PWM Signal Generation and Monitoring

CENG 355 Laboratory Report

Report Submitted To : Saman Khoshbakht

Report Submitted On : November 30th, 2016

Section : B05

Names : Jakob Roberts v00484900
Ram Wierzbicki v00806259

Problem Description

A pulse-width-modulated (PWM) signal must be generated by an external timer. The PWM signal must be controlled, and monitored using an embedded system.

Requirements

The microcontroller on the STMF0 Discovery board will be used to measure the voltage across the PBMCUSLK board's potentiometer. This information will then be relayed to an optocoupler to control the PWM signal frequency. The frequency of the PWM and the resistance of the potentiometer will be displayed on the PBMCUSLK's LCD display.

Components

Controllers:	1x NE555 Timer
	1x 4N35 Optocoupler
Board:	1x STMF0 Discovery
	1x PBMCUSLK
Resistors:	As necessary

Design Solution

In researching the design we referred to the data sheet and manuals for each component to determine the specific functionality. The circuit diagram in the Ceng 355 Lab Manual was a good basis for interfacing components:

- PA1 pin from the discovery board connected to the output pin of the NE555
- PA4 connected to pin1 of the optocoupler
- PA3 connected to the POT pin on the PBMCUSLK

The rest of the circuit connections can be observed from the interfacing lab notes. Once the circuit was built software was written to take the power from the PBMCUSLK board, transfer it through the potentiometer then to the analog to digital converter (ADC). After the ADC the path then splits into two, going straight to the digital to analog converter (DAC) as well as being used to determine resistance which is displayed on the LCD using the SPI connection. The signal is required to go to the ADC first is so that the digital output can be processed onto the LCD. Continuing past the DAC, the signal travels to the optocoupler then the timer and then back to the microcontroller. The timer creates interrupt requests using EXTI and TIM2. The frequency is determined inside the requests. The frequency is then transmitted to the LCD using the SPI.

Functions were written to determine the frequency and resistance based off V=IR and f=1/T formulas where V=voltage, I = current, R= resistance, f=frequency and T= period. The period of calculation was set to the rising edges of the signal. Only 8 bits of data can be sent, when transmitting to the LCD.

When writing to the SPI, a delay was required since the microcontroller ran at a much higher speed. Without the delay the data would build up and cause an overload. Therefore a simple wait function was created.

When setting up the control registers, we followed the proposed design that was given in the course lecture slides. We wanted to follow this because it was very hard to determine what each specific value should be. Studying the microcontroller specification sheet was very difficult so we wanted to simplify this as much as possible. Each initialization function only set the necessary bits needed to produce the required functionality and nothing extra. Busy flags were only checked while in the main while loop, while sending data using SPI_SendData8() and inside the EXTI interrupt. These were the only times that this was necessary because these were the only times where important values were received and sent.

Diagrams

Figure 1 below shows an overview of the completed circuit; wired as described in the report descriptions requirements. Figure 2 shows the rate of change comparison between the frequency and the resistance. Figure 3 shows the Frequency as a function of the Resistance.

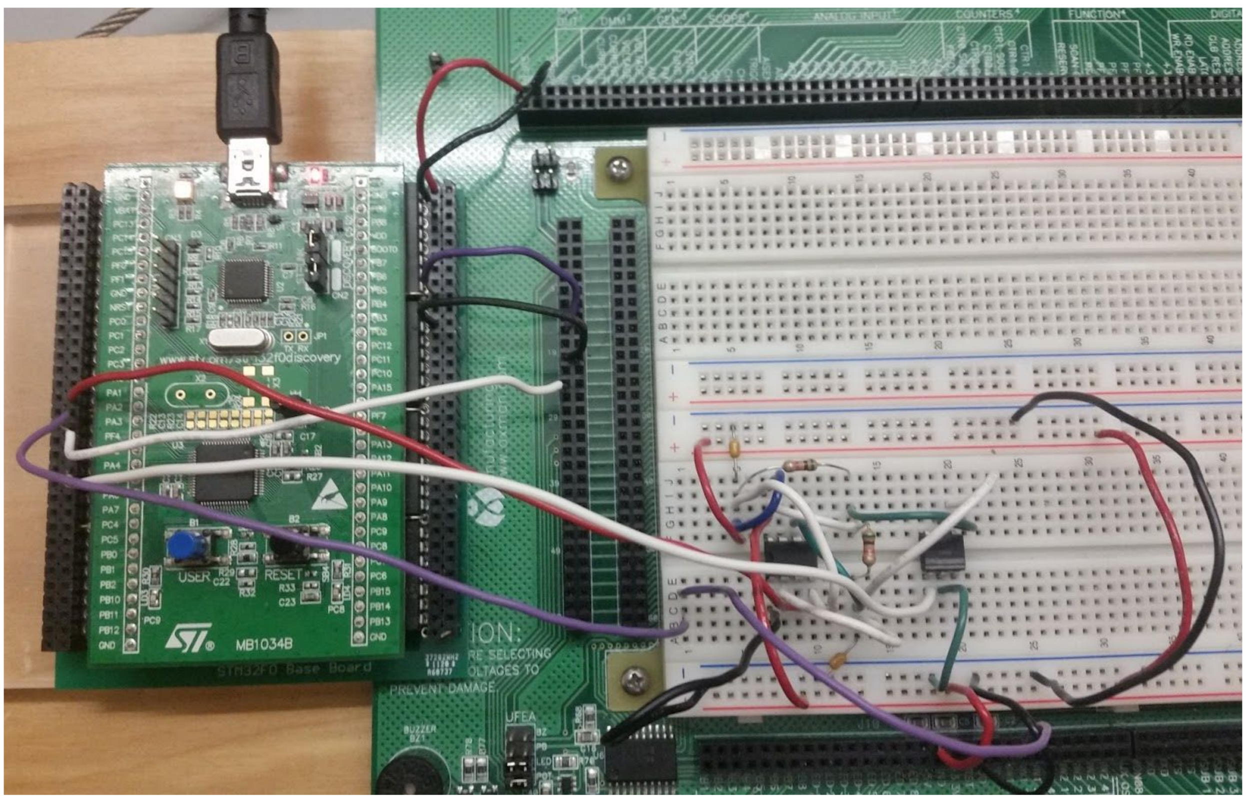

Figure 1: Full View of Completed Circuit

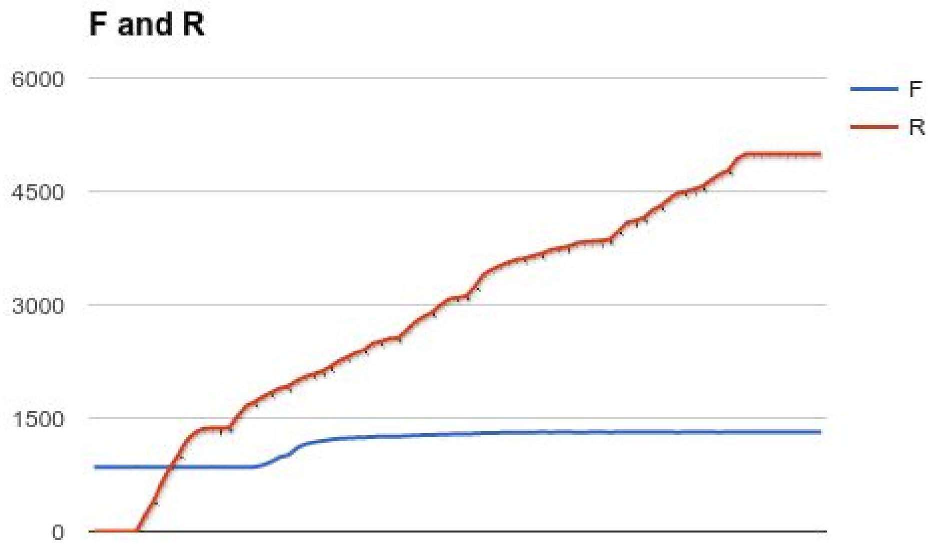

Figure 2: Rate of change comparison between Frequency and Resistance

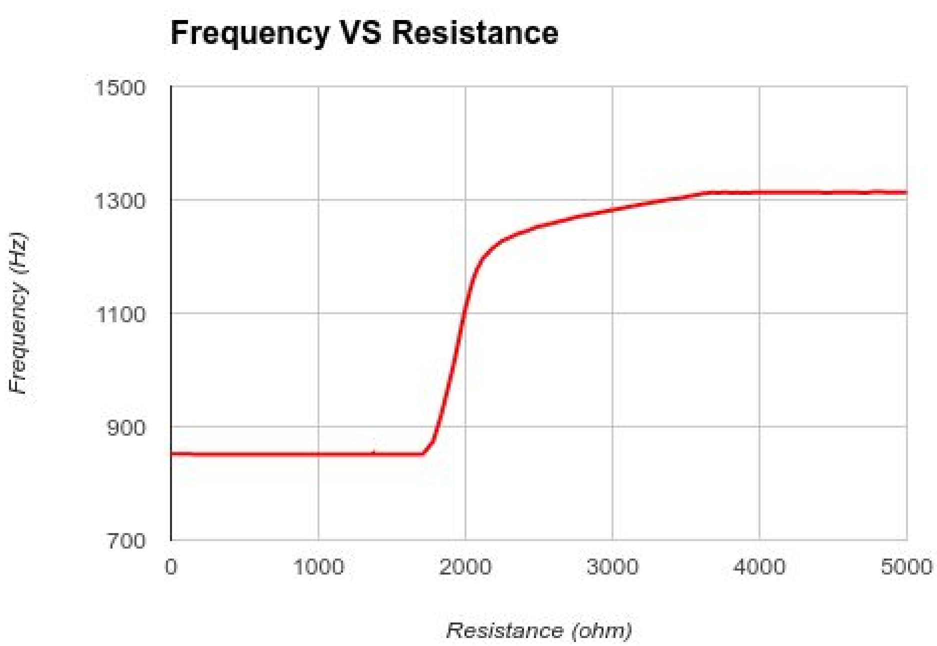

Figure 3: Frequency as a function of Resistance

Test Procedure and Results

To verify the behavior of the resistance display on the LCD we began with the potentiometer at its maximum position and saw that 5kΩ was printed on the display, then moved the potentiometer to its minimum position and saw a reading of zero on the LCD. Between the two extreme measurements we saw that the LCD was changing its resistance readings at a steady rate which matched our physical input with no fluctuation. The frequency was then changing accordingly as the resistance in the potentiometer changed.

As shown above in Figure 2, we can see that as the resistance increases, the frequency(F) doesn't change much until the resistance(R) hits about 1800ohms. At this point the resistance is increasing rapidly until the resistance reaches approximately 3600ohms and begins to slow. The frequency remains relatively constant as the resistance is above 3500ohms. Figure 3 shows further how the Frequency changes as a function of Resistance.

Discussion

The next paragraph will explain the main flow of the software. It will describe the initialization, to the calculation of the required values, to the displaying of the values on the provided display. The description will include the names of required function calls; please refer to the attached source code at the end of the document..

When the program starts up, it goes straight into the main function. Main calls the 8 initialization functions which set the correct bits for each corresponding hardware piece will work and initializing the LCD. Main then goes into an infinite while loop that starts the ADC then gets the resistance from the ADC register and outputs that value to the DAC. The frequency and resistance values then enter the process of getting displayed by calling DisplayFrequency(freq) and DisplayResistance(resistance).

Each of those functions are designed to split up the given value into 4 digits, thousands, hundreds, tens, and ones. Since each individual character must be sent to the LCD one at a time. Those functions then set the display to the correct line then send each digit along with characters for resistance and frequency. These characters and digits are sent using the SendToLCD(<digit in hex>) function. This function splits the characters into high and low order then are sent to the next step using DataEnable(uint8_t) which send data to the Serial Port Interface(SPI), this is complemented by the CommandEnable() function which sends commands to the SPI. These functions split the words and set the RS and EN bits. The words are sent 3 times in each function, this is done to ensure that it is properly sent. Waiting must happen after each SPI_SendData() because the SPI is much slower than the microcontroller.

SPI_SendData() takes data as its input. This function forces the LCD signal to 0, waits until SPI is ready then send the data using SPI_SendData8() (built-in function), this function sends data to the LCD using the SPI. After that, the LCK is forced back to 1 and waiting happens to ensure that the SPI is ready before moving on. This all continues in a while loop in the main. It continually determines the frequency and resistance values and updates them on the display.

There were no features added in this design, although the main flow was nicely split up into separate functions for better logical flow. The given design proposes that all the work is to be done in the main function, but based off of proper software engineering methodologies, the work should be split up into different segregated functions where each function performs one task. The logical breakup of functions also makes the code simpler to follow and debug

The only errors that were encountered during testing were that we would have to occasionally re-plug in the board and do a power reset in order to clear the display properly.

The final code design was successfully demonstrated to our lab TA. The biggest challenge we had was correctly wiring the circuit and getting the 555 timer to work properly. This challenge was overcome by asking the proper questions to our TA to receive assistance with debugging the circuit.

Conclusion

When the final design for the CEng 355 project was completed, we had a better understanding on how to properly develop an embedded system for monitoring and controlling a pulse-width-modulated signal.

References

[1] CENG 355: Laboratory Manual
[2] STM32F051 Data Sheet
[3] STM32F051 Schematic
[4] STM32F0 Reference Manual
[5] 555 Timer Data Sheet
[6] 4N35 Optocoupler Data Sheet

Source Code

```c
// This file is part of the GNU ARM Eclipse distribution.
// Copyright (c) 2014 Liviu Ionescu.
// ----------------------------------------------------------------------------
// School: University of Victoria, Canada.
// Course: CENG 355 "Microprocessor-Based Systems".
//
// Jakob Roberts  - v00484900
// Ram Wierzbicki - v00806259
//
// See "system/include/cmsis/stm32f0xx.h" for register/bit definitions.
// See "system/src/cmsis/vectors_stm32f0xx.c" for handler declarations.
// ----------------------------------------------------------------------------
#include <stdio.h>
#include "diag/Trace.h"
#include "cmsis/cmsis_device.h"
// ----------------------------------------------------------------------------
// STM32F0 empty sample (trace via $(trace)).
//
// Trace support is enabled by adding the TRACE macro definition.
// By default the trace messages are forwarded to the $(trace) output,
// but can be rerouted to any device or completely suppressed, by
// changing the definitions required in system/src/diag/trace_impl.c
// (currently OS_USE_TRACE_ITM, OS_USE_TRACE_SEMIHOSTING_DEBUG/_STDOUT).
//
// ----- main() ---------------------------------------------------------------
// Sample pragmas to cope with warnings. Please note the related line at
// the end of this function, used to pop the compiler diagnostics status.
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wmissing-declarations"
#pragma GCC diagnostic ignored "-Wreturn-type"

/* Clock prescaler for TIM2 timer: no prescaling */
#define myTIM2_PRESCALER ((uint16_t)0x0000)
/* Maximum possible setting for overflow */
#define myTIM2_PERIOD ((uint32_t)0xFFFFFFFF)
/* SPI useful definitions */
#define SPI_Direction_1Line_Tx ((uint16_t)0xC000)
#define SPI_Mode_Master ((uint16_t)0x0104)
#define SPI_DataSize_8b ((uint16_t)0x0700)
#define SPI_CPOL_Low ((uint16_t)0x0000)
#define SPI_CPHA_1Edge ((uint16_t)0x0000)
#define SPI_NSS_Soft SPI_CR1_SSM
#define SPI_FirstBit_MSB ((uint16_t)0x0000)
#define SPI_CR1_SSM ((uint16_t)0x0200)

void myGPIOA_Init(void);
void myTIM2_Init(void);
```

```c
void myEXTI_Init(void);
void myADC_Init(void);
void myDAC_Init(void);
void mySPI_Init(void);
void myLCK_Init(void);
void myLCD_Init(voids);
void SPI_SendData(uint8_t);
void Wait(volatile long);
void CommandEnable(uint8_t);
void CommandSend(uint8_t);
void SendToLCD(uint8_t);
void DisplayResistance(int);
void DisplayFrequency(int);
void DataEnable(uint8_t);

// ----- globals -------------------------------------------------------------
volatile int firstedge = 0;
int freq = 0;
int resistance = 0;

int main(int argc, char* argv[]){
    myGPIOA_Init();     /* Initialize I/O port PA */
    myTIM2_Init();      /* Initialize timer TIM2 */
    myEXTI_Init();      /* Initialize EXTI */
    myADC_Init();       /* Initialize ADC */
    myDAC_Init();       /* Initialize DAC */
    mySPI_Init();       /* Initialize SPI */
    myLCK_Init();       /* Initialize LCK */
    myLCD_Init();       /* Initialize LCD */

    while (1){
        //Pot Voltage = 3.26v , VCC voltage = 2.97v  (POT/Vcc)*4095 = 4494.84
        ADC1->CR |= ADC_CR_ADSTART ;                  //ADC start
        while ((ADC1 -> ISR & ADC_ISR_EOSMP) != 0 ){
            DAC->DHR12R1 = ADC1->DR;                  // Writes output of ADC to input of DAC
            resistance = (ADC1->DR)*5000/4095;        // Resistance calculation
            DisplayFrequency(freq);                   //Sends freq to be displayed
            DisplayResistance(resistance);            //Sends resistance to be displayed
            // trace_printf("F: %u R: %u\n", freq, resistance);
        }
    }
    return 0;
}

/*****************************************************************************
 * Calcualtes the resistance and sends out to LCD, uses:
 *    CommandSend(), SendToLCD()
 * Note: using +'0' converts int to ascii form
 */
void DisplayResistance(int Resistance){
```

```
    int tho = 0;       //Init 1000s digit
    int hun = 0;       //Init 100s digit
    int ten = 0;       //Init 10s digit
    int one = 0;       //Init 1s digit

    tho = (Resistance/1000) % 10;                //Splits resistance into 4 digits to send
    hun = (Resistance/100 ) % 10;
    ten = (Resistance/10  ) % 10;
    one = (Resistance     ) % 10;

    CommandSend(0xC0);          //Set LCD display to second line
    SendToLCD(0x52);            //"R"
    SendToLCD(0x3A);            //":"
    SendToLCD(tho+'0');         //Send 1000s digit
    SendToLCD(hun+'0');         //Send 100s digit
    SendToLCD(ten+'0');         //Send 10s digit
    SendToLCD(one+'0');         //Send 1s digit
    SendToLCD(0x4F);            //"O"
    SendToLCD(0x68);            //"h"/
}

/******************************************************************************
  * Calcualtes the frequency and sends out to LCD, uses:
  *    CommandSend(), SendToLCD()
  * Note: using +'0' converts int to ascii form
 */
void DisplayFrequency(int Frequency){
    int tho = 0;      //Init 1000s digit
    int hun = 0;      //Init 100s digit
    int ten = 0;      //Init 10s digit
    int one = 0;      //Init 1s digit

    tho = (Frequency/1000) % 10;     //Splits frequency into 4 digits to send
    hun = (Frequency/100 ) % 10;
    ten = (Frequency/10  ) % 10;
    one = (Frequency     ) % 10;

    CommandSend(0x80);          //Set LCD display to first line
    SendToLCD(0x46);            //"F"
    SendToLCD(0x3A);            //":"
    SendToLCD(tho+'0');         //Send 1000s digit
    SendToLCD(hun+'0');         //Send 100s digit
    SendToLCD(ten+'0');         //Send 10s digit
    SendToLCD(one+'0');         //Send 1s digit
    SendToLCD(0x48);            //"H"
    SendToLCD(0x7A);            //"z"
}

/******************************************************************************
 * Used by DisplayFrequency and DisplayResistance to change line on Display
```

```
 * Uses:
 *    CommandEnable()
 */
void CommandSend(uint8_t Word){
    uint8_t HighOrder = ((Word >> 4) & 0x0F);       //Shifts most significant bits to the
least significant side and masks the most significant bits with 0
    CommandEnable(HighOrder);                       //Sends command value

    uint8_t LowOrder = (Word & 0x0F);               //No need to shift bits, mask the higher
order bits
    CommandEnable(LowOrder);
}

/******************************************************************************
 * Creates the High and Low filter to send to the SPI
 * Used by DisplayFrequency and DisplayResistance to change line on Display
 */
void CommandEnable(uint8_t Word){
    uint8_t EN = Word | 0x80; //For sending commands - RS = 0 EN = 1

    SPI_SendData(Word);     //Send the enable word to the SPI
    Wait(300);              //Delay for SPI to process

    SPI_SendData(EN);
    Wait(300);

    SPI_SendData(Word);
    Wait(300);
}

/******************************************************************************
 * Creates the High and Low filter to send to the SPI
 * Used by DisplayFrequency and DisplayResistance to display the digits
 */
void SendToLCD(uint8_t Word){
    uint8_t HighOrder = ((Word >> 4) & 0x0F);   //Shifts most significant bits to the least
significant side and masks the most significant bits with 0
    DataEnable(HighOrder);                      //Sends data value

    uint8_t LowOrder = (Word & 0x0F);           //No need for shift
    DataEnable(LowOrder);
}

/******************************************************************************
 * Creates the High and Low filter to send to the SPI
 * Used by SendToLCD to display the digits
 */
void DataEnable(uint8_t Word){
    uint8_t HighpulseEN = Word | 0xC0;      //Sets high word for sending data - RS = 1 EN = 1
    uint8_t LowpulseEN = Word | 0x40;       //Sets low word for sending data - RS = 1 EN = 0
```

```c
    SPI_SendData(LowpulseEN);               //Sends low to SPI - EN = 0 RS = 1
    Wait(300);                              //Delay for SPI

    SPI_SendData(HighpulseEN);              //Sends high to SPI - EN = 1 RS = 1
    Wait(300);

    SPI_SendData(LowpulseEN);               //Sends low again
    Wait(300);
}

/*****************************************************************************
 * Sends required data to Serial Port Interface for the Display.
 * Is used by all functions interacting with the display.
 */
void SPI_SendData(uint8_t Data){
    GPIOA->BSRR |= GPIO_BSRR_BR_3; //Force LCK signal to 0
    Wait(5000);

    while ((SPI1->SR & SPI_SR_BSY ) != 0 ); // Wait until SPI1 is ready (TXE = 1 or BSY = 0)

    SPI_SendData8(SPI1,Data); //Built-in function

    while ((SPI1->SR & SPI_SR_BSY ) != 0 ); //Wait until SPI is ready

    GPIOA->BSRR |= GPIO_BSRR_BS_3; //Force LCK signal to 1
    Wait(5000);
}

/*****************************************************************************
 * Wait function to chew up clock cycles.
 */
void Wait(volatile long Time){
    while(Time >= 0){
        Time--;
    }
}

/*****************************************************************************
 * Fixes and init for PA1 & also configure PA2 and PA4
 */
void myGPIOA_Init(){
    /* Enable clock for GPIOA peripheral */
    //RCC->AHBENR = 0x10000;
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN;  //configure pa1 as input
    RCC->AHBENR |= RCC_AHBENR_GPIOBEN;
    //Configure PB1 as input
    GPIOB->AFR[2] = ((uint32_t)0x00000000);
    /* Configure PA1, PA2, PA4 to analog mode*/
    GPIOA->MODER |= (GPIO_MODER_MODER2 | GPIO_MODER_MODER4);
```

```
    GPIOB->MODER |= GPIO_MODER_MODER3_1 | GPIO_MODER_MODER4_0 | GPIO_MODER_MODER5_1;
    /* Ensure no pull-up/pull-down for PA1 */
    //relevant register: GPIOA->PUPDR
    GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR1 | GPIO_PUPDR_PUPDR2 | GPIO_PUPDR_PUPDR4);
    GPIOB->PUPDR &= ~(GPIO_PUPDR_PUPDR3 | GPIO_PUPDR_PUPDR4 | GPIO_PUPDR_PUPDR5);
}

/*****************************************************************************
 * Used for Frequency timer
 */
void myTIM2_Init(){
    /* Enable clock for TIM2 peripheral */
     // Relevant register: RCC->APB1ENR
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
    /* Configure TIM2: buffer auto-reload, count up, stop on overflow, * enable update events,
interrupt on overflow only */
    // Relevant register: TIM2->CR1
    TIM2->CR1 = ((uint16_t)0x008C);
    /* Set clock prescaler value */
    TIM2->PSC = myTIM2_PRESCALER;
    /* Set auto-reloaded delay */
    TIM2->ARR = myTIM2_PERIOD;
    /* Update timer registers */
    // Relevant register: TIM2->EGR
    TIM2->EGR = ((uint16_t)0x0001);
    /* Assign TIM2 interrupt priority = 0 in NVIC */
    // Relevant register: NVIC->IP[3], or use NVIC_SetPriority
    NVIC_SetPriority(TIM2_IRQn, 0);
    /* Enable TIM2 interrupts in NVIC */
    // Relevant register: NVIC->ISER[0], or use NVIC_EnableIRQ
    NVIC_EnableIRQ(TIM2_IRQn);
    /* Enable update interrupt generation */
    // Relevant register: TIM2->DIER
    TIM2->DIER |= TIM_DIER_UIE;
}

/*****************************************************************************
 * initialize PA1
 */
void myEXTI_Init(){
     /* Map EXTI1 line to PA1 */
     // Relevant register: SYSCFG->EXTICR[0]
    SYSCFG->EXTICR[0] &= ((uint32_t)0xFFFFFF0F);
    SYSCFG->EXTICR[0] |= 0x80;
    /* EXTI1 line interrupts: set rising-edge trigger */
    // Relevant register: EXTI->RTSR
    EXTI->RTSR |= EXTI_RTSR_TR1;
    /* Unmask interrupts from EXTI1 line */
    // Relevant register: EXTI->IMR
    EXTI->IMR |= EXTI_IMR_MR1;
```

```c
    /* Assign EXTI1 interrupt priority = 0 in NVIC */
    //NVIC_SetPriority;
    NVIC->IP[1] = ((uint32_t)0x00000000);
    /* Enable EXTI1 interrupts in NVIC */
    //NVIC_EnableIRQ;
    //NVIC->ISER[0] = ((uint32_t)0x00000800);//
    //--- /* Assign TIM2 interrupt priority = 0 in NVIC */
    //NVIC_SetPriority(EXTI0_1_IRQn, 0);
    // Same as: NVIC->IP[3] = ((uint32_t)0x00FFFFFF);
    NVIC->ISER[0] = ((uint32_t)0xFFFFFFFF);
    /* Enable TIM2 interrupts in NVIC */
    //NVIC_EnableIRQ(EXTI0_1_IRQn);
    // Same as: NVIC->ISER[0] = ((uint32_t)0x00000800) */ //---
}

/*****************************************************************************
 * This handler is declared in system/src/cmsis/vectors_stm32f0xx.c
 */
void TIM2_IRQHandler(){
     /* Check if update interrupt flag, bit 0, is 1 if it is then theres an overflow */
     if ((TIM2->SR & TIM_SR_UIF) != 0){
        trace_printf("\n*** Overflow! ***\n");
        /* Clear update interrupt flag */
        // Relevant register: TIM2->SR
        /* Clear update interrupt flag */
        TIM2->SR &= ~(TIM_SR_TIF);
            /* Restart stopped timer */
        TIM2->CR1 &= ~(TIM_CR1_CEN);
     }
}

/*****************************************************************************
 * Get frequency from square wave machine.
 * This handler is declared in system/src/cmsis/vectors_stm32f0xx.c
 */
void EXTI0_1_IRQHandler(){
    unsigned int clockcycles;
    clockcycles = TIM2->CNT;
    if((EXTI->PR & EXTI_PR_PR1) != 0){
        if(firstedge){
            firstedge = 0;
            TIM2->CNT = 0;
            TIM2->CR1 |= TIM_CR1_CEN;
        }else{
            firstedge = 1;
            EXTI->IMR &= ~(EXTI_IMR_MR1);
            TIM2->CR1 &= ~(TIM_CR1_CEN);
            if(SystemCoreClock/clockcycles > 4)
              freq = SystemCoreClock/clockcycles;
            EXTI->IMR |= EXTI_IMR_MR1;
```

```c
        }
        EXTI->PR |= 0x2; //Clear EXTI1 interrupt pending flag.
    }
}

/******************************************************************************
 * initialize LCD and clear it
 */
void myLCD_Init(){
    SPI_SendData(0x02); //Initializing EN = 0
    Wait(300);

    SPI_SendData(0x82); //Initializing EN = 1
    Wait(300);

    CommandSend(0x28);  //DL = 0, N = 1, F = 0
    CommandSend(0x0C);  //D = 1, C = 0, B = 0
    CommandSend(0x06);  //I/D = 1, S = 0
    CommandSend(0x01);  //Clear display
    Wait(50000);        //Delay for initialization for LCD
}

/******************************************************************************
 * initialize PA3 to LCK
 */
void myLCK_Init(){ //Configure PA3 as output for the LCK
    GPIOA->MODER |=  GPIO_MODER_MODER3_0;    // Configure PA3 to output
    GPIOA->OSPEEDR |= GPIO_OSPEEDR_OSPEEDR3; // GPIO port output Speed Register 00 or 01 low
speed,
    GPIOA->PUPDR &= ~(0xC0);                 // Configure  no pull up -pull down:
    GPIOA->BSRR |= GPIO_BSRR_BS_3;           //Drive PA3 High, using BSRR
}

/******************************************************************************
 * initialize connection to ADC
 */
void myADC_Init(){
    RCC->APB2ENR |= RCC_APB2ENR_ADCEN;      //ADC1 clock enable
    ADC1->CFGR1 |= ADC_CFGR1_CONT;          //This bit will ensure the ADC is in  continuous
mode and not single conversion mode
    ADC1->CHSELR |= ADC_CHSELR_CHSEL2;      //Channel 2 is selected for conversion
    ADC1->SMPR |= ADC_SMPR_SMP;             //sampling time selection (239.5 ADC cycles)
    ADC1->CR |= ADC_CR_ADEN;
    while((ADC1->ISR & ADC_ISR_ADRDY) == 0);//wait until ADC is ready
}

/******************************************************************************
 * initialize connection to DAC
 */
void myDAC_Init(){
```

```c
    RCC->APB1ENR |= RCC_APB1ENR_DACEN;      //DAC clock enable
    DAC->CR |= DAC_CR_EN1;                  //DAC channel 1 enable and set software trigger
    //Wake-up time
    DAC->CR |= DAC_CR_TSEL1;                //Setting timer trigger source
    DAC->SWTRIGR |= DAC_SWTRIGR_SWTRIG1;    //Enabling software trigger register
}

/******************************************************************************
 * initialize connection to Serial Port Interface
 */
void mySPI_Init(){
    RCC-> APB2ENR |= RCC_APB2ENR_SPI1EN ;
    SPI_InitTypeDef SPI_InitStructInfo;
    SPI_InitTypeDef* SPI_InitStruct = &SPI_InitStructInfo;
    SPI_InitStruct->SPI_Direction = SPI_Direction_1Line_Tx;
    SPI_InitStruct->SPI_Mode = SPI_Mode_Master;
    SPI_InitStruct->SPI_DataSize = SPI_DataSize_8b;
    SPI_InitStruct->SPI_CPOL = SPI_CPOL_Low;
    SPI_InitStruct->SPI_CPHA = SPI_CPHA_1Edge;
    SPI_InitStruct->SPI_NSS = SPI_NSS_Soft;
    SPI_InitStruct->SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_256 ;
    SPI_InitStruct->SPI_FirstBit = SPI_FirstBit_MSB;
    SPI_InitStruct->SPI_CRCPolynomial = 7;
    SPI_Init(SPI1, SPI_InitStruct);
    SPI_Cmd(SPI1, ENABLE);
}

#pragma GCC diagnostic pop
//============================================================================
```