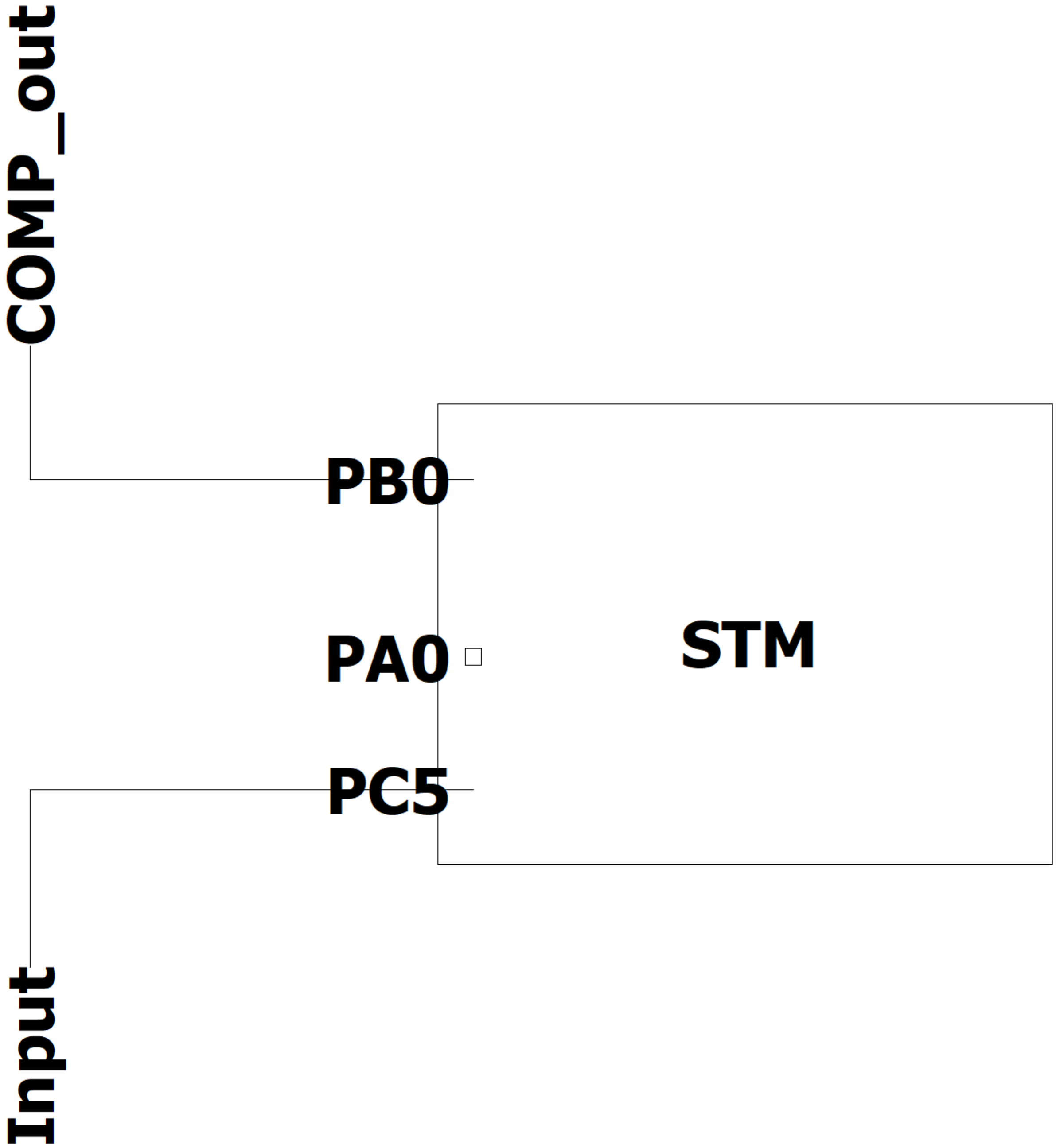
Benchohra, Sereen
Lopez, Luke
CPE 329
12/2/21

A9 - Frequency Measurement

Link to video : https://youtu.be/lpKCvWMNuv8

A. Schematic:



C - Code

USART.h

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

#ifndef SRC_USART_H_
#define SRC_USART_H_

#define AF7 0x7
#define ESC 0x1B
/* "Public" Stuff
------------------------------------------------------------*/
void UART_init(void);
void UART_print_string(char*);
void UART_send_esc_code(char*);
/* "Private" Stuff
-----------------------------------------------------------*/
void GPIO_config(void);
void UART_print_char(char);
void USART2_IRQHandler(void);

#endif /* SRC_UART_H_ */
```

USART .c

```c
#include "main.h"
#include "USART.h"

/*
```

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

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 34
      USART2->BRR = 277;
      // 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);
      GPIO_config();
      // Enable USART2 transmit
      USART2->CR1 |= (USART_CR1_TE);
      // Enable USART2 receive
      USART2->CR1 |= (USART_CR1_RE);
}

void UART_print_string(char *str)
{
       // Print string over UAR
      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 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)
{

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

UART.h

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

#ifndef SRC_USART_H_
#define SRC_USART_H_

#define AF7 0x7
```

```
#define ESC 0x1B
/* "Public" Stuff
--------------------------------------------------------------*/
void UART_init(void);
void UART_print_string(char*);
void UART_send_esc_code(char*);
/* "Private" Stuff
-------------------------------------------------------------*/
void GPIO_config(void);
void UART_print_char(char);
void USART2_IRQHandler(void);

#endif /* SRC_UART_H_ */
```

Comp.c

```
/*
 * COMP.c
 *
 *  Created on: Nov 24, 2021
 *      Author: Luke Lopez
 */
#include"main.h"
#include"COMP.h"
void COMPInit(void);
uint8_t readCOMP(void);

void COMPInit(void)
{
      //Initialize gpio PB0 for comparitor
      //RCC->AHB2ENR   |=  (RCC_AHB2ENR_GPIOCEN); //enable GPIOC

      //Setup Input GPIO
      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

    //Setup output GPIO
    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
    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

    //Intialize Comparator
    COMP1->CSR &= ~(0xFF);  //clear control register
    COMP1->CSR &= ~COMP_CSR_HYST;  //Set medium comparator hysterisis for
variation due to noise (prob make it 2 or 2
    COMP1->CSR |= COMP_CSR_INMSEL_2; //Set input minus selection to
VREFINT (set to DAC channel)
    COMP1->CSR |=  COMP_CSR_EN;         //Enable comparator
    //COMP_CSR_VALUE //Macro for status of comparator
    COMP1->CSR &= ~COMP_CSR_PWRMODE;// set high speed
    //COMP1->CSR |= COMP_CSR_PWRMODE_0; // set to medium speed
    //Input set to PC5
    //Output set to PB0
}



uint8_t readCOMP(void)
{
    uint8_t Value;
    Value = COMP1->CSR && COMP_CSR_VALUE; //return status of comparitor
    return Value;
}
```

Comp.h

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

#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
 *      Author: Luke Lopez
 */
/* 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 = 3;                                          //DAC is
connected to peripherals and external pin

      //Setup Input GPIO
      RCC->AHB2ENR   |=  (RCC_AHB2ENR_GPIOAEN);       //enable GPIOB
      DAC_PORT->MODER &= ~(GPIO_MODER_MODE4);   //clear mode register
      DAC_PORT->MODER |= (GPIO_MODER_MODE4);          //Set mode to Analog
mode
```

```
	DAC_PORT->PUPDR &= ~(GPIO_PUPDR_PUPD4);   //Set to no Pull-up,
pull-down
	DAC->CR |= DAC_CR_EN1;                            //enable DAC Ch.1

}

void DAC_write(uint16_t Volt )
{
	DAC->DHR12R1 &= ~(0xFFF);           //clear DAC
	DAC->DHR12R1 = Volt;                //Write to DAC
}
//DAC->DHR12R1 //DAC 1 data register
```

DAC.h

```
/*
 * DAC.h
 *
 *  Created on: Nov 26, 2021
 *      Author: Luke Lopez
 */
#include <stdint.h>

#ifndef SRC_DAC_H_
#define SRC_DAC_H_

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

#endif /* SRC_DAC_H_ */
```

ADC.c

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

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 DAC_volt_conv(uint16_t mVolt)
{
	int DAC_count;
	DAC_count = (mVolt * DAC_RES)/ (VREF * 1000);
	return DAC_count;
}



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

#endif /* SRC_ADC_H_ */

---------------------------------------------------------------------
```

Timer.h

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

#ifndef SRC_TIMER_H_
#define SRC_TIMER_H_

void TIM2init(void);

#endif /* SRC_TIMER_H_ */

-----------------------------------------------------------------
Timer.c
----------------------------------------------------------


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

// output comp medium speed
// timer 2 ic4f

// initialize TIM3 for sampling Vref for ideally 1 s
```

```
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);

      TIM2->SR &= ~(TIM_SR_CC4IF);

      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

}
```

```
------------------------------
Main.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>

/* Private function prototypes
-----------------------------------------------*/
void SystemClock_Config(void);
void delay1_us(const uint16_t time_us);


#define THREE_VOLTS 3722


//void Get_digit(void);
uint16_t findavg(void);
uint16_t volt_conv(uint16_t digital);
uint16_t findmax(void);
uint16_t findmin(void);
char freq[4];
char average[4];

// Global Variables
uint16_t ADC_Value; // global to read in ISR and main
// uint8_t ADC_flag = 0;

//uint16_t **ADC_Values; // global array to store ADC_Value

uint32_t max, min;
uint16_t avg;
```

```
uint8_t ADC_Flag = 0;
//uint32_t samples = 45000;
uint32_t frequency;
uint32_t edge1, edge2;
uint32_t sum = 0;
int j = 0;
uint16_t samples = 100000;
//int samples = 5;
int main(void) {
      HAL_Init();                   //HAL Configuration
      SystemClock_Config();   //Clock Cinfiguration
      UART_init();                  //USART configuration
      ADC_init();                   //ADC config
      DAC_init();
      COMPInit();                   //why does COMP start triggered with no
VREF?
      TIM2init();
      __enable_irq();               //enable interrupts

      UART_send_esc_code("[2J"); // clears screen
      UART_send_esc_code("[H"); // go to top left position


      while (1)
      {

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

            if((j > samples ) && (ADC_Flag == 1))
            {
                  char summ[4];

                  sprintf(summ, "%" PRId32, sum);

                  avg = sum/samples;

                  DAC_write(avg);         //Output Vref
                  sprintf(average, "%" PRId16, avg);

                  j = 0;

            }
            if (frequency != 0)     //Clear overcapture flag)
            {
                  sprintf(freq, "%" PRId32, frequency); // converts 32 bit
integer into a string

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

                  UART_print_string("Frequency: ");

                  UART_print_string(freq);

                  UART_print_string(" Hz ");


            }

      }
}

void delay1_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
}

uint16_t volt_conv(uint16_t digital)
{
      uint32_t calvolt;
      calvolt = (801 * digital - 4670)/1000;
      return calvolt;
}

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 += 640;
      }
      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 = (32000000 / 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
      }
}

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 */
```