_1_IRQHandler(void)
{
  // TODO: Add code to switch LED7 delay frequency
  currentTick = HAL_GetTick();
  if(currentTick - lastTick > 150){
    lastTick = currentTick;
    HAL_TIM_Base_Stop(&htim6);
    if(LED7_Freq%2 == 0){
      htim6.Init.Period = 1000-1;
    }
    else{
      htim6.Init.Period = 500-1;
    }
    HAL_TIM_Base_Init(&htim6);
    HAL_TIM_Base_Start(&htim6);
    LED7_Freq++;
  }
  HAL_GPIO_EXTI_IRQHandler(Button0_Pin); // Clear interrupt flags
}

void TIM6_IRQHandler(void)
{
  // Acknowledge interrupt
  HAL_TIM_IRQHandler(&htim6);

  // Toggle LED7
  HAL_GPIO_TogglePin(GPIOB, LED7_Pin);
}

void TIM16_IRQHandler(void)
{
  // Acknowledge interrupt
  HAL_TIM_IRQHandler(&htim16);

  // TODO: Initialise a string to output second line on LCD
  uint8_t number = read_from_address(currentAddress);
  char stringToWrite[3]; // Ensure this buffer is large enough for the largest
      expected number
  sprintf(stringToWrite, "%d", number);
  lcd_command(0xC0);
  for (int i = 0; i < 3; i++) { // Assuming a 16x2 LCD
    lcd_putchar(' '); // Write a space character
  }
  if(number != bitVals[currentAddress]){
    lcd_command(0xC0);
    lcd_putstring("SPI ERROR!");
  }
  else{
    writeLCD(stringToWrite);
    // TODO: Change LED pattern; output 0x01 if the read SPI data is incorrect
  }
  currentAddress ++;
  currentAddress =  currentAddress % 6;
}

// TODO: Complete the writeLCD function
void writeLCD(char *char_in){
  lcd_command(0xC0); // 0xC0 is the command to set the cursor to the beginning of
      Line 2
  // Write each character of the string to the LCD
  while (*char_in) {
```

```
522       lcd_putchar(*char_in); // Write each character to LCD
523       char_in++;
524   }
525   delay(3000);
526 }
527
528 // Get ADC value
529 uint32_t pollADC(void){
530   HAL_ADC_Start(&hadc); // start the adc
531   HAL_ADC_PollForConversion(&hadc, 100); // poll for conversion
532   uint32_t val = HAL_ADC_GetValue(&hadc); // get the adc value
533   HAL_ADC_Stop(&hadc); // stop adc
534   return val;
535 }
536
537 // Calculate PWM CCR value
538 uint32_t ADCtoCCR(uint32_t adc_val){
539   // TODO: Calculate CCR value (val) using an appropriate equation
540   uint32_t brightness = (adc_val * 47999) / 4096;
541   //return val;
542   return brightness;
543 }
544
545 void ADC1_COMP_IRQHandler(void)
546 {
547   //adc_val = HAL_ADC_GetValue(&hadc); // read adc value
548   HAL_ADC_IRQHandler(&hadc); //Clear flags
549 }
550
551 // Initialise SPI
552 static void init_spi(void) {
553
554   // Clock to PB
555   RCC->AHBENR |= RCC_AHBENR_GPIOBEN;  // Enable clock for SPI port
556
557   // Set pin modes
558   GPIOB->MODER |= GPIO_MODER_MODER13_1; // Set pin SCK (PB13) to Alternate
          Function
559   GPIOB->MODER |= GPIO_MODER_MODER14_1; // Set pin MISO (PB14) to Alternate
          Function
560   GPIOB->MODER |= GPIO_MODER_MODER15_1; // Set pin MOSI (PB15) to Alternate
          Function
561   GPIOB->MODER |= GPIO_MODER_MODER12_0; // Set pin CS (PB12) to output push-pull
562   GPIOB->BSRR |= GPIO_BSRR_BS_12;     // Pull CS high
563
564   // Clock enable to SPI
565   RCC->APB1ENR |= RCC_APB1ENR_SPI2EN;
566   SPI2->CR1 |= SPI_CR1_BIDIOE;                    // Enable output
567   SPI2->CR1 |= (SPI_CR1_BR_0 |  SPI_CR1_BR_1);          // Set Baud to fpclk / 16
568   SPI2->CR1 |= SPI_CR1_MSTR;                   // Set to master mode
569   SPI2->CR2 |= SPI_CR2_FRXTH;                    // Set RX threshold to be 8 bits
570   SPI2->CR2 |= SPI_CR2_SSOE;                   // Enable slave output to work in
          master mode
571   SPI2->CR2 |= (SPI_CR2_DS_0 | SPI_CR2_DS_1 | SPI_CR2_DS_2);  // Set to 8-bit mode
572   SPI2->CR1 |= SPI_CR1_SPE;                    // Enable the SPI peripheral
573 }
574
575 // Implements a delay in microseconds
576 static void spi_delay(uint32_t delay_in_us) {
577   volatile uint32_t counter = 0;
578   delay_in_us *= 3;
579   for(; counter < delay_in_us; counter++) {
580     __asm("nop");
581     __asm("nop");
582   }
583 }
584
585 // Write to EEPROM address using SPI
586 static void write_to_address(uint16_t address, uint8_t data) {
587
588   uint8_t dummy; // Junk from the DR
589
590   // Set the Write Enable latch
591   GPIOB->BSRR |= GPIO_BSRR_BR_12; // Pull CS low
592   spi_delay(1);
```

```
  *((uint8_t*)(&SPI2->DR)) = WREN;
  while ((SPI2->SR & SPI_SR_RXNE) == 0); // Hang while RX is empty
  dummy = SPI2->DR;
  GPIOB->BSRR |= GPIO_BSRR_BS_12; // Pull CS high
  spi_delay(5000);

  // Send write instruction
  GPIOB->BSRR |= GPIO_BSRR_BR_12;       // Pull CS low
  spi_delay(1);
  *((uint8_t*)(&SPI2->DR)) = WRITE;
  while ((SPI2->SR & SPI_SR_RXNE) == 0);    // Hang while RX is empty
  dummy = SPI2->DR;

  // Send 16-bit address
  *((uint8_t*)(&SPI2->DR)) = (address >> 8);  // Address MSB
  while ((SPI2->SR & SPI_SR_RXNE) == 0);    // Hang while RX is empty
  dummy = SPI2->DR;
  *((uint8_t*)(&SPI2->DR)) = (address);     // Address LSB
  while ((SPI2->SR & SPI_SR_RXNE) == 0);    // Hang while RX is empty
  dummy = SPI2->DR;

  // Send the data
  *((uint8_t*)(&SPI2->DR)) = data;
  while ((SPI2->SR & SPI_SR_RXNE) == 0); // Hang while RX is empty
  dummy = SPI2->DR;
  GPIOB->BSRR |= GPIO_BSRR_BS_12; // Pull CS high
  spi_delay(5000);
}

// Read from EEPROM address using SPI
static uint8_t read_from_address(uint16_t address) {

  uint8_t dummy; // Junk from the DR

  // Send the read instruction
  GPIOB->BSRR |= GPIO_BSRR_BR_12;       // Pull CS low
  spi_delay(1);
  *((uint8_t*)(&SPI2->DR)) = READ;
  while ((SPI2->SR & SPI_SR_RXNE) == 0);    // Hang while RX is empty
  dummy = SPI2->DR;

  // Send 16-bit address
  *((uint8_t*)(&SPI2->DR)) = (address >> 8);  // Address MSB
  while ((SPI2->SR & SPI_SR_RXNE) == 0);    // Hang while RX is empty
  dummy = SPI2->DR;
  *((uint8_t*)(&SPI2->DR)) = (address);     // Address LSB
  while ((SPI2->SR & SPI_SR_RXNE) == 0);    // Hang while RX is empty
  dummy = SPI2->DR;

  // Clock in the data
  *((uint8_t*)(&SPI2->DR)) = 0x42;      // Clock out some junk data
  while ((SPI2->SR & SPI_SR_RXNE) == 0);    // Hang while RX is empty
  dummy = SPI2->DR;
  GPIOB->BSRR |= GPIO_BSRR_BS_12;       // Pull CS high
  spi_delay(5000);

  return dummy;               // Return read data
}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}
```

```
668 #ifdef  USE_FULL_ASSERT
669 /**
670   * @brief  Reports the name of the source file and the source line number
671   *         where the assert_param error has occurred.
672   * @param  file: pointer to the source file name
673   * @param  line: assert_param error line source number
674   * @retval None
675   */
676 void assert_failed(uint8_t *file, uint32_t line)
677 {
678   /* USER CODE BEGIN 6 */
679   /* User can add his own implementation to report the file name and line number,
680      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
681   /* USER CODE END 6 */
682 }
683 #endif /* USE_FULL_ASSERT */
```



UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD
DEPARTMENT OF ELECTRICAL ENGINEERING

EEE3095S/EEE3096S Practical 3 Demonstrations/Solutions 2024

Total Marks Available: 15

Group No.	30	
	Stn 1	Stn2
Student no.	RTHMAT003	
Name	Matthew Rothenburg	
Signature		

NB Please take a photo of this mark sheet and submit it with your report!

Action + Mark Allocation	Mark
Pressing PA0 should toggle the flashing frequency of LED PB7 from 0.5 seconds to 1 second, or from 1 second back to 0.5 seconds.	2 /2
The LCD should display the "EEPROM byte" with the correct formatting. This should vary between the values 10101010, 01010101, 11001100, 00110011, 11110000, and 00001111 — changing every 1 second. Check code: SPI **must** be used for this; if not, student gets **zero** for this task.	4 /4
The brightness of LED PB0 should vary based on the current value being read from POT1, i.e., off when POT1 is turned fully anticlockwise and maximum brightness when POT1 is turned fully clockwise.	3 /3
Check code: PA0 should have some form of debouncing enabled (see Marking Notes).	1 /1
Check code: an EXTI interrupt is used to handle PA0 presses.	1 /1
Check code: CRR is calculated correctly (see Marking Notes).	2 /2
Check code: "pollADC" and "writeLCD" functions are correctly implemented and used.	2 /2

Tutor Name:	TYRAN
Tutor Signature:	