P3 - Digital Multimeter

Sereen Benchohra
Jose Velez

EE 329-3

Prof. Paul Hummel

Table of Contents

Behavior Description

This system is designed to act as a digital multimeter. The digital multimeter is designed to be multifunctional. The multimeter uses a function generator to generate different types of waves. The digital multimeter measures voltages from 0 to 3.3 Volts with an accuracy of +/- 25 mv, updating every 3.5 seconds. The digital multimeter has AC and DC modes that are determined by the user. The Digital Multimeter DC mode displays a measurement that is an average of multiple measurements over a time period greater than 1 ms and less than 100 ms.In the AC mode, the digital multimeter displays the true- RMS value that includes the DC offset. In addition, the AC mode displays peak-to-peak value and works for the Sine Waves, Saw-tooth waves, and Square Waves. The digital multimeter reads from a voltage range from 0V to 3V and reads a minimum of 0.5 V. The maximum DC offset value that shall be measured is 2.75 V. The digital multimeter measures and displays the frequencies ranging from 1 to 1000 Hz with an accuracy of +/- 5 Hz from various waveforms (Sine , Sawtooth, Square, etc). Finally, the DC voltage and true RMS voltage are displayed on a created bar graph in a manner of our choosing to correspond with the values read in the multimeter.

System Specification

The system specification can be found in Table 1, and indicate the components and relative information for the Function Generator

STM32L476RG	Power Supply	MiniUSB entry, 5.0 V
	Maximum Frequency	32 MHz
	PIN capability	4-digit, numerical 0-9
Function Generator	Wiring Capability	2, one for input and GND
	Waveforms	At least 3 types of waveforms, Sine, Saw-tooth, Square
	Pin Capability	Up to 6 pins

Table 1: System Specs

System Schematic

The system schematic as shown in Figure 1, includes the connection between the board and the function generator.

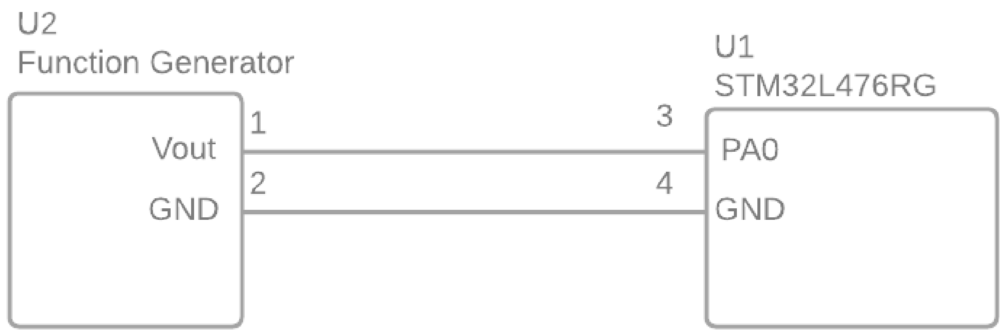

Figure 1: System Schematic Diagram for Digital Multimeter

Software Architecture

The software architecture for this product was modeled mostly initializing hardware internal components on the STM32 board. Hence, in the software architecture it initializes the hardware component and retrieves values and utilizes the values from the board's hardware components. The software architecture uses conditional statements, to check if certain flags have been set to run the program. For more detailed explanations, see descriptions in the Flowcharts.

Flowchart for main Digital Multimeter

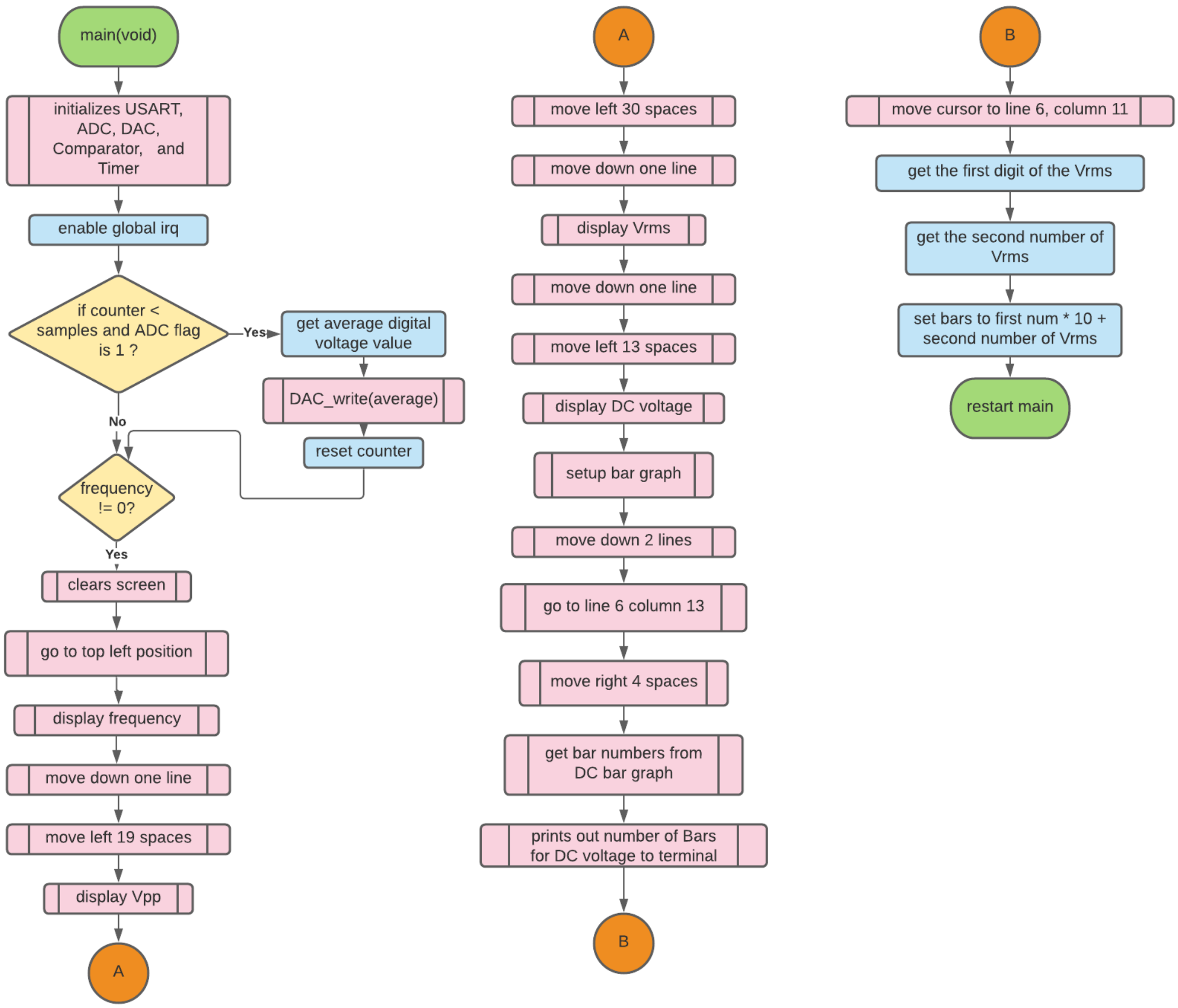

Figure 2 Digital Multimeter main initialization and implementation

The core implementation of the Digital Multimeter is shown in Figure 2. The multimeter first starts initializing the hardware components for ADC, USART, DAC, Comparator, and Timer(See their respective flowchart for more details). The program then enables the global interrupt to enable the irq handlers for the ADC and Timer. The program then checks if there were enough samples collected. If so, the average voltage is calculated and used as a voltage reference to input into the DAC_write where the output of the DAC would input into the comparator. The counter would reset, and check if there is a frequency that is read, if so the screen is cleared and the cursor moves to the appropriate position on the terminal to calculate and display the Peak to Peak voltage. Once displayed, the cursor on the terminal moves to the appropriate position and displays the Voltage RMS value. Again, the cursor moves again for the destined position, and displays the DC voltage. After the voltages are displayed , the program sets up the Bar Graph and displays the RMS and DC voltages real time as their value changes.

Helper functions for Digital Multimeter

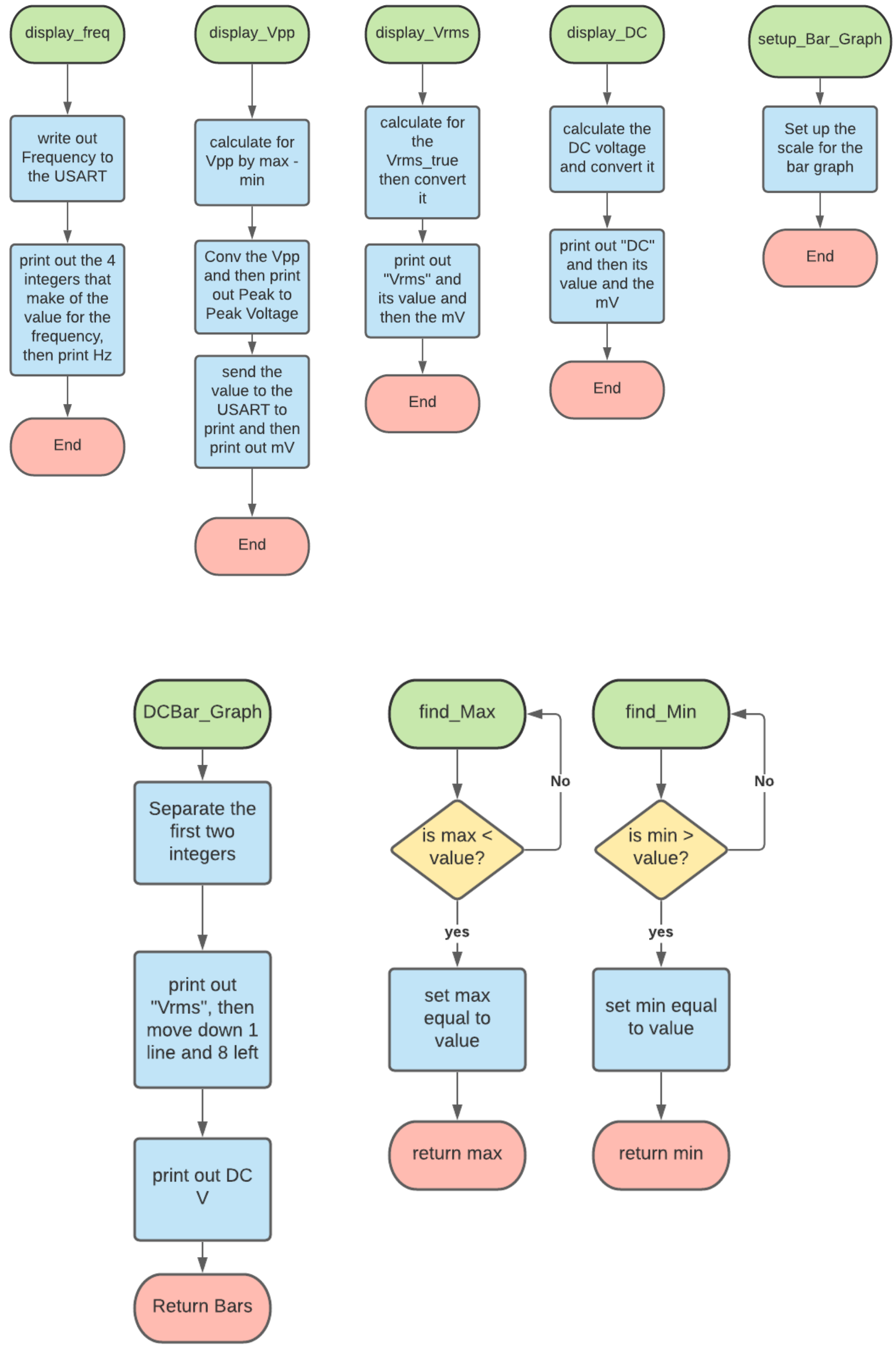

Figure 3 Helper Functions for Main

The display_freq function prints out the frequency to the USART. The Display_Vpp function will calculate the voltage from peak to peak and then print it out to the USART. Display_Vrms function will calculate the Vrms, convert it and finally will print it out to the USART.

Display_DC will calculate the DC voltage, convert it and then will print it out. Setup_Bar_Graph will set up the scale for the Vrms and the DC voltage by increments of 1.0 until the end where we add an increment of 0.5. DCBar_Graph will separate the first two integers from the DC value and then will print out “Vrms |”. Then it will move the cursor down one line and over to the left eight spaces. It will then print out “DC V | “. Finally it will return the variable Bars that holds the value of the integer for the DC voltage. The find_Max and find_Min functions do very similar functions where they compare their variable max or min, with the value being read from the ADC to see if it is greater than the max or less than the min. If it does find that the value meets those requirements, it will set the max equal to the new greater value and the min to the new lesser value. At the end of their respective functions, they will return the max and min to the main.

Timer AND IRQ Handler FLOWCHART

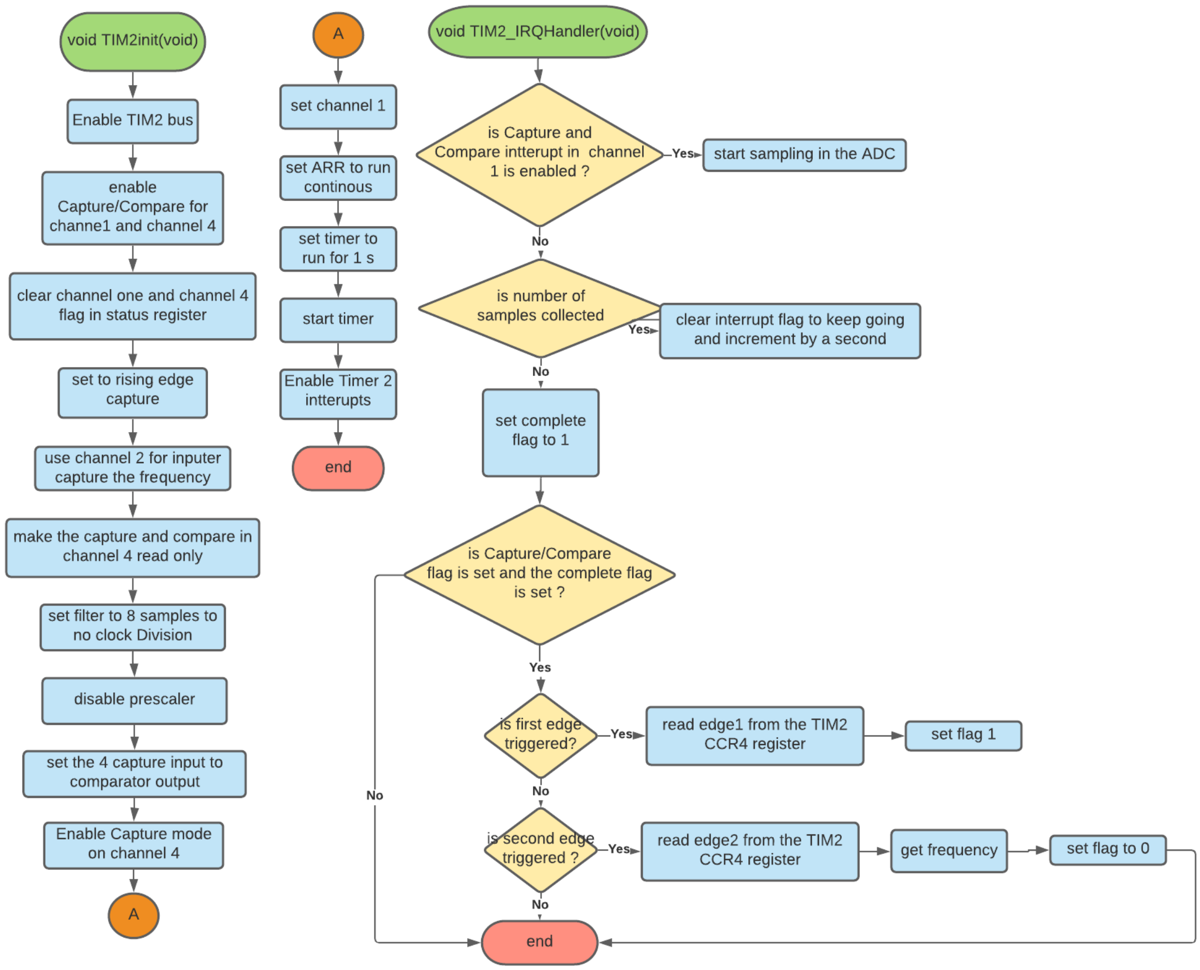

Figure 4 Flowchart for TIM2_init and TIM2_IRQHandler

Figure 4 shows the flowchart for the setup and execution of the Timer module, where the sampling is initiated and frequency is calculated. The TIM2_init sets up two channels, channel 1 and channel 4. Channel 1 is set up to utilize the counter and get the proper period for sampling. To ensure that the sampling happens in the proper period, the TIM2_IRQ handler checks for the Capture and Compare Flag for Channel 1, if the condition is satisfied, then the sampling from the ADC is started. Channel 4 is set up in input capture mode to get the frequency from the comparator. The input capture is set up so when it detects a rising edge, the Capture and

Compare flag for channel 4 is raised. The TIM2_IRQ handler checks for the channel 4 flag , if raised, the handler then checks if the flag is raised for the first time or not. If it is the first time, the clock is read from the CCR4 register of TIM2 and saved to the first edge. If the second flag is raised, the CCR4 register is saved into the second edge. The frequency is then calculated by subtracting the edge 2 from edge 1 and dividing the master clock(32 MHZ) by the result. Once that is done the flag is reset to get a new frequency.

UART functions

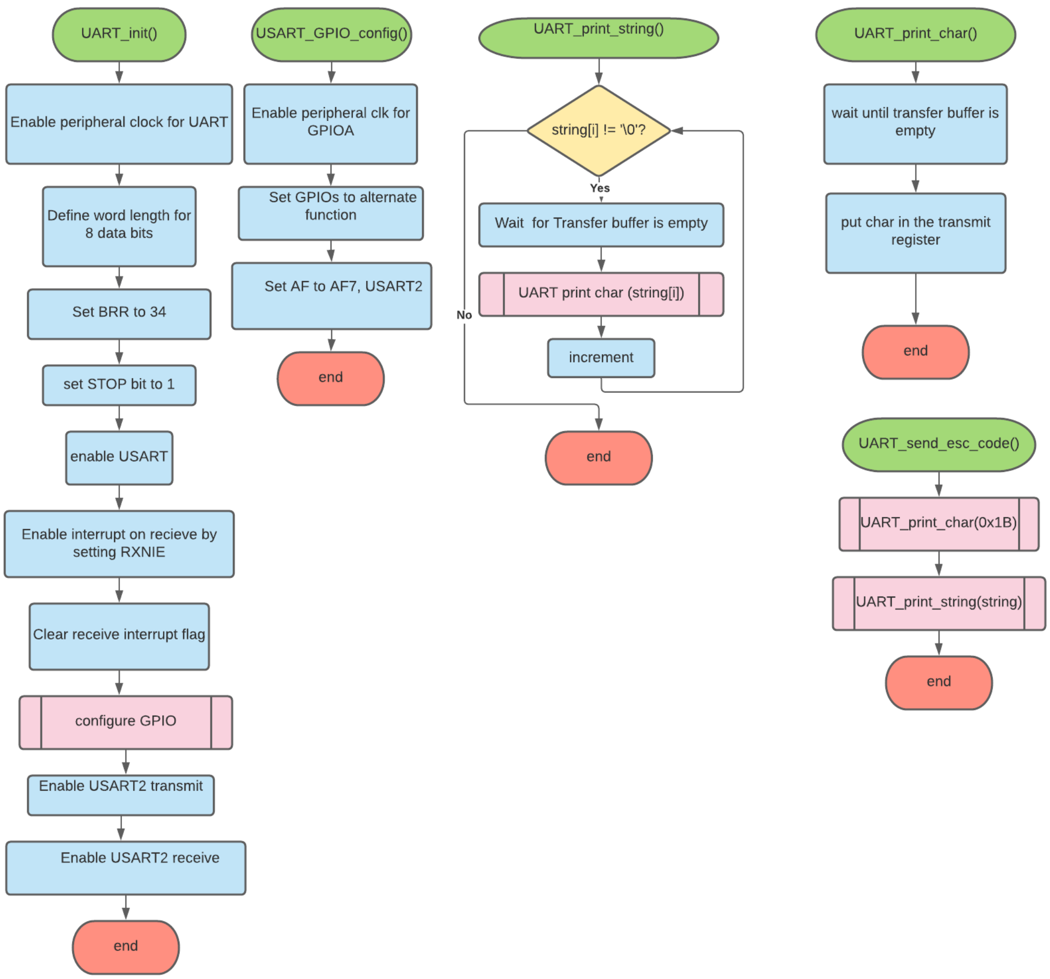

Figure 5 Flowchart for USART functions

USART utilizes and creates a serial interface that can connect to the software terminal on the host computer. By doing this, the USART can be used to write to the serial terminal for the digital multimeter. The USART is used to show and read the value for the frequency, DC voltage, Vrms and also display a bar graph created. The setup for the USART is as follows, the UART_init, USART_GPIO_config, UART_print_string, UART_print_char. The UART_init is a series of steps to be able to use the USART.

COMP_Init and readCOMP

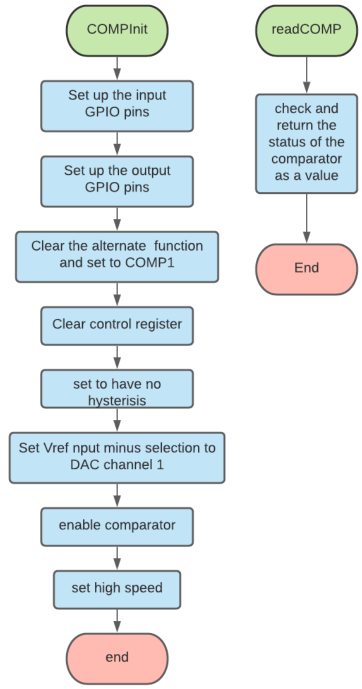

Figure 6 Flowchart Comparator and read Comparator

This flowchart shows the initialization for the comparator as well as how it's read in the code. The beginning steps are just setting up the comparator such as the input GPIO pins and the output GPIO pins. Afterwards, start the actual initialization by clearing the control register, setting the comparator to no hysteresis, set input minus selection for VREFINT and input to DAC channel 1 , enable the comparator, and set it to high speed. The read comparator will take in two inputs where one is a signal and the other is a voltage reference. It will see if the voltage ref is within the voltage range of the wave and then convert the wave into a square wave allowing for the frequency of the wave to be easily attainable through the use of the timer in input capture mode (See Figure 4 to see how the Frequency is calculated from the Square Wave).

ADC_init and ADC_IRQ_Handler FLOWCHART:

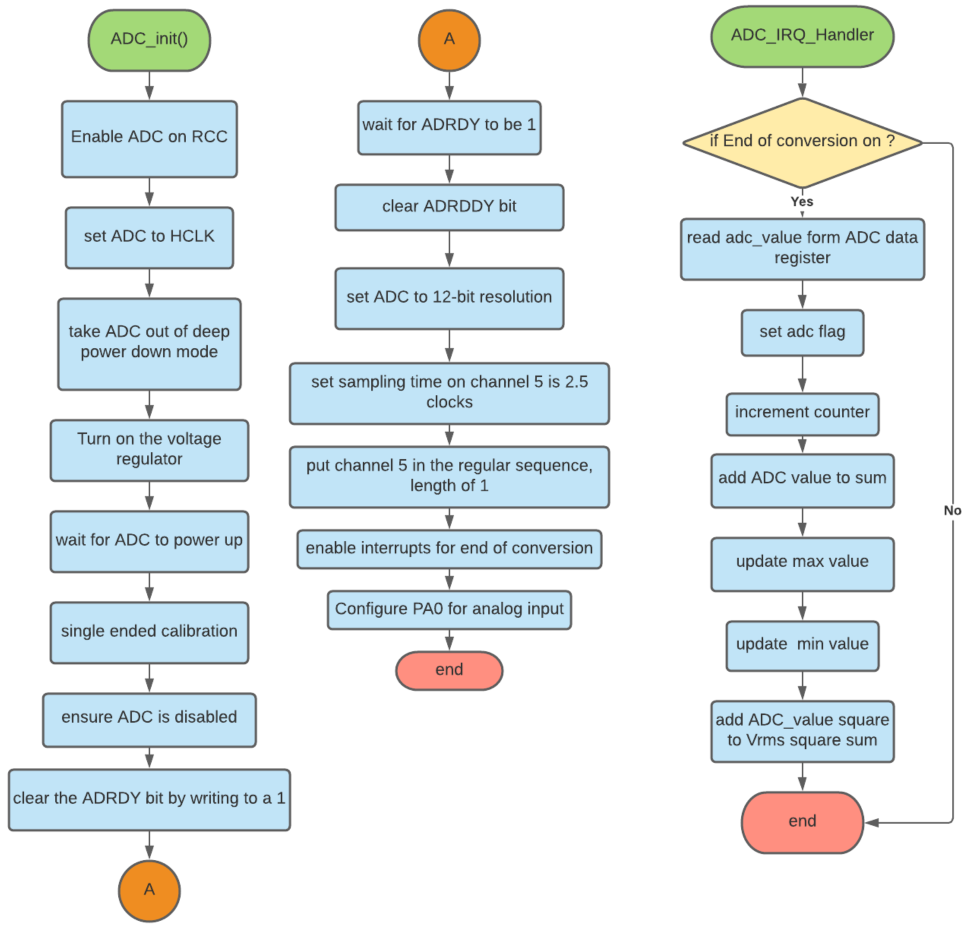

Figure 7 Flowchart for ADC_init and ADC_IRQHander

This flowchart shows the basic thought process when initializing the ADC and creating the IRQ Handler for it. The initialization allows for the ADC to be initialized and to be ready to take in analog input voltages from various waves from the Function Generator. The ADC_IRQHandler will check if a value has been written to the ADC and interrupt it. After doing so, the value from the ADC1 data register will be written to the adc_value. Afterwards, set the ADC_Flag equal to 1. Getting the ADC values in the IRQ handler allows us to calculate the the Rms voltages, the minimum and maximum voltages needed for Peak to Peak.

DAC init and DAC write FLOWCHART

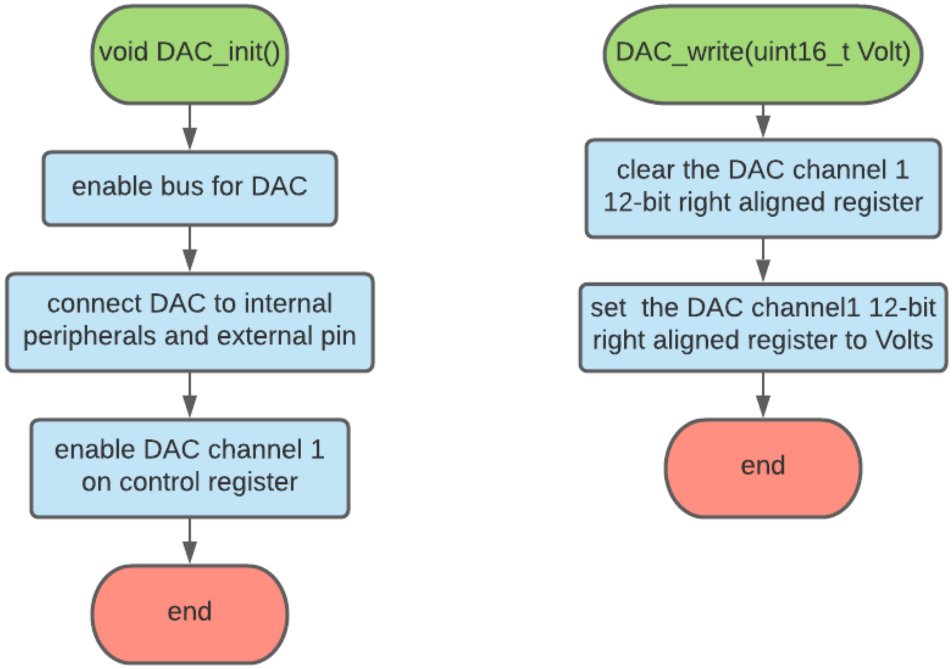

Figure 8, Flowcharts for DAC_write and DAC_init

Figure 8 shows how the internal DAC works. To initialize the DAC, the BUS for the DAC needs to be enabled. Set it the mode control register to connect to the Comparator (See Figure 6 for more detail on the comparator) and enable the DAC on channel 1. The DAC_write is where we write the Digital Voltage we got from the ADC and once we get the average digital voltage, it is set into the DAC register and the output of the DAC would be used and a reference Voltage would go to the Comparator.

Appendix - Code

Main.c

```c
#include"main.h"
#include"USART.h"
#include"ADC.h"
#include"COMP.h"
#include"Timer.h"
#include"DAC.h"
#include<stdlib.h>
#include<stdio.h>
#include <inttypes.h>
#include <math.h>

/* Private function prototypes
-----------------------------------------------*/
void SystemClock_Config(void);


int AVG_FirstNum;
int AVG_SecondNum;

uint16_t findavg(void);
uint16_t volt_conv(uint16_t digital);
uint16_t findmax(void);
uint16_t findmin(void);
int DCBar_Graph(void);
void display_freq(void);
void display_Vpp(void);
void display_Vrms(void);
void display_Vrms(void);
void display_DC(void);
void setup_Bar_Graph(void); // sets up scale for the Bar Graph

// Global Variables
uint16_t ADC_Value; // global to read in ISR and main

uint32_t max = 0;
uint32_t min = 4095;
uint64_t Vrms_sum_sqr = 0;
uint16_t Vrms_true;
uint16_t calib_rms;
uint16_t calib_DC;

uint16_t avg;
uint8_t ADC_Flag = 0;
```

```
uint32_t frequency;
uint32_t edge1, edge2;
uint32_t sum = 0;
uint16_t Vpp;

int j = 0;
uint16_t samples = 100000;

int main(void)
{
	// initializes the Hardware components
	HAL_Init();				//HAL Configuration
	SystemClock_Config(); //Clock Cinfiguration
	UART_init();			//USART configuration
	ADC_init();				//ADC config
	DAC_init();
	COMPInit();
	TIM2init();
	__enable_irq();			//enable interrupts

	while (1)
	{

		if((j > samples ) && (ADC_Flag == 1))
		{
			avg = sum/samples; // gets average num of samples
			DAC_write(avg);		//Output Vref
			j = 0;
		}

		if (frequency != 0)	//Clear overcapture flag)
		{
			UART_send_esc_code("[2J"); // clears screen

			UART_send_esc_code("[H");// go to top left position

			HAL_Delay(3);	//delay before sending to UART

			// display Frequency
			display_freq();


			UART_send_esc_code("[1B");// move down one line
			UART_send_esc_code("[19D");//move left 19 spec
```

```
                // Display Vpp
                display_Vpp();


                UART_send_esc_code("[30D");// move left 30 spaces
                UART_send_esc_code("[1B"); // go down 1 line

                // display Vrms
                display_Vrms();

                UART_send_esc_code("[1B");//move down 1 lines
                UART_send_esc_code("[13D"); // move left 30 spaces

                // display DC voltage
                display_DC();

                // Bar Graph setup
                setup_Bar_Graph();

                UART_send_esc_code("[2B"); // move down two lines
                UART_send_esc_code("[6,13H"); // go to line 6 column
13
                UART_send_esc_code("[4C"); // move right 4 spaces

                int bar_num = DCBar_Graph();

                for(int i = 0; i < bar_num; i++) // prints out the
number of bars for DC voltage
                        USART_write("*");

                UART_send_esc_code("[6;11H"); // move cursor in
terminal to line 6 , column 11

                uint16_t Vrms_FirstNum = (calib_rms / 1000);//get
the first digit

                uint16_t Vrms_SecondNum = ((calib_rms / 100) % 10);
                int one = Vrms_FirstNum * 10;
                int Barz = one + Vrms_SecondNum; // gets number of
Bars for Vrms

                for(int j = 0; j < Barz;j++) // prints the number of
Bars for Vrms
                        USART_write("*");
```

```
            }

        }
}

void display_freq()
{
        USART_write("Frequency: ");
        char freq[4];
        sprintf(freq, "%" PRId32, frequency); // convert freq into
string
        USART_write(freq); // output string into terminal
        USART_write(" Hz ");
}

void display_Vpp()
{
        Vpp = max - min; // calculate Peak to Peak
        uint16_t calib_Vpp  =  ADC_volt_conv(Vpp); // calibrate Vpp
        USART_write("Peak to Peak Voltage: ");
        char Vpp_str[4];
        sprintf(Vpp_str, "%" PRId16, calib_Vpp); // convert Vpp into
string
        USART_write(Vpp_str); // output string into terminal
        USART_write(" mV");

}

void display_Vrms()
{
        Vrms_true = sqrt(Vrms_sum_sqr/samples); // calculate Vrms

        calib_rms = ADC_volt_conv(Vrms_true);

        USART_write("Vrms: ");
        char Vrms_str[4];
        sprintf(Vrms_str, "%" PRId16, calib_rms);  // convert Vrms into
string
        USART_write(Vrms_str); // output string into terminal
        USART_write("mV");
}

void display_DC()
{
```

```
	uint32_t DC = (max+min)/2; // calculate DC voltage
	calib_DC = ADC_volt_conv(DC); // calibrate DC voltage

	char DC_str[4];
	sprintf(DC_str, "%" PRId16, calib_DC); // convert DC voltage
into string
	USART_write("DC: ");
	USART_write(DC_str); // output string into terminal
	USART_write("mV");
}

/*

*/
void setup_Bar_Graph() // sets up X axis scale for the Bar Graph
starting from 0 and ending at 3.5 Volts
{
	UART_send_esc_code("[1B");//go down one line
	USART_write("0"); // write 0
	UART_send_esc_code("[8C"); // go right 8 spaces
	USART_write("1.0"); // write 1
	UART_send_esc_code("[7C"); // go right 7 spaces
	USART_write("2.0"); // write 2 Volts
	UART_send_esc_code("[7C"); // go right 7 spaces
	USART_write("3.0"); // write 3 Volts
	UART_send_esc_code("[7C"); // go right 7 spaces
	USART_write("3.5"); // write 3.5 Volts
}

int DCBar_Graph(void)
{
	AVG_FirstNum = (calib_DC/ 1000);//get the first digit
	AVG_SecondNum = ((calib_DC/ 100) % 10);//get the second digit
	USART_write("Vrms |");
	int First = AVG_FirstNum * 10;
	int Bars = First + AVG_SecondNum;
	UART_send_esc_code("[1B"); // move down one line
	UART_send_esc_code("[6D"); // go left 6 spaces
	USART_write("DC V |"); // write the DC V in the Y axis
	return Bars;
}

uint16_t find_Max(uint16_t value) // get max value
{
```

```
	if(max < value)
		max = value;

	return max;
}

uint16_t find_Min(uint16_t value) // get min value
{
	if(min > value)
		min = value;

	return min;
}


void TIM2_IRQHandler(void)
{
	static uint8_t flag = 0;
	static uint8_t sflag = 0;

	uint32_t CC;
	if (TIM2->SR & TIM_SR_CC1IF)	// start sequence when Flag
starts
		ADC1->CR |= ADC_CR_ADSTART; // start regular sequence

	if( j > (samples) )
	{
		TIM2->SR &= ~(TIM_SR_CC1IF);	//Clear CCR1 flag
		TIM2->CCR1 += ONE_SEC;
	}
	else
		sflag = 1;

	if (TIM2->SR & TIM_SR_CC4IF && sflag == 1 )
	{
		if (flag == 0) {
			edge1 = TIM2->CCR4;	//collect first edge
			flag = 1;  //Set first edge captured flag

		} else if (flag == 1) {
			edge2 = TIM2->CCR4;	//collect second edge
			CC = (edge2 - edge1); //Clock cycles in between
rising edges
			frequency = (MCLK / CC);	//calculate frequency
			flag = 0;		//reset flag
```

```
		}

	}
}

void ADC1_2_IRQHandler(void) {
	if (ADC1->ISR & ADC_ISR_EOC) {
		ADC_Value = ADC1->DR; // read conversion
		j++;
		sum += ADC_Value;
		ADC_Flag = 1; // might be our issue
		max = find_Max(ADC_Value);
		min = find_Min(ADC_Value);
		Vrms_sum_sqr += (ADC_Value * ADC_Value);

	}
}

void SystemClock_Config(void) {
	RCC_OscInitTypeDef RCC_OscInitStruct = { 0 };
	RCC_ClkInitTypeDef RCC_ClkInitStruct = { 0 };

	/** Configure the main internal regulator output voltage
	 */
	if
(HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1)
			!= HAL_OK) {
		Error_Handler();
	}
	/** Initializes the RCC Oscillators according to the specified
parameters
	 * in the RCC_OscInitTypeDef structure.
	 */
	RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
	RCC_OscInitStruct.MSIState = RCC_MSI_ON;
	RCC_OscInitStruct.MSICalibrationValue = 0;
	RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_10;
	RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
	if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) {
		Error_Handler();
	}
	/** Initializes the CPU, AHB and APB buses clocks
	 */
	RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK |
RCC_CLOCKTYPE_SYSCLK
```

```
                  | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
      RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
      RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
      RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
      RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

      if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) !=
HAL_OK) {
            Error_Handler();
      }
}

void Error_Handler(void) {
      /* USER CODE BEGIN Error_Handler_Debug */
      /* User can add his own implementation to report the HAL error
return state */
      __disable_irq();
      while (1) {
      }
      /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT

void assert_failed(uint8_t *file, uint32_t line)
{

}
#endif /* USE_FULL_ASSERT */
```

ADC.c

```
#include "main.h"
#include "ADC.h"

#define m 785
#define b 1400

void ADC_init(void)
{

      // enable ADC on RCC
```

```
	RCC->AHB2ENR |= RCC_AHB2ENR_ADCEN;
	// set ADC to use HCLK / 1 clock speed
	ADC123_COMMON->CCR = (ADC123_COMMON->CCR & ~(ADC_CCR_CKMODE)) |
	            (1 << ADC_CCR_CKMODE_Pos);
	// take ADC out of deep power down mode
	// and turn on the voltage regulator
	ADC1->CR &= ~(ADC_CR_DEEPPWD);
	ADC1->CR |= (ADC_CR_ADVREGEN);
	delay_us(20); // wait 20us for ADC to power up
	// Calibration time
	// single ended calibration, ensure ADC is disabled
	ADC1->CR &= ~(ADC_CR_ADEN | ADC_CR_ADCALDIF);
	ADC1->CR |= (ADC_CR_ADCAL);
	while(ADC1->CR & ADC_CR_ADCAL);  // wait for ADCAL to become 0
	// configure single ended for channel 5
	ADC1->DIFSEL &= ~(ADC_DIFSEL_DIFSEL_5);
	// enable ADC FINALLY!!!!
	// clear the ADRDY bit by writing a 1
	ADC1->ISR |= (ADC_ISR_ADRDY);
	ADC1->CR |= (ADC_CR_ADEN);
	while(!(ADC1->ISR & ADC_ISR_ADRDY)); // wait for ADRDY to be 1
	ADC1->ISR |= (ADC_ISR_ADRDY); // clear ADRDY bit
	// Configure ADC
	// 12-bit resolution
	ADC1->CFGR &= ~(ADC_CFGR_RES);
	// sampling time on channel 5 is 2.5 clocks
	ADC1->SMPR1 &= ~(ADC_SMPR1_SMP5);
	// put channel 5 in the regular sequence, lenght of 1
	ADC1->SQR1 = (ADC1->SQR1 & ~(ADC_SQR1_SQ1 | ADC_SQR1_L)) |
	            (5 << ADC_SQR1_SQ1_Pos);
	// enable interrupts for end of conversion
	ADC1->IER |= ADC_IER_EOC;
	ADC1->ISR &= ~(ADC_ISR_EOC); // clear the flag
	NVIC->ISER[0] = (1 << (ADC1_2_IRQn & 0x1F));

	// Configure GPIO PA0 for analog input
	RCC->AHB2ENR |= (RCC_AHB2ENR_GPIOAEN);
	GPIOA->MODER |= (GPIO_MODER_MODE0);  // analog mode PA0
	GPIOA->ASCR |= GPIO_ASCR_ASC0;  // connect analog PA0


}


uint16_t ADC_volt_conv(uint16_t dig_val) {
```

```
	// conversion equation derived from calibration data
	uint32_t uv_dec = m * dig_val + b;
	// converts uV decimal to mV decimal
	uint16_t mv_dec = uv_dec / 1000;
	return mv_dec;
}


void SysTick_Init(void){
	SysTick->CTRL |= (SysTick_CTRL_ENABLE_Msk |	// enable
SysTick Timer
			SysTick_CTRL_CLKSOURCE_Msk);	// select CPU clock
	SysTick->CTRL &= ~(SysTick_CTRL_TICKINT_Msk);	// disable
interrupt, breaks HAL delay function
}

void delay_us(const uint16_t time_us) {
	// set the counts for the specified delay
	SysTick->LOAD = (uint32_t)((time_us * (SystemCoreClock /
1000000)) - 1);
	SysTick->VAL = 0;					//
clear the timer count
	SysTick->CTRL &= ~(SysTick_CTRL_COUNTFLAG_Msk);		//
clear the count flag
	while (!(SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk));	//
wait for the flag to be set
}
```

ADC.h

```
#ifndef SRC_ADC_H_
#define SRC_ADC_H_
#define DAC_RES 4095
#define VREF 3.3

void ADC_init(void);
uint16_t ADC_volt_conv(uint16_t dig_val);
uint16_t ADC_volt_conv1(uint16_t dig_val);


void SysTick_Init(void);
void delay_us(const uint16_t time_us);
```

```
#endif /* SRC_ADC_H_ */
```

Comp.c

```
/*
 * COMP.c
 *
 *  Created on: Nov 24, 2021
 */
#include"main.h"
#include"COMP.h"


void Input_GPIO_COMP_setup()
{
	//Setup Input GPIO, PC5
	RCC->AHB2ENR	|=  (RCC_AHB2ENR_GPIOCEN);	//enable GPIOB
	GPIOC->MODER &= ~(GPIO_MODER_MODE5);	//clear mode register
	GPIOC->MODER |= (GPIO_MODER_MODE5);		//Set mode to Analog mode
	GPIOC->OTYPER &= ~(GPIO_OTYPER_OT5);		//Set type to Push-pull
	GPIOC->OSPEEDR &= ~(GPIO_OSPEEDR_OSPEED5);		//set speed (lowspeed)
	GPIOC->PUPDR |= (GPIO_PUPDR_PUPD5);	//Set to no Pull-up, pull-down

}

void Output_GPIO_COMP_setup()
{
	//Setup output GPIO, PB0
	RCC->AHB2ENR	|=  (RCC_AHB2ENR_GPIOBEN); //enable GPIOB
	GPIOB->MODER &= ~(GPIO_MODER_MODE0);	//clear mode register
	GPIOB->MODER |= (2 << GPIO_MODER_MODE0_Pos);	//Set mode to Alternate function, AFRL 2
	GPIOB->OTYPER &= ~(GPIO_OTYPER_OT0);		//Set type to Push-pull
	GPIOB->OSPEEDR &= ~(GPIO_OSPEEDR_OSPEED0);		//set speed (lowspeed)
	GPIOB->PUPDR &= ~(GPIO_PUPDR_PUPD0);	//Set to no Pull-up, pull-down
	GPIOB->AFR[0] &= ~(GPIO_AFRL_AFSEL0); //Clear Alternate
```

```
function register
      GPIOB->AFR[0] |= (12 << GPIO_AFRL_AFSEL0_Pos);    //Set to COMP1
output

}


void COMPInit(void)
{

      Input_GPIO_COMP_setup(); //Setup Input GPIO, PC5

      Output_GPIO_COMP_setup(); //Setup output GPIO, PB0

      //Intialize Comparator
      COMP1->CSR &= ~(0xFF);      //clear control register
      COMP1->CSR &= ~COMP_CSR_HYST;   // Have no hysterisis
      COMP1->CSR |= COMP_CSR_INMSEL_2; //Set input minus selection to
VREFINT (set to DAC channel)
      COMP1->CSR |=  COMP_CSR_EN;      //Enable comparator
      COMP1->CSR &= ~COMP_CSR_PWRMODE;// set high speed

}


uint8_t readCOMP(void) // used to read values for comparator,
primarily used for testing
{
      uint8_t Value;
      Value = COMP1->CSR && COMP_CSR_VALUE;       //return status of
comparitor
      return Value;
}
```

Comp.h

```
/*
 * COMP.h
 *
 *  Created on: Nov 24, 2021
 */

#ifndef INC_COMP_H_
```

```
#define INC_COMP_H_
void COMPInit(void);
uint8_t readCOMP(void);
#endif /* INC_COMP_H_ */
```

DAC.c

```
/*
 * DAC.c
 *
 *  Created on: Nov 26, 2021
 */
/* Initialize DAC*/
#include"DAC.h"
#include"main.h"

#define DAC_PORT GPIOA
void DAC_init() {
	RCC->APB1ENR1 |= RCC_APB1ENR1_DAC1EN; //enable bus for Dac &
TIM2
	DAC->MCR &= (DAC_MCR_MODE1_Pos);		//  clear mode
control register
	DAC->MCR |= (3 << DAC_MCR_MODE1_Pos); //DAC is set to  connect
to peripherals and external pin
	DAC->CR |= DAC_CR_EN1;				//enable DAC
Ch.1

}

void DAC_write(uint16_t Volt) {
	DAC->DHR12R1 &= ~(0xFFF);//clear DAC in right aligned, 12 -bit
data regidter for DAC 1
	DAC->DHR12R1 = Volt;//Write to DAC right aligned, 12 -bit data
regidter for DAC 1
}
```

DAC.h

```
/*
 * DAC.h
 *
 *  Created on: Nov 26, 2021
 */
```

```
#include <stdint.h>

#ifndef SRC_DAC_H_
#define SRC_DAC_H_

void DAC_init();
void DAC_write( uint16_t Volt );

#endif /* SRC_DAC_H_ */
```

Timer.c

```
/*
 * Timer.c
 *
 *  Created on: Nov 26, 2021
 */
#include"main.h"
#include"Timer.h"


void TIM2init(void)
{


	RCC->APB1ENR1 |= (RCC_APB1ENR1_TIM2EN);     //Enable TIM2 bus

	TIM2->DIER |= (TIM_DIER_CC1IE | TIM_DIER_CC4IE );      //enable
capture/compare for Channel 1 and 4

	TIM2->SR &= ~(TIM_SR_CC1IF | TIM_SR_CC4IF); // clear channel 1
and channel 4 interrupt flags

	TIM2->CCER &= ~(TIM_CCER_CC1NP | TIM_CCER_CC1P); //Set to
rising edge capture

	// use channel 2 for input capture the frequency setup

	TIM2->CCMR2 |= (TIM_CCMR2_CC4S_0);     //Make CCR4 and TI4 read
only

	TIM2->CCMR2 |= (3 << TIM_CCMR2_IC4F_Pos);   //set filter (no
clock Div) 8 samples
```

```
	TIM2->CCER &= ~(TIM_CCER_CC4NP | TIM_CCER_CC4P); //Set to rising edge capture

	TIM2->CCMR2 &= ~(TIM_CCMR2_IC4PSC);   //disable prescaler

	TIM2->OR1 |= (TIM2_OR1_TI4_RMP_0); //TIM2 capture 4 input connects to comparator1 output

	TIM2->CCER |= (TIM_CCER_CC4E);   //Enable Capture mode


	// set channel 1

	TIM2->ARR = 0xFFFFFFFF;    //Set ARR to run continuously

	TIM2->CCR1 = 640-1;

	TIM2->CR1 |= (TIM_CR1_CEN);// start timer

	NVIC->ISER[0] = (1 << (TIM2_IRQn & 0x1F));  //Enable Timer2 interrupts

}
```

Timer.h

```
/*
 * Timer.h
 *
 *  Created on: Nov 26, 2021
 */

#ifndef SRC_TIMER_H_
#define SRC_TIMER_H_

#define THREE_VOLTS 3722
#define ONE_SEC 640
#define MCLK 32000000

void TIM2init(void);
```

```
#endif /* SRC_TIMER_H_ */
```

USART.c

```
 ----------------------
#include "main.h"
#include "USART.h"
#include <string.h>

/*
 * UART.c
 *
 *  Created on: Nov 9, 2021
 */

#define BAUDRATE_32_MCKLK 277

void UART_init()
{
	// Enable peripheral clk for UART
	RCC->APB1ENR1 |= (RCC_APB1ENR1_USART2EN);
	// Define word length for 8 data bits: M[1:0] = 0b00
	USART2->CR1 &= ~(USART_CR1_M0 | USART_CR1_M1);
	// Set bitrate to 115.2 kbps by setting BRR to 277 for 32 MHZ
	USART2->BRR = BAUDRATE_32_MCKLK ;
	// Set 1 stop by by seeing STOP[1:0] = 0b00
	USART2->CR2 &= ~(USART_CR2_STOP_0 | USART_CR2_STOP_1);
	// Enable USART
	USART2->CR1 |= (USART_CR1_UE);
	// Enable interrupt on recieve by setting RXNEIE
	USART2->CR1 |= (USART_CR1_RXNEIE);
	NVIC->ISER[1] = (1 << (USART2_IRQn & 0x1F));
	// Clear receive interrupt flag
	USART2->ISR &= ~(USART_ISR_RXNE);
	USART_GPIO_config();
	// Enable USART2 transmit
	USART2->CR1 |= (USART_CR1_TE);
	// Enable USART2 receive
	USART2->CR1 |= (USART_CR1_RE);
}

void UART_print_string(char *str)
{
```

```c
	// Print string over UART
	for (uint8_t i = 0; str[i] != '\0'; i++)
	{
		while (!(USART2->ISR & USART_ISR_TXE));
		UART_print_char(str[i]);
	}
}

void UART_send_esc_code(char *str)
{
	// Print an ESC Code over UART by sending an ESC code before
sending the string
	UART_print_char(ESC);
	UART_print_string(str);
}

void USART2_IRQHandler(void)
{

	// Check if USART2 RXNE caused the interrupt
	if (USART2->ISR & USART_ISR_RXNE)
	{
		// Read the received data
		uint8_t receivedChar = USART2->RDR;
		switch (receivedChar) {
		case 'R': {
			UART_send_esc_code("[31m");
			break;
		}
		case 'B': {
			UART_send_esc_code("[34m");
			break;
		}
		case 'G': {
			UART_send_esc_code("[32m");
			break;
		}
		case 'W': {
			UART_send_esc_code("[37m");
			break;
		}
		default: {
			UART_print_char(receivedChar);
		}
		}
```

```
			// Clear ISR flag
			USART2->ISR &= ~(USART_ISR_RXNE);
		}
}


void USART_GPIO_config()
{

	// Enable peripheral clk for GPIOA
	RCC->AHB2ENR |= (RCC_AHB2ENR_GPIOAEN);
	// Set GPIOs to alternate function
	GPIOA->MODER |= ( GPIO_MODER_MODE2_1 | GPIO_MODER_MODE3_1);

	GPIOA->MODER &= ~( GPIO_MODER_MODE2_0 | GPIO_MODER_MODE3_0);
	// Set AF to AF7, USART2
	GPIOA->AFR[0] |= ((AF7 << GPIO_AFRL_AFSEL2_Pos)
			| (AF7 << GPIO_AFRL_AFSEL3_Pos));
}

void UART_print_char(char charToPrint) // prints char to Terminal
{

	// Wait until TX buffer is empty
	while (!(USART2->ISR & USART_ISR_TXE));
	USART2->TDR = charToPrint;
}

void UART_send_esc_code(char* (string)) {
	while (!(USART2->ISR & USART_ISR_TXE));	//if the transmit is
empty, send the data
	USART2->TDR = (0x1B);
	while (!(USART2->ISR & USART_ISR_TXE));	//if the transmit is
empty, send the data
	USART2->TDR = '[';
	for(int i = 0; i< strlen(string);i++){
		while (!(USART2->ISR & USART_ISR_TXE));	//if the
transmit is empty, send the data

		USART2->TDR = string[i];
	}
}

void USART_write(char* (string)) {
	while (!(USART2->ISR & USART_ISR_TXE));	//if the transmit is
```

```
empty, send the data
	for (int i = 0; i < strlen(string); i++) {
		while (!(USART2->ISR & USART_ISR_TXE));	//if the
transmit is empty, send the data
		USART2->TDR = string[i];
	}

}
```

USART.h

```
/*
 * UART.h
 *
 *  Created on: Nov 9, 2021
 */

#ifndef SRC_USART_H_
#define SRC_USART_H_

#define AF7 0x7
#define ESC 0x1B

#define BAUDRATE_32_MCKLK 277

/* "Public" Stuff
------------------------------------------------------------*/
void UART_init(void);
void UART_print_string(char*);
void UART_send_esc_code(char*);
/* "Private" Stuff
-----------------------------------------------------------*/
void USART_GPIO_config(void);
void UART_print_char(char);
void USART2_IRQHandler(void);
void UART_send_esc_code(char*(string));
void USART_write(char*(string));
void hex_write(uint16_t num);
//int Bar_Graph(void);

#endif /* SRC_UART_H_ */
```