GEBZE TECHNICAL UNIVERSITY

ELECTRONIC ENGINEERING

ELEC335

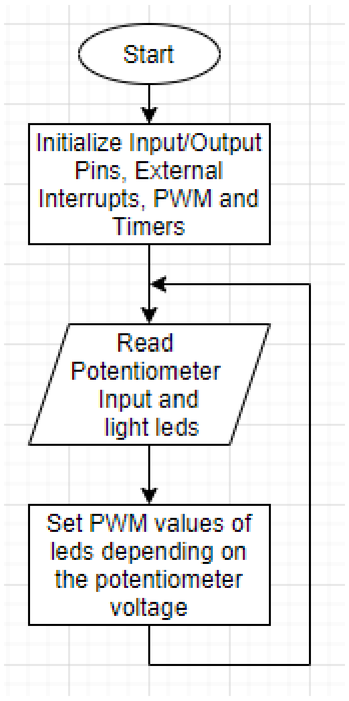
MICROPROCESSORS LAB

LAB 7 Report

Prepared by
171024086 Berat KIZILARMUT

1. Introduction

In this lab firstly I will be designing a led dimmer that dims and lights a pair of leds depending on the analog potentiometer input. On the second problem I will be designing a knock counter that will counts the knocks recorded by the microphone and increases the counter.

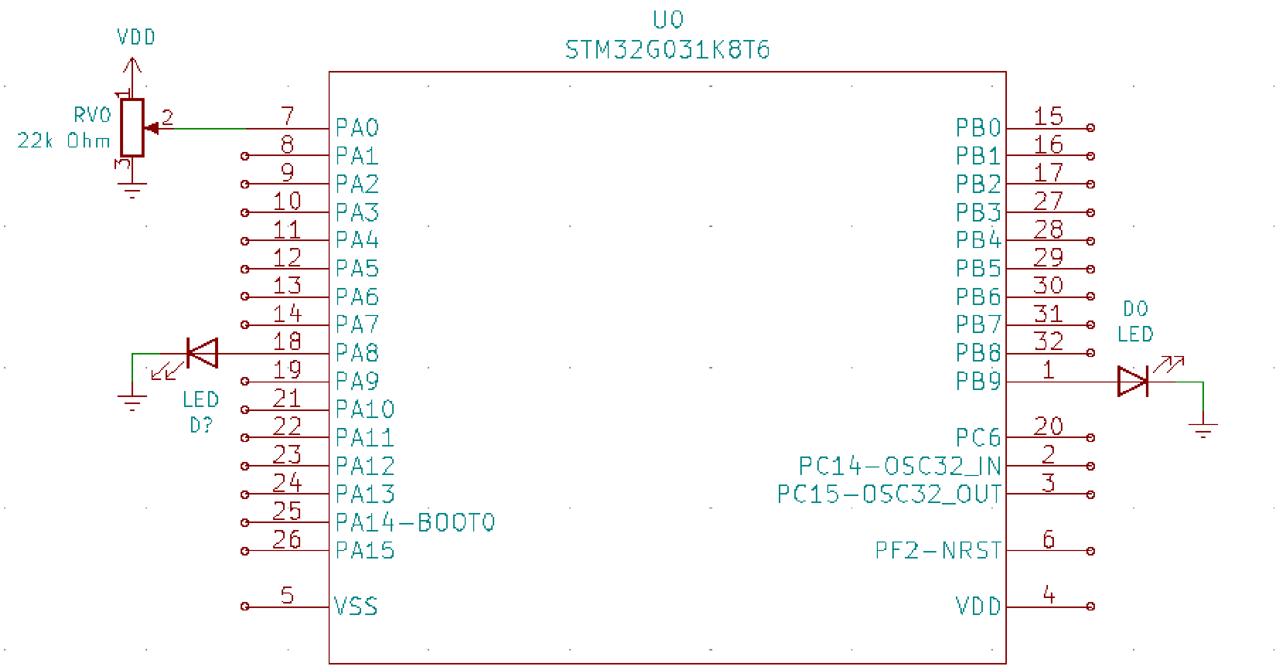


Start
Initialize Input/Output Pins, External Interrupts, PWM and Timers
Read Potentiometer Input and light leds
Set PWM values of leds depending on the potentiometer voltage

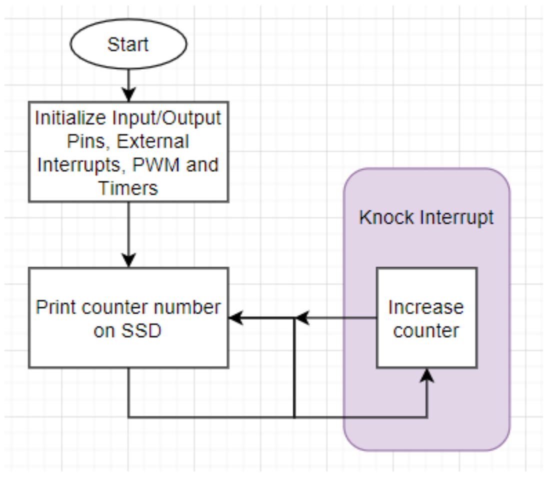
2. Problem 1

I will be designing a light dimmer that will dim and light a pair of leds depending on the analog input from a potentiometer. Potentiometer values will change the duty cycles of the leds. When the potentiometer knob is all the way to the left, towards the first led, that said led will be lit fully and second led will be disabled. They will be half ways on duty cycle when the knob is on the middle. As expected led 1 will turn of when knob is all the day towards led2 way.

Firstly, we need two different PWM function to use on individually lit leds. For the first led I chose pin PB9 and set TIM17 to use as PWM. For the second led I chose pin PA8 and set TIM1 to use as PWM. Analog potentiometer is connected as, first leg to 5V, second leg to ADC pin PA0.



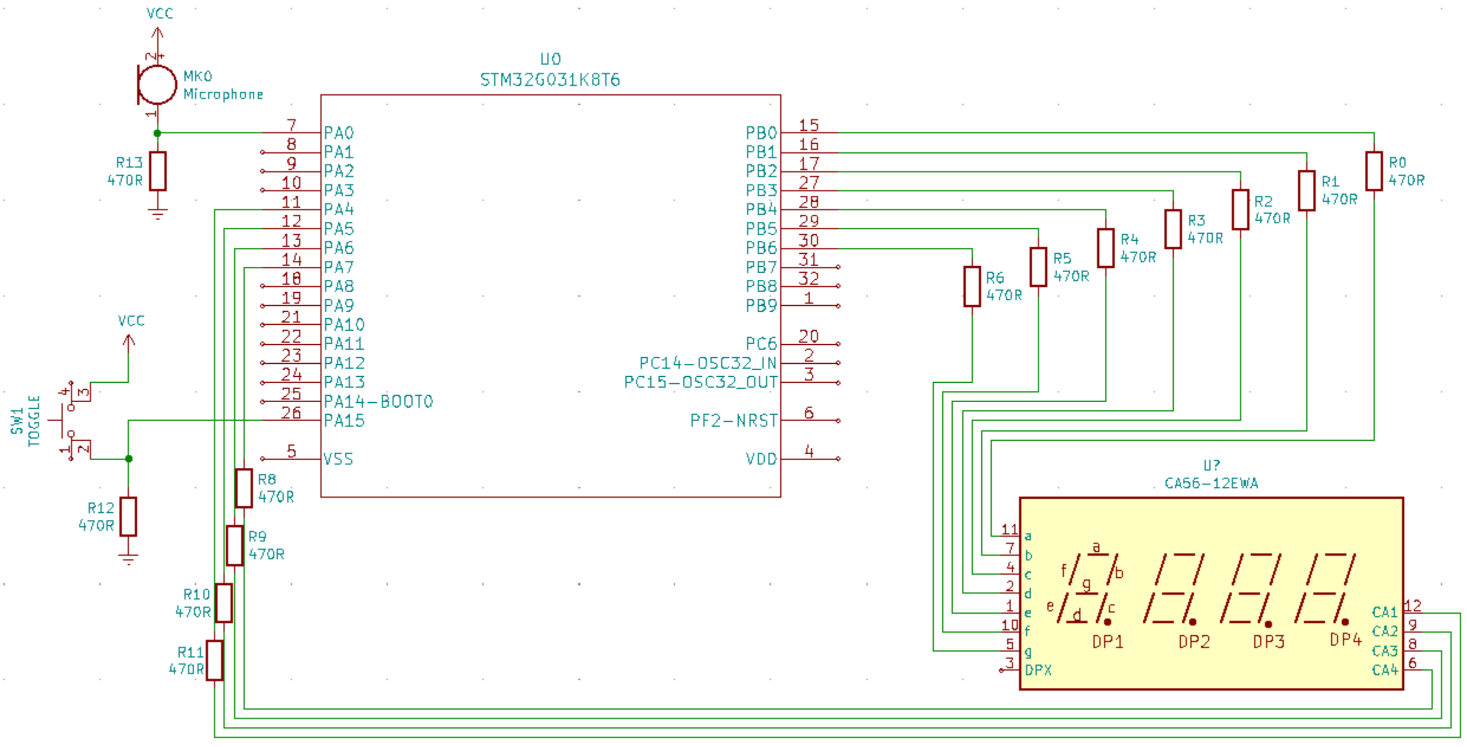
```c
/* problem1.c Lab 7, Problem 1, Led Dimmer
    Berat Kizilarmut, 171024086 */
#include "stm32g0xx.h"
volatile uint32_t delaycount = 0; /* Global delay counter */

void delay_ms(volatile uint32_t s) /* Set as volatile to prevent optimization */
{
    delaycount = 0; /* Set counter as zero */
    while(1) /* Continuously checks the counter */
    {
        if(delaycount == s)
        return;
    }
}

void TIM2_BRK_UP_TRG_COM_IRQHandler (void) /* Configuring TIM2 */
{
    delaycount++;
    TIM2->SR &= ~(1U << 0); /* Clear pending status */
}

void system_initialize(void) /* Setting up the board and subsystems */
{
    /* Enable GPIOA and GPIOB clock */
    RCC->IOPENR |= (3U); /* 3U = 11 */

    /* Setup PA8 as Alternate function mode */
    GPIOA->MODER &= ~(3U << 2*8);
    GPIOA->MODER |= (2U << 2*8);
    /* Choose AF2 from the Mux */
    GPIOA->AFR[1] &= ~(0xFU << 4*0);
    GPIOA->AFR[1] |= (2U << 4*0);
    /* Setting TIM1 for PWM on PA8 */
    RCC->APBENR2 |= (1U << 11); /* Enable TIM1 Clock */
    TIM1->CR1 = 0; /* Clearing the control register */
    TIM1->CR1 |= (1U << 7);  /* Auto Reload Preload Enable */
    TIM1->CNT = 0; /* Zero the counter */
    TIM1-
>PSC = 15999; /* Setting prescaler as 16000 to achieve a millisecond on my delay fun
ction */
    TIM1->ARR = 1; /* Auto Reload Register */
    TIM1->DIER |= (1U << 0); /* Updating interrupt enabler */
    TIM1->CR1 |= (1U << 0); /* Enable TIM1 */
    TIM1->CCMR1 |= (6U << 4); /* Capture/Compare mode register */
    TIM1->CCMR1 |= (1U << 3);
    TIM1->CCER |= (1U << 0); /* Capture/Compare enable register */
    TIM1->CCR1 |= (1U << 0); /* Capture/Compare register 1 */
    TIM1->BDTR |= (1U << 15); /* Break and Dead-Time register */

    /* Setup PB9 as Alternate function mode */
    GPIOB->MODER &= ~(3U << 2*9);
    GPIOB->MODER |= (2U << 2*9);
    /* Choose AF2 from the Mux */
    GPIOB->AFR[1] &= ~(0xFU << 4*1);
    GPIOB->AFR[1] |= (2U << 4*1);
    /* Setting TIM17 for PWM on PB9 */
    RCC->APBENR2 |= (1U << 18); /* Enable TIM17 Clock */
    TIM17->CR1 = 0; /* Clearing the control register */
    TIM17->CR1 |= (1U << 7);  /* Auto Reload Preload Enable */
    TIM17->CNT = 0; /* Zero the counter */
    TIM17-
>PSC = 15999; /* Setting prescaler as 16000 to achieve a millisecond on my delay fun
ction */
    TIM17->ARR = 1; /* Auto Reload Register */
    TIM17->CR1 |= (1U << 0); /* Enable TIM17 */
    TIM17->CCMR1 |= (6U << 4); /* Capture/Compare mode register */
    TIM17->CCMR1 |= (1U << 3);
    TIM17->CCER |= (1U << 0); /* Capture/Compare enable register */
    TIM17->CCR1 |= (1U << 0); /* Capture/Compare register 1 */
    TIM17->BDTR |= (1U << 15); /* Break and Dead-Time register */

    /* Setting TIM2 for Delays*/
    RCC->APBENR1 |= (1U << 0); /* Enable TIM2 Clock */
    TIM2->CR1 = 0; /* Clearing the control register */
    TIM2->CR1 |= (1U << 7);  /* Auto Reload Preload Enable */
    TIM2->CNT = 0; /* Zero the counter */
    TIM2-
>PSC = 15999; /* Setting prescaler as 16000 to achieve a millisecond on my delay fun
ction */
    TIM2->ARR = 1; /* Auto Reload Register */
    TIM2->DIER |= (1U << 0); /* Updating interrupt enabler */
    TIM2->CR1 |= (1U << 0); /* Enable TIM2 */
    NVIC_SetPriority(TIM2_BRK_UP_TRG_COM_IRQn, 0); /* Setting highest priority */
    NVIC_EnableIRQ(TIM2_BRK_UP_TRG_COM_IRQn); /* Enabling interrupt */

    /* Setup PA0 as Analog mode */
    GPIOA->MODER |= (3U << 0);
    /* Setting up ADC1/0 for Analog input on PA0 */
    RCC->APBENR2 |= (1U << 20); /* Enable ADC Clock */
    ADC1->CR = 0; /* Clearing the control register */
    ADC1->CFGR1 = 0; /* Clearing the configrator register */
    ADC1->CR |= (1U << 28); /* Enabling voltage regulator */
    delay_ms(1000); /* Waiting for second to voltage to regulate */
    ADC1->CR |= (1U << 31); /* Initialize Calibration Operation */
    while(!((ADC1->ISR >> 11) & 1)); /* If ISR is 1 break */
    ADC1->IER |= (1U << 11); /* Stop Calibration Operation */
    ADC1->CFGR1 |= (2U << 3); /* Setting resolution on configration register */
    ADC1->CFGR1 |= (1U << 13); /* Continuous conversion mode enabled */
    ADC1->CFGR1 &= ~(1U << 16); /* Discontinuous mode disabled */
    ADC1-
>SMPR |= (0 << 0); /* Sampling Time Register is set as 1.5 ADC Clock Cycles */
    ADC1->CFGR1 &= ~(3U << 10); /* External trigger is disabled */
    ADC1->CFGR1 &= ~(7U << 6); /* External trigger selected as TRG0 */
    ADC1->CHSELR |= (1U << 0); /* Channel is set as PA0 */
    ADC1->ISR |= (1 << 0); /* ADC Ready bit is set as 1 */
    ADC1->CR |= (1 << 0); /* ADC Enable bit is set as 1 */
    while( !(ADC1->ISR & (1 << 0))); /* If ISR is 0 break */
    ADC1->CR |= (1U << 2); /* ADC start conversion bit is set as 1 */
    active_run(); /* Run active running function */
}

void active_run(void)
{
    while (1)
    {
        /* Read ADC voltage here */
        /* Set PWM of the leds */
    }
}

int main(void) {
    system_initialize(); /* Calling the system initializer */
    while(1) { } /* Endless loop */
    return 0;
}
```


Start
Initialize Input/Output Pins, External Interrupts, PWM and Timers
Print counter number on SSD
Knock Interrupt
Increase counter

3. Problem 2

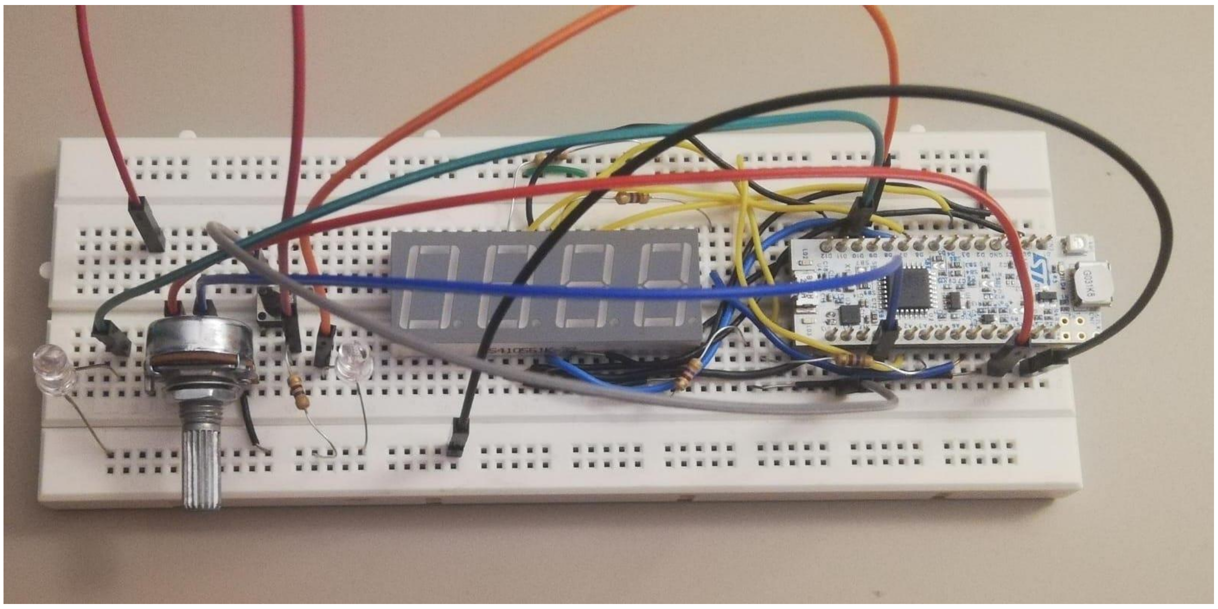
In this problem I will be designing a knock counter that will increase a counter which is being displayed on a 4-digit Seven Segment Display. Used pins PA4 to PA7 as Digit pins of the SSD, used pins PB0 to PB6 as A B C D E F G pins of the SSD. Digit pins are working active high, A B C D E F G pins are working active low. Connected a button to PA0 to use as a counter reset button.

I could not get this working because I have realized I don't have the microphone module I think I have. I have checked my receipts; I have purchased a MAX4466 Microphone Module from direnc.net with my other module purchases. I have realized that they didn't ship it with my other modules and could not get a replacement in time. Apologies.



Explaining my code, created a ssd print and clearing function that prints and clears given values to the 4-digit Seven Segment Display. Set and configurated an external interrupt routine that sets the knock counter to zero when triggered. Set up TIM1 to use on delay operations inside interrupt handler functions.

```c
/* problem1.c Lab 7, Problem 2, Knock Counter
    Berat Kizilarmut, 171024086 */
#include "stm32g0xx.h"

volatile uint32_t delaycount = 0; /* Global delay counter */
volatile uint32_t count = 0; /* Global knock counter */

void ssd_clear(void) /* Clear the display */
{
    GPIOA->ODR &= ~(15U << 4); /* Clear Digits */
    GPIOB->ODR |= (255U); /* Clear A B C... */
}

void delay_ms(volatile uint32_t s) /* Set as volatile to prevent optimization */
{
    delaycount = 0; /* Set counter as zero */
    while(1) /* Continuously checks the counter */
    {
        if(delaycount == s)
        return;
    }
}

void ssd_printdigit(uint32_t s) /* Prints the values depending on the number */
{
    switch(s)
    {
        case 0: /* Printing 0 */
            GPIOB->ODR &= (0xC0U);
            break;

        case 1: /* Printing 1 */
            GPIOB->ODR &= (0xF9U);
            break;

        case 2: /* Printing 2 */
            GPIOB->ODR &= (0xA4U);
            break;

        case 3: /* Printing 3 */
            GPIOB->ODR &= (0xB0U);
            break;

        case 4: /* Printing 4 */
            GPIOB->ODR &= (0x99U);
            break;

        case 5: /* Printing 5 */
            GPIOB->ODR &= (0x92U);
            break;

        case 6: /* Printing 6 */
            GPIOB->ODR &= (0x82U);
            break;

        case 7: /* Printing 7 */
            GPIOB->ODR &= (0xF8U);
            break;

        case 8: /* Printing 8 */
            GPIOB->ODR &= (0x80U);
            break;

        case 9: /* Printing 9 */
            GPIOB->ODR &= (0x90U);
            break;
    }
}

void ssd_print(float value)
{
    /* Gather the digit by digit values using integer rounding */
    uint32_t d1=0, d2=0, d3=0, d4=0;
    d1=value/1000;
    value=value-(d1*1000);
    d2=value/100;
    value=value-(d2*100);
    d3=value/10;
    value=value-(d3*10);
    d4=value;
    ssd_clear();
    /* Print Thousands Digit */
    GPIOA->ODR |= (1U << 4); /* Set D1 High */
    ssd_printdigit(d1); /* Print Value to D1 */
    ssd_clear();
    /* Print Hundreds Digit */
    GPIOA->ODR |= (1U << 5); /* Set D2 High */
    ssd_printdigit(d2); /* Print Value to D2 */
    ssd_clear();
    /* Print Tens Digit */
    GPIOA->ODR |= (1U << 6); /* Set D3 High */
    ssd_printdigit(d3); /* Print Value to D3 */
    ssd_clear();
    /* Print Ones Digit */
    GPIOA->ODR |= (1U << 7); /* Set D4 High */
    ssd_printdigit(d4); /* Print Value to D4 */
    ssd_clear();
}

void EXTI0_1_IRQHandler (void) {
    counter = 0; /* Zeroing the counter */
    delay_ms(500); /* Delay so function does not run continiously */
    EXTI->RPR1 |= (1U << 0); /* Clear pending status */
}

void TIM1_BRK_UP_TRG_COM_IRQHandler (void) /* Configuring TIM1 */
{
    delaycount++;
    TIM1->SR &= ~(1U << 0); /* Clear pending status */
}

void system_initialize(void) /* Setting up the board and subsystems */
{
    /* Enable GPIOA and GPIOB clock */
    RCC->IOPENR |= (3U); /* 3U = 11 */

    /* Setting up the Seven Segment Display*/
    /* Set PB0 to PB7 as output to use as A B C D E F G and decimal point pins of th
e SSD */
    GPIOB->MODER &= ~(65535U); /* Setting first 16 bits as zero */
    GPIOB->MODER |= (21845U); /* Setting odd bits as one */
    /* Set PA4 to PA7 as output to use as D1 D2 D3 D4 pins of the SSD */
    GPIOA->MODER &= ~(65280U); /* Setting bits 8th to 16th as zero */
    GPIOA->MODER |= (21760U); /* Setting odd bits as one from 8th to 16th */

    /* Setting External Interrupt */
    /* Setup PA0 as input */
    GPIOA->MODER &= ~(3U);
    /* Setting up the interrupt operation for PA0 */
    EXTI->RTSR1 |= (1U); /* Setting the trigger as rising edge */
    EXTI->EXTICR[0] &= ~(1U << 8*0); /* EXTICR 0 for 0_1 */
    EXTI->IMR1 |= (1U << 0); /* Interrupt mask register */
    EXTI-
>RPR1 |= (1U << 0); /* Rising edge pending register, Clearing pending PA0 */
    NVIC_SetPriority(EXTI0_1_IRQn, 1); /* Setting priority */
    NVIC_EnableIRQ(EXTI0_1_IRQn); /* Enabling the interrupt function */

    /* Setting TIM1 for Delays*/
    RCC->APBENR2 |= (1U << 11); /* Enable TIM1 Clock */
    TIM1->CR1 = 0; /* Clearing the control register */
    TIM1->CR1 |= (1U << 7);  /* Auto Reload Preload Enable */
    TIM1->CNT = 0; /* Zero the counter */
    TIM1-
>PSC = 15999; /* Setting prescaler as 16000 to achieve a millisecond on my delay fun
ction */
    TIM1->ARR = 1; /* Auto Reload Register */
    TIM1->DIER |= (1U << 0); /* Updating interrupt enabler */
    TIM1->CR1 |= (1U << 0); /* Enable TIM1 */
    NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn, 0); /* Setting highest priority */
    NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn); /* Enabling interrupt */

    /* Setting TIM17 for PWM on PB9*/
    RCC->APBENR2 |= (1U << 18); /* Enable TIM17 Clock */
    TIM17->CR1 = 0; /* Clearing the control register */
    TIM17->CR1 |= (1U << 7);  /* Auto Reload Preload Enable */
    TIM17->CNT = 0; /* Zero the counter */
    TIM17-
>PSC = 15999; /* Setting prescaler as 16000 to achieve a millisecond on my delay fun
ction */
    TIM17->ARR = 1; /* Auto Reload Register */
    TIM17->DIER |= (1U << 0); /* Updating interrupt enabler */
    TIM17->CR1 |= (1U << 0); /* Enable TIM17 */
    TIM17->CCMR1 |= (6U << 4); /* Capture/Compare mode register */
    TIM17->CCMR1 |= (1U << 3);
    TIM17->CCER |= (1U << 0); /* Capture/Compare enable register */
    TIM17->CCR1 |= (1U << 0); /* Capture/Compare register 1 */
    TIM17->BDTR |= (1U << 15); /* Break and Dead-Time register */
    /* Setup PB9 as Alternate function mode */
    GPIOB->MODER &= ~(3U << 2*9);
    GPIOB->MODER |= (2U << 2*9);
    /* Choose AF2 from the Mux */
    GPIOB->AFR[1] &= ~(0xFU << 4*1);
    GPIOB->AFR[1] |= (2U << 4*1);

    /* Setup PA1 as Analog mode */
    GPIOA->MODER |= (3U << 1*2);
    /* Setting up ADC1/1 for Analog input on PA1 */
    RCC->APBENR2 |= (1U << 20); /* Enable ADC Clock */
    ADC1->CR = 0; /* Clearing the control register */
    ADC1->CFGR1 = 0; /* Clearing the configrator register */
    ADC1->CR |= (1U << 28); /* Enabling voltage regulator */
    delay_ms(1000); /* Waiting for second to voltage to regulate */
    ADC1->CR |= (1U << 31); /* Initialize Calibration Operation */
    while(!((ADC1->ISR >> 11) & 1)); /* If ISR is 1 break */
    ADC1->IER |= (1U << 11); /* Stop Calibration Operation */
    ADC1->CFGR1 |= (2U << 3); /* Setting resolution on configration register */
    ADC1->CFGR1 |= (1U << 13); /* Continuous conversion mode enabled */
    ADC1->CFGR1 &= ~(1U << 16); /* Discontinuous mode disabled */
    ADC1-
>SMPR |= (0 << 0); /* Sampling Time Register is set as 1.5 ADC Clock Cycles */
    ADC1->CFGR1 &= ~(3U << 10); /* External trigger is disabled */
    ADC1->CFGR1 &= ~(7U << 6); /* External trigger selected as TRG0 */
    ADC1->CHSELR |= (1U << 1); /* Channel is set as PA1 */
    ADC1->ISR |= (1 << 0); /* ADC Ready bit is set as 1 */
    ADC1->CR |= (1 << 0); /* ADC Enable bit is set as 1 */
    while( !(ADC1->ISR & (1 << 0))); /* If ISR is 0 break */
    ADC1->CR |= (1U << 2); /* ADC start conversion bit is set as 1 */
    /* Printing and prescaler setting loop */
    while(1)
    {
        ssd_print(counter);
    }
}

int main(void) {
    system_initialize(); /* Calling the system initializer */
    while(1) { } /* Endless loop */
    return 0;
}
```



4. References

a. RM0444 Reference Manual, STM32G0x1 advanced Arm®-based 32-bit MCUs, ST Microelectronics