MXEN2003: Ultrasonic Range Sensor Assignment

Written by Istvan Savanyo (21492387)

Purpose/Functionality of the Program

The purpose of the range sensor and code is to light up an LED whenever an object is detected within 38.7cm of the sensor. This is achieved by pinging the sensor to release a short burst of sound waves, the sensor then 'listens' for any echo response that may have rebounded from the waves reflecting off a solid surface. The time-length of this response is used to determine the relative distance of the object, as distance and signal length share a closely proportional relationship. Control structures in the algorithm dictate whether the LED will light up.

The general functionality of the program is to perform a task depending on the proximity of the sensor to other objects. In the current program, the task is simply turning on an LED. However, this functionality can be applied to other tasks, for example, if a robot detects an object too close to its sensors it could shut off its motors to prevent a collision.

How the Program & Sensor Operate (Code Logic)

The PING))) sensor works by sending out a short ultrasonic burst and then listening for any echo that may occur from the sound waves reflecting off a surface. An Input Trigger Pulse (t_{out}) is sent to the sensor to produce the sound, typically 5 microseconds in length. There is also a 200-microsecond delay between measurements to prevent the overlapping of signals.

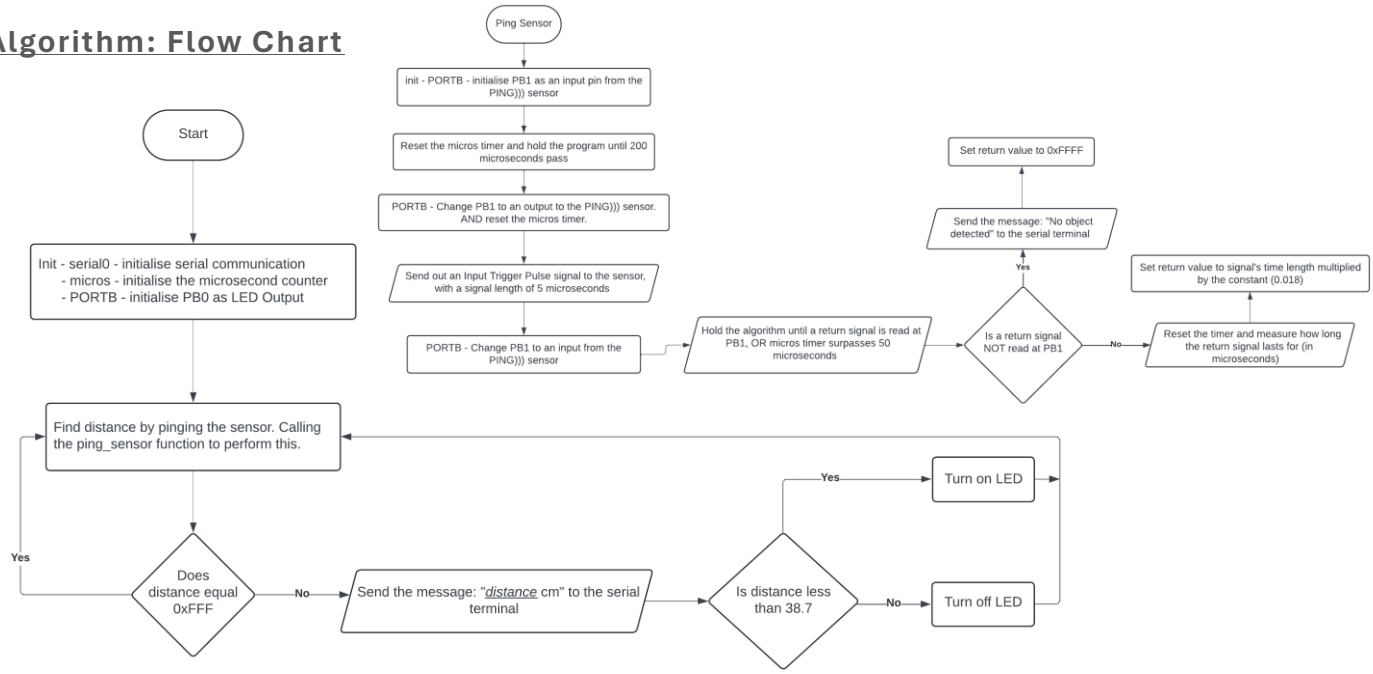
The ping_sensor() function in the code is responsible for pinging signals to the sensor and reading the response. The function first sets pin PB1, which is connected to the PING)))'s SIG pin, as an input. The program is held in this state using polling until 200 microseconds have passed, acting as the PING)))'s delay. Tracking the elapsed time was achieved by using one of the Arduino's internal timers, labelled *micros* in the code.

PB1 is then set to both output and HIGH, the program is then polled for 5 microseconds before the pin reverts to input and LOW. This is the Input Trigger Pulse sent out to the sensor. The program waits until either 50 microseconds pass have passed, or a signal is read at PB1. If the time is passed, it indicates that there is no object present. . An error message is then sent to terminal warning the user there is no object present, and 0xFFFF is returned to main. Otherwise, if a signal is read, then the micros timer is reset, and the time-length of the signal is recorded. This time is then multiplied by a predetermined constant to get the associated distance value, which is returned to main.

The main() program starts by initialising PB0 as the LED output, then calls ping_sensor() at the start of every loop. Control structures are used to determine how the program responds to the outputted distance value. If the distance is lower than 38.7 then PB0's output is set to HIGH, turning on the LED. Otherwise, the program will set PB0's output to LOW, turning the LED off.

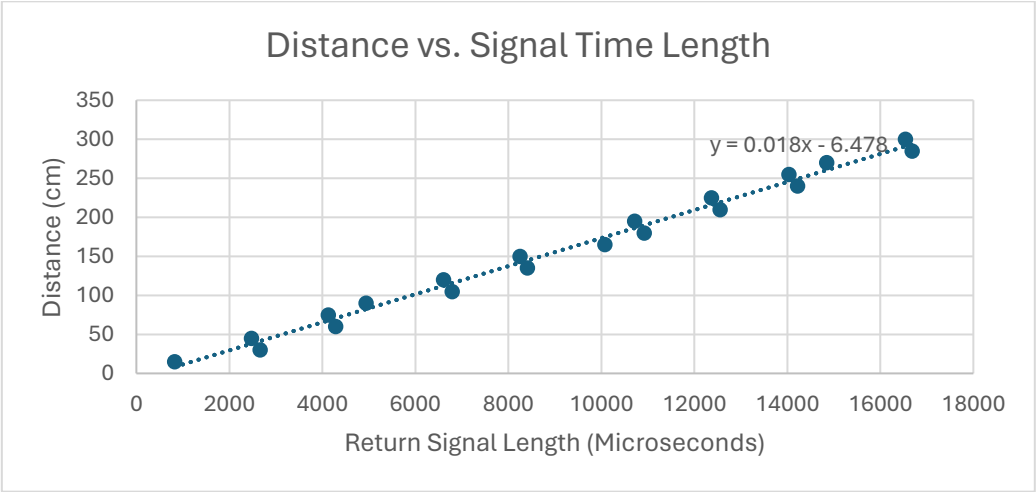
Following this algorithm, the code and sensor together are able to consistently track the distance of any object that is within the range of the sensor, and light up the LED when it becomes too close.

Algorithm: Flow Chart



Relevant Range Sensor Calculations

The original *constant* value (when multiplied by the signal length) produced distance values that were inaccurate within the TinkerCad space. To fix this problem, the constant was temporarily removed, and the raw microseconds signal length was recorded in 15cm intervals between the ranges of 15cm to 300cm. The data was then plotted on a graph with a line of best fit.

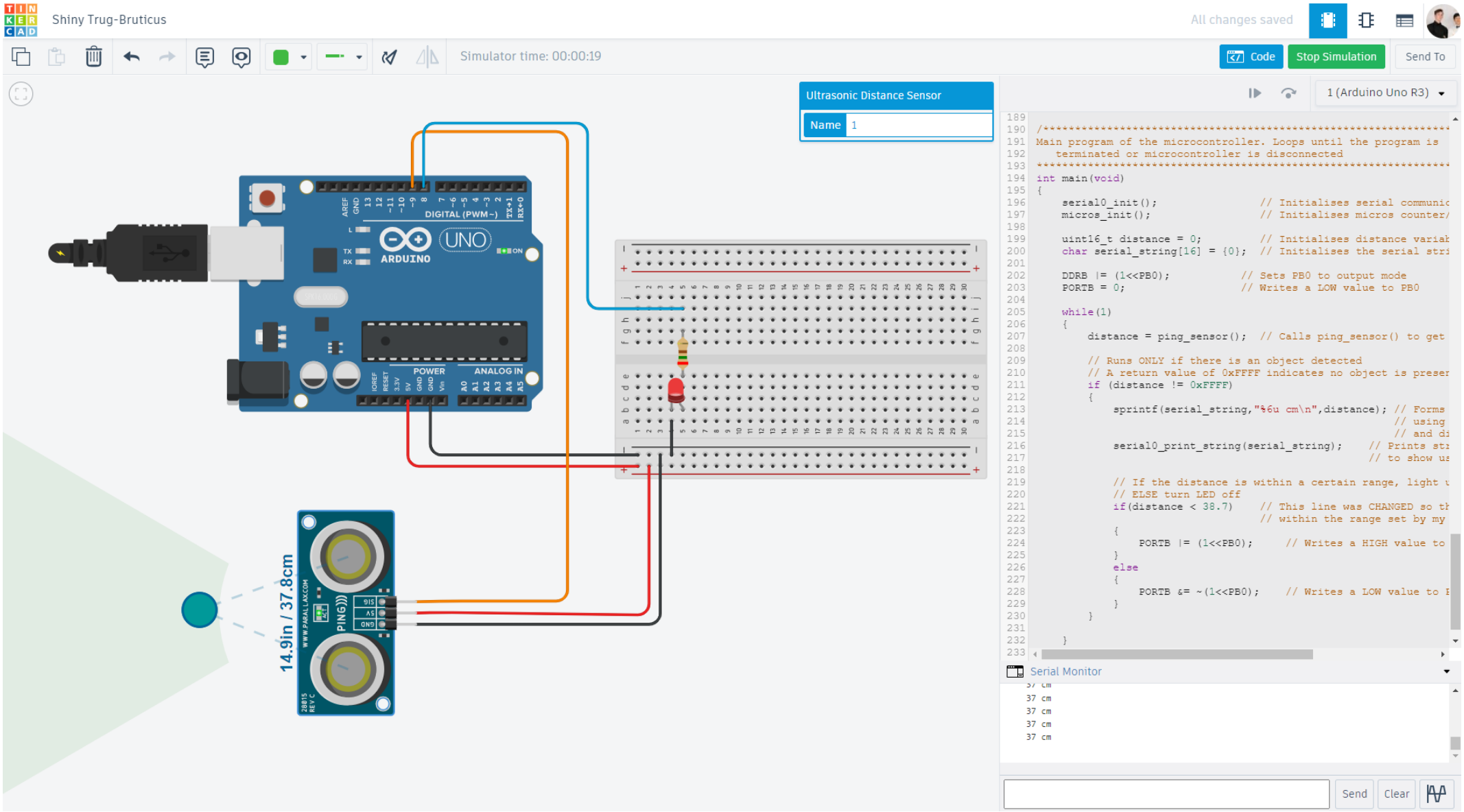


This line of best fit accurately represents the pattern between signal length and distance. The gradient of this line (0.018) ended up replacing the original constant so that the program produced a more accurate reading to the true distance.

The resistance of the resistor in the LED portion of the circuit also needed to be calculated. Using the "Standard LED Red Emitting Colour" datasheet, the continuous forward current through the LED should be 20mA. We assume a worst-case scenario where the LED has malfunctioned and acts as a short circuit when doing this calculation. With an input voltage of 5V, using ohms law the resistance value can be calculated as:

$$R = \frac{V}{I} = \frac{5}{20 * 10^{-3}} = 250\ \Omega$$

TinkerCad Screenshot



Range Sensor Code

```
//Define USART constants for ATmega328, see ATmega328P datasheet, pg 145
#define USART_BAUDRATE 9600
#define F_CPU 16000000
#define BAUD_PRESCALE ((((F_CPU/16)+(USART_BAUDRATE/2))/(USART_BAUDRATE))-1)

/*************************************************************************
Initialise USART 0
See ATmega328P datasheet for register descriptions, pg 159
Input: None
Output: None
*************************************************************************/
void serial0_init(void)
{
    UCSR0B = (1<<RXEN0)|(1<<TXEN0);     //Enable bits for transmit and recieve
    UCSR0C = (1<<UCSZ00)|(1<<UCSZ01);   //Use 8- bit character sizes
    UBRR0 = BAUD_PRESCALE;              //Load baud rate prescaler into register
}

/*************************************************************************
Print string via USART 0
See ATmega328P datasheet for register descriptions, pg 159
Input:      string_pointer      char array      string to be printed to serial 0
Output:     None
*************************************************************************/
void serial0_print_string(char * string_pointer)
{
    while(*string_pointer)              //While not null character (end of string)
    {
        while((UCSR0A&(1<<UDRE0))==0){} //Wait for register empty flag
        UDR0 = *string_pointer;         //Send what's at the string pointer to serial data
register
        string_pointer++;               //Increment string pointer to go to next letter in
string
    }
}

/*************************************************************************
Test serial sprinting accross USART 0
    Will initialise USART 0 if not initialised
Input: None
Output: None
*************************************************************************/
void test_print(void)
{
    if(UBRR0 != BAUD_PRESCALE)      //Check USART prescale set
    {
        //Run initialisation if not set and print test string
        serial0_init();
        serial0_print_string("Serial 0 was not initialised. Now initialised and working");
    }
    else
```

```
    {
        //Print test string
        serial0_print_string("Serial 0 is working");
    }
}



//Define global variable
volatile uint32_t microseconds = 0;

/*********************************************************************
Initialise microsecond timer using timer 1
    Increments in 1000 microsecond intervals
    Tracks microseconds using counter register
Input: None
Output: None
*********************************************************************/
void micros_init(void)
{
    cli();                          //Disable global interrupts
    TCCR1A = 0;                     //No pin outputs required
    TCCR1B = (1<<WGM12);            //Set CTC mode
    TCNT1 = 0;                      //Reset timer counter
    OCR1A = 15999;                  //Set comparison register A for 1000 microsecond
intervals
    TIMSK1 |= (1<<OCIE1A);          //Set Output Compare Interrupt Enable 1 A
    TCCR1B |= (1<<CS10);            //Set prescaler to 1, starting timer
    sei();                          //Enable global interrupts
}

/*********************************************************************
Reset microseconds timer
    Reset counter register and variable
Input: None
Output: None
*********************************************************************/
void micros_reset(void)
{
    uint8_t oldSREG = SREG;
    TCNT1 = 0;
    microseconds = 0;
    SREG = oldSREG;
}

/*********************************************************************
Returns the current microseconds count
Input: None
Output: uint32_t microseconds
*********************************************************************/
uint32_t micros_now(void)
{
    uint32_t m;
    uint8_t oldSREG = SREG;
```

```
    // disable interrupts while we read timer0_millis or we might get an
    // inconsistent value (e.g. in the middle of a write to timer0_millis)
    cli();
    m = microseconds;              //Assign microseconds
    m += (TCNT1>>4);               //Add counter value to microseconds, adjust for clock
speed

    SREG = oldSREG;

    return m;
}

/*****************************************************************
Timer 1 compare a interrupt service routine
    Increments global microseconds variable in 5 microsecond intervals
*****************************************************************/
ISR(TIMER1_COMPA_vect)
{
    microseconds += 1000;              //increment microseconds
}
/*****************************************************************/



// Define time constants for PING))) signals and delays. See PING))) datasheet, pg 2
#define  t1 5
#define  t2 50
#define  t3 200
// The constant was CHANGED from its original 352.4*100/1000000/2 value
// in order to more accurately display distance in the TinkerCad environment
#define  constant 0.018
#define  SIG_PIN     PB1

/*****************************************************************
Pings a signal from the PING))) Sensor and reads a response
Input: None
Output: distnce (which equals: Response signal time * constant)
*****************************************************************/
uint16_t ping_sensor(void){

        // Delays the sensor by 200 microseconds. Pg.2 of the PING)))
        // datasheet shows sensor requires this delay to operate correctly
        PORTB &= ~(1<<SIG_PIN);         // Writes a LOW value to PB1
        DDRB &= ~(1<<SIG_PIN);          // Sets PB1 into input mode
        micros_reset();                 // Resets micros timer to 0
        while((micros_now()) < t3){}    // Polling the code until time
                                        // is greater than 200 microseconds

        // Sends out the Input Trigger Pulse (T out) with a signal time of
        // 5 microseconds. As seen on Pg.2 of the PING))) datasheet.
        DDRB |= (1<<SIG_PIN);         // Sets PB1 into output mode
        PORTB |= (1<<SIG_PIN);        // Writes a HIGH value to PB1
        micros_reset();               // Resets timer to 0
```

```
        while((micros_now()) < t1){}  // Polling the code until timer is
                                      // greater than 5 microseconds


        // Turns off Input Trigger Pulse, then waits until a response
        // signal is detected
        PORTB &= ~(1<<SIG_PIN);       // Writes a LOW value to PB1
        DDRB &= ~(1<<SIG_PIN);        // Sets PB1 into input mode
        // Polls the code until a signal is read at PB1
        // OR too much time has elapsed on the micros timer (50 microseconds)
        while((!(PINB & (1<<SIG_PIN))|| (micros_now() < t2))){}


        // If a signal IS NOT read at PB1 within the 50 microseconds
        if(!(PINB & (1<<SIG_PIN)))
        {
            serial0_print_string("No object detected!");
            return 0xFFFF;                // 0xFFFF return value to indicate no object
was found
        }
        // If a signal IS read at PB1, program will measure the time length of the
        // signal and convert it to a distance by multiplying it by the 'constant'
        else
        {
            micros_reset();               // Resets timer to 0
            while(PINB & (1<<SIG_PIN)){}  // Polling the code until a signal
                                          // is no longer read at PB1

            return(micros_now()*constant); // Returns the distance where
                                          // distance = signal time * constant
        }
}


/*****************************************************************
Main program of the microcontroller. Loops until the program is
   terminated or microcontroller is disconnected.
*****************************************************************/
int main(void)
{
    serial0_init();               // Initialises serial communication
    micros_init();                // Initialises micros counter/timer

    uint16_t distance = 0;        // Initialises distance variable
    char serial_string[16] = {0}; // Initialises the serial string [16 characters]

    DDRB |= (1<<PB0);           // Sets PB0 to output mode
    PORTB = 0;                  // Writes a LOW value to PB0


    while(1)
    {
        distance = ping_sensor();  // Calls ping_sensor() to get distance variable
```

```
        // Code only runs when an object is detected and a valid distance is returned
        // 0xFFFF is only returned when an objectis NOT detected
        if (distance != 0xFFFF)
        {
            sprintf(serial_string,"%6u cm\n",distance); // Forms a serial string through
sprintf()to
                                                        // allow the changing distance
value to be
                                                        // printed

            serial0_print_string(serial_string); // Prints serial stirng to terminal to
show user
                                                 // the recorded distance

            // If statements turn on LED if distance is within a set range.
            if(distance < 38.7)    // This line was CHANGED so that the LED lights up
                                   // within the range set by my student ID: 21492[38.7]
            {
                PORTB |= (1<<PB0);     // Writes a HIGH value to PB0
            }
            else
            {
                PORTB &= ~(1<<PB0);    // Writes a LOW value to PB0
            }
        }

    }
    return(1);
}//end main
```