**Abstract**

The project is to develop a buggy to traverse in a maze while collecting distance and visual data. The buggy uses a microcontroller board to integrate an ultrasonic sensor for distance measurement, a camera for video recording, and motors for movement. The ultrasonic sensor is capable of measuring distance within a 5 cm to 200 cm range with ±4% error and transmitting data back to the computer console through USB connection. The video recording will be matched to distance data according to the timestamp and the LED flashing. Motor control is achieved through an H-bridge circuit, controlled by output pins from the microcontroller. For future improvements, we could integrate the Bluetooth module for wireless communication and redesign the chassis to accommodate additional motors.

**Goals**

The goal of this project is to create a buggy that can collect distance and visual information when traversing through a maze. The information will be later used for simulated neurons to form visual and spatial memory.

**Background**

Ultrasound is the sound with frequencies greater than 20 kHz. Such sound is beyond the audible range of healthy young adults.(1) Ultrasonic devices are used in many fields. One of the most important areas is to detect objects and measure distances. Ultrasonic sensing doesn’t require direct contact with the target, which is an advantage for medical, military, and general industries. (2) The common way of detection is based on the pulsed sound waves. The device would transmit signals consisting of short bursts of ultrasonic waves. After each burst, the receiver looks for an echo signal within a small window of time that corresponds to the previously transmitted signal. (3)

**Methods**

For the microcontroller board, we are using MSP432P401R. It has several pins that can be configured to different modes to communicate with external devices. To interact with the ultrasonic sensor and motor, we will be using simple I/O mode. In input mode, the pin can detect the rise and fall of the voltage connected to the pin. In output mode, the pins can be set to low(0V) or high (3.3V). The board has an embedded timer that operates based on the system clock signal, allowing for precise control over signal timing and accurate measurement of time intervals. Data from the board is transmitted via UART through micro-USB port to a connect computer, where results are displayed on the console.

For distance detection, we are using ultrasonic sensors RCWL-1601(5). It has the same parameter as HC-SR04(6) but with an additional 3.3V mode, so it can be connected directly to the MSP432 board. RCWL has four pins, Vcc, Trig, Echo, and GND. Vcc and GND are connected to the board for the voltage source and the ground. Trig is connected to an output pin, and Echo is connected to an input pin. The basic procedure of detecting objects and measuring distance is listed as below:

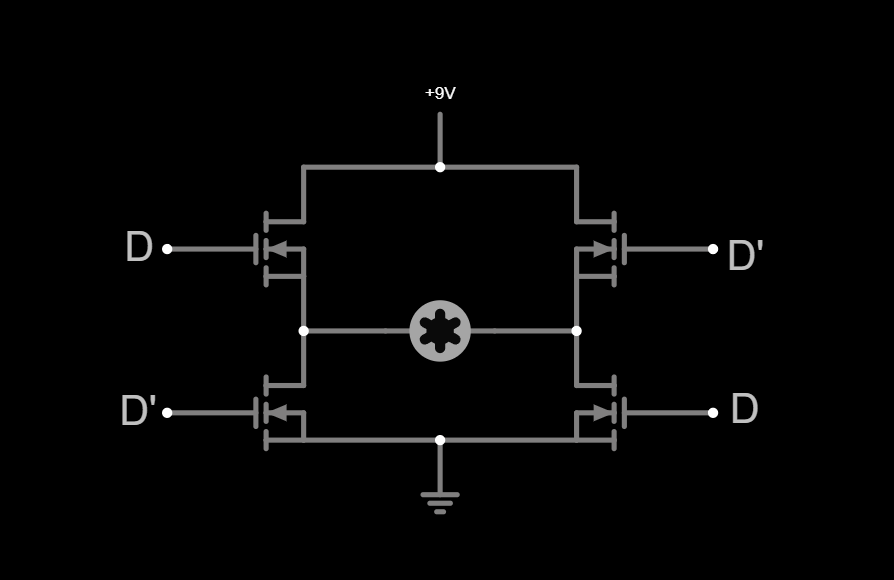
1. Set the output pin to Trig as high for at least 10us.
2. RCWL will send out an 8 cycle burst of ultrasound at 40 kHz and raise the Echo pin at high. It waits to detect any sound pulses coming back.
3. If it detects any sound pulses, it will set the Echo pin as low. So the time interval between the Echo pin rising to high and dropping to low is the echo time.

The distance D between the sensor and the detected object can be calculated by the equation below:

Where is the echo time, is the velocity of sound (340 m/s). The distance is divided by 2 because the sound pulse has to travel to and back from the object.

For the camera, we are using LF-C1T camera from littlelf(7). It can store the video to the cloud drive and the inserted SD card. To match the video frame with the distance data, the LED on the board will blink three times to mark the beginning of the measurement, and the timestamp is stored alongside with the distance data.

For motors of the buggy, we are using the DC motor F130SA-11200-38V. It can be powered by a 9V battery. A H-bridge composed by four MOSFET is used to control the motor. The circuit is shown in the diagram below:

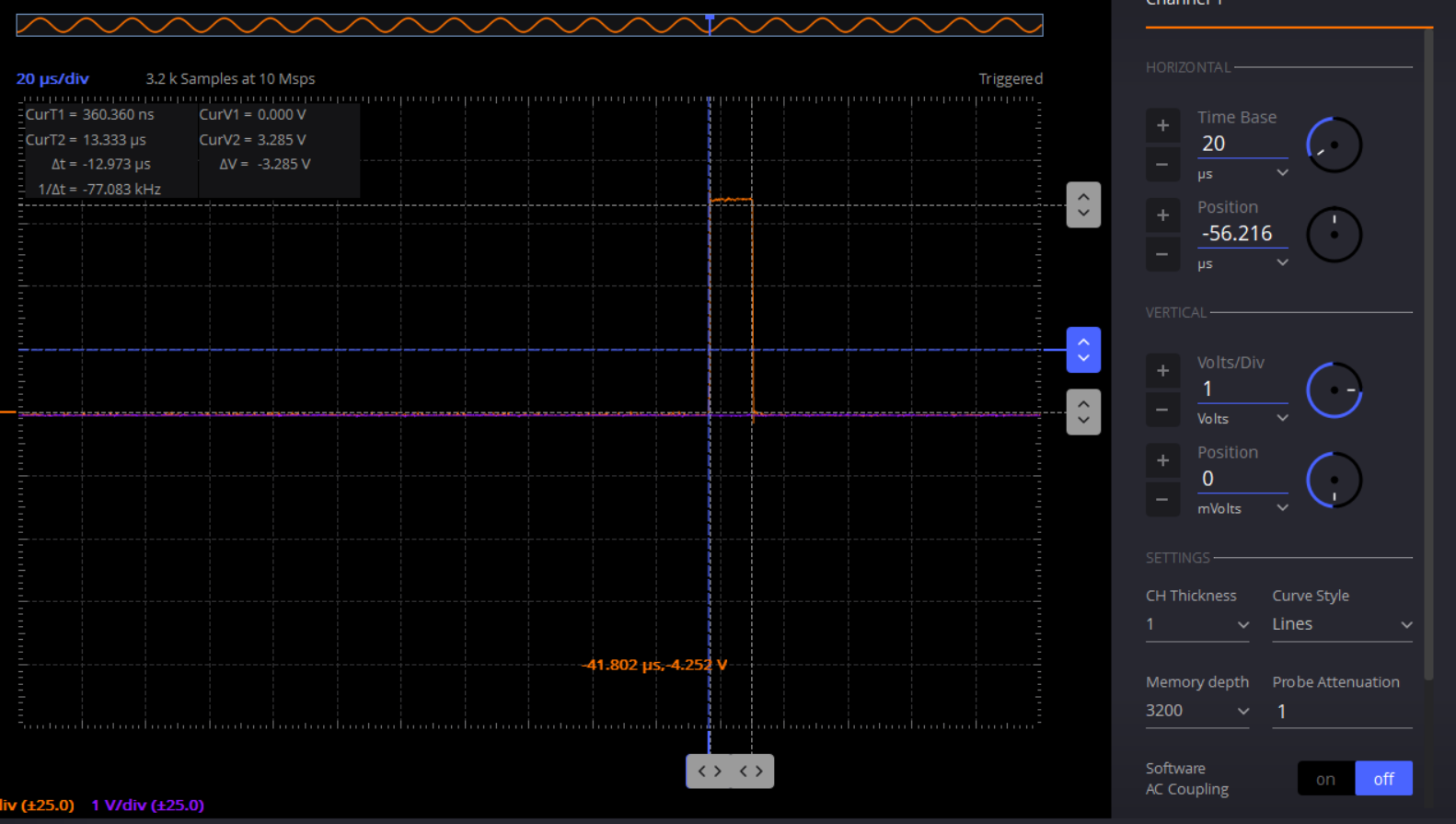


Where D and D’ are the output from the board to control the MOSFET. The direction of the motor is directly controlled by the direction of the current passing through. And the signal D will act as a switch to change the direction of the current. So the board can control the movement of the buggy by setting different combinations of signals to the MOSFET for two motors. For example, when both motors are set to the same direction, the buggy will move forward or backward. On the other hand, if the motors are set to different directions. The buggy will start turning. And the degree of the turn can be set by the length of the turning motion.

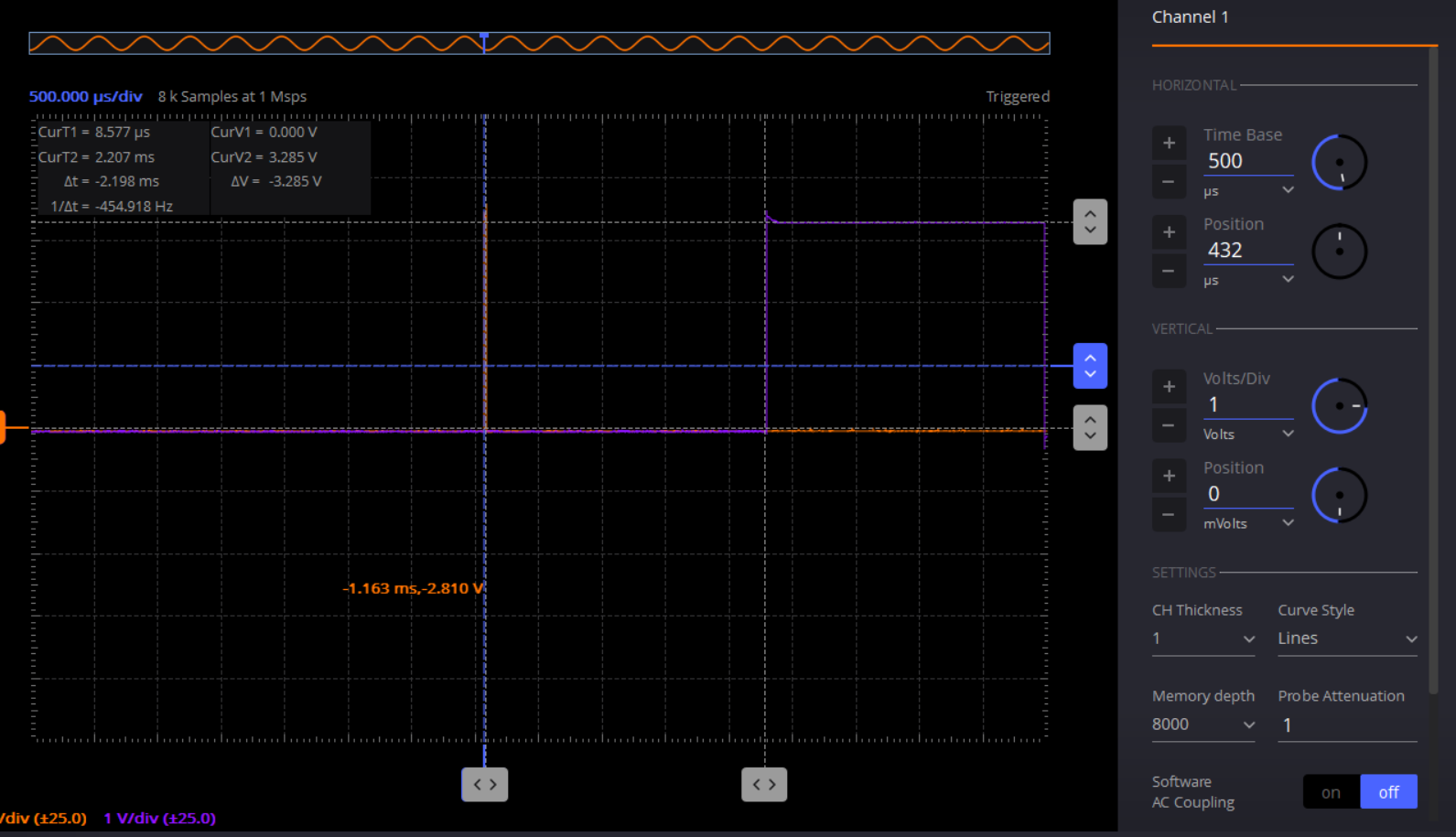
**Results**

Several parameters of the ultrasonic sensor were measured during the testing. To monitor the input and the output of the ultrasonic sensor, the ADALM2000 was connected to the Trig and Echo pin of the sensor.

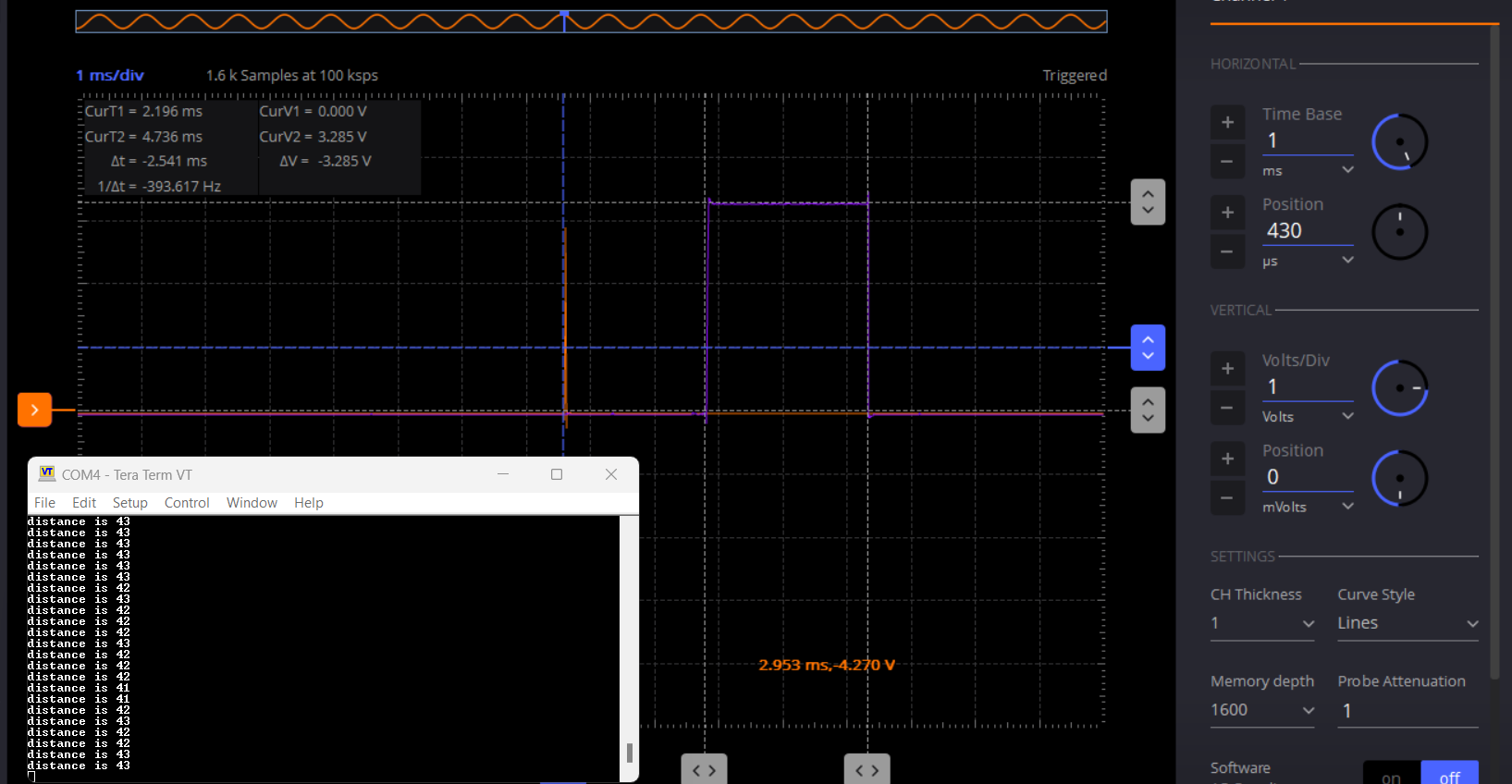
The first parameter analyzed was the length of the trigger pulse sent to the Trig pin. While the theoretical length is expected to be 10us. The measured minimum length we can set is 12us. This is likely due to the limitations imposed by the microcontroller’s system clock.



The second parameter is the time between the end of the trigger pulse and the rise of the echo pulse. This marks the waiting time for the board to receive a response from the ultrasonic sensor. And the average value is 2.2ms.



To calibrate the sensor and measure the accuracy, we set the ultrasonic sensor running and place objects at different distances. We compared the reading of the board, the distance calculated by the echo interval, and the physical distance. An example measurement is shown below:



An object is placed at 45cm away from the sensor. The reading from the board is 43cm. And the echo time is 2.54ms, which corresponds to 43.2cm. This shows that the board accurately measures the time interval and correctly calculates the distance based on it.

Based on several measurements around 5cm, 15cm, 45cm, 75cm, 100cm, 200cm, and 300cm. The valid measuring distance is between 5cm to 200cm. On average the error is ±4%.

For the motor, the board successfully controls the movement of the buggy through H-bridge and output pins. However, the speed is limited by the weight of the buggy, especially with all components including the camera and the power bank.

**Discussion and Conclusion**

In conclusion, the ultrasonic sensor demonstrated reliable accuracy within the expected tolerance range. The motor performed well with forward and backward movement, but the turning process was affected by the weight of the buggy, requiring further calibration.

There are several parts that need to be improved. Currently, the distance measurement is transmitted to the computer via cable connection. This heavily limits the buggy’s mobility within the maze. Wireless connection, such as Bluetooth can be a solution to this problem. While the current board doesn’t have an embedded Bluetooth module, an external module can be easily integrated.

Additionally, the motor’s strength may be insufficient to handle the buggy’s weight, which includes the camera and power bank. To resolve this issue, additional motors are needed. This adjustment may require a different motor kit or redesigning the buggy's chassis to accommodate the increased power demand.

**References** - from literature cite related work of others in the field.

1. Dance D, Christofides S, Maidment A, McLean I, Ng K, eds. (2014). "Physics of Ultrasound". Diagnostic Radiology Physics: A Handbook for Teachers and Students. International Atomic Energy Agency. p. 291. ISBN 978-92-0-131010-1.
2. Papadakis EP, ed. (1999). Ultrasonic Instruments & Devices. Academic Press. p. 752. ISBN 978-0-12-531951-5.
3. US Department of Commerce, National Oceanic and Atmospheric Administration. "NOAA Ocean Explorer: Technology: Observation Tools: SONAR". oceanexplorer.noaa.gov. Archived from the original on 2020-06-22. Retrieved 2020-06-22.
4. Texas Instrument, MSP432P401R, MSP432P401M Mixed-Signal Microcontrollers, <https://www.ti.com/lit/ds/slas826e/slas826e.pdf>
5. Adafruit Industries LLC, Ultrasonic Distance Sensor - 3V or 5V -HC-SR04 compatible <https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/307/4007_Web.pdf>
6. Handson Technology, User Guide: Ultrasonic Sensor V2.0, <https://www.handsontec.com/dataspecs/HC-SR04-Ultrasonic.pdf>
7. Littlelf, Home Camera, <https://www.littlelf.com/HomeCamera.html>
8. Keyestudio, KS0324 Keyestudio Motor Wheel Kit for Smart Car <https://fs.keyestudio.com/KS0324>

**Appendix**

1. Code for the ultrasonic sensor: <https://uofi.box.com/s/3v1e9cgza525ilnxt3yr73ql0z91aubv>
2. Code for the motor control:

<https://uofi.box.com/s/uc2gs46cj6fopchpzhufaf5k3k7hms7e>

**UStesting\_**

[UStesting\_Albert](https://uofi.app.box.com/s/3v1e9cgza525ilnxt3yr73ql0z91aubv/folder/293191254742)·Updated Dec 5, 2024 by Albert Jiang

#include "msp.h"

#include <stdint.h>

//higher number has lower priority

#define PORT2\_PRIO 4

#define TIMEA0\_PRIO 3

#define MCLKFREQ

#define INTR\_PRD 1000

#define DIST\_DIV 1400

//according to HCSR-04 datasheet, trigger length is 10us and max echo length is 60ms

//by my measurement, the echo length is from 135us to 55ms

int intrcnt = 0;

int distance = 0;

long clktick = 0;

void UART0init();

int uart\_puts(const char \*str);

void Delay(uint32\_t tick);

/\*\*

\* The main function initializes all port, including, UART, trigger, echo. And keep triggering

\* the ultrasonic sensor and send the distance converted from echo length to the console

\*/

int main(void)

{

WDT\_A->CTL = WDT\_A\_CTL\_PW | // Stop watchdog timer

WDT\_A\_CTL\_HOLD;

//setting up clock signals

CS->KEY = CS\_KEY\_VAL; // Unlock CS module for register access

CS->CTL0 = 0; // Reset tuning parameters

CS->CTL0 = CS\_CTL0\_DCORSEL\_3; // Set DCO to 12MHz (nominal, center of 8-16MHz range)

CS->CTL1 = CS\_CTL1\_SELA\_2 | // Select ACLK = REFO

CS\_CTL1\_SELS\_3 | // SMCLK = DCO

CS\_CTL1\_SELM\_3; // MCLK = DCO

CS->KEY = 0; // Lock CS module from unintended accesses

UART0init();

// Configure GPIO

// Setting up LED for distance detecting

P1->DIR |= BIT0; // Set P1.0 as output

//set p2 as GPIO

P2->SEL0 = 0;

P2->SEL1 = 0;

// uses p2.7 as echo

P2->DIR &= ~BIT7; // P2.7 as input pin

P2->REN |= BIT7; // P2.7 pull resistor enabled

P2->OUT &= ~BIT7; // P2.7 selected as pull down (active low)

// uses p2.6 as trigger

P2->DIR |= BIT6; // trigger pin as output

// receiving interrupt from p2.7

P2->IFG = 0; // clean pending interrupt flag

P2->IES &= ~BIT7; // enable rising edge interrupt

P2->IE |= BIT7; // enable interrupt

//setting up timer A0

TIMER\_A0->CCTL[0] = TIMER\_A\_CCTLN\_CCIE; // CCR0 interrupt enabled

TIMER\_A0->CCR[0] = INTR\_PRD - 1; // interrupt is raised for every 1000 clock tick

TIMER\_A0->CTL = TIMER\_A\_CTL\_TASSEL\_2 | TIMER\_A\_CTL\_MC\_\_UP | TIMER\_A\_CTL\_CLR; // SMCLK, upmode, TA clear

// Enable Port2 interrupt for echo and set priority as 4

NVIC\_SetPriority(PORT2\_IRQn, PORT2\_PRIO);

NVIC\_EnableIRQ(PORT2\_IRQn);

// Enable Timer interrupt for timing and set priority as 3

NVIC\_SetPriority(TA0\_0\_IRQn, TIMEA0\_PRIO);

NVIC\_EnableIRQ(TA0\_0\_IRQn);

\_\_enable\_irq(); // Enables interrupts to the system

while (1)

{

// uses pin 2.6 for trigger

P2->OUT |= BIT6; // generate pulse

Delay(120); // 120 cycle in 12mhz is about 10us

P2->OUT &= ~BIT6; // stop pulse

P2->IFG = 0; // clear P2 interrupt just in case anything happened before

P2->IES &= ~BIT7; // wait for rising edge on ECHO pin

Delay(660000); // wait for a maximum amount of time just in case there is no echo detected

distance = clktick / DIST\_DIV; // converting ECHO time into cm

char buffer[50];

sprintf(buffer, "distance is %d\n", distance); // format the output string

uart\_puts(buffer);

//turning LED on if distance is less than certain range and if distance isn't 0.

if (distance < 50 && distance != 0)

P1->OUT |= BIT0; //

else

P1->OUT &= ~BIT0;

Delay(30000);

}

}

/\*\*

\* This function is initializing the UART0 for sending distance message to the console with baud rate of 9600

\*

\*/

void UART0init()

{

// Configure UART pins

P1->SEL0 |= BIT2 | BIT3; // set 2-UART pin as secondary function

// Configure UART

EUSCI\_A0->CTLW0 |= EUSCI\_A\_CTLW0\_SWRST; // Put eUSCI in reset

EUSCI\_A0->CTLW0 |= EUSCI\_B\_CTLW0\_SSEL\_\_SMCLK; // Configure eUSCI clock source for SMCLK

// Baud Rate calculation

// 12000000/(16\*9600) = 78.125

// Fractional portion = 0.125

// User's Guide Table 21-4: UCBRSx = 0x10

// UCBRFx = int ( (78.125-78)\*16) = 2

EUSCI\_A0->BRW = 78; // 12000000/16/9600

EUSCI\_A0->MCTLW = (2 << EUSCI\_A\_MCTLW\_BRF\_OFS) |

EUSCI\_A\_MCTLW\_OS16;

EUSCI\_A0->CTLW0 &= ~EUSCI\_A\_CTLW0\_SWRST; // take eUSCI out of reset mode

EUSCI\_A0->IFG &= ~EUSCI\_A\_IFG\_RXIFG; // Clear eUSCI RX interrupt flag

EUSCI\_A0->IE &= ~EUSCI\_A\_IE\_RXIE; // Disable USCI\_A0 RX interrupt

}

/\*\*

\* This function puts a string to transmit buffer in the UART, which will be sent to console

\*/

int uart\_puts(const char \*str)

{

int status = -1;

if (str != '\0')

{

status = 0;

while (\*str != '\0')

{

/\* Wait for the transmit buffer to be ready \*/

while (!(EUSCI\_A0->IFG & EUSCI\_A\_IFG\_TXIFG))

;

/\* Transmit data \*/

EUSCI\_A0->TXBUF = \*str;

/\* If there is a line-feed, add a carriage return \*/

if (\*str == '\n')

{

/\* Wait for the transmit buffer to be ready \*/

while (!(EUSCI\_A0->IFG & EUSCI\_A\_IFG\_TXIFG))

;

EUSCI\_A0->TXBUF = '\r';

}

str++;

}

}

return status;

}

/\*\*

\* This function delay for the given amount of clock cycle

\*/

void Delay(uint32\_t tick)

{

// initialize timer32 1 with give amount of tick

TIMER32\_1->LOAD = tick;

//no prescaler, periodic wrapping mode, disable interrupt, 32-bit timer

TIMER32\_1->CONTROL = 0xc2;

//spin wait until the time is reached

while ((TIMER32\_1->RIS & 1) == 0)

;

TIMER32\_1->INTCLR = 0; //clear raw interrupt flag

}

/\*\*

\* Port2 interrupt service routine, start timing echo from rising edge to falling edge

\*/

void PORT2\_IRQHandler(void)

{

// check if interrupt is pending for p2.7

if (P2->IFG & BIT7)

{

//if we are checking for raising edge, we are at the beginning of the echo

if (!(P2->IES & BIT7 ))

{

//clears timer and starts counting for time

TIMER\_A0->CTL |= TIMER\_A\_CTL\_CLR; // clears timer A

intrcnt = 0;

//now checks for falling edge, which is the end of the echo

P2->IES |= BIT7;

}

else

{

clktick = (long) intrcnt \* 1000 + (long) TIMER\_A0->R; //calculating ECHO length

//now back to checking rising edge, which is the beginning of the echo

P2->IES &= ~BIT7;

}

P2->IFG &= ~BIT7; //clear flag

}

}

/\*\*

\* TimerA0 interrupt service routine, updates count of interrupt and clears interrupt flag

\*/

void TA0\_0\_IRQHandler(void)

{

// Interrupt gets triggered for every 1000 clock cycle in SMCLK

intrcnt++;

TIMER\_A0->CCTL[0] &= ~TIMER\_A\_CCTLN\_CCIFG;

}

[Company Logo](https://uofi.app.box.com/)

[UStesting\_Albert](https://uofi.app.box.com/s/3v1e9cgza525ilnxt3yr73ql0z91aubv/folder/293191254742)·Updated Dec 5, 2024 by Albert Jiang

#include "msp.h"

#include <stdint.h>

//higher number has lower priority

#define PORT2\_PRIO 4

#define TIMEA0\_PRIO 3

#define MCLKFREQ

#define INTR\_PRD 1000

#define DIST\_DIV 1400

//according to HCSR-04 datasheet, trigger length is 10us and max echo length is 60ms

//by my measurement, the echo length is from 135us to 55ms

int intrcnt = 0;

int distance = 0;

long clktick = 0;

void UART0init();

int uart\_puts(const char \*str);

void Delay(uint32\_t tick);

/\*\*

\* The main function initializes all port, including, UART, trigger, echo. And keep triggering \* the ultrasonic sensor and send the distance converted from echo length to the console

\*/

int main(void)

{

WDT\_A->CTL = WDT\_A\_CTL\_PW | // Stop watchdog timer

WDT\_A\_CTL\_HOLD;

//setting up clock signals

CS->KEY = CS\_KEY\_VAL; // Unlock CS module for register access

CS->CTL0 = 0; // Reset tuning parameters

CS->CTL0 = CS\_CTL0\_DCORSEL\_3; // Set DCO to 12MHz (nominal, center of 8-16MHz range)

CS->CTL1 = CS\_CTL1\_SELA\_2 | // Select ACLK = REFO

CS\_CTL1\_SELS\_3 | // SMCLK = DCO

CS\_CTL1\_SELM\_3; // MCLK = DCO

CS->KEY = 0; // Lock CS module from unintended accesses

UART0init();

// Configure GPIO

// Setting up LED for distance detecting

P1->DIR |= BIT0; // Set P1.0 as output

//set p2 as GPIO

P2->SEL0 = 0;

P2->SEL1 = 0;

// uses p2.7 as echo

P2->DIR &= ~BIT7; // P2.7 as input pin

P2->REN |= BIT7; // P2.7 pull resistor enabled

P2->OUT &= ~BIT7; // P2.7 selected as pull down (active low)

// uses p2.6 as trigger

P2->DIR |= BIT6; // trigger pin as output

// receiving interrupt from p2.7

P2->IFG = 0; // clean pending interrupt flag

P2->IES &= ~BIT7; // enable rising edge interrupt

P2->IE |= BIT7; // enable interrupt

//setting up timer A0

TIMER\_A0->CCTL[0] = TIMER\_A\_CCTLN\_CCIE; // CCR0 interrupt enabled

TIMER\_A0->CCR[0] = INTR\_PRD - 1; // interrupt is raised for every 1000 clock tick

TIMER\_A0->CTL = TIMER\_A\_CTL\_TASSEL\_2 | TIMER\_A\_CTL\_MC\_\_UP | TIMER\_A\_CTL\_CLR; // SMCLK, upmode, TA clear

// Enable Port2 interrupt for echo and set priority as 4

NVIC\_SetPriority(PORT2\_IRQn, PORT2\_PRIO);

NVIC\_EnableIRQ(PORT2\_IRQn);

// Enable Timer interrupt for timing and set priority as 3

NVIC\_SetPriority(TA0\_0\_IRQn, TIMEA0\_PRIO);

NVIC\_EnableIRQ(TA0\_0\_IRQn);

\_\_enable\_irq(); // Enables interrupts to the system

while (1)

{

// uses pin 2.6 for trigger

P2->OUT |= BIT6; // generate pulse

Delay(120); // 120 cycle in 12mhz is about 10us

P2->OUT &= ~BIT6; // stop pulse

P2->IFG = 0; // clear P2 interrupt just in case anything happened before

P2->IES &= ~BIT7; // wait for rising edge on ECHO pin

Delay(660000); // wait for a maximum amount of time just in case there is no echo detected

distance = clktick / DIST\_DIV; // converting ECHO time into cm

char buffer[50];

sprintf(buffer, "distance is %d\n", distance); // format the output string

uart\_puts(buffer);

//turning LED on if distance is less than certain range and if distance isn't 0.

if (distance < 50 && distance != 0)

P1->OUT |= BIT0; //

else

P1->OUT &= ~BIT0;

Delay(30000);

}

}

/\*\*

\* This function is initializing the UART0 for sending distance message to the console with baud rate of 9600

\*

\*/

void UART0init()

{

// Configure UART pins

P1->SEL0 |= BIT2 | BIT3; // set 2-UART pin as secondary function

// Configure UART

EUSCI\_A0->CTLW0 |= EUSCI\_A\_CTLW0\_SWRST; // Put eUSCI in reset

EUSCI\_A0->CTLW0 |= EUSCI\_B\_CTLW0\_SSEL\_\_SMCLK; // Configure eUSCI clock source for SMCLK

// Baud Rate calculation

// 12000000/(16\*9600) = 78.125

// Fractional portion = 0.125

// User's Guide Table 21-4: UCBRSx = 0x10

// UCBRFx = int ( (78.125-78)\*16) = 2

EUSCI\_A0->BRW = 78; // 12000000/16/9600

EUSCI\_A0->MCTLW = (2 << EUSCI\_A\_MCTLW\_BRF\_OFS) |

EUSCI\_A\_MCTLW\_OS16;

EUSCI\_A0->CTLW0 &= ~EUSCI\_A\_CTLW0\_SWRST; // take eUSCI out of reset mode

EUSCI\_A0->IFG &= ~EUSCI\_A\_IFG\_RXIFG; // Clear eUSCI RX interrupt flag

EUSCI\_A0->IE &= ~EUSCI\_A\_IE\_RXIE; // Disable USCI\_A0 RX interrupt

}

/\*\*

\* This function puts a string to transmit buffer in the UART, which will be sent to console

\*/

int uart\_puts(const char \*str)

{

int status = -1;

if (str != '\0')

{

status = 0;

while (\*str != '\0')

{

/\* Wait for the transmit buffer to be ready \*/

while (!(EUSCI\_A0->IFG & EUSCI\_A\_IFG\_TXIFG))

;

/\* Transmit data \*/

EUSCI\_A0->TXBUF = \*str;

/\* If there is a line-feed, add a carriage return \*/

if (\*str == '\n')

{

/\* Wait for the transmit buffer to be ready \*/

while (!(EUSCI\_A0->IFG & EUSCI\_A\_IFG\_TXIFG))

;

EUSCI\_A0->TXBUF = '\r';

}

str++;

}

}

return status;

}

/\*\*

\* This function delay for the given amount of clock cycle

\*/

void Delay(uint32\_t tick)

{

// initialize timer32 1 with give amount of tick

TIMER32\_1->LOAD = tick;

//no prescaler, periodic wrapping mode, disable interrupt, 32-bit timer

TIMER32\_1->CONTROL = 0xc2;

//spin wait until the time is reached

while ((TIMER32\_1->RIS & 1) == 0)

;

TIMER32\_1->INTCLR = 0; //clear raw interrupt flag

}

/\*\*

\* Port2 interrupt service routine, start timing echo from rising edge to falling edge

\*/

void PORT2\_IRQHandler(void)

{

// check if interrupt is pending for p2.7

if (P2->IFG & BIT7)

{

//if we are checking for raising edge, we are at the beginning of the echo

if (!(P2->IES & BIT7 ))

{

//clears timer and starts counting for time

TIMER\_A0->CTL |= TIMER\_A\_CTL\_CLR; // clears timer A

intrcnt = 0;

//now checks for falling edge, which is the end of the echo

P2->IES |= BIT7;

}

else

{

clktick = (long) intrcnt \* 1000 + (long) TIMER\_A0->R; //calculating ECHO length

//now back to checking rising edge, which is the beginning of the echo

P2->IES &= ~BIT7;

}

P2->IFG &= ~BIT7; //clear flag

}

}

/\*\*

\* TimerA0 interrupt service routine, updates count of interrupt and clears interrupt flag

\*/

void TA0\_0\_IRQHandler(void)

{

// Interrupt gets triggered for every 1000 clock cycle in SMCLK

intrcnt++;

TIMER\_A0->CCTL[0] &= ~TIMER\_A\_CCTLN\_CCIFG;

}

// Header to initialize SysTick for simple delay in ms or us

#include "msp.h"

#include <stdio.h>

void SysTick\_Init();

void SysTick\_Delayms();

void SysTick\_Delayus();

// Functions to initialize SysTick for simple delay in ms or us

#include "msp.h"

#include <stdio.h>

#include "SysTick0.h"

void SysTick\_Init();

void SysTick\_Delayms();

void SysTick\_Delayus();

// Function to initialize SysTick without interrupt enable

// No longer necessary, delay functions work indpendently

void SysTick\_Init(){

SysTick->CTRL = 0; // disable

SysTick->LOAD = 0x00FFFFFF; // set reload to max

SysTick->VAL = 0; // clear counter value

SysTick->CTRL = 0x00000005 ; // set CLKSOURCE to core clock and enable to 1

// clock is 3MHz, CTRL has interrupts disabled

// tech reference manual 2.4.4.1

return;

}

// Delay function with argument in milliseconds

void SysTick\_Delayus(int time){

SysTick->LOAD = (time \* 3 - 1); // 3000t / 3 MHz = t ms

// minimum 25 us, scales oddly until ~100 us

SysTick->VAL = 0; // any write to VAL clears VAL, restart

SysTick->CTRL = 0x00000005 ; // set CLKSOURCE to core clock and enable to 1

while(!(SysTick->CTRL & 0x00010000)); // stays in loop until COUNTFLAG is true

return;

}

// Delay function with argument in milliseconds

void SysTick\_Delayms(int time){

SysTick->LOAD = (time \* 3000 - 1); // 3000t / 3 MHz = t ms

SysTick->VAL = 0; // any write to VAL clears VAL, restart

SysTick->CTRL = 0x00000005 ; // set CLKSOURCE to core clock and enable to 1

while(!(SysTick->CTRL & 0x00010000)); // stays in loop until COUNTFLAG is true

return;

}