Department of

Electrical & Electronics Engineering

Abdullah Gül University

Lab Experiment 4 Report

EE3002 - EMBEDDED CONTROL SYSTEMS DESIGN CAPSULE

Submitted on: Submitted by: Pınar Yılmaz

Lab Partners: Ahmed Tamer

Lab Instructors: Atıf Kerem Şanlı

Grade: / 100

OBJECTIVE

The goal of this lab to use the STM32 microcontroller's Analog-to-Digital Converter (ADC) with interrupts. This system reads an analog signal through the ADC. It processes the ADC value to control the PWM (Pulse Width Modulation) output and uses an LED as an indicator. The brightness of the LED is adjusted based on the ADC value, and if the value reaches a certain threshold, the LED is toggled.

BACKGROUND

This lab focuses on the Analog-to-Digital functionality of the STM32F107xx microcontroller. ADC converts analog signals into digital values. Interrupts allows to handle tasks asynchronously. When a specific event occurs, interrupt is triggered. This lets the program respond to the event without having to check for it constantly.

DESIGN AND TEST PROCEDURES

Design Steps

1. **GPIO Clock Configuration:**
    - The system clock was chosen as HSI.
2. **GPIO Pin Configuration:**
    - PA0 is set up as an analog input to read the voltage from the external analog sensor. This pin is connected to the ADC.
    - For PWM output, PA6 is set up as an output. This pin controls the LED's brightness that is attached to this pin.
    - PC13 is configured as an output to control the external LED. If the ADC value is higher than threshold, the LED is toggled.
3. **ADC configuration:**
    - ADC1 is set up to sample an analog input pin PA0 at a regular interval.
4. **PWM for LED Brightness Control**
    - PA6 is configured with PWM to control the brightness of the LED. TIM3 (Timer 3) is used to produce PWM signals on PA6.

- The duty cycle of the PWM is adjusted based on the ADC value to vary the LED brightness. It is mapped from the ADC result (0-4095) to a range of 0-1000, which corresponds to the PWM duty cycle.

5. **LED Control**

   - After each ADC conversion, the program checks if the ADC value reach a threshold of 3000. LED on PC13 is set up when the value is above the threshold, the PC13 is toggled.

Code Explanation

1. System Clock Initialization

For system clock configuration HIS was enabled. The system clock was chosen as HSI.

```
void SystemClock_Config(void) {
    RCC->CR |= RCC_CR_HSION; // Enable HSI
    while (!(RCC->CR & RCC_CR_HSIRDY)); // Wait until HSI is ready

    RCC->CFGR &= ~RCC_CFGR_SW;       // Clear SW bits
    RCC->CFGR |= RCC_CFGR_SW_HSI;    // Select HSI as system clock
    while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_HSI); // Wait for HSI
}
```

2. GPIO Configuration

- The clocks for GPIO ports A and C was enabled.

```
void GPIO_Init(void) {
    RCC->APB2ENR |= RCC_APB2ENR_IOPAEN | RCC_APB2ENR_IOPCEN |
RCC_APB2ENR_AFIOEN;
```

- PA0 configured as an analog input for the ADC.

```
  GPIOA->CRL &= ~GPIO_CRL_MODE0;  // Clear mode bits
  GPIOA->CRL &= ~GPIO_CRL_CNF0;   // Set as analog input
```

- PA6 set as output for PWM.

```
   /* Configure PA6 as Alternate Function Push-Pull (PWM) */
   GPIOA->CRL &= ~GPIO_CRL_MODE6;  // Clear mode bits
   GPIOA->CRL |= GPIO_CRL_MODE6_1; // Set output mode, 2 MHz
   GPIOA->CRL &= ~GPIO_CRL_CNF6;   // Clear CNF bits
   GPIOA->CRL |= GPIO_CRL_CNF6_1;  // Alternate function push-pull
```

- PC13 configured as a general-purpose output for an LED.

```
  GPIOC->CRH &= ~GPIO_CRH_MODE13;  // Clear mode bits
  GPIOC->CRH |= GPIO_CRH_MODE13_1; // Output mode, 2 MHz
  GPIOC->CRH &= ~GPIO_CRH_CNF13;   // Push-pull mode
}
```

3. ADC1 Initialization

- ADC is set up to convert an analog signal (in this case 3V) into a digital value. PA0 is configured to read data.

- The ADC1 clock was enabled using the **`RCC->APB2ENR |= RCC_APB2ENR_ADC1EN`** bit.
- Sampling time arranged for channel 0 (PA0) by adjusting the SMPR2 register and ADC is set to convert channel 0 by modifying the SQR3 register.

```
void ADC1_Init(void) {
    RCC->APB2ENR |= RCC_APB2ENR_ADC1EN; // Enable ADC1 Clock

    ADC1->CR1 = 0;                          // Single conversion mode
    ADC1->CR2 = ADC_CR2_ADON;               // Power on ADC
    ADC1->SMPR2 |= ADC_SMPR2_SMP0;          // Set sample time for channel 0
    ADC1->SQR3 = 0;                         // Select channel 0 for conversion
}
```

4. Timer Initialization for PWM

This function initializes Timer 3 to produce PWM signals by setting a prescaler to lower the timer clock frequency. It configures PWM mode for Channel 1 (PA6) and enables the output.

- Timer3 clock is enabled by using RCC_APB1ENR_TIM3EN.
- The timer's prescaler is set to 8.
- The duty cycle is initialized to 0% by setting CCR1 to 0.

```
void TIM3_Init(void) {
    RCC->APB1ENR |= RCC_APB1ENR_TIM3EN; // Enable TIM3 clock

    TIM3->PSC = 8;
    TIM3->ARR = 3000 - 1;
    TIM3->CCR1 = 0;         // Start with 0% duty cycle

    // Configure PWM mode on TIM3 Channel 1 (PA6)
    TIM3->CCMR1 = TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_2; // PWM mode 1
    TIM3->CCMR1 |= TIM_CCMR1_OC1PE;                    // Enable preload
    TIM3->CCER |= TIM_CCER_CC1E;                       // Enable channel 1
output
    TIM3->CR1 |= TIM_CR1_ARPE;                         // Enable auto-reload
preload
    TIM3->EGR |= TIM_EGR_UG;                           // Generate an update
event
    TIM3->CR1 |= TIM_CR1_CEN;                          // Enable the timer
}
```

5. Delay

```
void delay_ms(uint32_t ms) {
    for (uint32_t i = 0; i < ms * 1000; i++) {
        __NOP(); // No operation (busy wait)
    }
}
```

Full Code

```
#include "stm32f10x.h" // Include CMSIS library for STM32F1
void SystemClock_Config(void);
void GPIO_Init(void);
void ADC1_Init(void);
void TIM3_Init(void);
void delay_ms(uint32_t ms);

int main(void) {

   SystemClock_Config(); //  Configure System Clock

   /* Initialize GPIO, ADC, and Timer */
   GPIO_Init();
   ADC1_Init();
   TIM3_Init();

   //Start ADC Conversion
   ADC1->CR2 |= ADC_CR2_ADON;       // Enable ADC
   ADC1->CR2 |= ADC_CR2_SWSTART;   // Start ADC conversion

   /* Infinite loop */
   while (1) {
      // Wait for ADC conversion to complete
      while (!(ADC1->SR & ADC_SR_EOC)); // Wait for End of Conversion

      uint16_t adcValue = ADC1->DR; // Read ADC value


      // Map ADC value (0-4095) to PWM duty cycle (0-1000)
      uint32_t dutyCycle = (adcValue * 1000) / 4095; // Scale to PWM range

      // Update PWM duty cycle (CCR1 for TIM3 Channel 1)
      TIM3->CCR1 = dutyCycle;

      // Blink LED if ADC value exceeds a threshold
      if (adcValue > 3000) {
         GPIOC->ODR ^= GPIO_ODR_ODR13; // Toggle PC13 (LED)
         delay_ms(200);              // Simple delay
      }
   }
}

/* System Clock Configuration */
void SystemClock_Config(void) {
   RCC->CR |= RCC_CR_HSION; // Enable HSI
   while (!(RCC->CR & RCC_CR_HSIRDY)); // Wait until HSI is ready

   RCC->CFGR &= ~RCC_CFGR_SW;      // Clear SW bits
   RCC->CFGR |= RCC_CFGR_SW_HSI;   // Select HSI as system clock
   while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_HSI); // Wait for HSI
}
```

```
/* GPIO Initialization */
void GPIO_Init(void) {
   /* Enable GPIO Clocks */
   RCC->APB2ENR |= RCC_APB2ENR_IOPAEN | RCC_APB2ENR_IOPCEN |
RCC_APB2ENR_AFIOEN;

   /* Configure PA0 as Analog Input (ADC) */
   GPIOA->CRL &= ~GPIO_CRL_MODE0;  // Clear mode bits
   GPIOA->CRL &= ~GPIO_CRL_CNF0;   // Set as analog input

   /* Configure PA6 as Alternate Function Push-Pull (PWM) */
   GPIOA->CRL &= ~GPIO_CRL_MODE6;  // Clear mode bits
   GPIOA->CRL |= GPIO_CRL_MODE6_1; // Set output mode, 2 MHz
   GPIOA->CRL &= ~GPIO_CRL_CNF6;   // Clear CNF bits
   GPIOA->CRL |= GPIO_CRL_CNF6_1;  // Alternate function push-pull

   /* Configure PC13 as General Purpose Output (LED) */
   GPIOC->CRH &= ~GPIO_CRH_MODE13;  // Clear mode bits
   GPIOC->CRH |= GPIO_CRH_MODE13_1; // Output mode, 2 MHz
   GPIOC->CRH &= ~GPIO_CRH_CNF13;   // Push-pull mode
}

/* ADC1 Initialization */
void ADC1_Init(void) {
   RCC->APB2ENR |= RCC_APB2ENR_ADC1EN; // Enable ADC1 Clock

   ADC1->CR1 = 0;                      // Single conversion mode
   ADC1->CR2 = ADC_CR2_ADON;           // Power on ADC
   ADC1->SMPR2 |= ADC_SMPR2_SMP0;      // Set sample time for channel 0
   ADC1->SQR3 = 0;                     // Select channel 0 for conversion
}

/* Timer3 Initialization for PWM */
void TIM3_Init(void) {
   RCC->APB1ENR |= RCC_APB1ENR_TIM3EN; // Enable TIM3 clock

   TIM3->PSC = 72 - 1;     // Prescaler (72 MHz / 72 = 1 MHz)
   TIM3->ARR = 1000 - 1;   // Auto-reload (1 kHz PWM frequency)
   TIM3->CCR1 = 0;         // Start with 0% duty cycle

   // Configure PWM mode on TIM3 Channel 1 (PA6)
   TIM3->CCMR1 = TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_2; // PWM mode 1
   TIM3->CCMR1 |= TIM_CCMR1_OC1PE;               // Enable preload
   TIM3->CCER |= TIM_CCER_CC1E;                  // Enable channel 1 output
   TIM3->CR1 |= TIM_CR1_ARPE;                    // Enable auto-reload preload
   TIM3->EGR |= TIM_EGR_UG;                      // Generate an update event
   TIM3->CR1 |= TIM_CR1_CEN;                     // Enable the timer
}

/* Simple delay function */
void delay_ms(uint32_t ms) {
   for (uint32_t i = 0; i < ms * 1000; i++) {
      __NOP(); // No operation (busy wait)
   }
}
```

RESULTS AND DISCUSSION

The system reads an analog input from PA0 and converts it through the ADC. ADC value controls the brightness of an LED using PWM on PA6. When the ADC value exceeds threshold value, the LED on PC13 is toggled. The LED's brightness is changes according to the input analog signal from potentiometer.

- **Challenges**:
    - **Pin Conflict**: During the experiment, we encountered a problem. We tried several pin to configure with PWM to control the brightness of an LED. However, only one of them that we tried is worked which is PA6. To solve this issue, instead of PA1 and PB4 we tried PA6, which is worked inputs for this project and worked correctly.

CONCLUSIONS

This lab successfully demonstrated how to use the Analog-to-Digital Converter (ADC) with interrupts. The analog input is used to adjust the PWM duty cycle, which controls the brightness of an LED. Beside, an LED toggles according to ADC value. In conclusion, the code successfully set up and shows the use of ADC, PWM, and GPIO pins for managing external hardware.

REFERENCES

[1] STMicroelectronics, "STM32F10xxx Reference Manual," [Online]. Available: https://www.st.com/resource/en/reference_manual/cd00171190-stm32f101xx-stm32f102xx-stm32f103xx-and-stm32f105xx-stm32f107xx-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf. Accessed: Oct. 14, 2024.