# Lab 6 – Ultrasonic Sensor

Prepared for:

Professor Bill Stefunak

Submitted by:

Graeme Judge and Sean Berkvens

04/07/2020

Electronic Systems engineering (ESE Year 2)

Conestoga College

## Main.c

```c
#include "stm32l476xx.h"
#include "SysClock.h"

#include "DistanceSensor.h"
#include "lcd.h"

#define DELAY   for(int i = 0; i < 10000000UL; i++)   //delay function

int main(void){
  System_Clock_Init(); // Switch System Clock = 80 MHz

  initDistanceSensor();
  LCDInit();

  while(1){
    uint32_t distance = getDistanceCM();    //get the distance
    if(distance != -1){                     //as long as the value is within range
      LCDprintf("Distance: %d", distance);  //print distance
    }
    else{
      LCDprintf("Invalid");
    }
    DELAY;
    LCDclear();
  }
}
```

## DistanceSensor.c

```c
#include "DistanceSensor.h"

static volatile uint32_t pulseWidth = 0;

#define MAX_ECHO 15000
#define TRIG_PIN 6

void initDistanceSensor(){
  initTrigger();
  initInterrupts();
}

static void initTrigger(){
  uint32_t pulseWidth_uS = 200;
  uint32_t pulseDelay_uS = 500000;
  float dutyCycle = (float)pulseWidth_uS / (float)pulseDelay_uS;  //calculating duty cycle

  SET_BITS(RCC->AHB2ENR, RCC_AHB2ENR_GPIOBEN);                   //enable clock for port b
  FORCE_BITS(GPIOB->MODER, 3UL<< (2*6), 2UL << (2*6));           //set mode to alternate function
  FORCE_BITS(GPIOB->AFR[0], 0XF<<(4*6), 2UL<<(4*6));             //select the alternate function 2
  FORCE_BITS(GPIOB->PUPDR, 3UL << (2*6), 0);                     //no pull up or down
  SET_BITS(RCC->APB1ENR1, RCC_APB1ENR1_TIM4EN);                  //enable the clock
  CLR_BITS(TIM4->CR1, TIM_CR1_DIR);                              //set the counting direction to up
  TIM4->PSC = 80000-1;                                           //prescalar 1khz
  TIM4->ARR = 5000-1;                                            //auto reload every 0.5ms
  CLR_BITS(TIM4->CCMR1, TIM_CCMR1_OC1M);                         //clear channel 1 compare registers
  TIM4->CCR1 = dutyCycle * (TIM4->ARR + 1);                      //setting the duty cycle
  SET_BITS(TIM4->BDTR, TIM_BDTR_MOE);                            //main output enable
  SET_BITS(TIM4->CCMR1, TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_2);    //set the pwm mode
  CLR_BITS(TIM4->CCER, TIM_CCER_CC1P);                           //active high for the output
  SET_BITS(TIM4->CCER, TIM_CCER_CC1E);                           //selecting channel 1
  SET_BITS(TIM4->CR1,TIM_CR1_CEN);                               //enable the counter
}
```

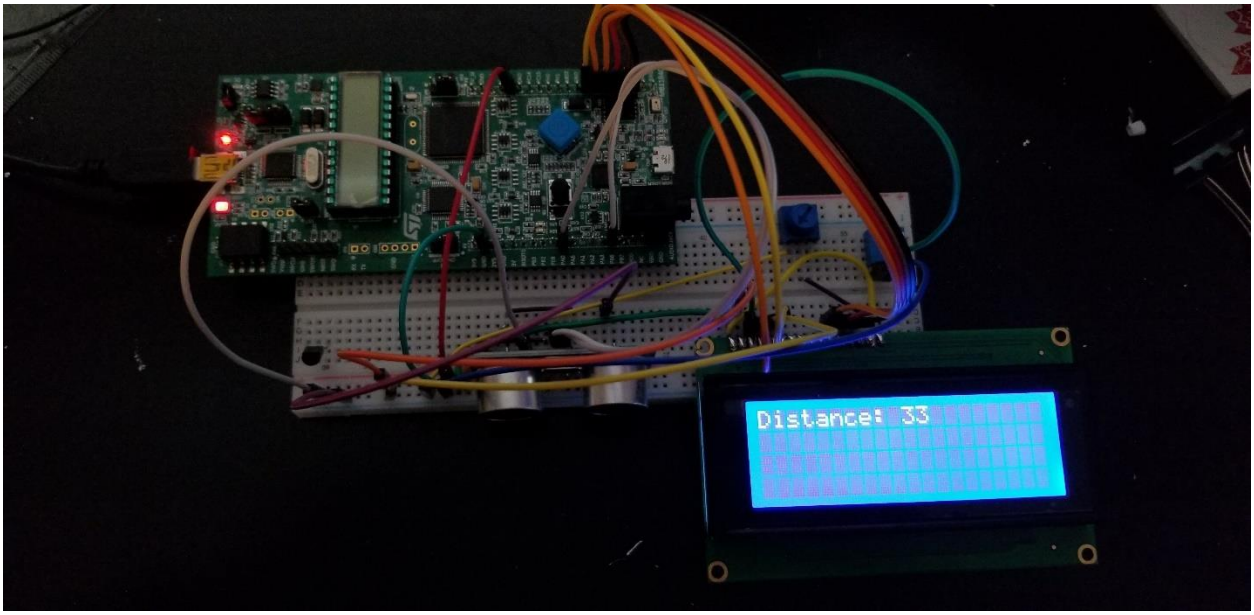
```c
static void initInterrupts(){
  //port set up
  SET_BITS(RCC->AHB2ENR, RCC_AHB2ENR_GPIOAEN);                   //enable clock for port A
  GPIOA->MODER &= ~(3UL);                                        //clear mode
  GPIOA->MODER |= 2UL;                                           //alternate function mode
  GPIOA->AFR[0] &= ~(0XF);                                       //clear alternate function selection
  GPIOA->AFR[0] |= 1UL;                                          //select alternate function
  GPIOA->OSPEEDR &= ~3UL;
  GPIOA->OSPEEDR |= 2UL;
  GPIOA->PUPDR &= ~3UL;
  //interrupt set up
  RCC->APB1ENR1 |= RCC_APB1ENR1_TIM2EN;                          //ENABLE CLOCK
  TIM2->PSC = 80UL;                                              //PRESCALAR 1MHz
  TIM2->ARR = 0XFFFF;                                            //AUTORELOAD AT MAX VALUE
  TIM2->CCMR1 &= ~TIM_CCMR1_CC1S;                                //CLEAR CAPTURE AND COMPARE SELECTIONS BITS
  TIM2->CCMR1 |= TIM_CCMR1_CC1S_0;                               //CC1S[1:0] FOR CHANNEL 1
  TIM2->CCMR1 &= ~TIM_CCMR1_IC1F;                                //NO FILTERING
  TIM2->CCER &= ~(TIM_CCER_CC1P|TIM_CCER_CC1NP);                 //CLEAR POLARITY
  TIM2->CCER |= TIM_CCER_CC1P;                                   //ONLY FALLING EDGE CREATES INTERRUPTS
  TIM2->SMCR &= ~TIM_SMCR_TS;                                    //CLEAR TRIGGER SELECTION
  TIM2->SMCR |= 4UL << 4;                                        // T1 EDGE DETECTOR
  TIM2->SMCR &= ~TIM_SMCR_SMS;                                   //CLEAR SLAVE MODE BIT
  TIM2->SMCR |= 4;                                               //1000 SLAVE MODE WITH RESET
  TIM2->CCMR1 &= ~(TIM_CCMR1_IC1PSC);                            //CLEAR INPUT PRESCALAR
  TIM2->CCER |= TIM_CCER_CC1E;                                   //ENABLE CAPTURE FOR CHANNEL 1
  TIM2->DIER |= TIM_DIER_CC1IE;                                  //ALLOW TIMER 2 CHANNEL 1 TO GENERATE INTERRUPTS
  TIM2->DIER |= TIM_DIER_CC1DE;
  TIM2->CR1 |= TIM_CR1_CEN;                                      //ENABLE COUNTING
  NVIC_SetPriority(TIM2_IRQn, 0);                                //SETS PRIORITY TO 0 (HIGHEST)
  NVIC_EnableIRQ(TIM2_IRQn);                                     //ENABLE THE NVIC
}
```

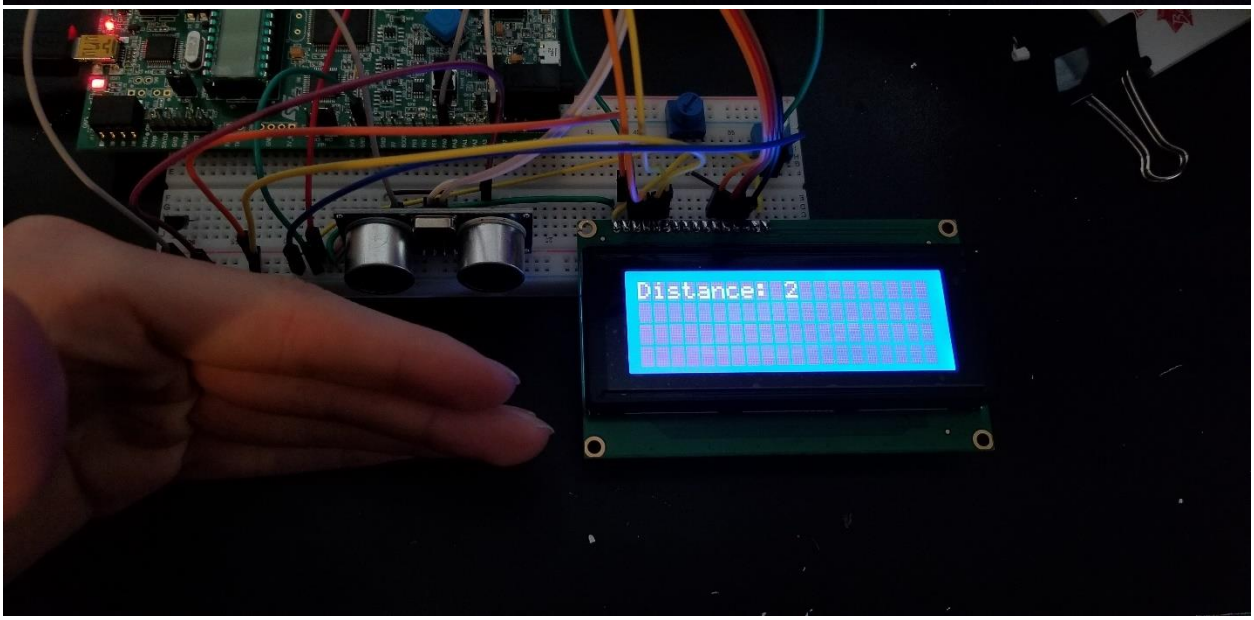
```
67 void TIM2_IRQHandler(void){
68   if((TIM2->SR & TIM_SR_UIF) != 0){
69       TIM2->SR &= ~TIM_SR_UIF;
70   }
71   if((TIM2->SR & TIM_SR_CC1IF) != 0){
72     pulseWidth = TIM2->CCR1;
73   }
74 }
75
76 uint32_t getDistanceCM(){
77   if(pulseWidth <= MAX_ECHO){
78     return(pulseWidth/58);
79   }else{
80     return(-1);
81   }
82
83 }
84
```

**DistanceSensor.h**

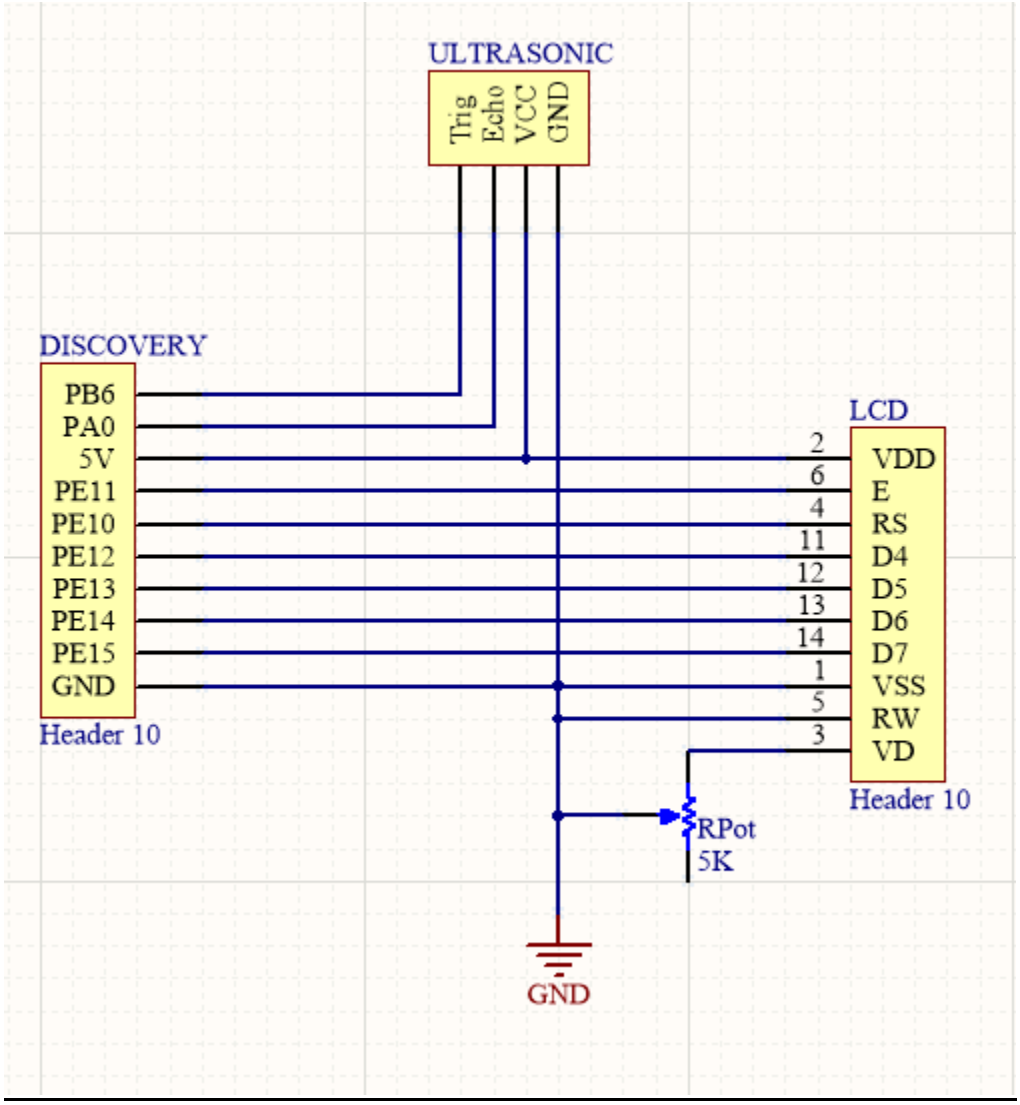
```
1  #include "stm321476xx.h"
2  #include "utils.h"
3
4  void initDistanceSensor();
5
6  static void initTrigger();
7
8  static void initInterrupts();
9
10 uint32_t getDistanceCM();
```

## Working images



**Circuit diagram**



**Demo**

https://youtu.be/ohV1teXcHhY