Project Report

Reverse Parking Sensor using TIVA TM4C123GH6PM

Embedded Systems and Designs

Name: Kartik Khandelwal and Viren Sharma

Roll Number: 21UEC071 and 21UEC143

Instructor: Dr. Deepak Nair

Department of Electronics and Communication Engineering
The LNMIIT Jaipur, Rajasthan

May 02, 2024

Declaration

This report has been prepared based on my work. Where other published and unpublished source materials have been used, these have been acknowledged.

Student Name: **Kartik Khandelwal and Viren Sharma**
Roll Number: **21UEC071 and 21UEC143**
Date of Submission: 02/05/2024

Table of Contents

Abstract

This project aims to develop a reverse parking monitor using Tiva C TM4C123GH6PM board and an ultrasonic sensor (HC-SR04) using Embedded C. The system uses ultrasonic sensor to detect the distance between the vehicle and an obstacle, and based on the measured distance, it activates a visual and audio alert to warn the driver. The C code presented in this project initializes the required GPIO pins and includes functions for measuring distance, UART communication and setting up PWM for buzzer. The system can detect obstacles within a range of 3cm to 300cm and provides feedback through status LEDs and buzzer. Additionally, a UART module enables data transmission to a computer for monitoring and analysis. The reverse parking assistance system aims to enhance driving safety and convenience by providing visual and audible cues to the driver during reverse parking.

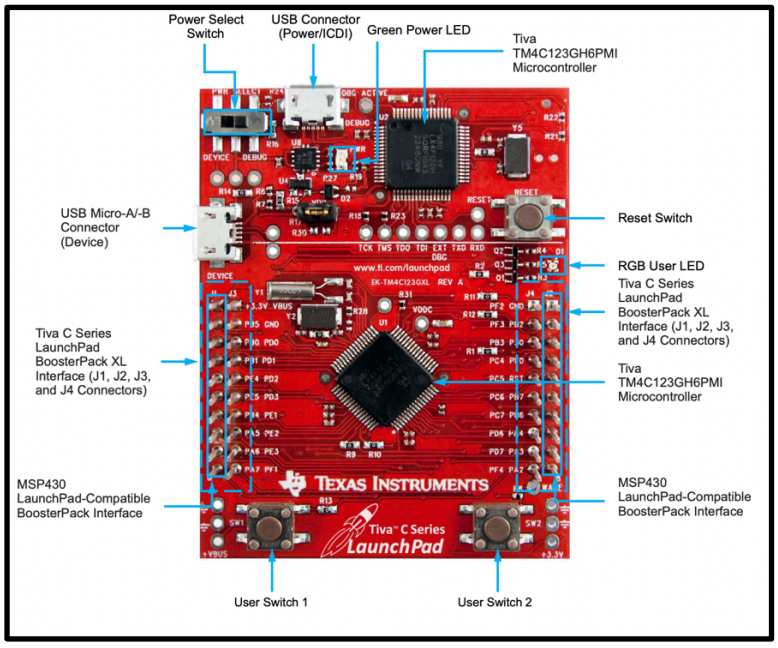
Component Name

1. **Tiva C (TM4C123GH6PM) Microcontroller board**:

The TM4C123GH6PM is a microcontroller from Texas Instruments, often referred to as the Tiva C Series TM4C123G LaunchPad Evaluation Kit. It's part of TI's ARM Cortex-M4F-based microcontrollers.

The TM4C123GH6PM board features:

- **ARM Cortex-M4F Processor**: The core of the microcontroller, capable of running at up to 80 MHz.
- **On-Chip Memory**: Including Flash memory for program storage and SRAM for data storage.
- **Peripherals**: Such as GPIO (General Purpose Input/Output) pins, UART (Universal Asynchronous Receiver/Transmitter), SPI (Serial Peripheral Interface), I2C (Inter-Integrated Circuit), PWM (Pulse Width Modulation), ADC (Analog-to-Digital Converter), and more.
- **USB Connectivity**: USB micro-A and micro-B connector for USB device, host, and on-the-go (OTG) connectivity
- **Supported by TivaWare**: for C Series software including the USB library and the peripheral driver library.



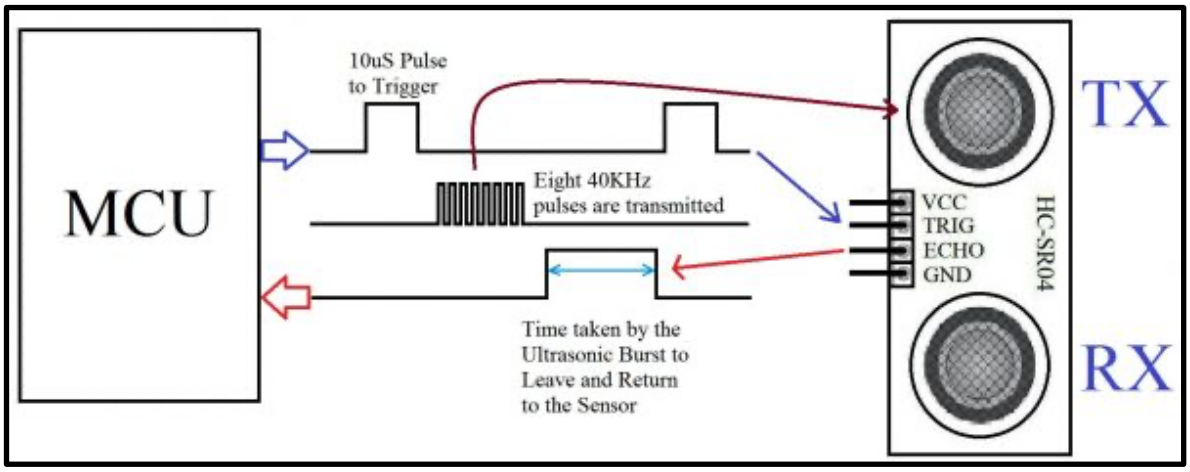
Tiva C Series TM4C123G LaunchPad Evaluation Board

2. Ultrasonic Sensor HC-SR04:

The HC-SR04 ultrasonic distance sensor comprises two ultrasonic transducers: one acts as a transmitter, converting electrical signals into 40 kHz ultrasonic pulses, while the other acts as a receiver, detecting these pulses. When the receiver detects the transmitted pulses, it generates an output pulse whose duration correlates with the distance to an object.

The process begins by setting the trigger pin high for 10 microseconds. In response, the sensor emits an ultrasonic burst of eight pulses at 40 kHz. This unique 8-pulse pattern aids the receiver in distinguishing the transmitted pulses from ambient ultrasonic noise. These pulses propagate through the air away from the transmitter, while the echo pin is set high to initiate the echo-back signal. If the transmitted pulses are not reflected back, the echo signal times out and returns low after 38 milliseconds. Therefore, a pulse duration of 38 milliseconds indicates no obstruction within the sensor's range.

The width of the received pulse is used to calculate the distance from the reflected object.
Distance = Speed X Time
(Speed – Speed of sound(340m/s))
(Time-Width of Pulse received on Echo Pin)



HC-SR04 Ultrasonic Sensor

3. LEDs:

3 LEDs of Red, Green and Yellow colour have been used.

4. Buzzer(MH-FMD):

Active Piezo-Electric Buzzer module produces a single-tone sound when trigger. The Active Buzzer module consists of a piezoelectric buzzer with a built-in oscillator. It generates a sound of approximately 2.5 kHz when activated.

- Resonance Frequency: 2500Hz±300Hz
- Sound Output Level: 85dB @10cm

Chapter 1: Proposed Solution and Working

The provided code implements a reverse parking sensor system using an ultrasonic sensor **(HC-SR04)** connected to a microcontroller **(TM4C123GH6PM)**. The system measures the distance between the sensor and an object and provides feedback to the driver using LEDs and a buzzer. The system clock of microcontroller is set at **16MHz**.

1.1 Ultrasonic Sensor Integration

Timer0A used to measure distance by measuring pulse duration of Echo Output signal. Trigger Pin is given approx 10us signal(9.932us) using **SysCtlDelay((SysCtlClockGet()/(3*173500)));.**

Timer0ACapture_init initializes Timer0A to capture consecutive rising and falling edges of echo signal. It is used to configure PA4 as digital output pin for trigger, PB6 as Input pin and use its alternate function of Timer 0 -T0CCP0. **GPIOB->AFSEL |= 0x40;**

Timer0 is then setup as follows:

TIMER0->CTL &= ~1;- Disables Timer0A during setup.

TIMER0->CFG = 4;- Configures Timer0A in 16-bit timer mode.

TIMER0->TAMR = 0x17; - Sets Timer0A to up-count, edge-time, capture mode.

TIMER0->CTL |=0x0C;- Configures Timer0A to capture the rising edge.

TIMER0->CTL |= (1<<0); - Enables Timer0A.

Measure_distance function is used to get the positive pulse width at Echo Pin by returning value **(thisEdge - lastEdge)** which is the Timer A counter value.

Clear Timer0A capture flag. **TIMER0->ICR = 4;**

Wait until capture flag is set. **while((TIMER0->RIS & 4) == 0);**

Check for a rising edge on pin PB6. **if(GPIOB->DATA & (1<<6))**

If rising edge detected, save timestamp as **lastEdge = TIMER0->TAR;** Clear capture flag. **TIMER0->ICR = 4;**

Wait until capture flag is set again. **while((TIMER0->RIS & 4) == 0);**

Save timestamp of falling edge as **thisEdge = TIMER0->TAR;**

Return time difference between thisEdge and lastEdge.- **return (thisEdge - lastEdge)**

This difference is then stored in time in main function where it is converted into distance by the formula distance = ((int64_t)(time) * 10625)/10000000; /* convert pulse duration into distance */.

This factor is obtained by dividing speed of sound by 16MHz system clock and by 2.

1.2 Buzzer Integration using PWM

SysCtlPWMClockSet(SYSCTL_PWMDIV_64);- This sets the PWM clock divisor to divide the system clock by 64. This determines the frequency of the PWM signal.

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);-- This enables the PWM peripheral and GPIO port A peripheral where the PWM pin is located.

PWMGenConfigure(PWM1_BASE,PWM_GEN_1,PWM_GEN_MODE_DOWN|PWM_GEN_MODE_DB_NO_SYNC);-This configures PWM generator 1 to operate in down-count mode with dead-band synchronization disabled.

ui32Load = (ui32PWMClock / PWM_FREQUENCY) - 1;: This calculates the PWM load value to set the desired PWM frequency. PWM_FREQUENCY is a constant representing the desired PWM frequency. Here the value of load register is **249**;

16,000,000(ClkFreq) / 64(Division_Factor) / 1000(Required_Freq) – 1 = 249

PWMGenPeriodSet(PWM1_BASE, PWM_GEN_1, ui32Load);- This sets the PWM period (load value) for PWM generator 1.

PWMOutputState(PWM1_BASE, PWM_OUT_2_BIT, true);- This enables PWM output 2 of PWM module 1. The PWM signal will now be generated on pin GPIO_PA6.

1.3 Status LEDs using GPIO:

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE); - This line enables the system clock on the GPIO E peripheral, allowing the microcontroller to use it.

GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3); - This line configures pins PE1, PE2, and PE3 as output pins, used for driving status LEDs

1.4 Distance Readings using UART:

SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0); - This line enables the clock for the UART (Universal Asynchronous Receiver/Transmitter) 0 peripheral, allowing the microcontroller to use it for serial communication.

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA); - This line enables clock for the GPIO A peripheral, allowing the microcontroller to configure GPIO pins for UART communication.

GPIOPinConfigure(GPIO_PA0_U0RX); - This line configures pin PA0 to work as the UART0 receive pin (RX).

GPIOPinConfigure(GPIO_PA1_U0TX); - This line configures pin PA1 to work as the UART0 transmit pin (TX).

GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);- This line configures the GPIO pins specified (PA0 and PA1) to be used for UART functionality.

UARTConfigSetExpClk(UART_MODULE,SysCtlClockGet(),UART_BAUDRATE, (UART_CONFIG_WLEN_8|UART_CONFIG_STOP_ONE|UART_CONFIG_PAR_NONE)) ; - This line configures the UART0 module with the specified parameters: the UART module used, the system clock frequency, baud rate, data length (8 bits), stop bits (1 bit), and parity (none).

9600 = (16MHz / 16 x baud divisor)

Baud divisor = 1000000/9600 = 104.1667

UART0->IBRD = 104;/* 16MHz/16=1MHz, 1MHz/104=9600 baud rate */

1.5 Main Function:

The **main** function begins by setting up the system clock using **SysCtlClockSet** to configure the clock frequency at **16MHz**. Following this, initialize **Timer0A** to capture edge-to-edge time intervals at Echo pin through **Timer0ACapture_init**. GPIO pins for controlling LEDs are initialized next via **GPIO_Init**, followed by PWM initialization for the buzzer through **PWM_Init**. The UART0 module is then initialized using **UART_Init** to enable data transmission to a computer. Inside the main loop, the function measures the pulse duration using **Measure_distance**, which is then used to calculate the distance.

Depending on the calculated distance, LEDs are controlled accordingly, with specific LED configurations for different distance ranges. Additionally, the PWM signal for the buzzer is adjusted based on the distance.

GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 ,GPIO_PIN_1);

PWMPulseWidthSet(PWM1_BASE, PWM_OUT_2, ui8Adjust * ui32Load / 1000);

PWMGenEnable(PWM1_BASE, PWM_GEN_1);

If the distance exceeds a threshold of 250cm, all LEDs are turned off, and a "null" message is transmitted to the computer. Finally, the measured distance data is transmitted via UART, and a 1-second delay is introduced before the loop iterates again.

SysCtlDelay((SysCtlClockGet()*1/(3)));/*1s delay */ as SysCtlDelay takes about 3 Clock cycles.

Chapter 2: Source Code

Header File, Pin Declarations and Function Prototyping:

```
#include "TM4C123GH6PM.h"
#include <stdio.h>
#include <intrinsics.h>
#include <stdint.h>
#include <stdbool.h>
#include "driverlib/pwm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/gpio.h"
```

```
#define UART_MODULE UART0_BASE
#define UART_BAUDRATE 9600
#define PWM_FREQUENCY 1000  // Define PWM frequency
//trigger pa4
//echo pb6
//uart pa0 pa1
//pwm pa6
//led pe1 pe2 pe3
uint32_t sysclock1;
/*Function prototype for Timer0A, GPIO ,UART module initialization */
void UART_Init(void);
void GPIO_Init(void);
void PWM_Init(void);
uint32_t Measure_distance(void);
void Timer0ACapture_init(void);
void printstring(char *str);
```

Global Variables:

```
/* global variables to store and display distance in cm */

uint32_t time; /*stores pulse on time */

uint64_t distance; /* stores measured distance value */

char mesg[200];  /* string format of distance value */

uint32_t ui32Load; // PWM load value

uint32_t ui32PWMClock; // PWM clock frequency

uint16_t ui8Adjust; // PWM duty cycle adjustment

uint32_t sysclock;

uint32_t sysclock1;

uint32_t sysclock2;
```

UART_Init Function:

```
void UART_Init(void)

{

 /* UART Configuration and GPIO Pins configuration to work as UART */

 // Enable the UART0 peripheral

 SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

 // Enable the GPIO A Peripheral

 SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

 // Configure GPIO pins for UART (Refer Table 14-1.) PA0-RX, PA1-TX

 GPIOPinConfigure(GPIO_PA0_U0RX);

 GPIOPinConfigure(GPIO_PA1_U0TX);

 GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);



 // Configure UART

 UARTConfigSetExpClk(UART_MODULE,  SysCtlClockGet(),UART_BAUDRATE,  (UART_CONFIG_WLEN_8
|UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));

//set baud rate, clock, 8 bits from a byte selected, 1 stop bit , no parity check

}
```

Timer0A Capture Function:

```
/* Timer0A initialization function */
/* Initialize Timer0A in input-edge time mode with up-count mode */
void Timer0ACapture_init(void)
{  SYSCTL->RCGCTIMER |= 1;     /* enable clock to Timer Block 0 */
   SYSCTL->RCGCGPIO |= 2;      /* enable clock to PORTB */
   GPIOB->DIR &= ~0x40;        /* make PB6 an input pin */
   GPIOB->DEN |= 0x40;         /* make PB6 as digital pin */
   GPIOB->AFSEL |= 0x40;       /* use PB6 alternate function */
   GPIOB->PCTL &= ~0x0F000000;  /* configure PB6 for T0CCP0 */(Page-650)
   GPIOB->PCTL |= 0x07000000;
 /* PA4 as a digital output signal to provide trigger signal */
    SYSCTL->RCGCGPIO |= 1;      /* enable clock to PORTA */
    GPIOA->DIR |=(1<<4);        /* set PA4 as a digital output pin */
    GPIOA->DEN |=(1<<4);        /* make PA4 as digital pin */

   TIMER0->CTL &= ~1;          /* disable timer0A during setup */
   TIMER0->CFG = 4;            /* 16-bit timer mode */
   TIMER0->TAMR = 0x17;        /* up-count, edge-time, capture mode */
   TIMER0->CTL |=0x0C;         /* capture the rising edge */
   TIMER0->CTL |= (1<<0);          /* enable timer0A */
}
```

PrintString Function:

```
void printstring(char *str)
{ while(*str)
    {
        UARTCharPut(UART_MODULE, *str++);
    }
}
```

Measure_distance Function:

```
        /* This function captures consecutive rising and falling edges of a periodic signal */

        /* from Timer Block 0 Timer A and returns the time difference in form of counter value(the period of the signal).
*/      uint32_t Measure_distance(void)

        {

          int lastEdge, thisEdge;

          /* Given 10us trigger pulse */

          GPIOA->DATA &= ~(1<<4); /* make trigger  pin low in case high */

          SysCtlDelay((SysCtlClockGet()/(3*173500)));/*approx. 10 us delay --- 100000 gives 14.3 us delay*/

          GPIOA->DATA |= (1<<4); /* make trigger  pin high */

          SysCtlDelay(SysCtlClockGet()/(3*173500)); /*approx 10 us delay */

          GPIOA->DATA &= ~(1<<4); /* make trigger  pin low */

          while(1)

          {

           TIMER0->ICR = 4;           /* clear timer0A capture flag */

           while((TIMER0->RIS & 4) == 0);    /* wait till captured */

           if(GPIOB->DATA & (1<<6)){ /*check if rising edge occurs */

             lastEdge = TIMER0->TAR;     /* save the timestamp --Timer A value*/

             TIMER0->ICR = 4;           /* clear timer0A capture flag */

             while((TIMER0->RIS & 4) == 0);    /* wait till captured */ /* detect falling edge */

             thisEdge = TIMER0->TAR;     /* save the timestamp */

             return (thisEdge - lastEdge); /* return the time difference */

           }

          }

}
```

GPIO_Init Function:

```
void GPIO_Init(void)

{

 // Enable GPIOE peripheral for switch reading

 SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);

 GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);

//PE1, PE2,PE3 for LEDs

}
```

PWM_Init Function for Buzzer:

```
void PWM_Init(void){

 SysCtlPWMClockSet(SYSCTL_PWMDIV_64);// Set PWM clock divisor to divide system clock by 64

 sysclock2 = SysCtlPWMClockGet();// Retrieve the current PWM clock frequency for reference

 SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);// Enable PWM peripheral and GPIO port for PWM pin

 SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

 GPIOPinTypePWM(GPIO_PORTA_BASE, GPIO_PIN_6);// Configure the GPIO pin as a PWM output(PA6)

 GPIOPinConfigure(GPIO_PA6_M1PWM2);// Configure the GPIO pin to use PWM module 1, PWM generator 2

 ui32PWMClock = SysCtlClockGet() / 64;// Calculate the PWM clock frequency by dividing system clock frequency by
64

 ui32Load = (ui32PWMClock / PWM_FREQUENCY) - 1;// Calculate the PWM load value to set the desired PWM
frequency

 PWMGenConfigure(PWM1_BASE, PWM_GEN_1, PWM_GEN_MODE_DOWN |
PWM_GEN_MODE_DB_NO_SYNC);// Configure PWM generator 1, set it to operate in down-count mode with dead-
band synchronization disabled

 PWMGenPeriodSet(PWM1_BASE, PWM_GEN_1, ui32Load);// Set the PWM period (load value) for PWM generator 1

 PWMOutputState(PWM1_BASE, PWM_OUT_2_BIT, true);// Enable the PWM output for PWM generator 1, PWM
output 2

}
```

Main Function:

```
/* main code to take distance measurement and send data to UART terminal and turn on buzzer and leds*/
int main(void){
   SysCtlClockSet(SYSCTL_SYSDIV_12_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ | SYSCTL_OSC_MAIN);
 // 16 MHz CLK set
   Timer0ACapture_init();  /*initialize Timer0A in edge edge time */
   GPIO_Init();/*initialize GPIO for LED*/
   PWM_Init();/*initialize PWM for Buzzer*/
   UART_Init(); /* initialize UART0 module to transmit data to computer */
   sysclock1=SysCtlClockGet();
   while(1)
   {
      time = Measure_distance(); /* take pulse duration measurement */
      distance = ((int64_t)(time) * 10625)/10000000; /* convert pulse duration into distance */
      sprintf(mesg, "\r\nDistance = %lld cm", distance); /*convert float type distance data into string */
//       printf(" %d\n",distance);// printing on terminal I/O
      if(distance <=100){
        ui8Adjust = 200;
        GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 ,GPIO_PIN_1);
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_2, ui8Adjust * ui32Load / 1000);
        PWMGenEnable(PWM1_BASE, PWM_GEN_1);
//         SysCtlDelay((SysCtlClockGet()*1/(3*100)));/* 10 ms delay */
      }
      else if(distance > 100 && distance <=200){
        ui8Adjust = 500;
        GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 ,GPIO_PIN_2);
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_2, ui8Adjust * ui32Load / 1000);
        PWMGenEnable(PWM1_BASE, PWM_GEN_1);
//         SysCtlDelay((SysCtlClockGet()*1/(3*100)));/*10 ms delay */
      }
```

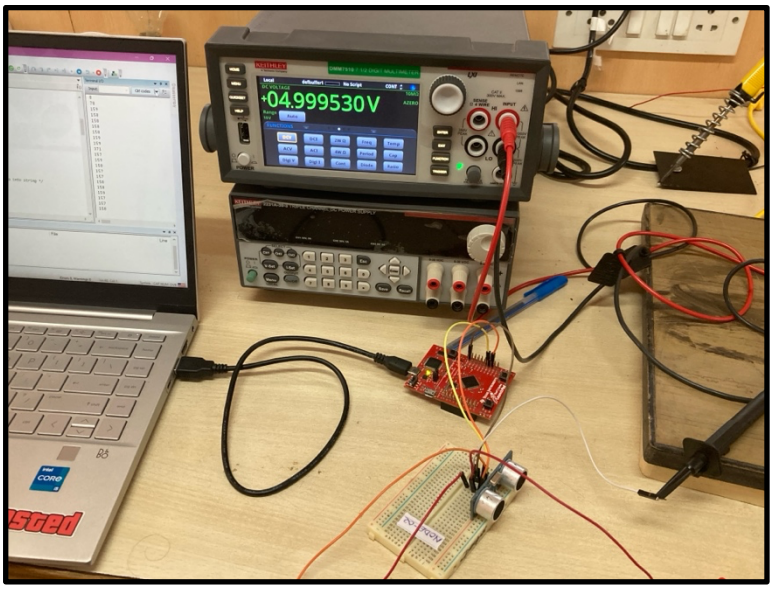
```
else if(distance > 200 && distance <=250){
        ui8Adjust = 900;
        GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 ,GPIO_PIN_3);
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_2, ui8Adjust * ui32Load / 1000);
        PWMGenEnable(PWM1_BASE, PWM_GEN_1);
//        SysCtlDelay((SysCtlClockGet()*1/(3*100)));/* 10 ms delay */
      }
      else{
        GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 ,0);
        printstring("\r\nnull");/*transmit data to computer */
        PWMGenDisable(PWM1_BASE, PWM_GEN_1);
        SysCtlDelay((SysCtlClockGet()*1/(3)));/* 1 s delay */
        continue;
      }
      printstring(mesg);/*transmit data to computer */
      SysCtlDelay((SysCtlClockGet()*1/(3)));/*1s delay */
```

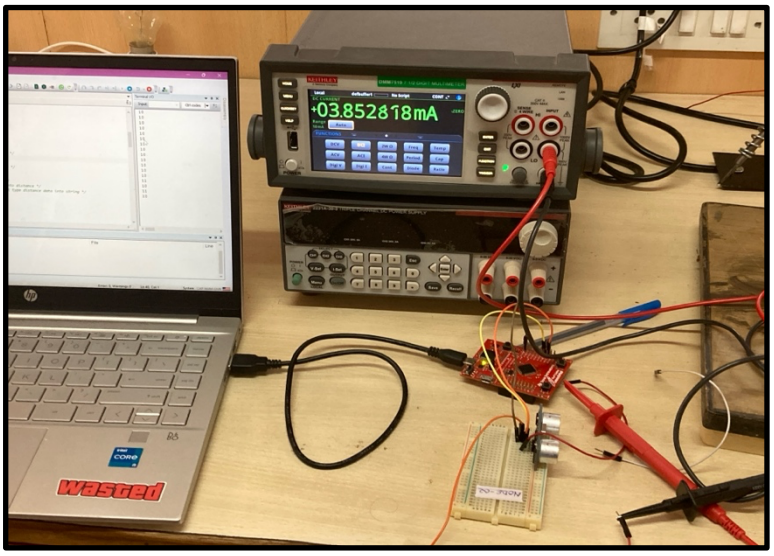
Chapter 3: Measurements:

Various measurements relating to power consumption by the ultrasonic sensor module, entire system under different conditions; measurements relating to Ultrasonic sensor (HC-SR04) for its calibration have been conducted.

Power Measurements:

1. **Ultrasonic Sensor Individual Power Consumption-**

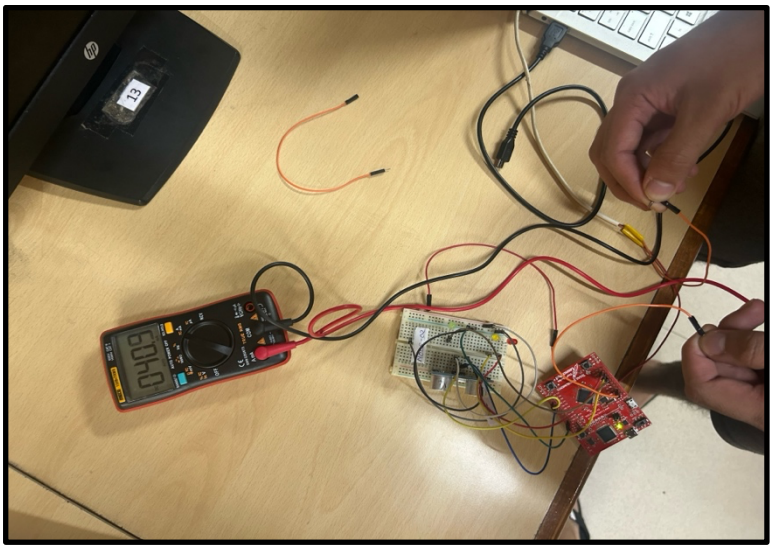




Power consumed=4.99V*3.85mA

Power consumed=0.019W

2. **Entire System Current Consumption in OFF state:**

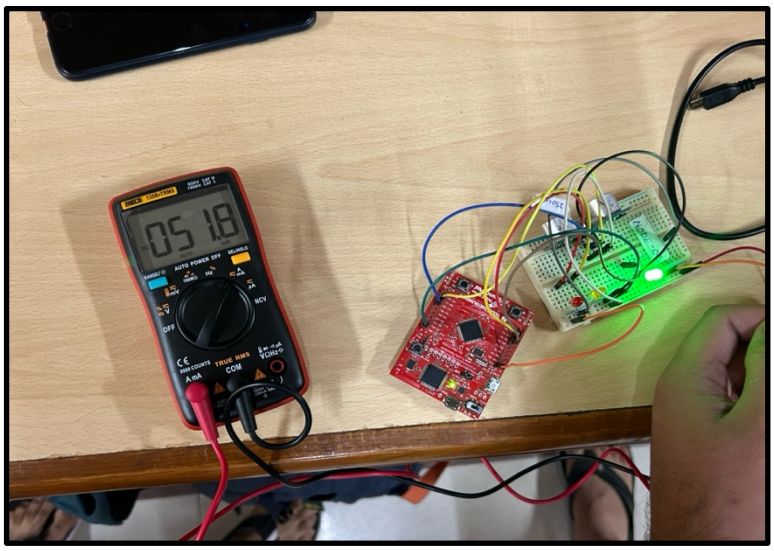


Current Consumption=40.9mA

Power consumed=5V*40.9mA

Power consumed=0.205W

3. **Entire System Current Consumption in ON state with Green LED -**

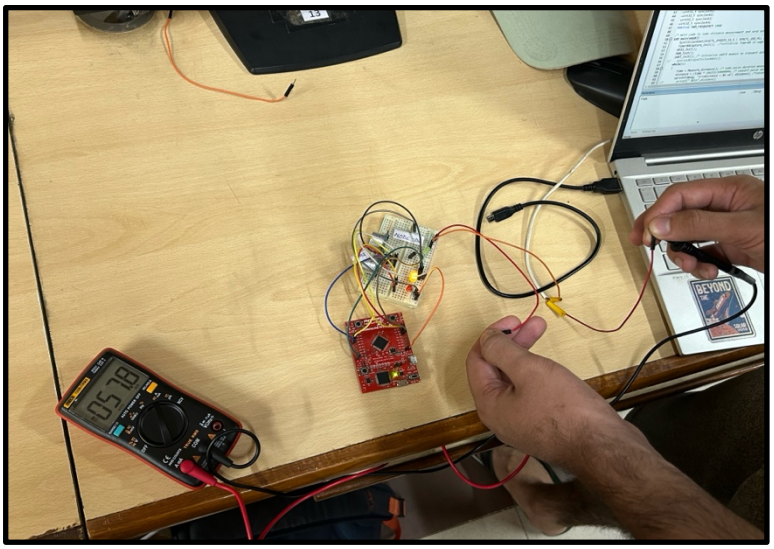


Current Consumption=51.8mA

Power consumed=5V*51.8mA

Power consumed=0.259W

4. **Entire System Current Consumption in ON state with Yellow LED –**



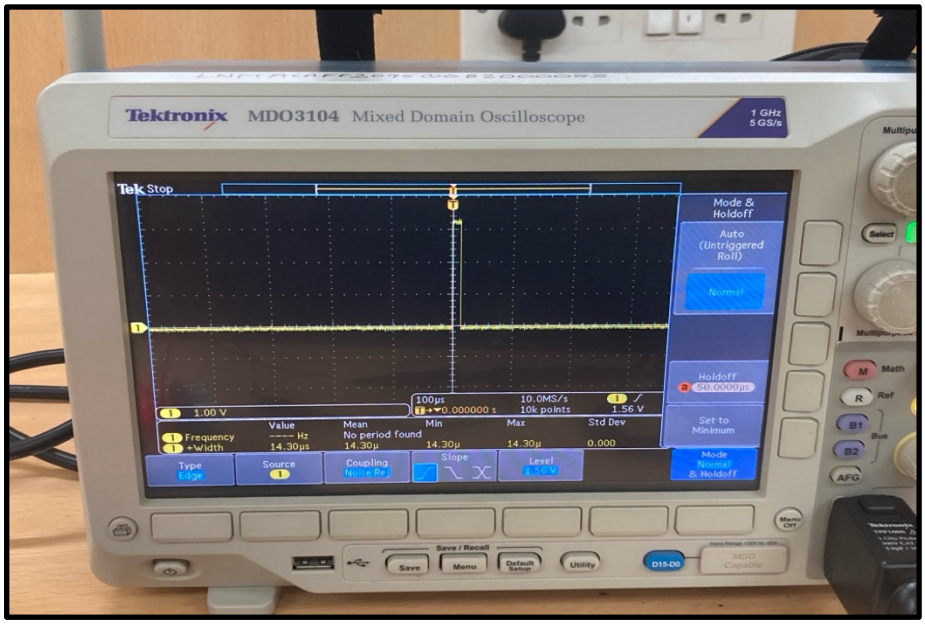
Current Consumption=57.8mA

Power consumed=5V*57.8mA

Power consumed=0.289W

Ultrasonic Sensor Measurements:

1. **Input Trigger Signal from Microcontroller-**

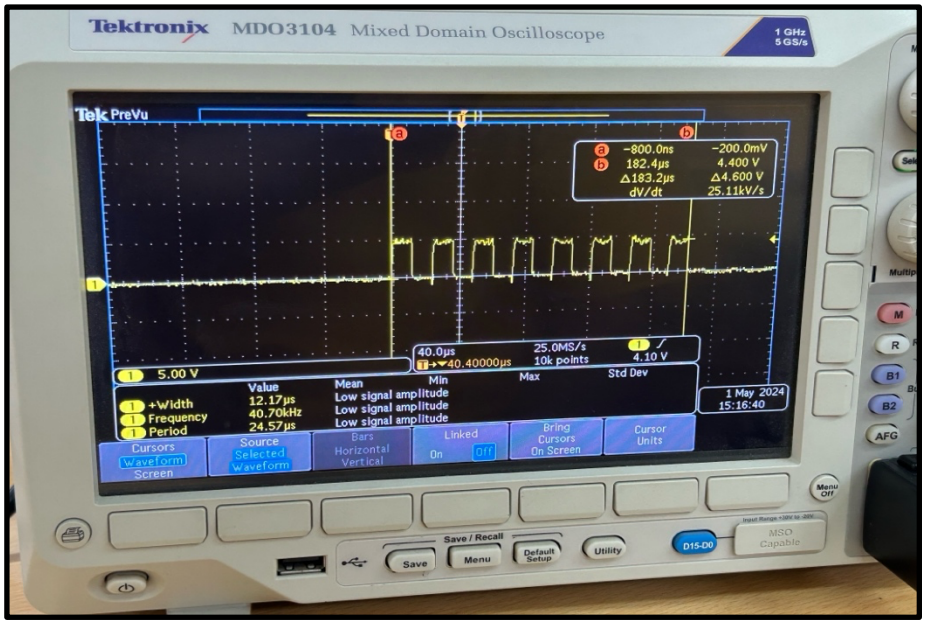


Trigger Signal by Microcontroller to Trigger Pin of HC-SR04

SysCtlDelay((SysCtlClockGet()/(3*173500)));/*approx. 10 us delay

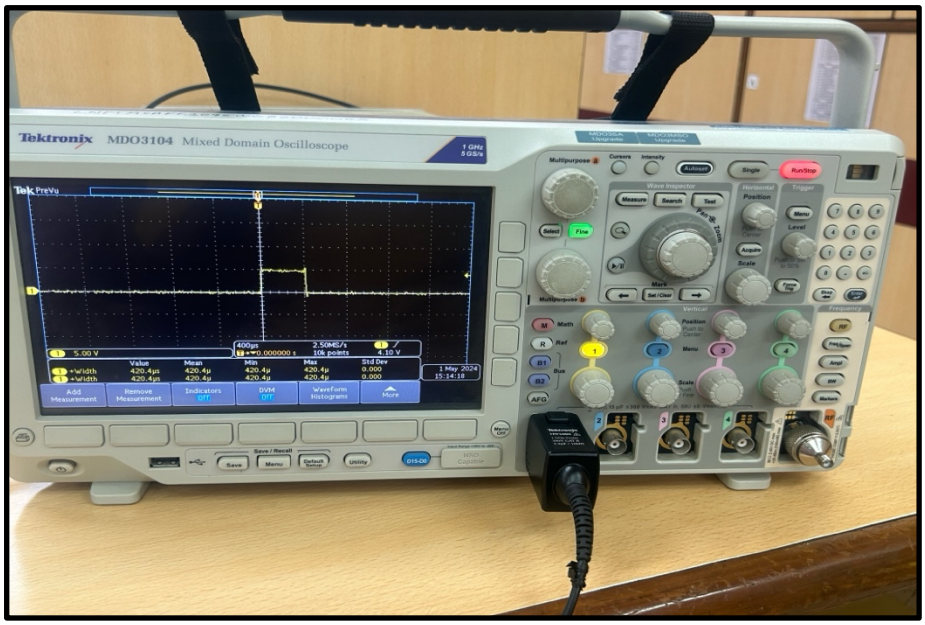
100000 produces 14.3 us delay; does not produce perfect 10us signal so changes made to make it 173500 instead of 100000.

2. **Tx generates 8 CLK pulses of 40Khz each-**



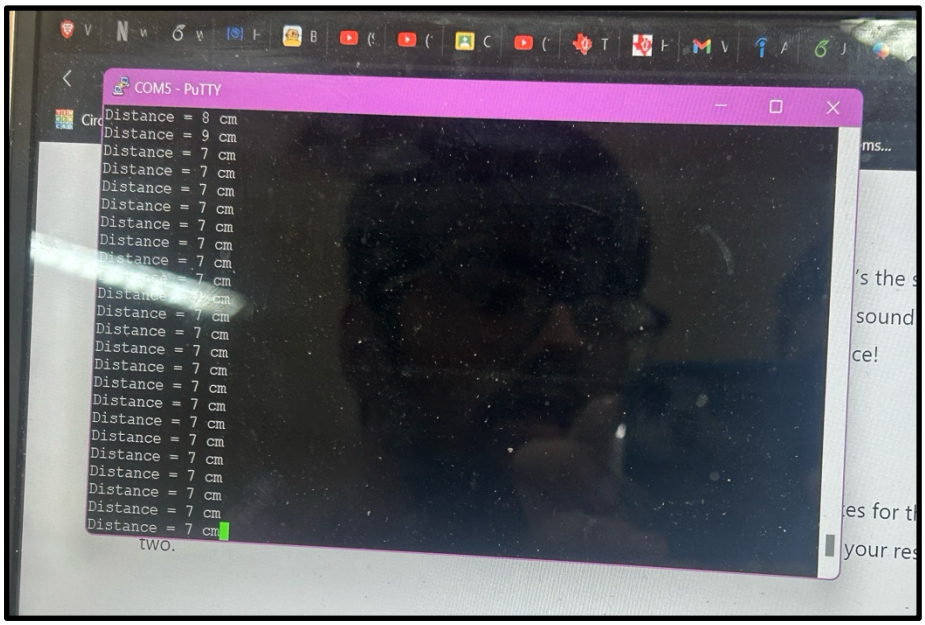
Each CLK pulse has a width of 12.17us and CLK frequency of 40.70KHz. These pulses are generated for a time interval of 183.2us.

3. **Echo Pin Pulse Width-**

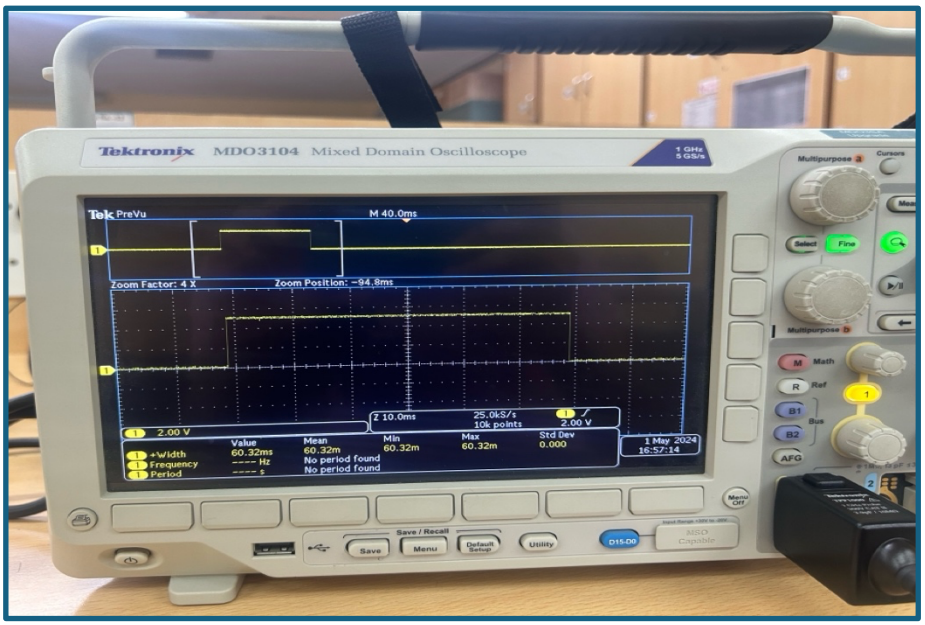


Pulse Width of 420.4us for an object kept 7cm far from the ultrasonic sensor.
Speed of light=0.034cm/us
Theoretical Distance = (0.034cm/us * 420.4us)/2
= 7.1468cm

Actual Distance = 7cm as observed on PuTTY Serial Monitor using UART communication



After 250cm the pulse width generated gives wrong results. Therefore, our sensor is set to detect ranges from 3cm to 250cm. It will give NULL readings at a distance greater than 250cm.



60.32ms pulse width is observed for an object at approximately 265cm which equates to a theoretical distance = (0.034cm/us*60.32*1000)/2 cm

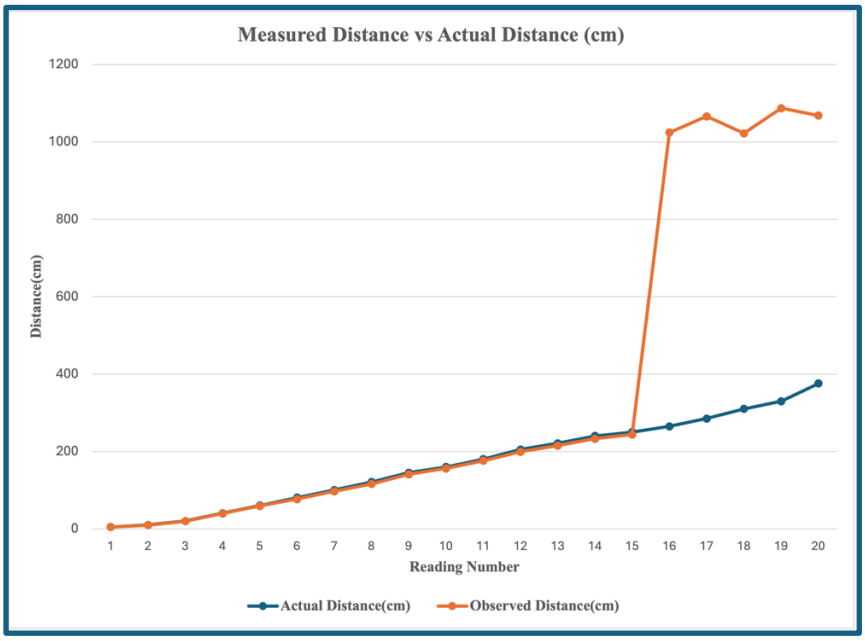
= 1020cm

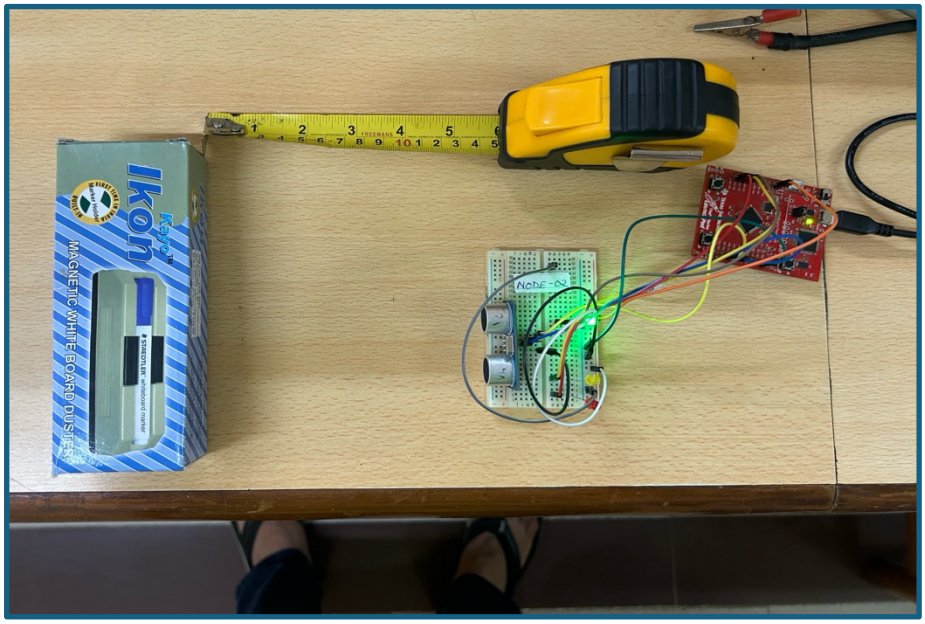
Thus, the system has been calibrated to take up readings only upto 250cm.

Chapter 4: Results

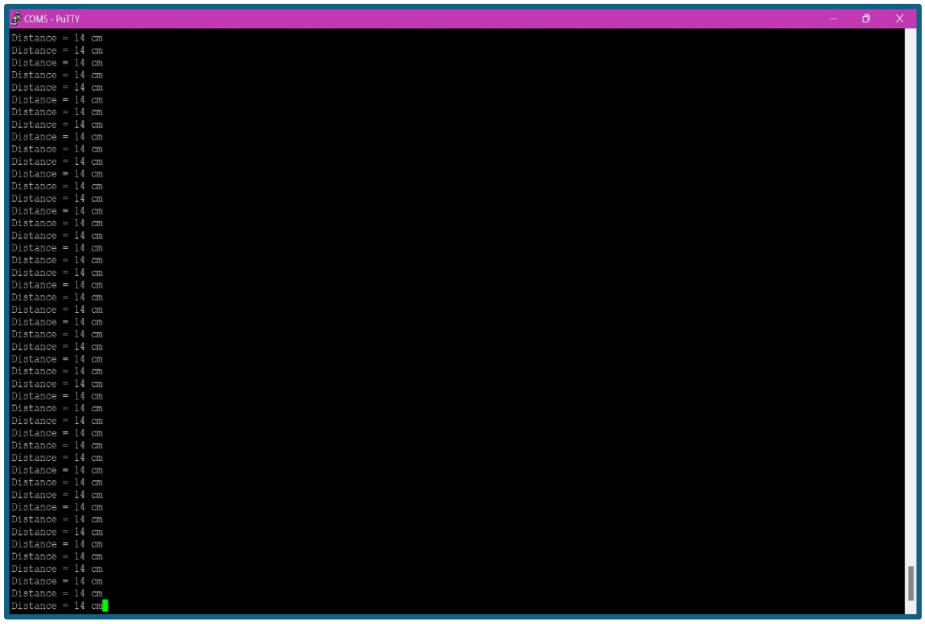
Comparison Table and Graph:

S.No.	Actual Distance(cm)	Observed Distance(cm)
1	5	5
2	10.1	10
3	20.5	20
4	40.5	40
5	60.5	59
6	80.5	77
7	100.5	97
8	121	116
9	145	141
10	160	156
11	180	176
12	205	199
13	221	215
14	240	233
15	250	244
16	265	1024
17	285	1066
18	310	1022
19	330	1087
20	376	1068





Object kept at 14cm from Ultrasonic Sensor



Distance Displayed on PuTTY monitor-14cm.

Link for the demo Video.

Bibliography

[1] HC-SR04 interfacing with TM4C123-Microcontrollerslab

[2] Tiva™ C Series TM4C123GH6PM Microcontroller Datasheet

[3] Texas Instruments ARM-based Microcontroller Forum

[4] Tiva C Series TM4C123x ROM User's Guide

[5] IAR Technical Notes