Implementation of ADC, PWM, and SPI Communication on STM32 Microcontroller for Embedded Systems Control

Georgene de Wet[†] and Arend JAcques du Preez[‡]
EEE3096S **Group 36**
University of Cape Town
South Africa
[†]DPRARE001 [‡]DWTGEO002

***Abstract*—This practical demonstrates the use of an Analog-to-Digital Converter (ADC), with Pulse Width Modulation (PWM), and Serial Peripheral Interface (SPI) on an STM32 microcontroller. The objectives of this lab includes reading the STM's potentiometers voltage using ADC, by controlling an LED's brightness through PWM, and finally utilizing SPI to write to and read from an EEPROM. Furthermore, data is displayed on an LCD, to showcase the integration of these embedded system components.**

I. Introduction

Embedded systems often require interaction with analogue devices such as sensors and user interfaces like potentiometers. To convert a potentiometers analogue signal into digital data, an Analogue-to-Digital Converter (ADC) is used. To control device parameters like LED brightness Pulse Width Modulation (PWM) is used. Furthermore, communication with memory devices like EEPROMs (which is a non-volatile form of memory) is achieved using a Serial Peripheral Interface (SPI). This practical focuses on combining these features on the STM32 microcontroller to control LEDs, read their analogue values, and display the data on an LCD.

The source code for this project can be found on GitHub: EmbeddedSystems$_{G}roup$36.

[1]

II. Methodology

A. Hardware Setup

To implement the Analog-to-Digital Converter (ADC) and Pulse Width Modulation (PWM) on the STM32 microcontroller [2], the following hardware components were used:

- **STM32 Development Board**: The CPU responsible for handling the ADC, PWM signal generation, and SPI communication.
- **Potentiometer**: Analogue input used to generate a variable voltage for the ADC.
- **LEDs (PB0 and PB7)**: PB0 was used for PWM-based brightness control, while PB7 toggled at variable frequencies.
- **Pushbutton (PA0)**: Configured to generate an interrupt to toggle the frequency of LED PB7.
- **EEPROM**: Used to store and retrieve data using SPI communication as an external memory module .
- **LCD Display**: To display the data read from the EEPROM.

B. Pseudocode

The following pseudocode was used to conceptualize what needed to be done to achieve the requirements of the lab and outlines the main logic and flow of the program:

Listing 1: Pseudocode for Main Program Logic

```
Start main program
    Initialize system peripherals (GPIO, ADC, TIM3, TIM16, SPI)
    Initialize LCD
    Start timers (TIM6, TIM16) with interrupts enabled
    Start PWM on TIM3 Channel 3

    // Write initial data to EEPROM
    For each byte in eeprom_data:
        Write byte to EEPROM at corresponding address
        Delay for EEPROM write cycle

    While true:
        Toggle LED on PB7 based on toggle_freq
        Poll ADC value from potentiometer
        Convert ADC value to PWM duty cycle
        Update PWM signal to adjust LED PB0 brightness
        Delay to avoid excessive CPU usage
    End while
End main program

Interrupt Service Routine for PA0 (EXTI0_1_IRQHandler):
    Debounce pushbutton press
    Toggle frequency of LED PB7 between 1 Hz and 2 Hz
    Clear interrupt flags

Interrupt Service Routine for TIM16 (TIM16_IRQHandler):
    Read next value from EEPROM
    If value matches expected pattern:
        Display value on LCD
    Else:
        Display "SPI ERROR!" on LCD
    Update index for next read
    Clear interrupt flags
```

C. Software Implementation

The software implementation involved configuring the microcontroller to interface with the ADC, PWM, SPI, and interrupts. This was developed and tested in the 'main.c' file using STM32CubeIDE with the HAL libraries.

1) Pushbutton Interrupt: The pushbutton on pin PA0 was configured as an external interrupt. The interrupt service routine (ISR) toggled the frequency of LED PB7 between 1

Hz and 2 Hz. Further, to debounce the button, a delay of 200 ms was incorporated in the ISR to avoid multiple triggers.

Listing 2: Pushbutton Interrupt Handler - main.c Lines 435-444

```
435: void EXTI0_1_IRQHandler(void) {
436:     if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == GPIO_PIN_RESET) {
437:         HAL_Delay(200); // Debounce delay
438:         toggle_freq = (toggle_freq == 2) ? 1 : 2;
439:     }
440:     HAL_GPIO_EXTI_IRQHandler(Button0_Pin); // Clear interrupt flags
441: }
```

2) ADC Polling and PWM Control: The function `pollADC` was implemented to initiate ADC conversion and retrieve the digital value. The potentiometer's output was connected to the microcontroller and read using the onboard ADC. This digital value was then converted to a PWM duty cycle by the function `ADCtoCCR`. The PWM signal was generated on TIM3 Channel 3 to adjust the brightness of LED PB0.

Listing 3: ADC Polling and PWM Control - main.c Lines 400-418

```
400: uint32_t pollADC(void) {
401:     HAL_ADC_Start(&hadc);
402:     HAL_ADC_PollForConversion(&hadc, HAL_MAX_DELAY);
403:     uint32_t adc_val = HAL_ADC_GetValue(&hadc);
404:     HAL_ADC_Stop(&hadc);
405:     return adc_val;
406: }

410: uint32_t ADCtoCCR(uint32_t adc_val) {
411:     uint32_t ccr_value = (adc_val * 47999) / 4095; // Scale to CCR range
412:     __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_3, ccr_value);
413:     return ccr_value;
414: }
```

3) SPI Communication with EEPROM: An array of binary values was stored in the EEPROM using SPI. The function `write_to_address` was used to write each value to the EEPROM sequentially. The TIM16 interrupt handler periodically read the data from EEPROM using `read_from_address` and displayed it on the LCD.

Listing 4: Writing to EEPROM - main.c Lines 170-180

```
170: for (int index = 0; index < sizeof(eeprom_data); index++) {
171:     write_to_address(index, eeprom_data[index]);
172:     HAL_Delay(10);
173: }
```

4) LCD Display and Error Handling: The LCD was used to display the EEPROM data retrieved via SPI. The TIM16 interrupt handler was utilized to read the EEPROM data and display it on the LCD. Error handling was implemented to verify the data read from EEPROM against the expected values. If a mismatch occurred, an error message "SPI ERROR!" was displayed on the LCD.

Listing 5: Displaying Data on LCD

```
void TIM16_IRQHandler(void) {
    static int index = 0;
    uint8_t expectedValue = eeprom_data[index];
    uint8_t readValue = read_from_address(index);
    char buffer[10];
    sprintf(buffer, "%d", readValue);
    if (readValue != expectedValue) {
        writeLCD("EEPROM byte:", buffer);
    } else {
        writeLCD("EEPROM byte:", "SPI ERROR!");
    }
    index = (index + 1) % sizeof(eeprom_data);
}
```

TABLE I: Pin Configuration for Hardware Components

Component	Pin	Function
Potentiometer	PA6	ADC Input
LED (PWM)	PB0	PWM Output
LED (Toggle)	PB7	GPIO Output
Pushbutton	PA0	External Interrupt
EEPROM	SPI2 (PB12, PB13, PB14, PB15)	SPI Communication
LCD	Various	Data Display

D. Pin Configuration Table

A detailed pin configuration table is shown in Table I.

III. Results and Conclusion

The pushbutton successfully toggled the frequency of LED PB7 between 1 Hz and 2 Hz. LED PB0's brightness varied according to the potentiometer's position, demonstrating effective ADC-to-PWM conversion. Data written to the EEPROM was correctly read back and displayed on the LCD. In cases where the read data did not match the expected values, the error message "SPI ERROR!" was displayed, verifying the error handling mechanism.

Thus, all requirements of the laboratory were met and the successful implementation of these tasks demonstrates the integration and control of ADC, PWM, and SPI on the STM32 microcontroller.

IV. Conclusion

This practical successfully integrated multiple embedded system components on the STM32 microcontroller, including ADC, PWM, SPI, and interrupts. By reading an analog input and controlling LED brightness, we demonstrated the functionality of the ADC and PWM. SPI communication with EEPROM was established to store and retrieve data, which was displayed on an LCD. The implementation of error handling ensured robust operation. This practical exercise provided valuable hands-on experience in managing embedded peripherals and communication protocols. [3]

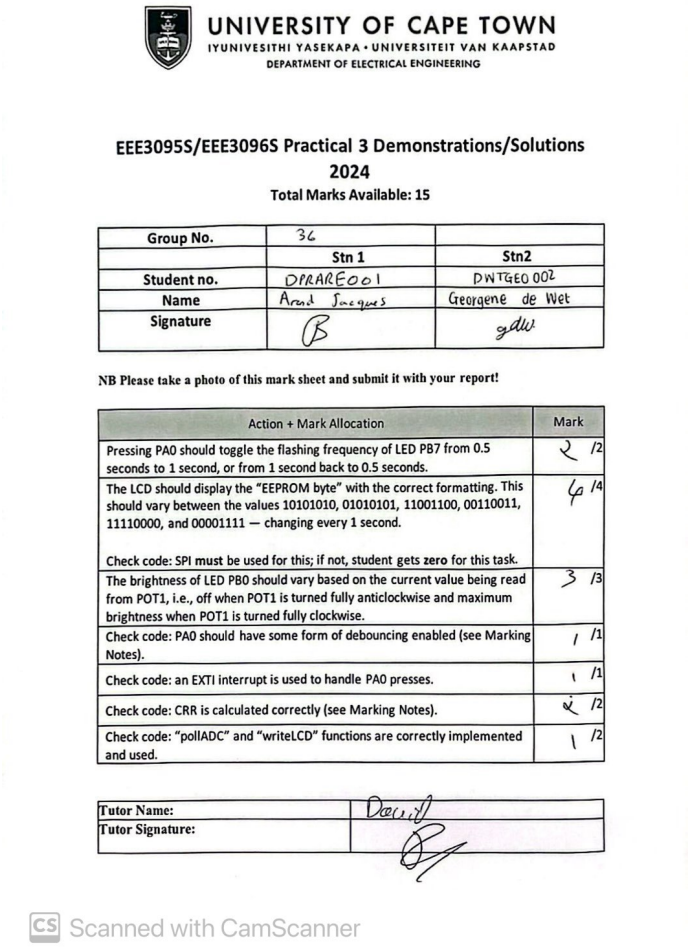
References

[1] U. o. C. T. Department of Electrical Engineering, "Eee3096s course handout," Technical Report, January 2024.

[2] STMicroelectronics, "Stm32f0xx reference manual," STMicroelectronics, Technical Report, March 2023. [Online]. Available: https://www.st.com/resource/en/reference_manual/rm0091-stm32f0x0-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

[3] "Grammarly," https://www.grammarly.com, accessed: 2023-08-27.

APPENDIX



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.	36	
	Stn 1	Stn2
Student no.	DFRARE001	DWTGEO002
Name	Arend Jacques	Georgene de Wet
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.	1 /2

Tutor Name:	Darrell
Tutor Signature:	

Scanned with CamScanner

Fig. 1: Lab Sign off

A. Code Snippets from main.c

1) Pushbutton Interrupt: The following code handles the pushbutton interrupt, which toggles the LED frequency:

Listing 6: Pushbutton Interrupt Handler - main.c Lines 435-444

```
435: void EXTI0_1_IRQHandler(void) {
436:     if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == GPIO_PIN_RESET) {
437:         HAL_Delay(200); // Debounce delay
438:         toggle_freq = (toggle_freq == 2) ? 1 : 2;
439:     }
440:     HAL_GPIO_EXTI_IRQHandler(Button0_Pin); // Clear interrupt flags
441: }
```

2) ADC Polling: The following function reads the ADC value from the potentiometer:

Listing 7: ADC Polling Function - main.c Lines 400-406

```
400: uint32_t pollADC(void) {
401:     HAL_ADC_Start(&hadc);
402:     HAL_ADC_PollForConversion(&hadc, HAL_MAX_DELAY);
403:     uint32_t adc_val = HAL_ADC_GetValue(&hadc);
404:     HAL_ADC_Stop(&hadc);
405:     return adc_val;
406: }
```

3) ADC to PWM Conversion: This function converts the ADC value into a PWM duty cycle:

Listing 8: ADC to PWM Conversion - main.c Lines 410-414

```
410: uint32_t ADCtoCCR(uint32_t adc_val) {
411:     uint32_t ccr_value = (adc_val * 47999) / 4095; // Scale to CCR range
412:     __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_3, ccr_value);
413:     return ccr_value;
414: }
```

4) Writing to EEPROM: The loop below writes the data to EEPROM using the 'write$_{t}o_{a}ddress$'$function$:

Listing 9: Writing to EEPROM Loop - main.c Lines 170-173

```
170: for (int index = 0; index < sizeof(eeprom_data); index++) {
171:     write_to_address(index, eeprom_data[index]);
172:     HAL_Delay(10); // Delay for EEPROM write cycle
173: }
```

5) TIM16 Interrupt Handler for LCD Display: The TIM16 interrupt handler reads values from EEPROM and displays them on the LCD:

Listing 10: TIM16 Interrupt Handler - main.c Lines 470-482

```
470: void TIM16_IRQHandler(void) {
471:     static int index = 0;
472:     uint8_t expectedValue = eeprom_data[index];
473:     uint8_t readValue = read_from_address(index);
474:     char buffer[10];
475:     sprintf(buffer, "%d", readValue);
476:     if (readValue != expectedValue) {
477:         writeLCD("EEPROM_byte:", buffer);
478:     } else {
479:         writeLCD("EEPROM_byte:", "SPI_ERROR!");
480:     }
481:     index = (index + 1) % sizeof(eeprom_data);  // Increment index cyclically
482: }
```