Practical 3

Abdul-Mateen Kader[†] and Chris Scheepers[‡]
EEE3096S Class of 2024
University of Cape Town
South Africa
[†]KDRABD004 [‡]SCHCHR077

***Abstract*—This practical employs the STM32 microcontroller's Analog-to-Digital Converter (ADC), Pulse Width Modulation (PWM), pushbutton interrupts, and Serial Peripheral Interface (SPI) to communicate with a variety of components. The key functions of this practical include reading an analogue voltage from a potentiometer to alter the duty cycle of an LED's PWM signal, toggling another LED's frequency with a pushbutton interrupt, and handling EEPROM data via SPI. The practical also includes presenting data on an LCD and handling errors during SPI communication.**

I. Introduction

This practical investigates the use of the STM32 microcontroller to interface analogue and digital components. The ADC will be used to read the voltage of a potentiometer and alter the brightness of LED D0 using PWM. A pushbutton interrupt changes the frequency of LED D7. In addition, SPI will be used to write to and read from an EEPROM, and the results will be outputted on the LCD. The aim of this practical is to learn about:

- PWM
- ADCs
- SPI
- Pushbutton interrupts

II. Methodology

This section will describe how the practical was conducted by referring to hardware, implementation, and experimental procedure.

A. Hardware

The hardware items used included:

- UCT STM32F051C6

B. Implementation

1) Pushbutton Interrupt and Debouncing: An EXTI interrupt was implemented to change the frequency of LED D7 when the SW0 was pressed. Debouncing was also implemented to ensure that no false signals are generated when pressing SW0.

```
void EXTI0_1_IRQHandler(void)
{
  // TODO: Add code to switch LED7 delay frequency
  static uint32_t lastDebounceTime = 0; // To store the last debounce time
  uint32_t debounceDelay = 200;

  // Check if enough time has passed since the last debounce
  if (HAL_GetTick() - lastDebounceTime > debounceDelay)
  {
    if (HAL_GPIO_ReadPin(GPIOA, Button0_Pin) == GPIO_PIN_RESET)
    {
      // Toggle interval based on the current value
      if (toggleInterval == 1000)
      {
        toggleInterval = 500; // Change toggle interval to 2Hz
      }
      else if (toggleInterval == 500)
      {
        toggleInterval = 1000; // Change toggle interval to 1Hz
      }
      // Update the timer period
      UpdateTimer16Period(toggleInterval);
      // Update the last debounce time
      lastDebounceTime = HAL_GetTick();
    }
  }

  HAL_GPIO_EXTI_IRQHandler(Button0_Pin); // Clear interrupt flags
}
```

2) Varying LED Brightness: The function *pollADC* is used to read the ADC value from the potentiometer, which is then used in the *ADCtoCCR* function which then updates the CCR value which sets the PWM duty cycle, allowing the potentiometer to vary the brightness of D0. The code is shown below.

```
uint32_t pollADC(void)
{
  HAL_ADC_Start(&hadc);                          // Start the ADC
  HAL_ADC_PollForConversion(&hadc, HAL_MAX_DELAY); // Wait for conversion to finish
  uint32_t val = HAL_ADC_GetValue(&hadc);        // Get the converted value
  HAL_ADC_Stop(&hadc);                           // Stop the ADC
  return val;
}
```

```
uint32_t ADCtoCCR(uint32_t adc_val)
{
  // TODO: Calculate CCR value (val) using an appropriate equation
  uint32_t CCR_val = (adc_val * (htim3.Init.Period + 1)) / 4096;
  return CCR_val;
}
```

3) SPI Interaction with EEPROM: The template given to us already had the SPI initialised. The next step was to create an array of 8-bit integers that could hold 6 binary values and then use the write_to_address() function to send these values to EEPROM via the SPI.

```
    // TODO: Define input variables
    uint8_t eepromData[6] = {0b10101010, 0b01010101, 0b11001100,
    0b00110011, 0b11110000, 0b00001111}; // initialise 8-bit array for EEPROM
    ...
     // TODO: Write all bytes to EEPROM using "write_to_address"
    for (uint16_t address = 0; address < 6; address++)
    {
        write_to_address(address, eepromData[address]); // Writing pattern to EEPROM
    }

    ...
```

The interrupt timer given to via the template already had a one-second timer that was used to implement the read_from_address() function to read these values one at a time from EEPROM via the SPI within the TIM16_IRQHandler(void) function. An address counter to keep count of the current address in EEPROM was used to compare with the expected result from the original array of data for SPI error checking. The values from the read_from_address() function were then sent to a buffer and written into the LCD as decimal values using the writeLCD() function.

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

  // TODO: Initialise a string to output second line on LCD
  uint8_t readValue = 0;

  // Read from EEPROM
  readValue = read_from_address(currentAddress);

  // Display value on LCD
  char buffer[17];
  sprintf(buffer, "%d", readValue);
  writeLCD(buffer);

  // TODO: Change LED pattern; output 0x01 if the read SPI data is incorrect
  if (readValue != eepromData[currentAddress]) // SPI error check
  {
    writeLCD("SPI ERROR!");
    HAL_GPIO_WritePin(GPIOB, LED7_Pin, GPIO_PIN_SET); // Set LED pattern
  }

  // Increment address, wrap around after reaching array size
  currentAddress = (currentAddress + 1) % 6;
}
```

The writeLCD() function was then completed.

```
    // TODO: Complete the writeLCD function
void writeLCD(char *char_in)
{
  lcd_command(CLEAR); // Clear LCD screen
  lcd_putstring("EEPROM byte:");
  lcd_command(LINE_TWO);
  lcd_putstring(char_in);
}
```

III. CONCLUSION

The results from our practical demonstration show that our methods and implementation of this practical were successful. Our demonstration met the criteria of the practical as we created a working LED dimmer using a potentiometer. We successfully integrated SPI with EEPROM to display values via the LCD every second and finally, we were able to change the frequency of LED7 via the push of a button using interrupts and debouncing. The final implementation of our solution worked well and we were pleased with our debouncing effects and use of interrupts for the pushbutton functionality. The practical marking sheet can be found in the appendix.

The use of the LCD did not seem like much of the learning objective that it could have been, so an improvement to this practical could be to implement a harder challenge or more interesting use of LCD functionality. Possibly utilising CGGRAM for custom characters.

Appendix

GitHub Link

```
/* USER CODE BEGIN Header */
/**
 ******************************************************************************
 * @file           : main.c
 * @brief          : Main program body
 * @authors        : Abdul-Mateen Kader, Chris Scheepers
 ******************************************************************************
 * @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>
#include "string.h"
/* 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
// initialise 8-bit array for EEPROM
uint8_t eepromData[6] = {0b10101010, 0b01010101, 0b11001100, 0b00110011, 0b11110000, 0b00001111};
uint32_t adc_val = 0;
uint32_t toggleInterval = 500; // Initial toggle interval for LED7 (2Hz)
uint16_t currentAddress = 0;   // EEPROM Address counter
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
```

```c
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 TIM6_IRQHandler(void);
void writeLCD(char *char_in);
//Added function to update Timer Period
void UpdateTimer16Period(uint32_t period);

// 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();

  // 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

```

```c
  // TODO: Write all bytes to EEPROM using "write_to_address"
  for (uint16_t address = 0; address < 6; address++)
  {
    write_to_address(address, eepromData[address]); // Writing pattern to EEPROM
  }

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */

  while (1)
  {

    // TODO: Poll ADC
    uint32_t adc_val = pollADC();

    // TODO: Get CRR
    CCR = ADCtoCCR(adc_val);

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

    /* 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
```

```c
 * @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 */
```

```c
  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 */
```

```c
}

/**
 * @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;
```

```c
  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
  static uint32_t lastDebounceTime = 0; // To store the last debounce time
  uint32_t debounceDelay = 200;

  // Check if enough time has passed since the last debounce
  if (HAL_GetTick() - lastDebounceTime > debounceDelay)
  {
    if (HAL_GPIO_ReadPin(GPIOA, Button0_Pin) == GPIO_PIN_RESET)
    {
      // Toggle interval based on the current value
      if (toggleInterval == 1000)
      {
        toggleInterval = 500; // Change toggle interval to 2Hz
      }
      else if (toggleInterval == 500)
      {
        toggleInterval = 1000; // Change toggle interval to 1Hz
      }
      // Update the timer period
      UpdateTimer16Period(toggleInterval);
      // Update the last debounce time
      lastDebounceTime = HAL_GetTick();
    }
  }

  HAL_GPIO_EXTI_IRQHandler(Button0_Pin); // Clear interrupt flags
}

void UpdateTimer16Period(uint32_t period)
{
    __HAL_TIM_SET_AUTORELOAD(&htim6, period - 1); // Set the timer period
    HAL_TIM_Base_Start(&htim6); // Restart the timer
}

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 readValue = 0;

  // Read from EEPROM
  readValue = read_from_address(currentAddress);

  // Display value on LCD
  char buffer[17];
  sprintf(buffer, "%d", readValue);
  writeLCD(buffer);

  // TODO: Change LED pattern; output 0x01 if the read SPI data is incorrect
  if (readValue != eepromData[currentAddress]) // SPI error check
  {
```

```c
    writeLCD("SPI ERROR!");
    HAL_GPIO_WritePin(GPIOB, LED7_Pin, GPIO_PIN_SET); // Set LED pattern
  }

  // Increment address, wrap around after reaching array size
  currentAddress = (currentAddress + 1) % 6;
}

// TODO: Complete the writeLCD function
void writeLCD(char *char_in)
{
  lcd_command(CLEAR); // Clear LCD screen
  lcd_putstring("EEPROM byte:");
  lcd_command(LINE_TWO);
  lcd_putstring(char_in);
}

// Get ADC value
uint32_t pollADC(void)
{
  HAL_ADC_Start(&hadc);                            // Start the ADC
  HAL_ADC_PollForConversion(&hadc, HAL_MAX_DELAY); // Wait for conversion to finish
  uint32_t val = HAL_ADC_GetValue(&hadc);          // Get the converted value
  HAL_ADC_Stop(&hadc);                             // Stop the ADC
  return val;
}

// Calculate PWM CCR value
uint32_t ADCtoCCR(uint32_t adc_val)
{
  // TODO: Calculate CCR value (val) using an appropriate equation
  uint32_t CCR_val = (adc_val * (htim3.Init.Period + 1)) / 4096;
  return CCR_val;
}

void ADC1_COMP_IRQHandler(void)
{
  adc_val = HAL_ADC_GetValue(&hadc); // read adc value
  HAL_ADC_IRQHandler(&hadc);         // Clear flags
}

// Initialise SPI
static void init_spi(void)
{

  // Clock to PB
  RCC->AHBENR |= RCC_AHBENR_GPIOBEN; // Enable clock for SPI port

  // Set pin modes
  GPIOB->MODER |= GPIO_MODER_MODER13_1; // Set pin SCK (PB13) to Alternate Function
  GPIOB->MODER |= GPIO_MODER_MODER14_1; // Set pin MISO (PB14) to Alternate Function
  GPIOB->MODER |= GPIO_MODER_MODER15_1; // Set pin MOSI (PB15) to Alternate Function
  GPIOB->MODER |= GPIO_MODER_MODER12_0; // Set pin CS (PB12) to output push-pull
  GPIOB->BSRR |= GPIO_BSRR_BS_12;       // Pull CS high

  // Clock enable to SPI
  RCC->APB1ENR |= RCC_APB1ENR_SPI2EN;
  SPI2->CR1 |= SPI_CR1_BIDIOE;                                 // Enable output
  SPI2->CR1 |= (SPI_CR1_BR_0 | SPI_CR1_BR_1);                  // Set Baud to fpclk / 16
  SPI2->CR1 |= SPI_CR1_MSTR;                                   // Set to master mode
  SPI2->CR2 |= SPI_CR2_FRXTH;                                  // Set RX threshold to be 8 bits
  SPI2->CR2 |= SPI_CR2_SSOE;                                   // Enable slave output to work in master mode
  SPI2->CR2 |= (SPI_CR2_DS_0 | SPI_CR2_DS_1 | SPI_CR2_DS_2); // Set to 8-bit mode
  SPI2->CR1 |= SPI_CR1_SPE;                                    // Enable the SPI peripheral
}

// Implements a delay in microseconds
static void spi_delay(uint32_t delay_in_us)
{
  volatile uint32_t counter = 0;
  delay_in_us *= 3;
  for (; counter < delay_in_us; counter++)
  {
    __asm("nop");
```

```
    __asm("nop");
  }
}

// Write to EEPROM address using SPI
static void write_to_address(uint16_t address, uint8_t data)
{

  uint8_t dummy; // Junk from the DR

  // Set the Write Enable latch
  GPIOB->BSRR |= GPIO_BSRR_BR_12; // Pull CS low
  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)
```

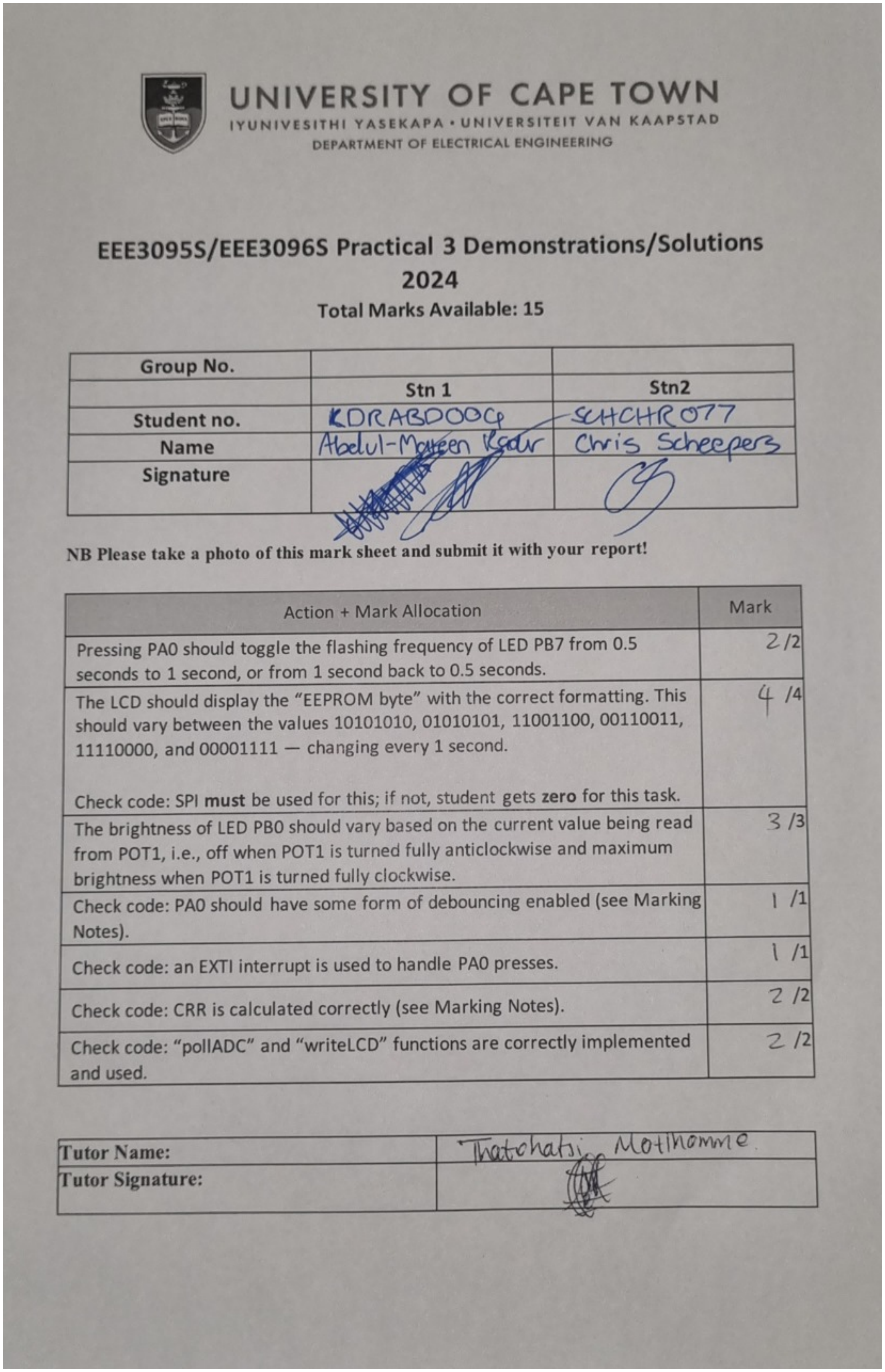
```c
    ; // Hang while RX is empty
  dummy = SPI2->DR;
  GPIOB->BSRR |= GPIO_BSRR_BS_12; // Pull CS high
  spi_delay(5000);

  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 */
}

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

Listing 1: main.c Prac3



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.		
	Stn 1	**Stn2**
Student no.	KDRABD009	SCHCHR077
Name	Abdul-Mateen Kadir	Chris Scheepers
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:	Thatchatsi Motlhomme
Tutor Signature:	

Fig. 1: Practical 3 Marking Sheet