GEBZE TECHNICAL UNIVERSITY

ELECTRONICS ENGINEERING DEPARTMENT

ELEC 458 - Embedded System Design

Spring 2019

Project 2-Waveform Generator

Group 3

Ali Fırat Arı - 131024074

İrfan Bilaloğlu - 151024095

Mehmet Sencer Altuntop – 141024065

Table of Contents

Introduction 3

Block diagram 3

Software Flowchart 3

Design overview 5

System photo 5

Conclusion 6

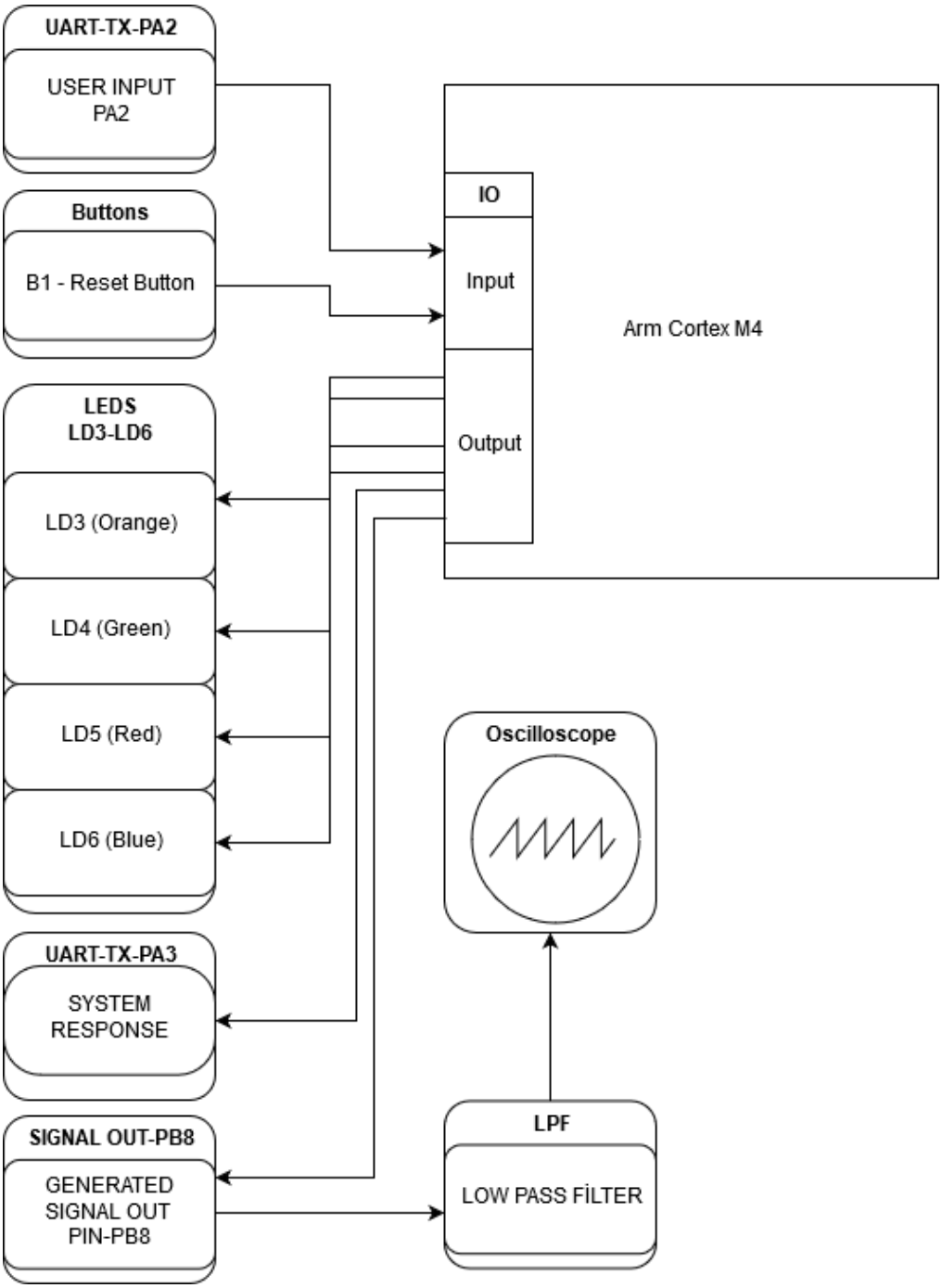
Grade sheet 6

Appendix 11

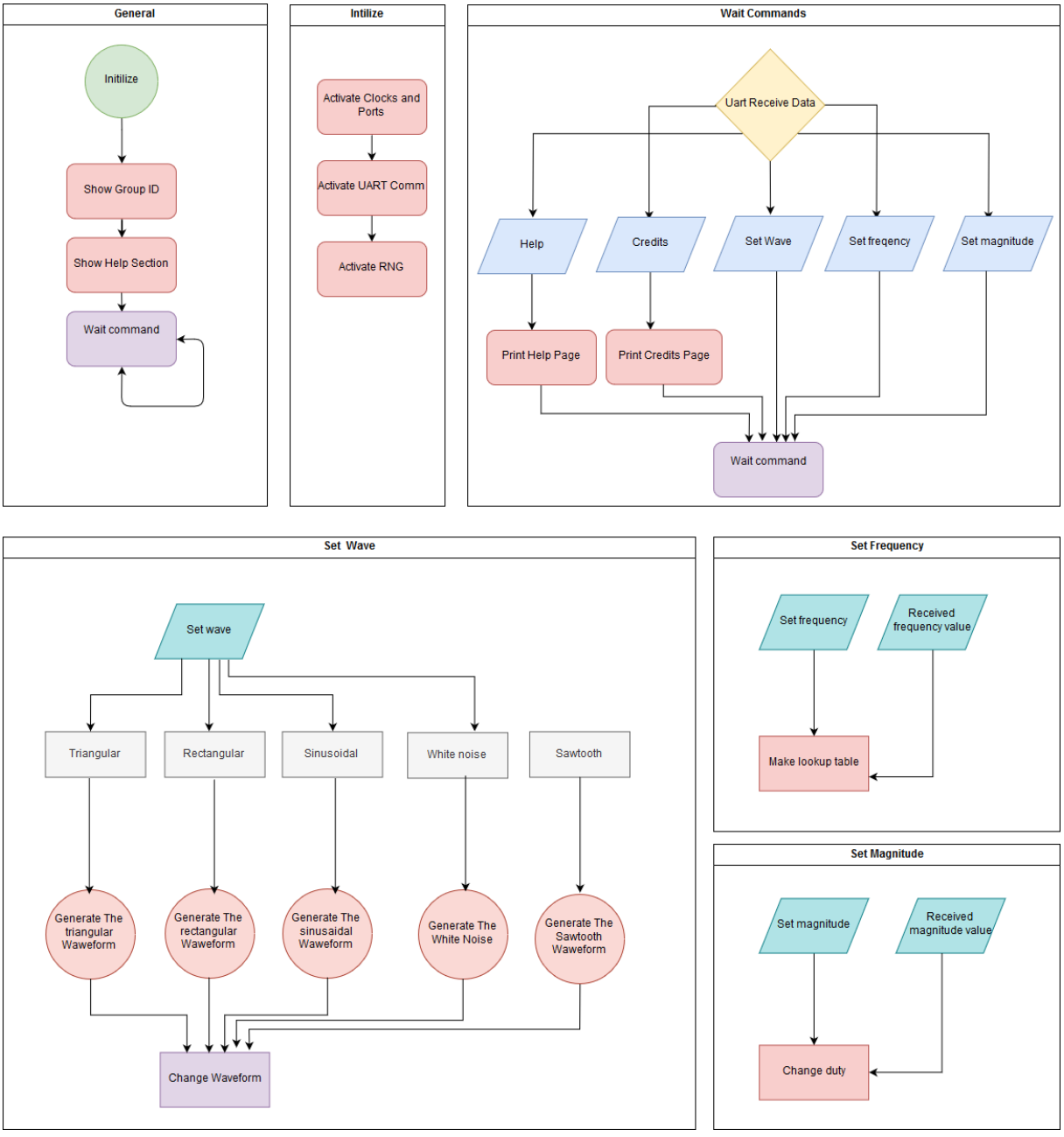
Introduction

In this project, we have designed a waveform generator that can produce sine, square, triangular, sawtooth (ramp) and white noise waveforms. That can be observe by using oscilloscope. We used C language as it is expected. System interface design supposed first ask for the waveform, then frequency, then amplitude. Once all three are selected, it should output the waveform, and return to the beginning.

Block diagram



Software Flowchart



Design overview

First of all, we tried to reach the maximum frequency value that we can produce from our board STM32f407. We achieved to generate 2 Ghz PWM by using Advance Timer which has high periperial clock.

Then we tried to produce wave forms by using the PWM with the aid of interrupt. We used the RNG (Random Number Generator) module to generate the White Noise. In this step to get the best waveform from the oscilloscope we designed and simulate the Low-Pass Filter (Given below).

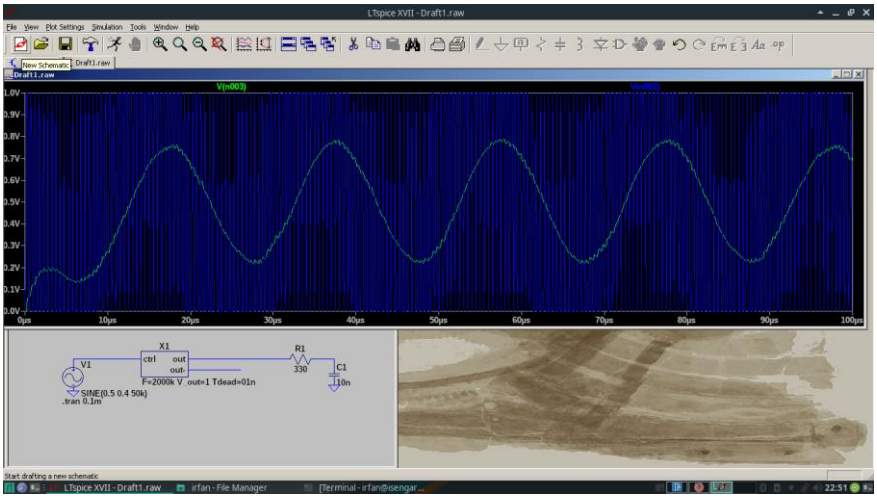


Figure.1 (LPF Simulation)

Then we noticed that maximum frequency of the system was not well enough. We figured out that it was happening because of code that works in an interrupt was working to slow (long period of time). To handle that problem, we made a look up tables for the signals in order to shorten the time period and that worked we achieved to generate signals with expected spacing. Because of the frequency dependent to look up tables in the end of the process we were able to change the frequency by look up tables.

After that managed to produce expected amplitude levels by changing the PWM's duty cycle. We noticed that although theoretically we are able to generate over 200K signals we were having problem with it. Then we also noticed that even if we shortened the interrupt duration with look up table the problem still the duty cycle was taking too long. İn this part we optimized our code by changing optimization to "2" from compiler so duty cycle shortened again after that we were able to see over 200Khz signals.

After all this sections one by one unit-tests in order to let the end user enter the command we have created a UART terminal. This command lets the user to change frequency and set waveform by changing the lookup table, to change the amplitude it changes the duty cycle.

System photo

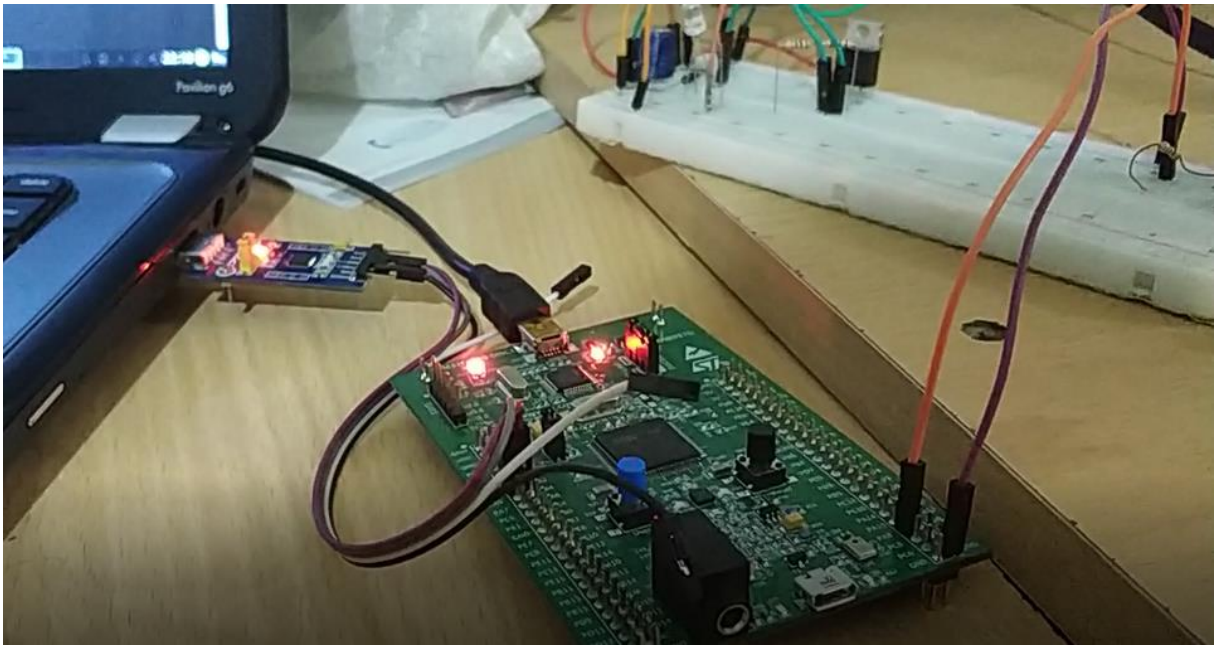


Figure.2 (System Photo)

System outputs

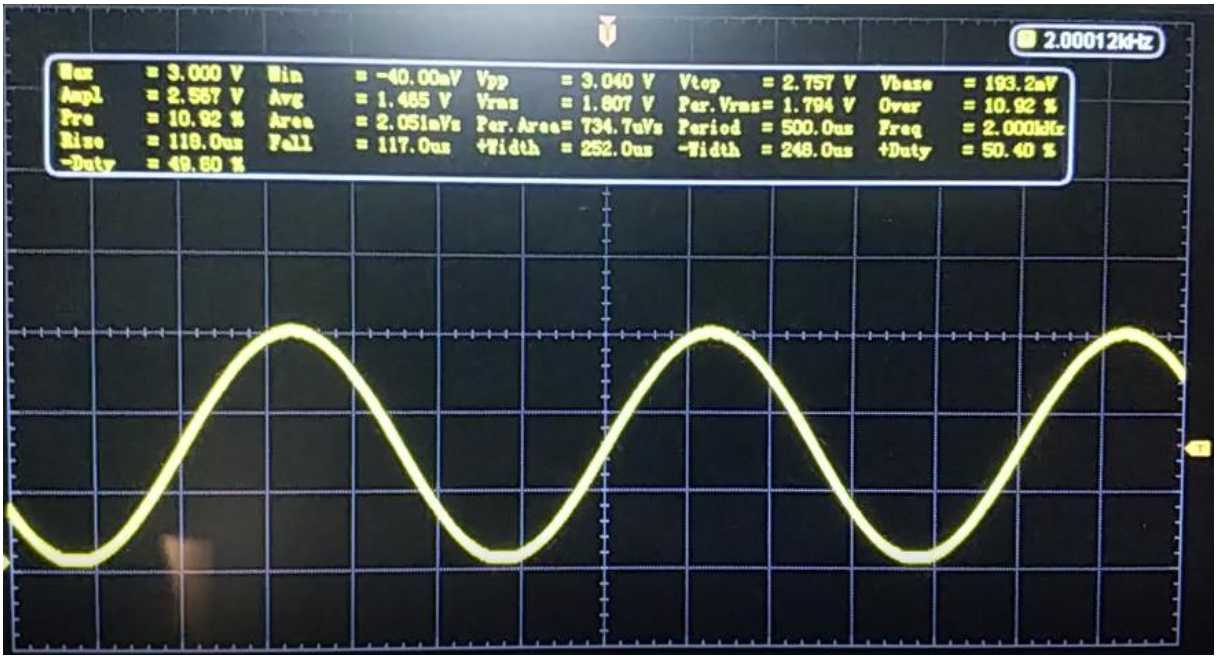


Figure.3 Sinusoidal wave (2Khz)

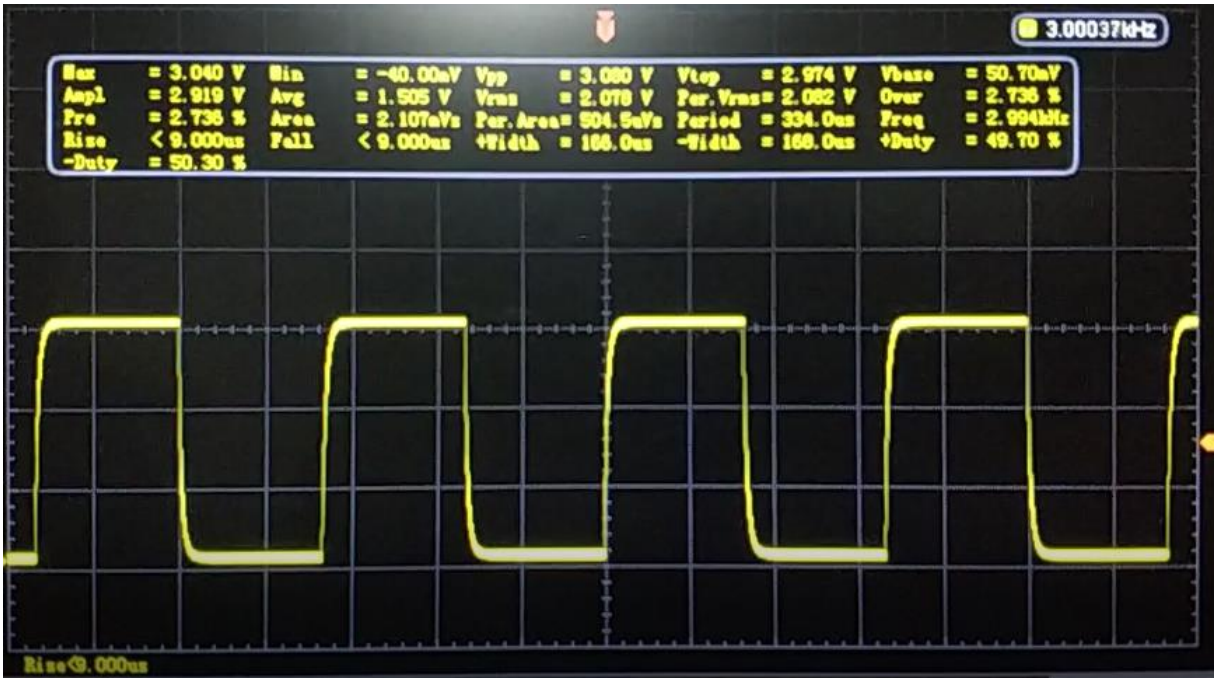


Figure.4 Rectangular wave (~3Khz)

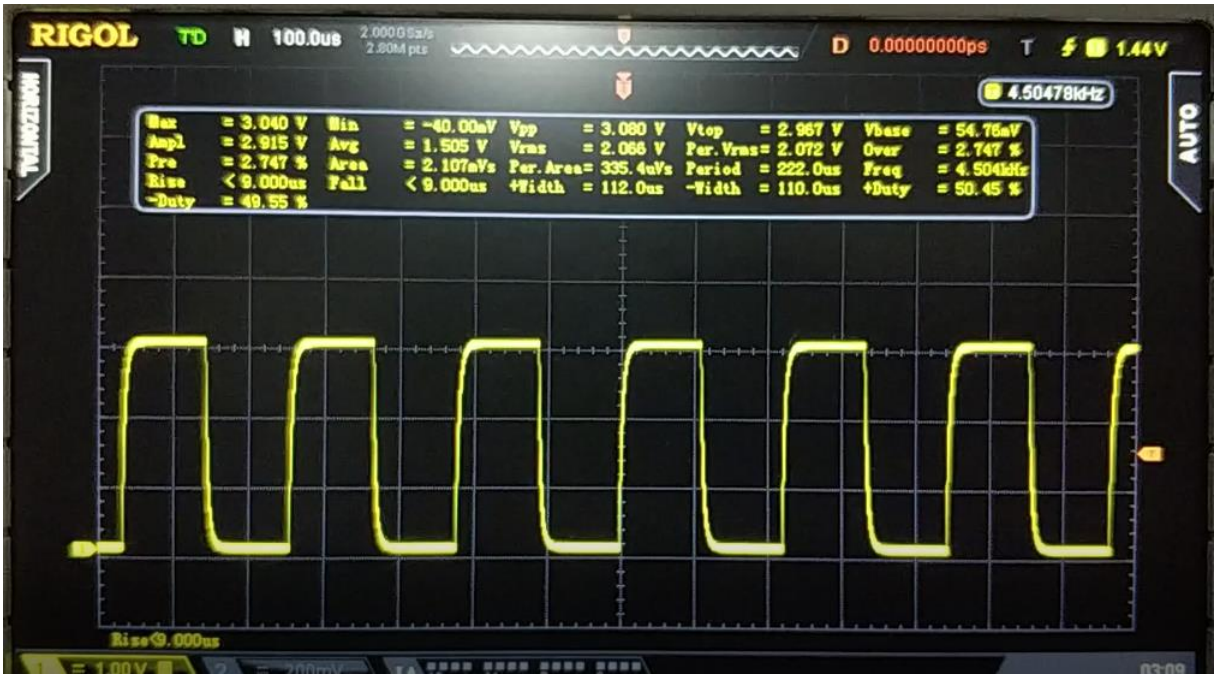


Figure.5 Rectangular wave (~4.5Khz)

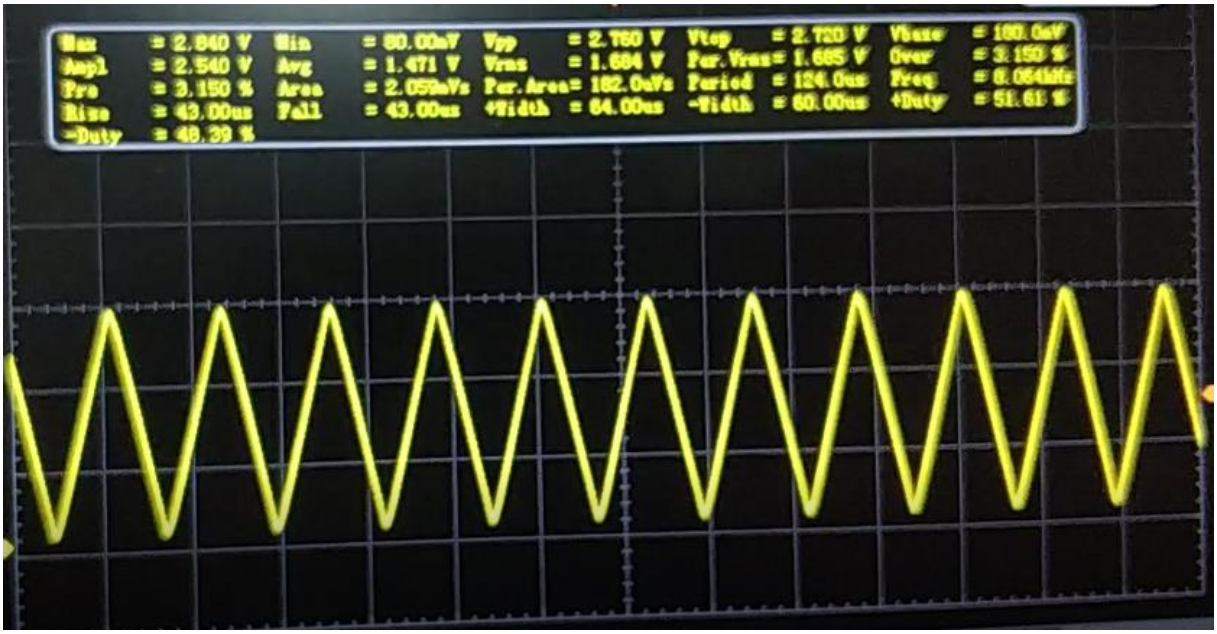


Figure.6 Triangular wave (~8Khz)

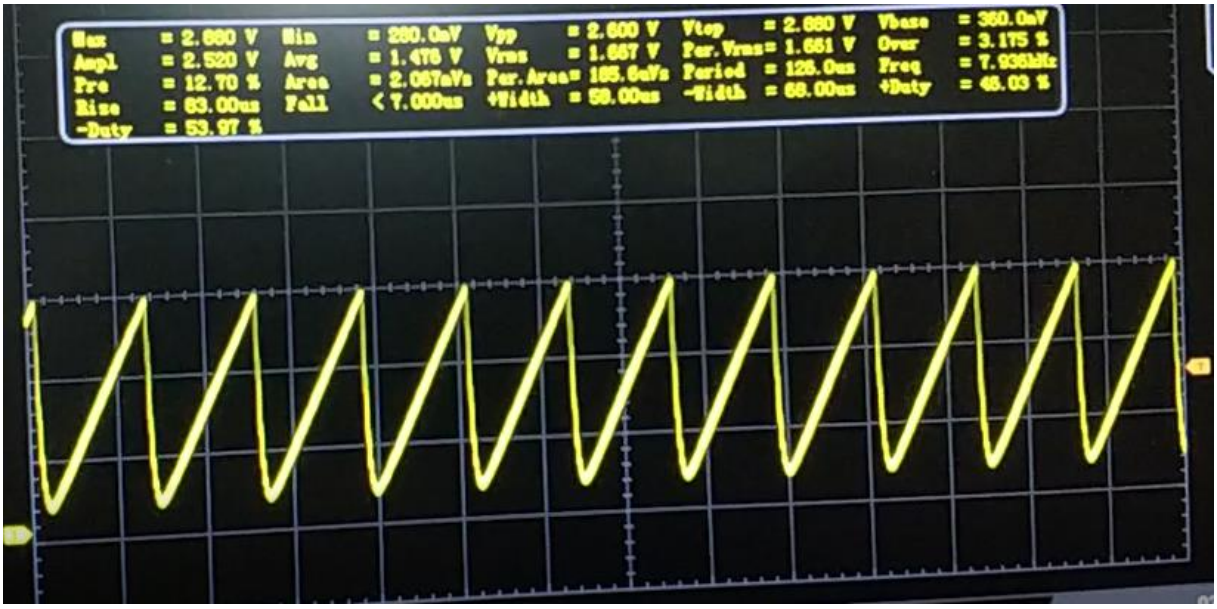


Figure.7 Sawtooth wave (~8Khz)

If you require any further information, video of the system has been attached to the file.

Conclusion

During the project we had chance to experiencing many thinks some listed below;

- Error handling,
- Interrupt Priority,
- Advance timer usage,
- Observing the benefits of optimization,
- PWM and UART, RNG experience,
- Waveform characteristics and measurements,
- **A lot more ☺**

Project lasted about 3-4 weeks although we had a problem after the we finished we thought that it's about our code, it took about a week to figure that out. After we checked everything working well, we noticed that because of we changed the trigger settings of the oscilloscope we were getting signal with 90 degree shifted version on screen at the same time.

Grade sheet

Appendix

Generator.h

```
/**
  ******************************************************************************
  * @file    generator.h
  * @author  Ali Firat ARI
  * @author  Irfan Bilaloglu
  * @author  Sencer Altintop
  * @version V1.0.1
  * @brief   Header for generator.c module
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; COPYRIGHT(c) 2019 Gebze Technical University</center></h2>
  *
  ******************************************************************************
  */


/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __GENERATOR_H
#define __GENERATOR_H

/************************************************
* Constants
************************************************/
#define SIN_WAVE 1
#define SQUARE_WAVE 2
#define TRIANGLE_WAVE 3
#define SAWTOOTH_WAVE 4
#define NOISE_WAVE 5

#define LEDDELAY 1000000
#define BLINK_COUNT      100

/************************************************
* function declarations
************************************************/
void Default_Handler(void);
void make_luts(int _wave_period);
void set_wave_type(int w_type);
void calculate_freq(uint32_t freq);
void sin_generate(void);
void sqr_generate(void);
void sawtooth_generate(void);
void triangle_generate(void);
void white_noise_generate(void);
void tim1_CCI(void);
void tim1_UI(void);
void basic_delay(volatile uint32_t s);
void NMI_Handler(void);

#endif
```

Terminal.h

```
/**
  ******************************************************************************
  * @file    terminal.h
  * @author  Ali Firat ARI
  * @author  Irfan Bilaloglu
  * @author  Sencer Altintop
  * @version V1.0.1
  * @brief   Header for terminal.c module
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; COPYRIGHT(c) 2019 Gebze Technical University</center></h2>
  *
  ******************************************************************************
  */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __TERMINAL_H
#define __TERMINAL_H

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx.h"
```

```
#include "system_stm32f4xx.h"
#include "math.h"

/* Exported functions ------------------------------------------------------- */
void Systick_Handler(void);

void Init_Systick(uint32_t s,uint8_t cen);
void delay_ms(volatile uint32_t s);

void UART2_Handler(void);
void __Clear(void);

void Init_Uart(void);
void UART_SendData(const char* str);
void Send_Info(void);
void Send_Help(void);

void Send_GroupId(uint8_t id);

int powInt(int x, int y);
int parseInt(char* chars);

#endif /* __TERMINAL_H */
```

Terminal.c

```
/**
  ******************************************************************************
  * @file    terminal.c
  * @author  Ali Firat ARI
  * @author  Irfan Bilaloglu
  * @author  Sencer Altintop
  * @version V1.0.1
  * @brief  Manage waveform genarator with USART
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; COPYRIGHT(c) 2019 Gebze Technical University</center></h2>
  *
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/

#include "stdio.h"
#include <stdlib.h>
#include "string.h"
#include "terminal.h"
#include "generator.h"


/* Private definitions ---------------------------------------------------------*/
#define MAX_FREQ 500000
#define MIN_FREQ 2000

#define MAX_MAGNITUDE 100
#define MIN_MAGNITUDE 0

/* Private variables ---------------------------------------------------------*/
static volatile uint32_t tDelay;
char rx_buffer[31] = {0};
char rx_temp[3] = {0};
int rx_done = 0;


void Systick_Handler(void)
{
    if (tDelay != 0x00)
    {
        tDelay--;
    }
}
void __Clear(void)
{
         for(int i = 0 ; i < 31 ; i++)
                 rx_buffer[i] = 0;
}


void UART2_Handler(void)
{
```

```
    if (((USART2->SR) >> 5) & 0x01)
    {
        static uint8_t i = 0;
        char data = (uint8_t) (USART2->DR & 0x000000FF);

        if(((data != '\n') && (data != '\n') && (data != 13) && (data != 10) ) && (i < 30))
        {
                        if (data == 8){
                                rx_buffer[i] = 0;
                                i--;
                                UART_SendData("\b");
                        }
                        else
                        {
                                rx_buffer[i++] = data;
                                rx_temp[0] = data;
                                UART_SendData(rx_temp);

                        }

        } else {
            rx_buffer[i] = '\0';
            USART2->SR &= 0xFFFFFFDF;

            if (strcmp(rx_buffer,"") == 0)
            {

            }

            i = 0;
                        UART_SendData("\r\n");
                        if ((strcmp(rx_buffer,"")==0) || (strcmp(rx_buffer,'\r')==0) ||
                                (strcmp(rx_buffer,'\n')==0) || (strcmp(rx_buffer,'\n\r')==0) ||
                                (strcmp(rx_buffer,'\r\n')==0) || (strcmp(rx_buffer, 13)==0) ||
(strcmp(rx_buffer, 10)==0) )
                                {

                                }
                        ////////////////////////CREDITS AND HELP//////////////////////////////
                        else if ((strcmp(rx_buffer,"help")==0) || (strcmp(rx_buffer,"h")==0)) {
                                Send_Help();
                                }
                        else if ((strcmp(rx_buffer,"credits")==0) || (strcmp(rx_buffer,"c")==0)) {
                                UART_SendData("Credits");
                                Send_Info();
                                }
                        else if ((strcmp(rx_buffer,"led group")==0) || (strcmp(rx_buffer,"l")==0)) {
                                UART_SendData("Showing group id with leds");
                                Send_GroupId(3);
                                }
                        ////////////////////////SET WAVEFORM//////////////////////////////
                        else if ((strcmp(rx_buffer,"set wave sine")==0) || (strcmp(rx_buffer,"s w
sin")==0)) {
                set_wave_type(SIN_WAVE);
                                UART_SendData("Cofiguration has been setting sinus waveform");
                                }
                        else if ((strcmp(rx_buffer,"set wave square")==0) || (strcmp(rx_buffer,"s w
squ")==0)) {
                set_wave_type(SQUARE_WAVE);
                                UART_SendData("Cofiguration has been setting square waveform");
                                }
                        else if ((strcmp(rx_buffer,"set wave triangular")==0) || (strcmp(rx_buffer,"s w
tri")==0)) {
                set_wave_type(TRIANGLE_WAVE);
                                UART_SendData("Cofiguration has been setting triangular waveform");
                                }
                        else if ((strcmp(rx_buffer,"set wave sawtooth")==0) || (strcmp(rx_buffer,"s w
saw")==0)) {
                set_wave_type(SAWTOOTH_WAVE);
                                UART_SendData("Cofiguration has been setting sawtooth waveform");
                                }
                        else if ((strcmp(rx_buffer,"set wave whitenoise")==0) || (strcmp(rx_buffer,"s w
noi")==0)) {
                                set_wave_type(NOISE_WAVE);
                                UART_SendData("Cofiguration has been setting whitenoise waveform");
                        }

                        ////////////////////////SET MAGNITUDE//////////////////////////////
                        else if ((strncmp(rx_buffer,"set magnitude", 13 )==0) || (strncmp(rx_buffer,"s
m", 3)==0)) {
```

```
                                if (((strlen(rx_buffer) == 3 || strlen(rx_buffer) == 4) &&
(strncmp(rx_buffer,"s m", 3)==0) ) ||
                                ((strlen(rx_buffer) == 13 || strlen(rx_buffer) == 14) &&
(strncmp(rx_buffer,"set magnitude", 13)==0) ))
                                )
                                {
                                        UART_SendData("Please give a magnitude level.");
                                }
                                else if
                                (!(     ((strlen(rx_buffer) > 14 && (strlen(rx_buffer) < 18)) &&
(strncmp(rx_buffer,"set magnitude", 13)==0)) ||
                                                ((strlen(rx_buffer) > 4 && (strlen(rx_buffer) < 8)) &&
(strncmp(rx_buffer,"s m", 3)==0))
                                        ))
                                {
                                        UART_SendData("Please give a correct level");
                                }

                                else if (
                                                ((strlen(rx_buffer) > 14 && (strlen(rx_buffer) < 18))
&& (strncmp(rx_buffer,"set magnitude", 13)==0)) ||
                                                ((strlen(rx_buffer) > 4 && (strlen(rx_buffer) < 8)) &&
(strncmp(rx_buffer,"s m", 3)==0))
                                                )
                                {

                                        if (strncmp(rx_buffer,"s m", 3)==0)
                                        {
                                                char *c_magnitude = NULL;
                                                c_magnitude = strtok(rx_buffer, "s m");
                                                int magnitude = parseInt(c_magnitude);

                                                if ( ( (MAX_MAGNITUDE + 1) > magnitude ) && (
(MIN_MAGNITUDE) < magnitude) )
                                                {
                                                        UART_SendData("Correct magnitude level value.
");
                                                        UART_SendData(c_magnitude);
                        calculate_amp((uint16_t)(magnitude));
                                                }
                                                else
                                                {
                                                        UART_SendData("Not correct magnitude level
value. ");
                                                        UART_SendData(c_magnitude);
                                                }
                                        }

                                        else if (strncmp(rx_buffer,"set magnitude", 13)==0)
                                        {
                                                char *c_magnitude = NULL;
                                                c_magnitude = strtok(rx_buffer, "set magnitude");
                                                int magnitude = parseInt(c_magnitude);

                                                if ( ( (MAX_MAGNITUDE + 1) > magnitude ) && (
(MIN_MAGNITUDE) < magnitude) )
                                                {
                                                        UART_SendData("Correct magnitude level value.
");
                                                        UART_SendData(c_magnitude);
                                                        calculate_amp((uint16_t)(magnitude));
                                                }
                                                else
                                                {
                                                        UART_SendData("Not correct magnitude level
value. ");
                                                        UART_SendData(c_magnitude);
                                                }
                                        }

                                }
                                else {
                                        UART_SendData("Unknown error!");
                                }

                        }
                        ////////////////////////SET FREQ//////////////////////////////
                        else if ((strncmp(rx_buffer,"set frequency", 13 )==0) || (strncmp(rx_buffer,"s
f", 3)==0)) {

                                if (((strlen(rx_buffer) == 3 || strlen(rx_buffer) == 4) &&
(strncmp(rx_buffer,"s f", 3)==0) ) ||
```

```
                                (((strlen(rx_buffer) == 13 || strlen(rx_buffer) == 14) &&
(strncmp(rx_buffer,"set frequency", 13)==0) ))
                                )
                                {
                                        UART_SendData("Please give a frequency.");
                                }
                                else if
                                (!(     ((strlen(rx_buffer) > 14 && (strlen(rx_buffer) < 22)) &&
(strncmp(rx_buffer,"set frequency", 13)==0)) ||
                                        ((strlen(rx_buffer) > 4 && (strlen(rx_buffer) < 12))
&& (strncmp(rx_buffer,"s f", 3)==0))
                                        ))
                                {
                                        UART_SendData("Please give a correct frequency");
                                }

                                else if (
                                        ((strlen(rx_buffer) > 14 && (strlen(rx_buffer) < 22))
&& (strncmp(rx_buffer,"set frequency", 13)==0)) ||
                                        ((strlen(rx_buffer) > 4 && (strlen(rx_buffer) < 12))
&& (strncmp(rx_buffer,"s f", 3)==0))
                                        )
                                {

                                        if (strncmp(rx_buffer,"s f", 3)==0)
                                        {
                                                char *c_frequency = NULL;
                                                c_frequency = strtok(rx_buffer, "s f");
                                                int frequency = parseInt(c_frequency);

                                                if ( ( (MAX_FREQ + 1) > frequency ) && ( (MIN_FREQ -
1) < frequency) )
                                                {
                                calculate_freq(frequency);
                                                        UART_SendData("Correct frequency value. ");
                                                        UART_SendData(c_frequency);
                                                }
                                                else
                                                {
                                                        UART_SendData("Not correct frequency value.
");
                                                        UART_SendData(c_frequency);
                                                }

                                        }
                                        else if (strncmp(rx_buffer,"set frequency", 13)==0)
                                        {
                                                char *c_frequency = NULL;
                                                c_frequency = strtok(rx_buffer, "set frequency");
                                                int frequency = parseInt(c_frequency);

                                                if ( ( (MAX_FREQ + 1) > frequency ) && ( (MIN_FREQ -
1) < frequency) )
                                                {
                                calculate_freq(frequency);
                                                        UART_SendData("Correct frequency value. ");
                                                        UART_SendData(c_frequency);
                                                }
                                                else
                                                {
                                                        UART_SendData("Not correct frequency value.
");
                                                        UART_SendData(c_frequency);
                                                }
                                        }

                                }
                                else {
                                        UART_SendData("Unknown error!");
                                }

                        }
                        ////////////////////////NEW COMMAND//////////////////
                        //else if ((strncmp(rx_buffer,"new command", 13 )==0) || (strncmp(rx_buffer,"s
f", 3)==0)) {
                        //
                        //}
                        ////////////////////////////////////////////////////
                        else {
                                UART_SendData("Unknown command");
                        }
                        UART_SendData("\r\n");
```

```
        }
    }
}


void Init_Systick(uint32_t s,uint8_t cen)
{
    //Clear CTRL register
    SysTick->CTRL = 0x00000;
    /*Main clock source is runnig with HSI by default which is at 8 Mhz.
     *Systick clock sorunce can be sat with CTRL register's second bit
     * 0: Processor clock/8 (AHB/8)
     * 1: Processor clock
    */
   SysTick->CTRL |= (0 << 2);
   //Enable syscallback
   SysTick->CTRL |= ((uint32_t)cen << 1);
   //Load value
   SysTick->LOAD = s;
   //Set the Current Value to 0
   SysTick->VAL  = 0;
   //Enable SysTick bit0
   SysTick->CTRL |= (1 << 0);
}

void Init_Uart(void)
{
    RCC->APB1ENR |= (1 << 17); //Enable clock for USART2
    RCC->AHB1ENR |= (1 << 0);  //USART2 is connected to GPIOA, enable GPIOA clock

    //set pins as alternate func (2nd and 3rd pins)
    GPIOA->MODER &= 0xFFFFFF0F; // Reset bits 10-15 to clear old values
    //GPIOA->MODER |= 0x000000A0; // Set 2nd and 3rd pins as alternate func mode.
        GPIOA->MODER |= (2 << 4); // Set 2nd and 3rd pins as alternate func mode.
        GPIOA->MODER |= (2 << 6); // Set 2nd and 3rd pins as alternate func mode.
    //USART pins speed are high
    //GPIOD->OSPEEDR |= 0x000000A0;
        GPIOA->OSPEEDR |= (3 << 4); // Set pin2 to very high speed
        GPIOA->OSPEEDR |= (3 << 6); // Set pin3 to very high speed

    //AF7 for USART2 in alternate func register
    GPIOA->AFR[0] |= (0x7 << 8);  // for pin 2
    GPIOA->AFR[0] |= (0x7 << 12); // for pin 3

    /*
     *USART2 word length M,bit 12
     *USART2->CR |= (0 << 12); // 0 - 1,8,n
    */

   USART2->CR1 |= (1 << 3);    //USART2_Tx enable, bit 3
   USART2->CR1 |= (1 << 2);    //USART2_Rx enable bit 2

   //set Rx Not Enable Ä°nterrupt Enable (RXNEIE)
   USART2->CR1 |=(1 << 5);

   NVIC_SetPriority(USART2_IRQn,1);
   NVIC_EnableIRQ(USART2_IRQn);
   /*
    * Baund_rate = fCk / (8 * (2 - OVER8) * USARTDIV)
    * Forc fCk = 42 Mhz, Baund = 115200, OVER8 = 0
    * USARTDIV = 42 Mhz / 115200 / 16 = 22.7865
    * We can alsa look at the table and 115.2 KBps baud
    * we need to set 22.8125
    * Fraction :16*0.8125 = 13
    * Mantissa : 22
    * 12-bit mantissa and 4-bit fraction
    */
   USART2->BRR |= (22 << 4);
   USART2->BRR |= 13;

   //Enable USART2
   USART2->CR1 |= (1 << 13);
}

void UART_SendData(const char* str)
{
    for (uint32_t i=0; i<strlen((const char*)str); i++)
    {
        //Send Data
        USART2->DR = str[i];
        //wait for transmi complate ,sixth bit of SR, TC
        while(!(USART2->SR & (1 << 6)));
```

```
    }
}

void Send_Info(void)
{
    const char* str = "\n\n\r\
    *************Gebze Technical University****************\n\n\r\
         Electronics Engineering ELEC458 \n\n\r\
         Embedded System Design\n\n\r\
                                 Ali Firat ARI 131024074\n\n\r\
                                 Irfan Bilaloglu 151024095\n\n\r\
                                 Sencer Altintop 141024065\n\n\r\
     ************Wave Generator ***********   \n\n\r";
    UART_SendData(str);
}

void Send_Help(void)
{
    const char* help_str =
         "\n\r\
         help                                : h                          : Shows help page \n\r\
         credits                             : c                          : Shows credits page \n\r\
         led group                           : l                          : Shows group id with leds \n\r\
         set wave sine                       : s w sin                    : Change waveform to sinus \n\r\
         set wave square                     : s w squ                    : Change waveform to square \n\r\
         set wave triangular                 : s w tri                    : Change waveform to triangular
\n\r\
         set wave sawtooth                   : s w saw                    : Change waveform to sawtooth
\n\r\
         set wave whitenoise                 : s w noi                    : Change waveform to whitenoise
\n\r\
         set magnitude  <0-100>              : s m <0-100>                : Set magnitude level \n\r\
         set frequency  <0-100000000>        : s f <0-100000000>          : Set frequency \n\r\
         ";
    UART_SendData(help_str);
}


void Send_GroupId(uint8_t id)
{
         if ((1 <= id) && (id <= 15))
         {
                  RCC->AHB1ENR |= (1 << 3);
                  //GPIOD->MODER &= 0x00FFFFFF;   // Reset bits 31-24 to clear old values
                  GPIOD->MODER |= 0x55000000;
                  GPIOD->ODR  = (uint16_t)(id << 12);
                  delay_ms(2000);
                  GPIOD->ODR  = (uint16_t)(0 << 12);
}
         else
                  printf("Error! \n The Group ID is out of range (1 to 15)");

}

void delay_ms(volatile uint32_t s)
{
         tDelay = s;
         while(tDelay != 0);
}

int parseInt(char* chars)
{
    int sum = 0;
    int len = strlen(chars);
    for (int x = 0; x < len; x++)
    {
        int n = chars[len - (x + 1)] - '0';
        sum = sum + powInt(n, x);
    }
    return sum;
}

int powInt(int x, int y)
{
    for (int i = 0; i < y; i++)
    {
        x *= 10;
    }
    return x;
}
```

Generator.c

```
/**
  ******************************************************************************
  * @file    generator.c
  * @author  Ali Firat ARI
  * @author  Irfan Bilaloglu
  * @author  Sencer Altintop
  * @version V1.0.1
  * @brief  Waveform genarator
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; COPYRIGHT(c) 2019 Gebze Technical University</center></h2>
  *
  ******************************************************************************
  */


#include "stm32f4xx.h"
#include "system_stm32f4xx.h"
#include <math.h>
#include "generator.h"
#include "terminal.h"


/*************************************************
* Vector Table
*************************************************/
// get the stack pointer location from linker
typedef void (* const intfunc)(void);
extern unsigned long __stack;

// attribute puts table in beginning of .vectors section
//   which is the beginning of .text section in the linker script
// Add other vectors -in order- here
// Vector table can be found on page 372 in RM0090
__attribute__ ((section(".vectors")))
void (* const vector_table[])(void) = {
        (intfunc)((unsigned long)&__stack), /* 0x000 Stack Pointer */
        Reset_Handler,                      /* 0x004 Reset         */
        Default_Handler,                    /* 0x008 NMI           */
        Default_Handler,                    /* 0x00C HardFault     */
        Default_Handler,                    /* 0x010 MemManage     */
        Default_Handler,                    /* 0x014 BusFault      */
        Default_Handler,                    /* 0x018 UsageFault    */
        0,                                  /* 0x01C Reserved      */
        0,                                  /* 0x020 Reserved      */
        0,                                  /* 0x024 Reserved      */
        0,                                  /* 0x028 Reserved      */
        Default_Handler,                    /* 0x02C SVCall        */
        Default_Handler,                    /* 0x030 Debug Monitor */
        0,                                  /* 0x034 Reserved      */
        Default_Handler,                    /* 0x038 PendSV        */
        Systick_Handler,                    /* 0x03C SysTick       */
        0,                                  /* 0x040 Window WatchDog Interrupt
*/
        0,                                  /* 0x044 PVD through EXTI Line detection Interrupt
*/
        0,                                  /* 0x048 Tamper and TimeStamp interrupts through the EXTI
line             */
        0,                                  /* 0x04C RTC Wakeup interrupt through the EXTI line
*/
        0,                                  /* 0x050 FLASH global Interrupt
*/
        0,                                  /* 0x054 RCC global Interrupt
*/
        0,                                              /* 0x058 EXTI Line0 Interrupt
*/
        0,                                  /* 0x05C EXTI Line1 Interrupt
*/
        0,                                  /* 0x060 EXTI Line2 Interrupt
*/
        0,                                  /* 0x064 EXTI Line3 Interrupt
*/
        0,                                  /* 0x068 EXTI Line4 Interrupt
*/
        0,                                  /* 0x06C DMA1 Stream 0 global Interrupt
*/
```

```
        0,                                  /* 0x070 DMA1 Stream 1 global Interrupt
*/
        0,                                  /* 0x074 DMA1 Stream 2 global Interrupt
*/
        0,                                  /* 0x078 DMA1 Stream 3 global Interrupt
*/
        0,                                  /* 0x07C DMA1 Stream 4 global Interrupt
*/
        0,                                  /* 0x080 DMA1 Stream 5 global Interrupt
*/
        0,                                  /* 0x084 DMA1 Stream 6 global Interrupt
*/
        0,                                  /* 0x088 ADC1, ADC2 and ADC3 global Interrupts
*/
        0,                                  /* 0x08C CAN1 TX Interrupt
*/
        0,                                  /* 0x090 CAN1 RX0 Interrupt
*/
        0,                                  /* 0x094 CAN1 RX1 Interrupt
*/
        0,                                  /* 0x098 CAN1 SCE Interrupt
*/
        0,                                  /* 0x09C External Line[9:5] Interrupts
*/
        0,                                  /* 0x0A0 TIM1 Break interrupt and TIM9 global interrupt
*/
        tim1_UI,                            /* 0x0A4 TIM1 Update Interrupt and TIM10 global interrupt
*/
        0,                                  /* 0x0A8 TIM1 Trigger and Commutation Interrupt and TIM11
global interrupt */
        0,                                          /* 0x0AC TIM1 Capture Compare Interrupt
*/
        0,                                           /* 0x0B0 TIM2 global Interrupt
*/
        0,                                  /* 0x0B4 TIM3 global Interrupt
*/
        0,                                              /* 0x0B8 TIM4 global Interrupt
*/
        0,                                  /* 0x0BC I2C1 Event Interrupt
*/
        0,                                  /* 0x0C0 I2C1 Error Interrupt
*/
        0,                                  /* 0x0C4 I2C2 Event Interrupt
*/
        0,                                  /* 0x0C8 I2C2 Error Interrupt
*/
        0,                                  /* 0x0CC SPI1 global Interrupt
*/
        0,                                  /* 0x0D0 SPI2 global Interrupt
*/
        0,                                  /* 0x0D4 USART1 global Interrupt
*/
        UART2_Handler,                      /* 0x0D8 USART2 global Interrupt
*/
        0,                                  /* 0x0DC USART3 global Interrupt
*/
        0,                                  /* 0x0E0 External Line[15:10] Interrupts
*/
        0,                                  /* 0x0E4 RTC Alarm (A and B) through EXTI Line Interrupt
*/
        0,                                  /* 0x0E8 USB OTG FS Wakeup through EXTI line interrupt
*/
        0,                                  /* 0x0EC TIM8 Break Interrupt and TIM12 global interrupt
*/
        0,                                  /* 0x0F0 TIM8 Update Interrupt and TIM13 global interrupt
*/
        0,                                  /* 0x0F4 TIM8 Trigger and Commutation Interrupt and TIM14
global interrupt */
        0,                                  /* 0x0F8 TIM8 Capture Compare global interrupt
*/
        0,                                  /* 0x0FC DMA1 Stream7 Interrupt
*/
        0,                                  /* 0x100 FSMC global Interrupt
*/
        0,                                  /* 0x104 SDIO global Interrupt
*/
        0,                                  /* 0x108 TIM5 global Interrupt
*/
        0,                                  /* 0x10C SPI3 global Interrupt
*/
        0,                                  /* 0x110 UART4 global Interrupt
*/
```

```
        0,                                  /* 0x114 UART5 global Interrupt
*/
        0,                                  /* 0x118 TIM6 global and DAC1&2 underrun error  interrupts
*/
        0,                                  /* 0x11C TIM7 global interrupt
*/
        0,                                  /* 0x120 DMA2 Stream 0 global Interrupt
*/
        0,                                  /* 0x124 DMA2 Stream 1 global Interrupt
*/
        0,                                  /* 0x128 DMA2 Stream 2 global Interrupt
*/
        0,                                  /* 0x12C DMA2 Stream 3 global Interrupt
*/
        0,                                  /* 0x130 DMA2 Stream 4 global Interrupt
*/
        0,                                  /* 0x134 Ethernet global Interrupt
*/
        0,                                  /* 0x138 Ethernet Wakeup through EXTI line Interrupt
*/
        0,                                  /* 0x13C CAN2 TX Interrupt
*/
        0,                                  /* 0x140 CAN2 RX0 Interrupt
*/
        0,                                  /* 0x144 CAN2 RX1 Interrupt
*/
        0,                                  /* 0x148 CAN2 SCE Interrupt
*/
        0,                                  /* 0x14C USB OTG FS global Interrupt
*/
        0,                                  /* 0x150 DMA2 Stream 5 global interrupt
*/
        0,                                  /* 0x154 DMA2 Stream 6 global interrupt
*/
        0,                                  /* 0x158 DMA2 Stream 7 global interrupt
*/
        0,                                  /* 0x15C USART6 global interrupt
*/
        0,                                  /* 0x160 I2C3 event interrupt
*/
        0,                                  /* 0x164 I2C3 error interrupt
*/
        0,                                  /* 0x168 USB OTG HS End Point 1 Out global interrupt
*/
        0,                                  /* 0x16C USB OTG HS End Point 1 In global interrupt
*/
        0,                                  /* 0x170 USB OTG HS Wakeup through EXTI interrupt
*/
        0,                                  /* 0x174 USB OTG HS global interrupt
*/
        0,                                  /* 0x178 DCMI global interrupt
*/
        0,                                  /* 0x17C RNG global Interrupt
*/
        0                                   /* 0x180 FPU global interrupt
*/
};



/************************************************
* Global Variables
************************************************/


uint32_t period = 84-1; // Amplitude of created signal
uint32_t pwm_period = 84-1; // Period of pwm signal
//uint32_t pwm_period = 168-1; // Period of pwm signal 186 MHz/ 168 Sample = 1 MHz
uint32_t wave_period = 2000; // Period of created signal
uint16_t amplitude = 99;

const uint_fast8_t max_lut = 10000;
volatile uint_fast8_t sin_lut[10000];


uint8_t wave_type = SIN_WAVE; // Default


/************************************************
* Interrupt Handlers
************************************************/
```

```
void Default_Handler(void)
{
        //TIM1->CR1 |= (0 << 1);
        //for (;;);  // Wait forever
        NMI_Handler();
}

void NMI_Handler(void)
{
        int bc = BLINK_COUNT;
        // Disable Timer 1 module (CEN, bit0)
        TIM1->CR1 &= (1 << 0);

        /* Enable GPIOD clock (AHB1ENR: bit 3) */
        // AHB1ENR: XXXX XXXX XXXX XXXX XXXX XXXX XXXX 1XXX
        RCC->AHB1ENR |= 0x00000008;

        GPIOD->MODER &= ~(0x3 << 28); // Reset bits 25:24 to clear old values
        GPIOD->MODER |=  (0x1 << 28); // Make PD14 Red Led output

        // Open led
        //GPIOD->ODR |= (1 << 12);

        //Blink PD12 RED LED
        while(bc > 0)
        {
                basic_delay(LEDDELAY);
                GPIOD->ODR ^= (1 << 14);  // Toggle LED
                bc--;
        }

        main();

}


/************************************************
* TIM1 Update Interrupt and TIM10 global interrupt
************************************************/
void tim1_UI(void)
{

        //Reset timer interrupt
        TIM1->SR = (uint16_t)(0x0000);

        switch(wave_type) {
                case SIN_WAVE:
                        sin_generate();
                        break;
                case SQUARE_WAVE:
                        sqr_generate();
                        break;
                case TRIANGLE_WAVE:
                        triangle_generate();
                        break;
                case SAWTOOTH_WAVE:
                        sawtooth_generate();
                        break;
                case NOISE_WAVE:
                        white_noise_generate();
                        break;

                //default: // Go to error function

        }

}


/************************************************
* On the fly calculator functions
************************************************/

void make_luts(int _wave_period) {

    NVIC_DisableIRQ(TIM1_UP_TIM10_IRQn);
    NVIC_ClearPendingIRQ(TIM1_UP_TIM10_IRQn);
    //TIM1->DIER &= (1 << 0); //Channel 1 update interrupt
    TIM1->SR = (uint16_t)(0x0000);
    TIM1->CR1 &= (0 << 0);
```

```
        for(int i=0; i<_wave_period; i++) {
                sin_lut[i] = (_wave_period/2) * ( sin(2 * M_PI * i / _wave_period) + 1);
        }

    TIM1->CR1 |= (1 << 0);
    NVIC_EnableIRQ(TIM1_UP_TIM10_IRQn);


}

void calculate_freq(uint32_t freq){
        // Base PWM period = 84 => 168/84 => 2MHz
        // Base wave_period = 100
        // Base freq = 20 Khz;
        const uint32_t base = 2000000;
        //const uint32_t base = 1000000;

        int _wave_period = base / freq;
        make_luts(_wave_period);
        wave_period = _wave_period;

}

void calculate_amp(uint16_t amp) {
        const uint16_t max_amp = 100;
        period = pwm_period * amp / max_amp;
}


void set_wave_type(int w_type)
{
    wave_type = w_type;
}

// A simple and not accurate delay function
// that will change the speed based on the optimization settings
void basic_delay(volatile uint32_t s)
{
        for(s; s>0; s--){
                __asm__("NOP");
        }
}


void init_signal_pin(void)
{
        // enable GPIOA clock
        RCC->AHB1ENR |= (1 << 0);

        GPIOA->MODER |= (0x2 << 16); // Set PA8 to alternate mode TIM1_CH1
        GPIOA->AFR[1] |= (0x1 << 0); // Choose Timer1 as Alternative Function for PA8 AF1
        GPIOA->OSPEEDR |= (0x3 << 16); //  PA8 to very high speed

}

void init_timer1(void)
{
        uint32_t duty = pwm_period/2;

        calculate_freq((uint32_t)(2000)); // 2Khz
        calculate_amp(100); // 100 / 100

        // enable TIM1 clock (bit0)
        RCC->APB2ENR |= (1 << 0);

        // Timer clock runs at ABP1 * 2
        //   since ABP1 is set to /4 of fCLK
        //   thus 168M/4 * 2 = 84Mhz
        // set prescaler to 83999
        //   it will increment counter every prescalar cycles
        // fCK_PSC / (PSC[15:0] + 1)
        // 84 Mhz / 8399 + 1 = 10 khz timer clock speed
        // 84 Mhz / 83 + 1 = 1 Mhz timer clock speed
        //TIM4->PSC = 83;

        // set prescaler to 167
        //   it will increment counter every prescalar cycles
        // fCK_PSC / (PSC[15:0] + 1)
        // 168 Mhz / 167 + 1 = 1 Mhz timer clock speed
        TIM1->PSC = 0;

        // set period
```

```
        TIM1->ARR = pwm_period;

        // set duty cycle on channel 1
        TIM1->CCR1 = duty;



        // enable channel 1 in capture/compare register
        // set oc1 mode as pwm (0b110 or 0x6 in bits 6-4)
        TIM1->CCMR1 |= (0x6 << 4);
        // enable oc1 preload bit 3
        TIM1->CCMR1 |= (1 << 3);

        TIM1->CCER |= (1 << 0); // enable capture/compare ch1 output

        TIM1->BDTR |= (1 << 15); // enable main output

        //TIM1->DIER |= (1 << 1); //Channel 1 capture compare
        TIM1->DIER |= (1 << 0); //Channel 1 update interrupt

        // priority TIM1 IRQ from NVIC
        //  TIM1 Update Interrupt and TIM10 global interrupt
        NVIC->IP[TIM1_UP_TIM10_IRQn] = 0x13; // Priority level

        // TIM1_CC_IRQn
        // TIM1 Capture Compare Interrupt
        //NVIC->IP[TIM1_CC_IRQn] = 0x12; // Priority level

        // enable TIM1 IRQ from NVIC
        // TIM1 Update Interrupt and TIM10 global interrupt
        NVIC_EnableIRQ(TIM1_UP_TIM10_IRQn);

        // TIM1 Capture Compare Interrupt
        // NVIC_EnableIRQ(TIM1_CC_IRQn);

        // Enable Timer 1 module (CEN, bit0)
        TIM1->CR1 |= (1 << 0);
}

/************************************************
* main code starts from here
************************************************/
int main(void)
{
        /* set system clock to 168 Mhz */
        set_sysclk_to_168();

    init_signal_pin();

    Init_Systick(21000, 1);
    Send_GroupId(3);
    Init_Uart();
    Send_Help();

    init_timer1();


        // For white noise enable Random Number Generator
        RCC->AHB2ENR |= 0x40;     // Enable Clock of RNG
    RNG->CR |= (1 << 2);    // RNG enable no interrupt Page 769


        while(1)
        {
                // Do nothing. let timer handler do its magic
        }

        return 0;
}


/************************************************
* Function Generators
************************************************/


/************************************************
* Sin wave Generator
************************************************/
void sin_generate(void)
{
        static uint32_t t = 0;
```

```
        uint32_t f = wave_period-1;

        if (t >= f)
        {
                t = 0;
        } else
        {
                ++t;
        }


        // set new duty cycle
        TIM1->CCR1 = (uint16_t)( (sin_lut[t] * period) / f );
}

/************************************************
* Square wave Generator
************************************************/
void sqr_generate(void)
{
        static uint32_t t = 0;
        uint32_t f = wave_period-1;

        if (t >= f)
        {
                t = 0;
        } else
        {
                ++t;
        }

        // Generate 1 with pwm
        if (t <= f/2) {
                TIM1->CCR1 = (uint16_t)((period + 1));
        }
        // Generate 0 with pwm
        else {
                TIM1->CCR1 = (uint16_t)(0);
        }
}

/************************************************
* Sawtooth Generator
************************************************/
void sawtooth_generate(void)
{
        static uint32_t t = 0;

        if (t >= wave_period)
        {
                t = 0;
        } else
        {
                ++t;
        }

        TIM1->CCR1 = (uint16_t)(t * period / (wave_period) );

}


/************************************************
* Triangle Generator
************************************************/
void triangle_generate(void)
{
        static uint32_t t = 0;
        uint32_t f = wave_period-1;


        if (t <= f/2) {
                TIM1->CCR1 = (uint16_t)(t* 2 * period / wave_period );
                ++t;
        }
        else {
                TIM1->CCR1 = (uint16_t)(abs(f - t) * 2 * period / (wave_period) );
                ++t;
                if(t >= f) t=0;
        }


}
```

```
/************************************************
* White Noise Generator
************************************************/
void white_noise_generate(void)
{

        uint32_t f;
        uint16_t random_number;
        float_t big_num = (uint16_t)(0xFFFF);

        f = period;

        //while (!(RNG->SR&1));    // Wait for usable Random Number

        // Scale 32bit random number to our period
        // Get our period to max number ratio
        // random_number = (RNG->DR >> 16) * (period / 0xFFFF);
        random_number =  (RNG->DR >> 16) * (float)( f / big_num );

        TIM1->CCR1 = (uint16_t)( random_number );

}
```