FALL 2022 ELEC335 MICROPROCESSORS LAB

Prepared By
1) 1901022038 – Selen Erdoğan
2) 200102002043 – Senanur Ağaç
3) 1901022050 - Merve Tutar

Aim

The purpose of this lab is to read, write and process analog values. Generally, we used C language.

Problem 1

In this problem, you will implement a light dimmer with a potentiometer.
•Connect a pot using a resistor divider setting.
•Connect two external LEDs. These LEDs will light up in opposite configuration.
•By changing the pot you will change the brightness of these LEDs. For example,if the pot is all they way down,first LED should light up, and second LED should be o_, and if the pot is all the way up, first LED should be o_and the second LED should light up. Their brightness should change in between.
•You will need PWM for the LED driving to change the brightness. 0 duty cycle will turn offthe LEDs and 100% duty cycle will light them up completely.

SOLUTION 1

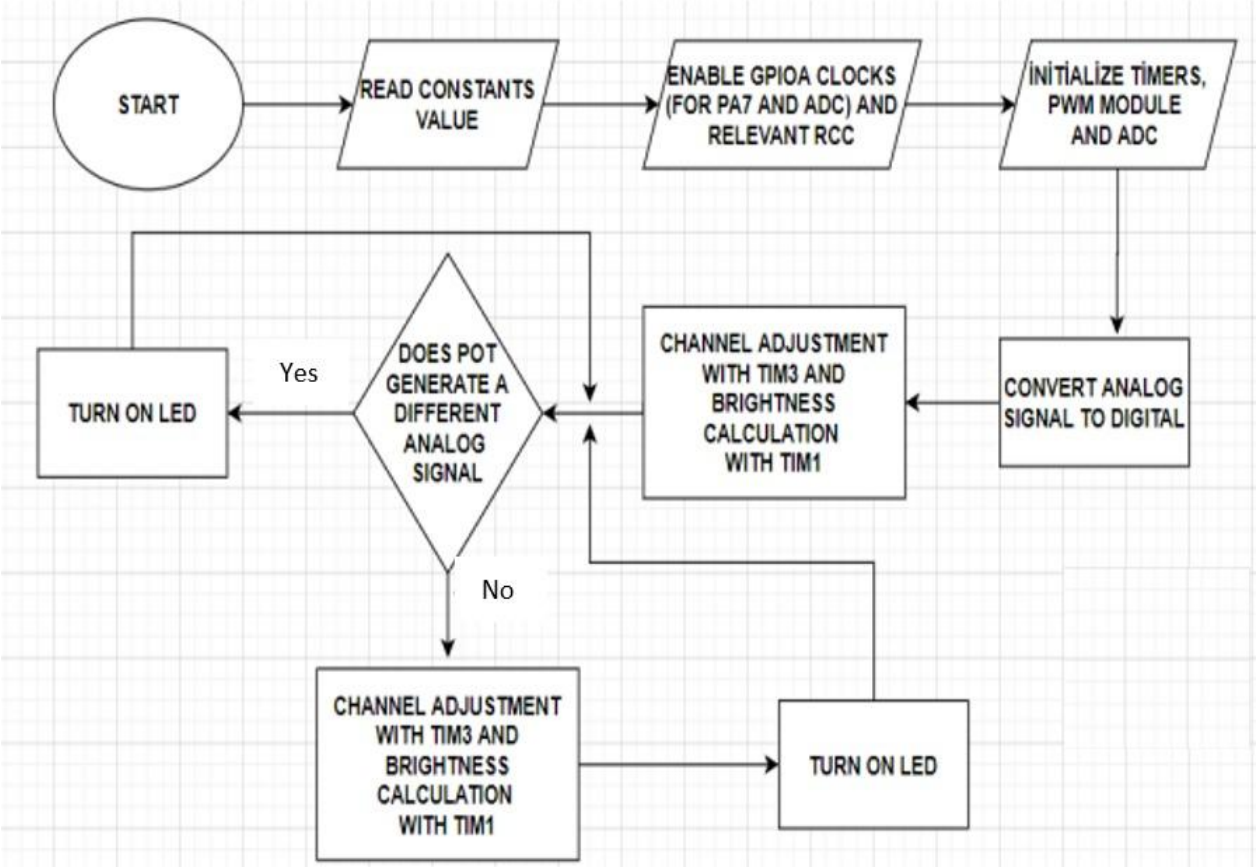

Figure 1. Flowchart

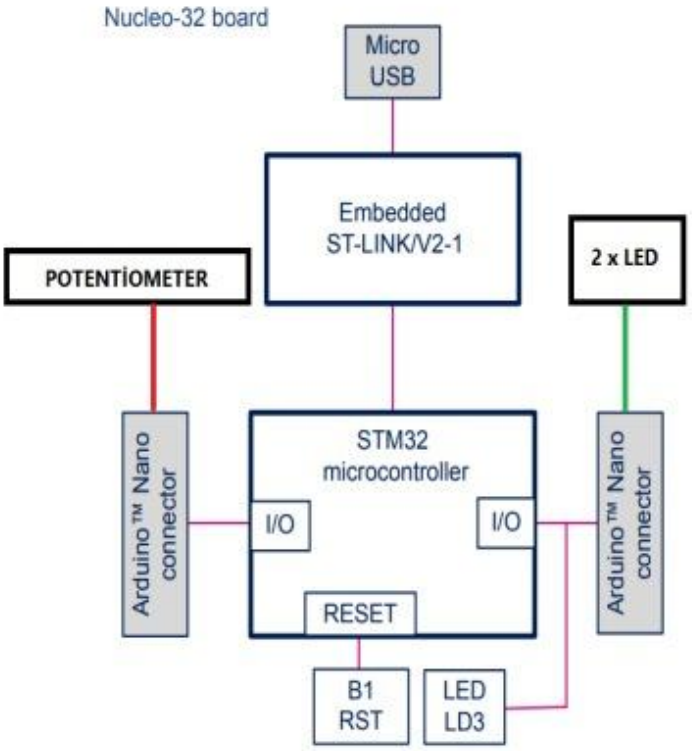

Figure 2. Block Diagram

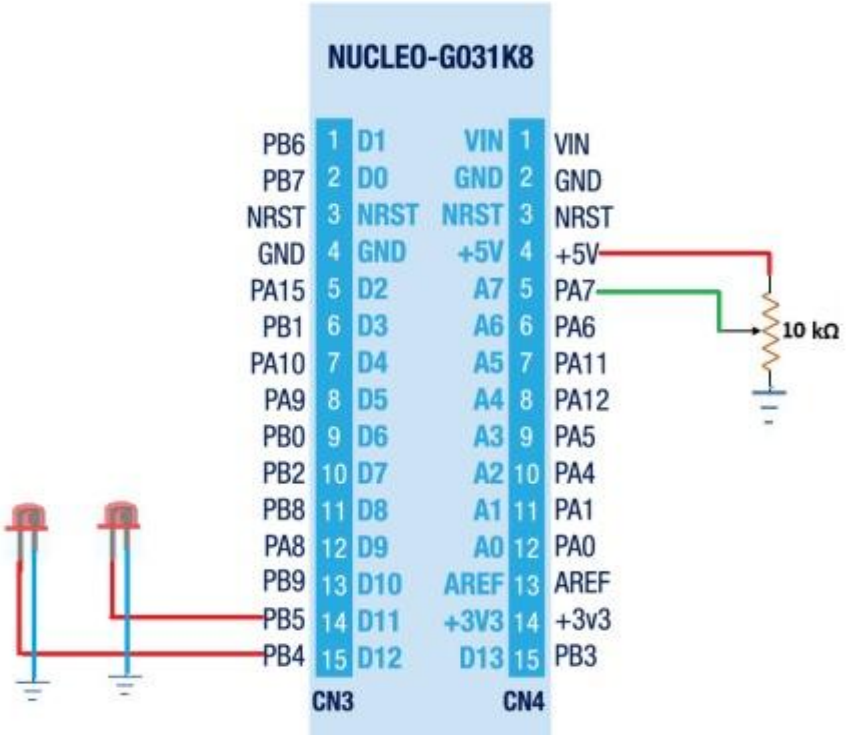

Figure 3. Schematic

➔ The analog value read through the potentiometer will be continuously processed in the ADC function and converted to a digital value. This digital value is processed in PWM module and transferred to LEDs. Since the PWM module will work together with the TIMER, brightness calculation in one of the TIMER_Handler functions is activated in one of the PWM mode channels. In this way, it is coordinated to convert the value received from the potentiometer to digital via ADC, this digital signal is parsed in the PWM module to determine which LEDe will be given how much brightness and which lede will be driven from which channel, and the loop is completed and constantly waits to receive a new analog signal through the potentiometer.

–main.c–

```
#include "stm32g0xx.h"
#include "bsp.h"
int main() {
init_tim1();
init_tim3();
init_ADC();
init_PWM();
while (1) {}
return 0;
}
```

–bsp.h–

```
#ifndef INC_BSP_H_
#define INC_BSP_H_
#include "stm32g0xx.h"
void init_tim1(void);
void TIM1_BRK_UP_TRG_COM_IRQHandler(void);
void init_tim3(void);
void TIM3_IRQHandler(void);
void init_PWM(void);
void init_ADC(void);
#endif
```

–bsp.c–

```
#include "stm32g0xx.h"
#include "bsp.h"
static volatile int freq = 16;
void delay(volatile uint32_t s)
{
for (; s > 0; s--)
;
}
void init_tim1(void)
```

```
{
/* enable TIM1 module clock */
RCC->APBENR2 |= (1U << 11); //golden rule for TIM1 module
//I chose TIM1 module 16 bit - 128MHz
TIM1->CR1 = 0; //control register is 0 for just in case
TIM1->CR1 |= (1 << 7); //arpe register is buffered
TIM1->CNT = 0; // counter register is 0
TIM1->PSC = 9;
TIM1->ARR = 1000; // Auto reload register
TIM1->DIER |= (1U << 0); // dier register is 1 because update interrupt enable
TIM1->CR1 |= (1U << 0); //control register is 1
NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn, 0);
NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn);
}
void TIM1_BRK_UP_TRG_COM_IRQHandler(void)
{
double led1 = 0;
double led2 = 0;
uint16_t read_voltage = 0;
ADC1->CR |= (1U << 2); // ADSTART register enable
while (!(ADC1->ISR & (1U << 2)))
;
read_voltage = (uint16_t)ADC1->DR;
led1 = (read_voltage / 2); // rate calc
led2 = 1000 - led1;
if (led1 < 1){
led1 = 0;
}
else if (led1 > 999){ // arranges LEDs stuation
led1 = 1000;
}
if (led2 < 1){
led2 = 0;
}
TIM3->CCR1 = (uint32_t)(freq * (int)led1 / 1000); // changes led compare registers value
ch1 and ch2
TIM3->CCR2 = (uint32_t)(freq * (int)led2 / 1000);
TIM1->SR &= ~(1U << 0);
}
void init_tim3(void)
{
RCC->APBENR1 |= (1U << 1); //TIM3 enable
TIM3->CR1 = 0; //control register is 0 for just in case
TIM3->CR1 |= (1U << 7); //arpe register is buffered
TIM3->CNT = 0; // counter register is 0
TIM3->PSC = 999;
TIM3->ARR = (uint32_t)freq; // //auto reload register = frequency
TIM3->DIER |= (1U << 0); // update interrupt enable
// ch1
TIM3->CCMR1 |= (1U << 3); // output compare 1 mode preload enable
TIM3->CCMR1 |= (1U << 5); // pwm mode output compare mode enable 110 ->OC1M
```

```
[0110]
TIM3->CCMR1 |= (1U << 6);
TIM3->CCMR1 &= ~(1U << 16); // pwm1 mode enable OC1M enable
TIM3->CCMR1 &= ~(1U << 4);
TIM3->CCER |= (1U << 0); // ccer register capture compare 1 output enable
TIM3->CCR1 = 1000; // for duty cycle
TIM1->CNT = 1000000;
TIM3->CCR1 = (uint32_t)(freq / 2);
// ch2
TIM3->CCMR1 &= ~(0x7U << 12); // clear OCM2
TIM3->CCMR1 &= ~(1U << 24); // clear
TIM3->CCMR1 |= (1U << 11);
TIM3->CCMR1 |= (0x6U << 12); // PWM Mode OC2M [0110]
TIM3->CCER |= (1U << 4);
TIM3->CCR2 = (uint32_t)(freq / 2); // capture compare mode 2 for channel 2
TIM3->CR1 |= (1U << 0); // TIM3 enable
NVIC_SetPriority(TIM3_IRQn, 0); // TIM3 NVIC enable
NVIC_EnableIRQ(TIM3_IRQn);
}
void TIM3_IRQHandler(void)
{
TIM3->SR &= ~(1U << 0); // status register
}
void init_ADC(void)
{
RCC->IOPENR |= (1U << 0); // enable gpioa for analog mode
RCC->APBENR2 |= (1U << 20); //enable rcc for adc
GPIOA->MODER &= ~(3U << 2 * 7); //pa7 as analog mode
GPIOA->MODER |= (3U << 2 * 7);
ADC1->CR = 0; //reset control register
ADC1->CFGR1 = 0;// reset cfgr1
ADC1->CR |= (1U << 28); //enable read_voltage regulator
delay(100);
ADC1->CR |= (1U << 31); //enable calibration
while (ADC1->CR & (1U << 31)) //wait for EOCAL == 1
;
ADC1->CFGR1 = (0U << 3); // 12 bits
ADC1->CFGR1 |= (1U << 13); // cont =0
ADC1->ISR &= ~(1U << 0); //clear the adrdy bit
ADC1->CR |= (1U << 0); //enable adc
while (ADC1->ISR & (1 << 0))
;
ADC1->CHSELR |= (1U << 7); //analog input channel 7 selection
ADC1->SMPR |= (2U << 4);
NVIC_SetPriority(ADC1_IRQn, 2);
NVIC_EnableIRQ(ADC1_IRQn);
ADC1->CR |= (1U << 2); // ADC start conversion command enable
}
void init_PWM(void)
{
RCC->IOPENR |= (1U << 1); //enable gpiob clock
```

```
GPIOB->MODER &= ~(3U << 2 * 4); //pb4
GPIOB->MODER |= (2U << 2 * 4);
GPIOB->AFR[0] &= ~(0xFU << 4 * 4); //PB4 AF1 = TIM3_CH1
GPIOB->AFR[0] |= (1U << 4 * 4);
GPIOB->MODER &= ~(3U << 2 * 5); //pb5
GPIOB->MODER |= (2U << 2 * 5);
GPIOB->AFR[0] &= ~(0xFU << 4 * 5); //PB5 AF1 = TIM3_CH2
GPIOB->AFR[0] |= (1U << 4 * 5);
}
```

Problem 2

In this problem, you will work on implementing aknock counter.
•Connect SSD that will show the number of knocks.
•Connect an external button that will reset the counter.
•Connect a microphone that will pick up the sounds.
•When you knock on the table, you should increment thecounter by one.
•There should be no mis-increments, or multiple incrementsas much as possible.
•You may include an IMU sensor readings to improve youraccuracy, but still need to incorporate ADC samples.

Solution 2

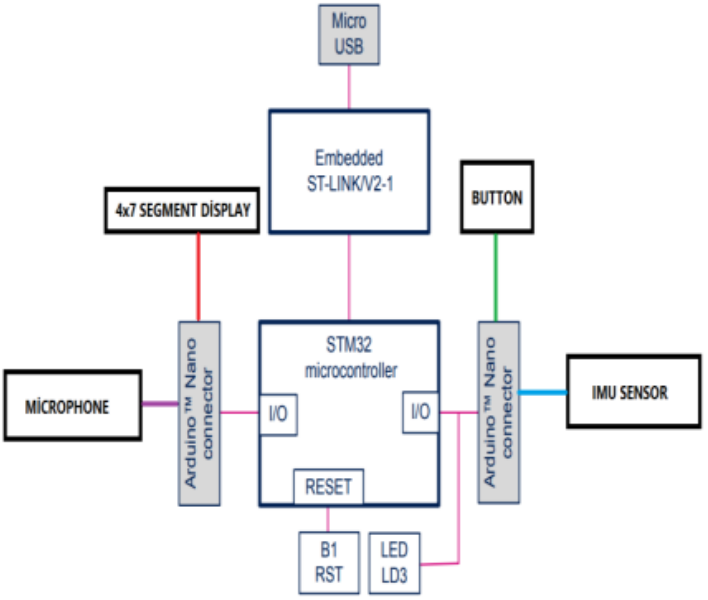

Figure 4. Block Diagram

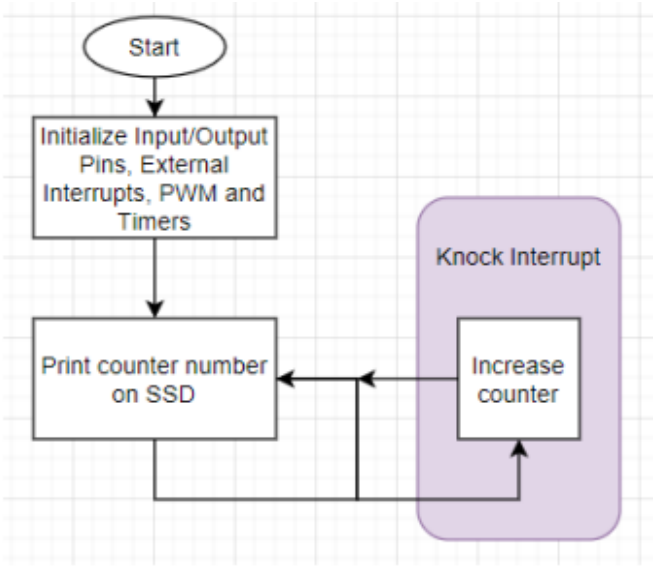

Figure 5. Flowchart

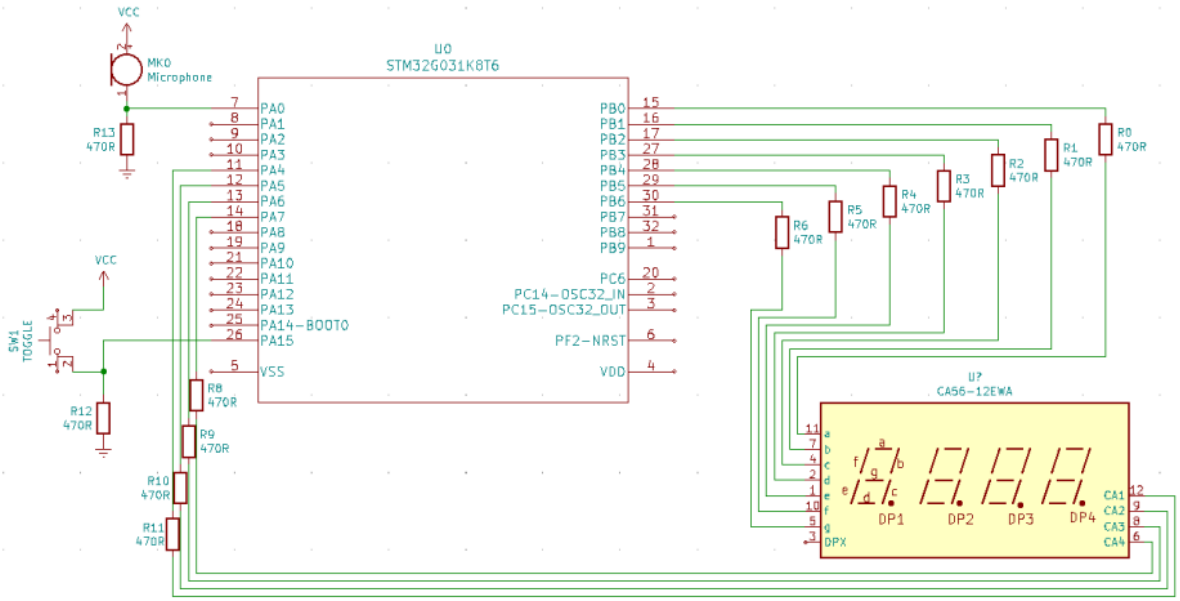

Figure 6. Schematic

```
#include "stm32g0xx.h"
volatile uint32_t delaycount = 0; /* Global delay counter */
volatile uint32_t counter = 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;
}
```