Lab 3 Report

Ultrasonic sensor

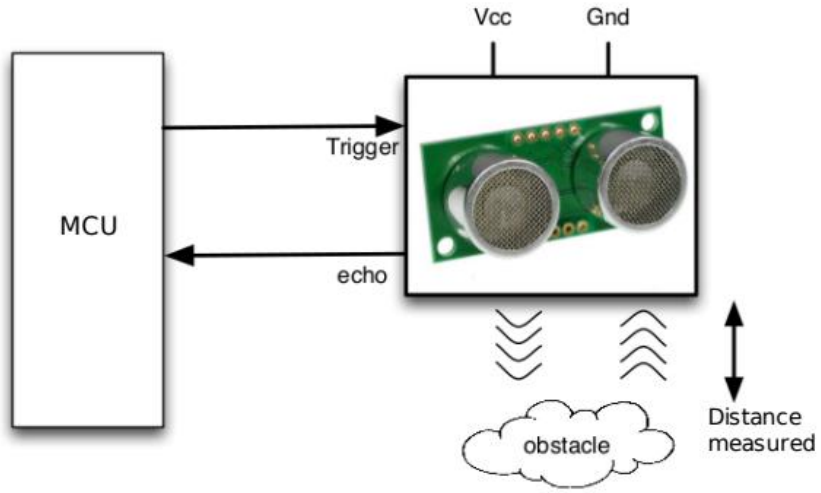
PRASANNA KUMAR Vinayak and BLAKUNOVS Sergejs

20.12.2019

Introduction

The purpose of this lab was to implement a driver for ultrasonic distance meter using interrupt implementation instead of polling approach. Drivers were tested for a simple application.

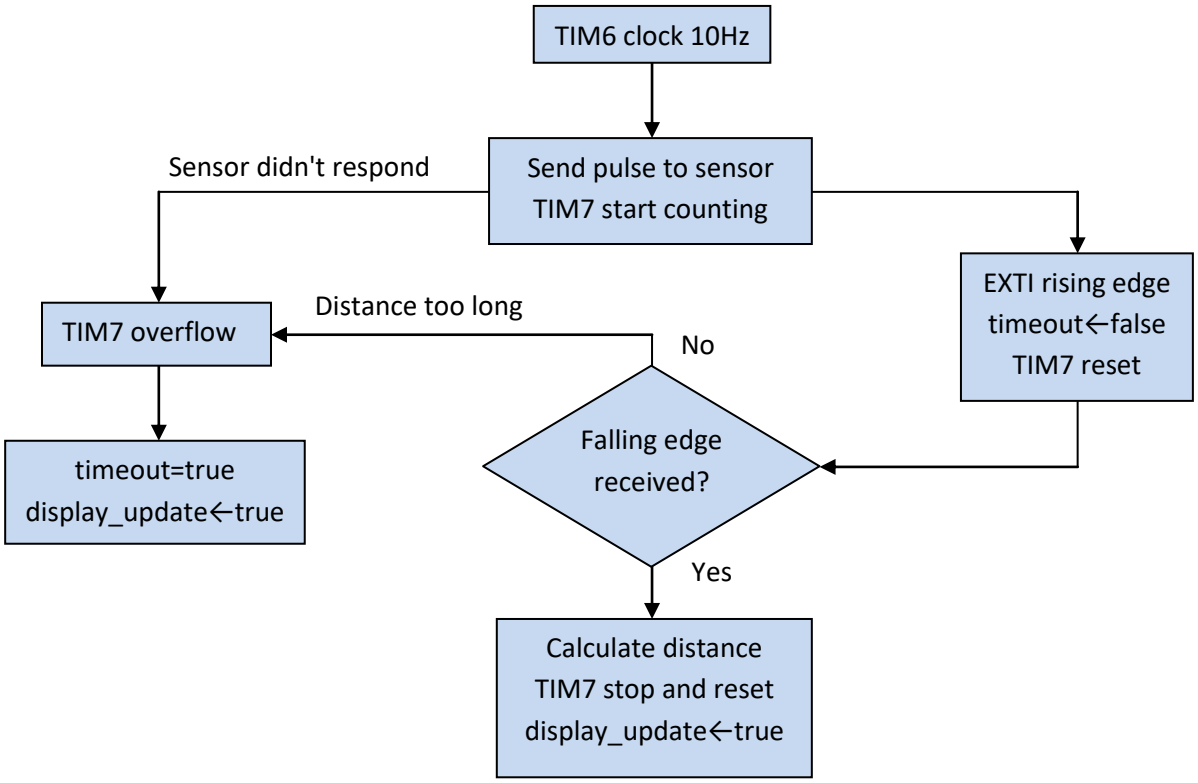
The sensor embeds an integrated circuit for an easy management by the MCU. The interface consists of only 2 logic signals (*Trigger and echo*), as in figure below.

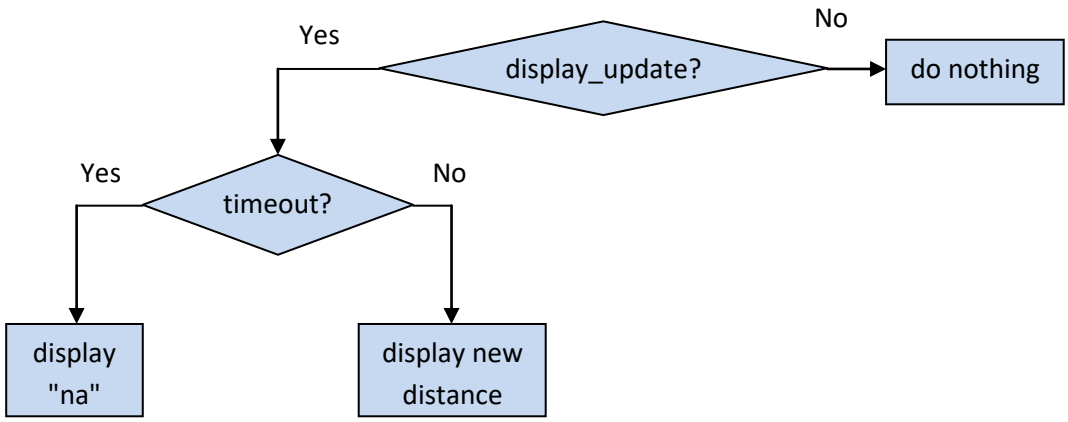


Pin PA10 of microcontroller was used both to send a *trigger* pulse to the microcontroller and also to receive an *echo* signal from the sensor.
Operation of the ultrasonic sensor module is as follows:
A short *trigger* pulse from MCU is a request to make a distance measurement. An ultrasonic burst is generated and *echo* line goes from "low" to "high". This rising edge is an indication of start of the measurement. Upon receiving the reflected ultrasonic burst, the sensor module sets *echo* line back to "low". This falling edge is an indication of the end of the measurement. MCU uses external interrupts to register "rising edge" and "falling edge" events on pin PA10 and a timer TIM7 to measure the elapsed time τ. The distance to the obstacle is then calculated as

$$d(cm) = \frac{\tau(\mu s)}{58(\mu s/cm)}$$

Measurements are made every 100ms, which is ensured by timer TIM6.
Logics of the program can be described by 2 block diagrams, one for interrupts and another for the main loop, see figure below.





Timer TIM6 is used to generate a clock signal at frequency 10Hz. At each clock cycle GPIO on pin PA10 is used to send a trigger pulse to the ultrasonic sensor. Timer TIM7 is used for 2 purposes - to measure the time taken for ultrasonic pulse to travel from transmitter to obstacle and back to receiver (and hence calculate the distance to the obstacle) and also to detect if sensor is disconnected/not responding.

When trigger pulse is sent, pin PA10 is set as a pull-up input, to receive the echo signal. Pull-up is needed to avoid pick-up of the noise when sensor is disconnected.

If sensor responds, to trigger pulse, PA10 goes from low to high state and rising edge is detected by an external interrupt handler for PA10. Counter of Timer TIM7 is set to zero and timeout flag is set to false as sensor has responded to trigger pulse.

TIM7 continues counting. In case a falling edge is detected by interrupt handler, counter value is

used to calculate distance to the obstacle. TIM7 is stopped to avoid TIM7 overflow, as this would result in "Sensor not available" message when the sensor is actually connected and distance value is measured.
If sensor does not respond to the trigger pulse or no reflected ultrasonic pulse is received, TIM7 will overflow. This results in "Sensor not available" message being displayed.

A *display_update* flag is used to detect if there is new data to be displayed. This prevents update of display on every single loop cycle and hence eliminates most of display flickering. *Timeout* flag is set if ultrasonic module does not respond to the trigger signal or no reflected signal is detected.

Next section gives details of implementation.

Implementation

1.Setup

```
void setup()
{
    tft.initR(INITR_BLACKTAB);   // initialize a ST7735S chip, black tab
    tft.setFont(&FONT_NAME);
    tft.fillScreen(ST7735_BLACK);
    tft.setTextColor(ST7735_RED);
    tft.setCursor(10, FONT_HEIGHT);
    tft.print("Hello World! ");
    ADCInit();

    RCC->AHBENR |= RCC_AHBENR_GPIOBEN_Msk; //clock for GPIOB
    asm("nop");                            //wait until GPIOB clock is Ok.
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN_Msk; //clock for GPIOA
    asm("nop");                            //wait until GPIOA clock is Ok
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;  //clock for the SYSCFG peripheral- external
                                           interrupts
    asm("nop");

    /*Initialization of GPIOA and B (Pins 10 and 0 respectively)*/
    GPIOA->MODER&= ~(3<<(10*2));
    GPIOB->MODER&= ~(3<<0*2);
    GPIOB->MODER|=(1<<0*2);
    GPIOA->MODER|= 1<<(10*2);

    //timer setup
    //input clock = 64MHz.
    /*Timer 6 initialization*/
    RCC->APB1ENR|= RCC_APB1ENR_TIM6EN;
    //reset peripheral (mandatory!)
    RCC->APB1RSTR |= RCC_APB1RSTR_TIM6RST;
    RCC->APB1RSTR &=~RCC_APB1RSTR_TIM6RST;
    asm("nop");

    /*Timer 7 initialization*/
    RCC->APB1ENR|= RCC_APB1ENR_TIM7EN;
    RCC->APB1RSTR |= RCC_APB1RSTR_TIM7RST;
    RCC->APB1RSTR &=~RCC_APB1RSTR_TIM7RST;
    asm("nop");
```

```
    /*Timer 6 register initialization*/
    TIM6->PSC=6400-1;  /*PSC to count in steps of 100us*/
    TIM6->ARR=1000-1;  /*Count up to 1000 to get a delay of 100ms, which is 10Hz*/
    TIM6->CR1|=TIM_CR1_CEN;  /*Start timer 6*/

    /*Interrupt enable*/
    TIM6->DIER |= TIM_DIER_UIE;  /*Enable interrupt for timer 6*/
    NVIC_EnableIRQ(TIM6_DAC1_IRQn);

    /*Timer 7 register initialization*/
    TIM7->PSC=64-1;    /*PSC to count in steps of 1us*/
    TIM7->ARR=50000-1;  /*Count up to 50ms (50,000 us) for time out condition*/

    TIM7->DIER |= TIM_DIER_UIE;  /*Enable interrupt for timer 7*/
    NVIC_EnableIRQ(TIM7_IRQn);

    SYSCFG->EXTICR[2] |= SYSCFG_EXTICR3_EXTI10_PA;  /*Enable external interrupt for
                                                      port A*/
    EXTI->FTSR|=EXTI_FTSR_TR10;  /*Enable interrupt on falling edge*/
    EXTI->RTSR|=EXTI_RTSR_TR10;  /*Enable interrupt on rising edge*/
    NVIC_EnableIRQ(EXTI15_10_IRQn);
}
```

The setup function initializes the required timers, interrupts and the peripherals needed to make use of the tft. The clocks needed to drive the GPIOs A and B are started. Pin 10 of GPIO A is initially configured as an output. The configuration of this pin will be changed multiple times during the rest of the program as it is used as both an input and an output. Pin 0 of GPIO B is configured as an output and this LED is merely used for debug purposes.

Timers 6 and 7 are enabled. The PRC for TIM6 is chosen such that we have ticks which count in steps of 100us. This timer is responsible for providing the 100ms delay between two successive trigger pulses. Hence the ARR is set to 1000-1 and the counter overflows every 100ms (100us*1000 = 100ms). The interrupt for TIM6 is enabled.

For TIM7, the PRC is chosen such that we have ticks which count in steps of 1us. This is done as we need a much smaller increments when measuring the echo width. The ARR of TIM7 is set as 50000 - 1. This is because we want the overflow to occur for delays greater than 50ms (i.e. when the echo signal is not received for more than 50ms.) The interrupt for TIM7 is also enabled.

The last 4 lines of the setup enable the interrupt for pin 10 of GPIOA. The interrupt is enabled for both the rising and falling edge as we need to detect both to determine the signal width.

2.Timer TIM6 interrupt handler

```
void TIM6_DAC1_IRQHandler()
{

    TIM6->SR &= ~TIM_SR_UIF; //Acknowledge interrupt

    EXTI->IMR &= ~(1<<10);  /*Temporarily disable external interrupt EXTI10
                              This is done to prevent unwanted trigger of interrupt
                               when we write to the same pin*/
    GPIOA->MODER|= (1<<(10*2));  //Set pin A10 to output to transmit pulse to the
                                    sensor
    asm("nop");
    /*Generate pulse to start data acquisition*/
    GPIOA->ODR|= 1<<10;   /*Pin A10 set to high*/
    for(volatile int i=0;i<20;i++);  /*Small delay to ensure that pulse width is
                                       sufficient*/
    GPIOA->ODR&= ~(1<<10);  /*Set pin A10 back to low*/
    pinMode(GPIOA,10,INPUT_PULLUP);  //Set pin A10 to output as it needs to read the
                                       echo signal
    EXTI->IMR |= (1<<10);   /*Enable external interrupt  EXTI10 again*/
    TIM7->CR1|=TIM_CR1_CEN;   /*Start TIM 7 counting*/
}
```

Interrupt is acknowledged first - the overflow flag of TIM6 is reset to ensure this interrupt is not executed again. External interrupt EXTI10 was temporarily disabled to make sure that writing "1" or "0" to the pin does not trigger rising/falling edge external interrupt for PA10. Next pin PA10 is set to output. A short pulse is generated on PA10 by setting its state to high, waiting in for some time in loop and setting PA10 it to low again. Then PA10 is set to input to be able to receive echo signal. External interrupt is enabled again. Pull-up is used to prevent external noise from triggering the external interrupt when sensor is disconnected. Timer TIM7 is started to detect if sensor is connected by using a timeout approach.

3.External interrupt handler

Below is the subroutine for the external interrupt, i.e. the echo signal received from the sensor. As seen from the waveforms, the length of the echo signal determines the distance between the sensor and the obstacle. This interrupt is configured for both the rising and falling edge of the echo signal. We first acknowledge the interrupt and then we read the pin to ascertain whether it is a rising edge or a falling edge.

In the case where the interrupt is invoked for the rising edge, we reset the timer 7 count to 0 and reset the timeout flag. This is because we now want to start a new count on timer 7 to measure the pulse width and the timeout flag is reset as it is no longer the timeout condition (The timeout condition is when the echo is not received for more than 50ms).

For the falling edge, we update the pulseWidth global variable with the actual timer 7 count and we calculate the distance in mm based on the knowledge that 58 ticks (here 1 tick is 1us) corresponds to 1cm. The timer is disabled and the display_update variable is set to indicate that a new sensor value is available.

```
void EXTI15_10_IRQHandler() //External interrupt handler
{
     EXTI->PR|=EXTI_PR_PR10;  //Acknowledge interrupt
     int pinValue = 0;
     //Read the GPIO value to determine whether it is rising edge or falling edge
     pinValue = (GPIOA->IDR)&(1<<10);
     if(pinValue)
     {
         /*If it is rising edge, reset count to 0, this is to start counting to
         determine width of pulse*/
         TIM7->CNT = 0;
         /* Clear timeout in case it was set earlier due to sensor disconnection.
            This line ensures that the system resumes function when the sensor is
            reconnected*/
         timeout=0;
     }
     else
     {  /*Falling edge detected*/
        /*Get pulse width, this is the timer 7 count*/
         pulseWidth = TIM7->CNT;
        /*Calculate distance based on pulse width. Distance in mm*/
        distance = (10*pulseWidth)/58;
        /*Disable timer 7. If this is not done then it continues to count and hits the
          interrupt*/
        TIM7->CR1&=~TIM_CR1_CEN;
        display_update=1;  /*Set variable to 1 indicating that display should be
                             updated -new value*/
        TIM7->CNT=0;  /*Reset count to 0 to ensure that the next count starts from 0*/
     }
}
```

4.TIM7 overflow interrupt handler

```
void TIM7_IRQHandler() //Timer7 overflow interrupt
{
       TIM7->SR &= ~TIM_SR_UIF;  /*Acknowledge interrupt*/
       timeout=1;  /*Set timeout indicating that the sensor did not respond with the
                     echo signal*/
       display_update=1; /*Set variable to 1 indicating that display should be
                           updated-sensor was disconnected*/
}
```

This interrupt routine handles the scenario wherein the sensor fails to respond back to the trigger signal. After sending the trigger pulse for the sensor, we start the timer 7 and wait for up to 50ms, the time within which we expect to have the response. The interrupt is acknowledged and the timeout flag is set to 1 indicating that the sensor did not respond back. This flag is later used in the main function when writing to the display. The display update flag is also set to 1 as we now need to update the display with the message “na”.

5.Main function

```
int main()
{
    setup();
       while(1)
       {
        /*Update the display only if a it should be updated*/
        if(display_update)
        {
        /*Display update available*/
        tft.fillRect(tft.width()/2-2*FONT_WIDTH, 4*FONT_HEIGHT+1, 4*FONT_WIDTH,
        FONT_HEIGHT,0);
        tft.setCursor(tft.width()/2-2*FONT_WIDTH,5*FONT_HEIGHT);
        tft.setTextColor(ST7735_GREEN);

            if(timeout) /*Check if sensor response timed out i.e. no response is
                         received*/
            {
                tft.print("na");   /*Display "Not available(na)"*/
            }
            else
            {
                tft.print(distance);  /*Else display the calculated distance*/
            }
              display_update=0;  /*Set flag to false as the latest value has now been
                                   updated to the display*/
        }
       }
}
```

The main function invokes the setup once and then enters an infinite while loop which takes care of refreshing the display. To prevent the display from being refreshed unnecessarily, the display update flag is used which ensures that the display is refreshed only when there is a change in output. If a timeout has occurred, the text “na” is displayed. This is done by checking the timeout flag. Otherwise, the actual sensor value (in mm) is displayed.

Testing and Result:

The ultrasonic sensor is tested by placing obstacles in front of it and observing the readings. The reading increased when the object was moved further away from the sensor, hence confirming its working. Disconnecting the sensor from the main board causes the program to rightly display “na”. GPIOB, pin 0 was also used for initial debugging, to check if the code flow entered the ISRs.

Appendix. Full code

```
#include "stm32f3xx.h"
#include "Adafruit_GFX.h"
#include "Adafruit_ST7735.h"
#include "pinAccess.h"

//font definition.
#define PROGMEM
#define FONT_WIDTH 11
#define FONT_HEIGHT 21
#include "tft/Adafruit-GFX-Library/fontconvert/veraMono9.h"
#define FONT_NAME VeraMono9pt7b

#ifdef __cplusplus
 extern "C" {
#endif /* __cplusplus */

Adafruit_ST7735 tft = Adafruit_ST7735();
volatile uint32_t pulseWidth = 0;   //Variable to measure the width of the echo signal
volatile uint32_t distance = 0;   // Variable which holds the final distance
int timeout=0;   //Variable to check if there is a timeout of 50ms after initiating
sensor data acquisition
int display_update=0; /*This flag indicates that display should be updated when it is
true */

void setup()
{
       tft.initR(INITR_BLACKTAB);   // initialize a ST7735S chip, black tab
       tft.setFont(&FONT_NAME);
       tft.fillScreen(ST7735_BLACK);
       tft.setTextColor(ST7735_RED);
       tft.setCursor(10, FONT_HEIGHT);
       tft.print("Hello World! ");
    ADCInit();

    RCC->AHBENR |= RCC_AHBENR_GPIOBEN_Msk; //clock for GPIOB
    asm("nop");                                    //wait until GPIOB clock is Ok.
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN_Msk; //clock for GPIOA
    asm("nop");                                    //wait until GPIOA clock is Ok
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;//clock for the SYSCFG peripheral- external
interrupts
    asm("nop");

    /*Initialization of GPIOA and B (Pins 10 and 0 respectively)*/
    GPIOA->MODER&= ~(3<<(10*2));
    GPIOB->MODER&= ~(3<<0*2);
    GPIOB->MODER|=(1<<0*2);
    GPIOA->MODER|= 1<<(10*2);

    //timer setup
    //input clock = 64MHz.
       /*Timer 6 initialization*/
    RCC->APB1ENR|= RCC_APB1ENR_TIM6EN;
    //reset peripheral (mandatory!)
    RCC->APB1RSTR |= RCC_APB1RSTR_TIM6RST;
    RCC->APB1RSTR &=~RCC_APB1RSTR_TIM6RST;
    asm("nop");
```

```
/*Timer 7 initialization*/
    RCC->APB1ENR|= RCC_APB1ENR_TIM7EN;
    RCC->APB1RSTR |= RCC_APB1RSTR_TIM7RST;
    RCC->APB1RSTR &=~RCC_APB1RSTR_TIM7RST;
    asm("nop");

    /*Timer 6 register initialization*/
    TIM6->PSC=6400-1;  /*PSC to count in steps of 100us*/
    TIM6->ARR=1000-1;  /*Count up to 1000 to get a delay of 100ms, which is 10Hz*/
    TIM6->CR1|=TIM_CR1_CEN;  /*Start timer 6*/

    /*Interrupt enable*/
    TIM6->DIER |= TIM_DIER_UIE;  /*Enable interrupt for timer 6*/
    NVIC_EnableIRQ(TIM6_DAC1_IRQn);

    /*Timer 7 register initialization*/
    TIM7->PSC=64-1;    /*PSC to count in steps of 1us*/
    TIM7->ARR=50000-1;  /*Count up to 50ms (50,000 us) for time out condition*/

    TIM7->DIER |= TIM_DIER_UIE;  /*Enable interrupt for timer 7*/
    NVIC_EnableIRQ(TIM7_IRQn);

    SYSCFG->EXTICR[2] |= SYSCFG_EXTICR3_EXTI10_PA;  /*Enable external interrupt for port
A/
    EXTI->FTSR|=EXTI_FTSR_TR10;  /*Enable interrupt on falling edge*/
    EXTI->RTSR|=EXTI_RTSR_TR10;  /*Enable interrupt on rising edge*/
    NVIC_EnableIRQ(EXTI15_10_IRQn);
}

void TIM6_DAC1_IRQHandler()
{
    //Acknowledge interrupt
    TIM6->SR &= ~TIM_SR_UIF;
    EXTI->IMR &= ~(1<<10);  /*Temporarily disable external interrupt EXTI10
                            This is done to prevent unwanted trigger of interrupt when we
write to the same pin*/
    GPIOA->MODER|= (1<<(10*2));  //Set pin A10 to output to transmit pulse to the sensor
    asm("nop");
       /*Generate pulse to start data acquisition*/
    GPIOA->ODR|= 1<<10;   /*Pin A10 set to high*/
    for(volatile int i=0;i<20;i++);  /*Small delay to ensure that pulse width is
sufficient*/
    GPIOA->ODR&= ~(1<<10);  /*Set pin A10 back to low*/
    pinMode(GPIOA,10,INPUT_PULLUP);  //Set pin A10 to output as it needs to read the echo
signal

    EXTI->IMR |= (1<<10);   /*Enable external interrupt  EXTI10 again*/
    TIM7->CR1|=TIM_CR1_CEN;   /*Start TIM 7 counting*/
}
```

```
void EXTI15_10_IRQHandler() //External interrupt handler
{
     EXTI->PR|=EXTI_PR_PR10;  //Acknowledge interrupt
     int pinValue = 0;  //Read the GPIO value to determine whether it is rising edge or
falling edge*/
     pinValue = (GPIOA->IDR)&(1<<10);
     if(pinValue)
     {
         /*If it is rising edge, reset count to 0, this is to start counting to determine
width of pulse*/
         TIM7->CNT = 0;
            /* Clear timeout in case it was set earlier due to sensor disconnection.
            This line ensures that the system resumes function when the sensor is
reconnected*/
         timeout=0;
     }
     else
     {  /*Falling edge detected*/
          /*Get pulse width, this is the timer 7 count*/
         pulseWidth = TIM7->CNT;
        /*Calculate distance based on pulse width. Distance in mm*/
        distance = (10*pulseWidth)/58;
        /*Disable timer 7. If this is not done then it continues to count and hits the
interrupt*/
        TIM7->CR1&=~TIM_CR1_CEN;
        display_update=1;  /*Set variable to 1 indicating that display should be updated
-new value*/
        TIM7->CNT=0;  /*Reset count to 0 to ensure that the next count starts from 0*/
     }
}

void TIM7_IRQHandler() //Timer7 overflow interrupt
{
    TIM7->SR &= ~TIM_SR_UIF;  /*Acknowledge interrupt*/
    timeout=1;  /*Set timeout indicating that the sensor did not respond with the echo
                  signal*/
    display_update=1; /*Set variable to 1 indicating that display should be updated-
                        sensor was disconnected*/
}
```

```
int main()
{
    setup();
       while(1)
       {

              /*This section was just used to debug, sets the debug LED B0 to high if
distance is greater than 10cm
        if(distance>100)
        {
            GPIOB->ODR= 1;
        }
        else
        {
            GPIOB->ODR= 0;
        }

        */
              /*Update the display only if a it should be updated*/
        if(display_update)
        {
        /*Display update available*/
        tft.fillRect(tft.width()/2-
2*FONT_WIDTH,4*FONT_HEIGHT+1,4*FONT_WIDTH,FONT_HEIGHT,0);
        tft.setCursor(tft.width()/2-2*FONT_WIDTH,5*FONT_HEIGHT);
        tft.setTextColor(ST7735_GREEN);

            if(timeout) /*Check if sensor response timed out i.e. no response is
received*/
            {
                tft.print("na");   /*Display "Not available(na)"*/
            }
            else
            {
                tft.print(distance);  /*Else display the calculated distance*/
            }
              display_update=0;  /*Set flag to false as the latest value has now been
updated to the display*/
        }

       }
}

#ifdef __cplusplus
 }
#endif /* __cplusplus */
```