Sensor Fusion: Ultrasonic Sensor

Microcontroller --> PGA460 --> Murata Ultrasonic Sensor

**Abstract**

Ultrasonic sensors operate on the principle of time-of-flight for the purpose of distance measurement. The time-of-flight principle refers to a measurement of time taken by a particle to travel through a medium. The duration is then used to calculate the distance between the ultrasonic sensor and the barrier by taking d = v\*t/2, for which v is the speed of sound (a constant) and t is the duration the signal takes to travel to the barrier and back to the emitter. The division by 2 in the calculation ensures that the result is the distance between the sensor and barrier, and not the total distance traveled by the signal.

**Materials**

Microcontroller: Arduino Mega

Ultrasonic Driver/ Transducer: PGA460

Ultrasonic sensor: Murata ma58mf14-7n

Step up transformer and signal amplifier circuitry integrated into the system

**Vision:**

Establish communication between microcontroller, transducer, and ultrasonic sensor. Eventual multiplexing between one transducer and multiple ultrasonic sensor.

**Procedure**

In order to interface with the ultrasonic sensor and process returning signals, an embedded system of a microcontroller, transducer, and ultrasonic senor is needed. Due to specific operating settings (voltage, current, etc) a step-up transformer is needed between PGA460 and ultrasonic sensor. Listed below are summary of specifications of PGA460 and Murata ultrasonic sensor.

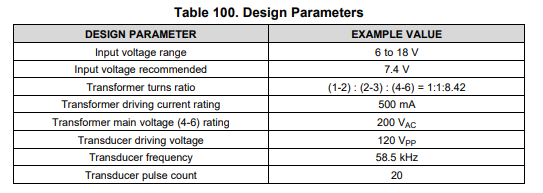
PGA460

Vin; 6-28V, optimum 7.4 V

Baud rate: up to 119200

Serial communication: SERIAL\_8N2

Distance range: Preset 1 (<1 meter) and Preset 2(>1 meter)



Murata ma58mf14-7n

Vin: 80-120Vpp (<20 pulses per second)

Center Frequency: 58 kHz

**Plans going forward:**

First stage:

* + Initializing settings for PGA460.
  + Establishing communication between PGA460 and microcontroller

Second stage:

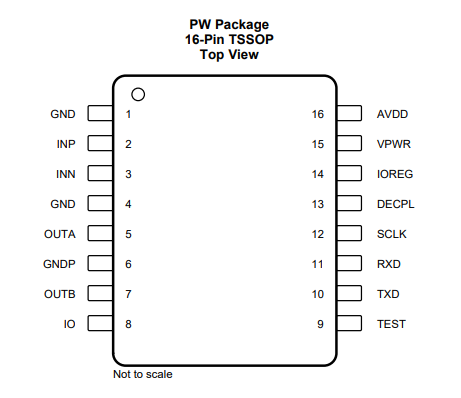
* + Interfacing PGA460 with sensor
  + Sensor to pulse signals given by PGA460
  + Sensor to detect reflected signal by barrier
  + PGA460 to process reflected signal to derive distance
  + Communicating result back to microcontroller

Third Stage:

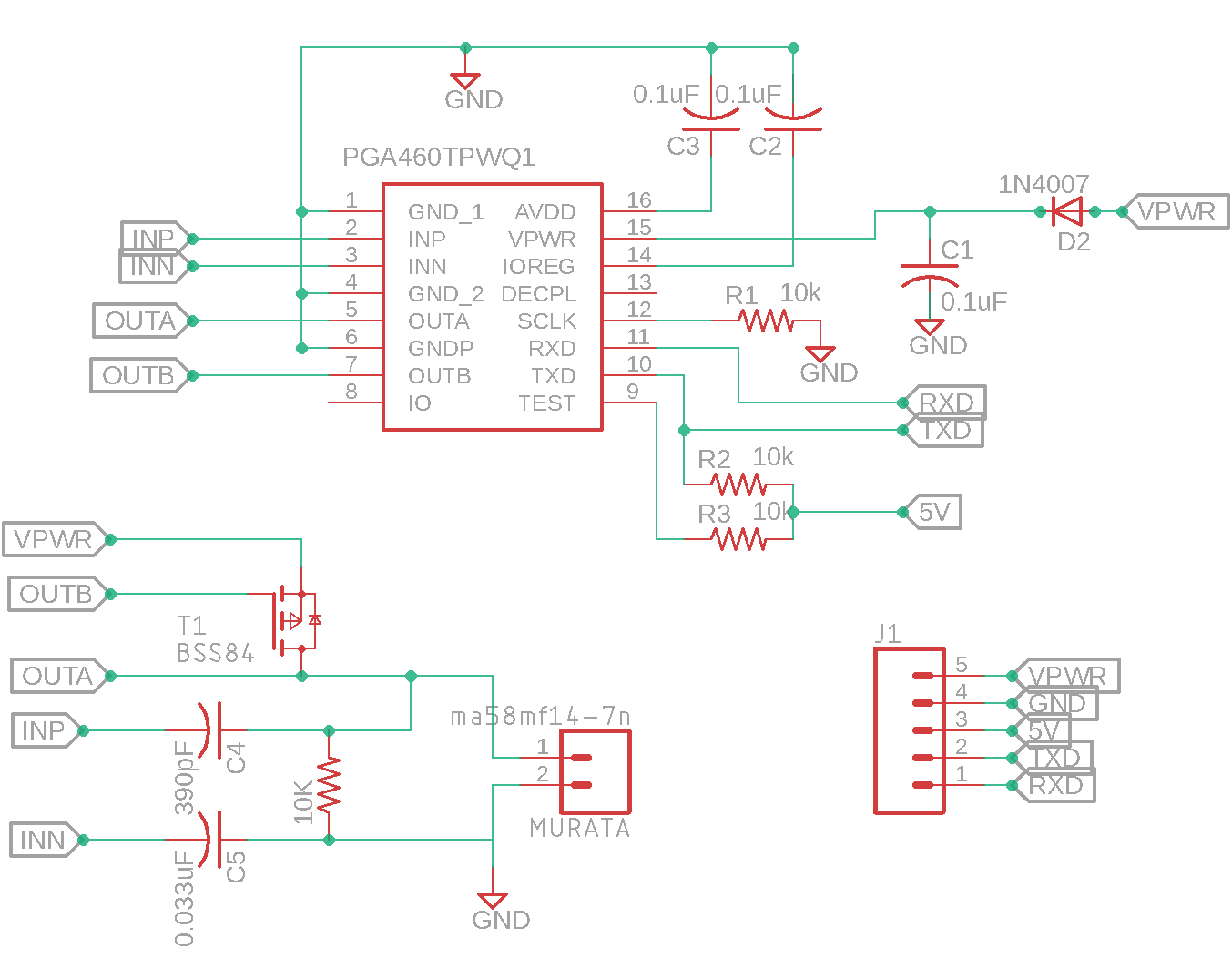
* + Driving multiple sensors with one transducer (Multiplexing)
  + Transmitting distance results back to microcontroller
  + Embedded system to run at rate more than 4Hz

**HARDWARE GUIDE**

***PGA460 Pinout***



***Direct-driven Schematics***



***Transformer-driven Schematics***

**INSERT IMAGE**

For reference on design requirements behind circuitry interfacing with PGA460 with Arduino Mega 2560 and Murata ma58mf14-7n ultrasonic sensor, refer to [PGA460 Ultrasonic Signal Processor and Transducer Driver](http://www.ti.com/lit/ds/symlink/pga460.pdf) datasheet under Section 8 Application and Implementation.

Due to variances in how each component is manufactured, some components used in the circuitry do not exactly follow the suggested values indicated in the PGA460 Ultrasonic Signal Processor and Transducer Driver datasheet. However, the [PGA460 Ultrasonic Module Hardware and Software Optimization](file:///C:\Users\myuyenn\AppData\Roaming\Microsoft\Word\:%20http:\www.tij.co.jp\jp\lit\an\slaa732\slaa732.pdf) guide, under Section 3.4 Passive Tuning, shows flexibility in how these values can be modified in order to best be tuned with the circuitry setup.

**SOFTWARE GUIDE**

***Communication mode***

PGA460 has four main modes of communication: UART, OWU, SPI, and TCI. For our purpose of measuring distances, we will be using UART, as it is the most common and simplest form of serial communication with Arduino MEGA 2560. The UART mode on PGA460 is designed to work in the baud rapture of between 2400-bps and 15200-bps, in which the baud rate is automatically detected based on the sync byte that is issued in the beginning of every command. In addition, PGA460 operates on 8 data, 1 start, 0 parity, and 2 stop bits.

***Energia Library and Example Code***

For first-time exposure of interfacing with PGA460, it is recommended to read the [PGA460 Software Development Guide](http://www.ti.com/lit/an/slaa730a/slaa730a.pdf). The same code is also available in the downloadable Energia Library example, which can be ported into the Arduino IDE Library and requires commenting out #include "Energia.h"to make the code compilable. Since SPI mode is not used, it is also suggested that #include "PGA460\_SPI.h" and related SPI functions and variables be commented out in PGA460\_USSC.h and PGA460\_USSC.cpp to prevent need of installing more unnecessary libraries. A cleaned-up version of the h and cpp files, without SPI functions can found PGA460, 0.25-0.5m folder.

***Initialization Steps***

The order of instructions for which PGA460 is set up to properly work is as follows:

1. **On power up,** configures EERPOM values to indicate which ultrasonic transducer to being used and how PGA460 should execute its commands. This step is **optional** **if** these EEPROM values have being burned once onto the PGA460.
2. Configures Threshold parameters by using threshold bulk write command (THRBW) or by independently writing a particular parameter by using register write command (SRW).
3. Configures Time-varying gain by using time-varying gain bulk write
4. **Once successfully configured**, program will execute the following commands in a loop:
   1. BURST+LISTEN (Preset1 or Preset 2)
   2. After record interval has expired, issue ultrasonic measurement result command(UMR) to retrieve data
   3. Use time-of-flight calculation on retrieved data to compute distance

*EEPROM (Electrically-erasable Programmable Read-only Memory)*

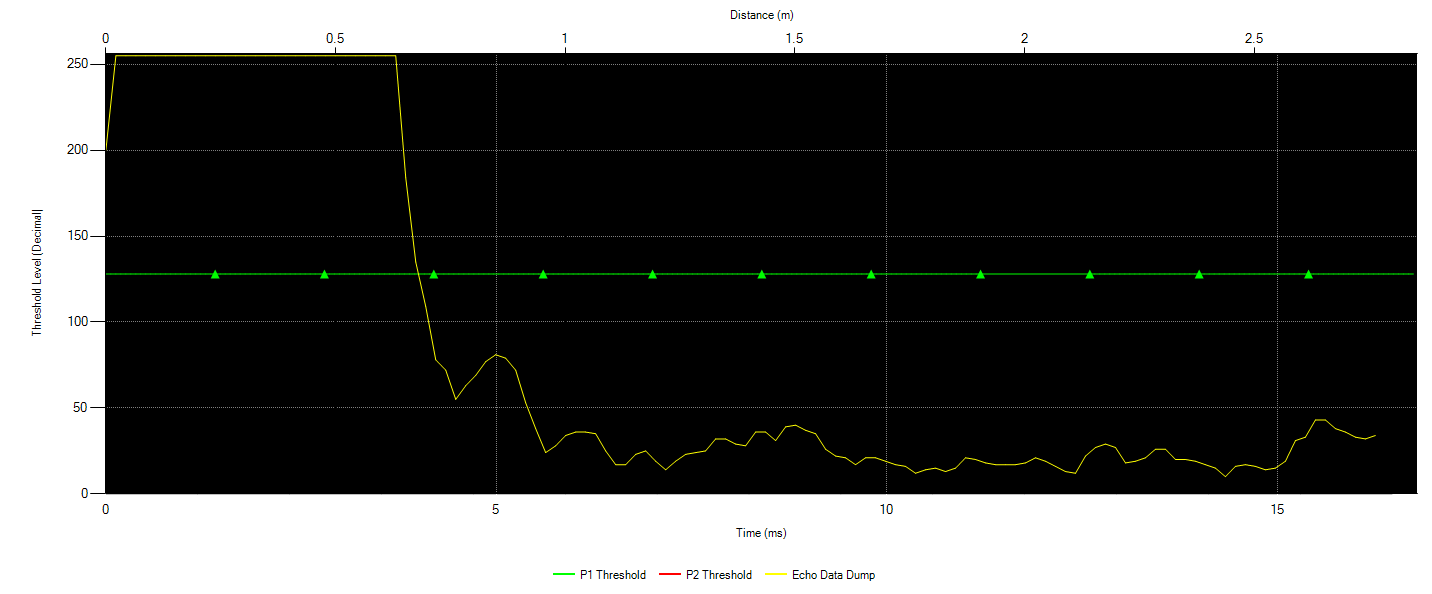
EEPROM is a non-volatile memory that contains parameters under which PGA460 recognizes type of transducer is being used and accordingly carries out its commands. The following parameters affect system diagnostics, burst and listen patterns, and filtering configurations:

* FREQUENCY, FREQ\_DIAG, DEADTIME determine how accurately the transducer’s frequency is analyzed and how PGA460 recognizes which echoing signal to process.
* REC\_LENGTH is associated with how long the PGA460 should wait and listen for the returning signals. During this period, no other commands should be issued to prevent interruption this listening period. Otherwise, the raw data will not be reported completely and cause diagnostic errors.
* PULSE\_1, PULSE\_2 determine the number of the bursting pulses for Preset 1 and Preset 2. The shorter the range of distance required to detect, the smaller the number of pulses should be to prevent a long ringing decay time that will interfere with object detection in short distances.
* P1\_GAIN\_CTRL, P2\_GAIN\_CTRL describes the amount of amplification throughout the listening period for Preset 1 and Preset 2

These parameters DO NOT have to be initialized every time on power up as long as it has been burned at least once. However, to experiment with different values and observe their effects on the results, the developer may use EEPROM bulk write command to override the burned settings.

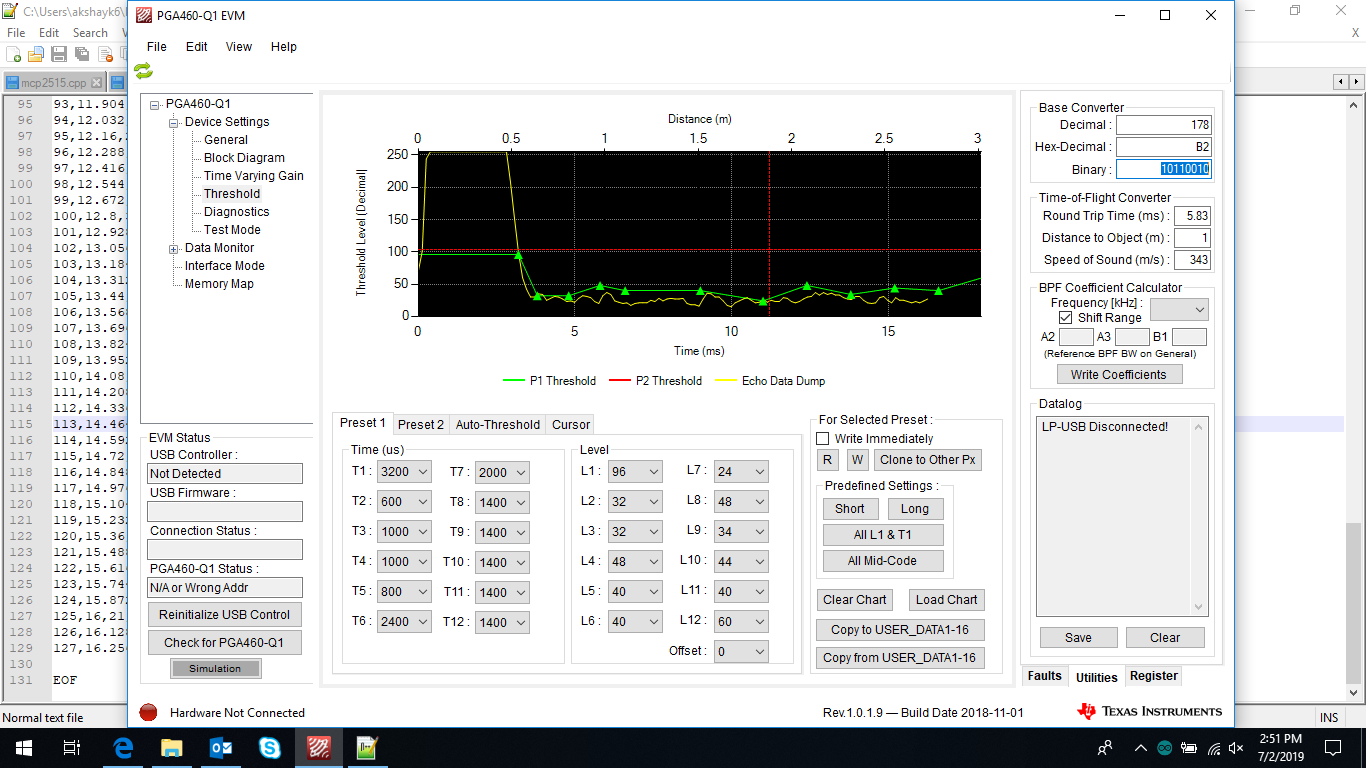
*THRESHOLD PARAMETERS*

Threshold parameters are stored on volatile memory of PGA460 and thus require to be re-written every time on power up in order for THR\_CRC\_ERR(see Register Write and Register Read section) check to be cleared (default to 1). Bursting will only be carried out after this check is cleared. Threshold parameters control the level at which the PGA460 is listening for and detecting reflected echo. If the levels are set too high, PGA460 may not recognize object detected if the raw echo data signals are small. On the other hand, if the levels are too low, PGA460 may detect false positives.



*In this graph, Preset 1 Threshold values are set at mid-code (50%) levels while echo data dump, with no-object detection, reports low-amplitude signals. As a result, PGA460 may not report any object detected as the echo data dump must cross the threshold levels in order to be recognized as detected object*

To optimally modify threshold values to prevent false positives and zero object detection, it is recommended that the threshold values should be set as close to the echo data dump base as possible while leaving some buffer for noise. The most efficient way to adjust the levels is use the [PGA460 EVM GUI](http://www.ti.com/tool/BOOSTXL-PGA460) Load Chart function, under Threshold tab, to better visualize the data and make appropriate modifications. The echo data dump can be retrieved from running pullEchoDataDump function and be reformatted into a file the GUI can import. The template can be found in Echo Data Dump Folder along with an Excel file that reorganizes the echo data dump. After the levels are visually

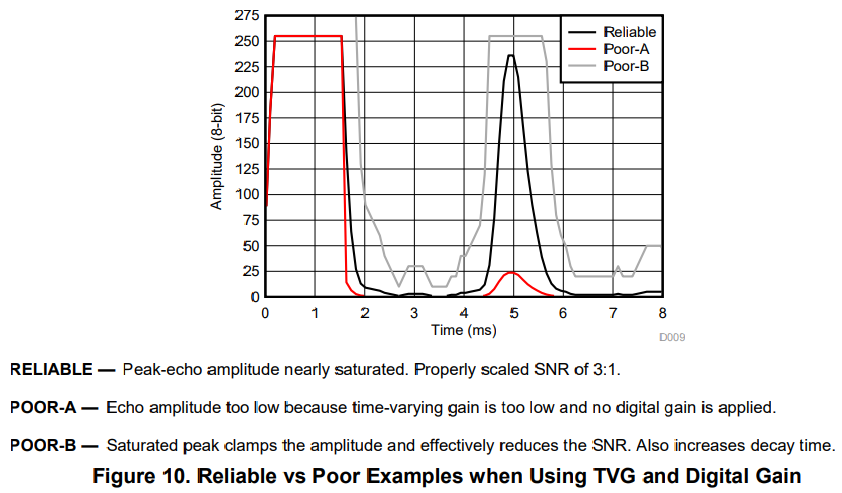


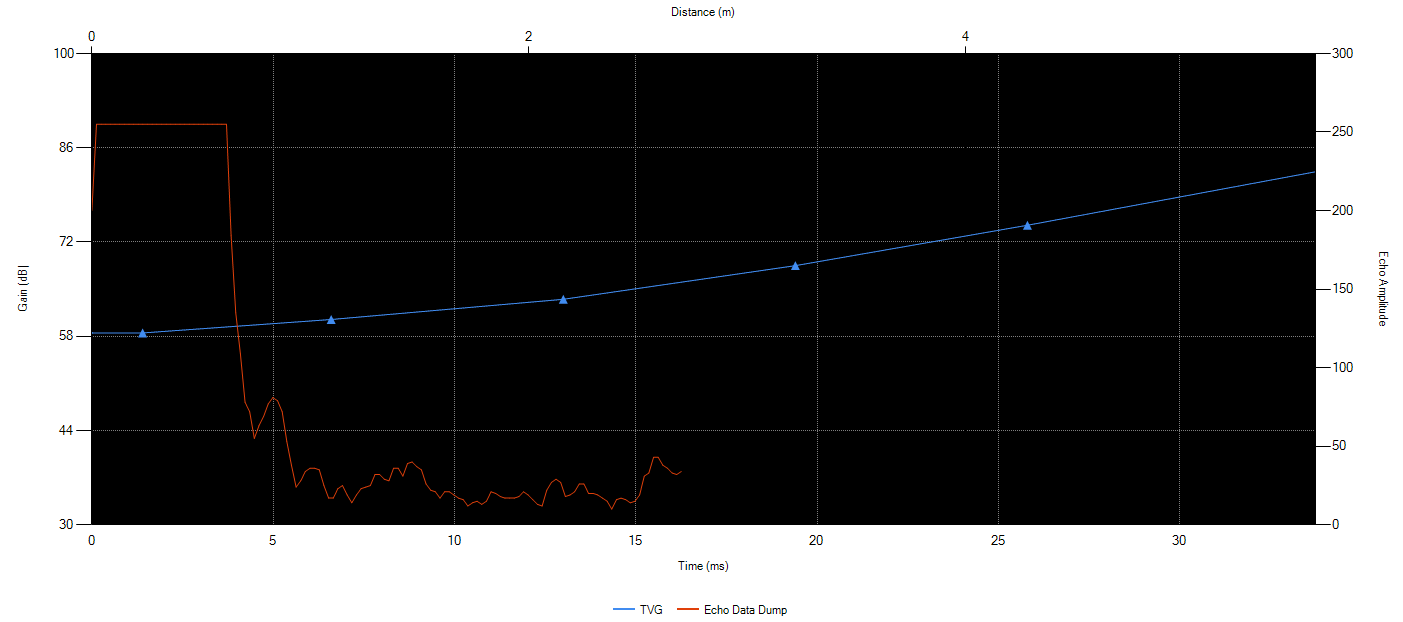
*The graph now shows the threshold values reconfigured to follow the base of echo data dump as closely as possible. As a warning, this level is susceptible to false positives with changing environments, but if the noise-level is known to be constant, this type of configuration will reliably detect objects.*

*ANALOG FRONT-END(AFE) and TIME VARYING GAINS(TVG)*

PGA460 has several built-in filtering stages for reflected signals, of which the first stage is the analog-front end gain (AFE), followed by time-varying gain (TVG). AFE gain filters the returned echo through a low-noise balanced amplifier with an initially fixed initial gain. There are four different ranges of AFE gain: 32-64dB, 46-78dB, 52-84dB, and 58-90dB, and the chosen range should be based on the overall amplitude of the received signals (i.e. if amplitude of received signals is already high, a lower range of AFE gain should be selected).

The amplified signals are then passed into a time-varying gain filter (TVG) to allow for uniform amplification of returned echo from different distances. It is suggested that TVG gain should be configured to saturate the peack echo without truncating the peak. In addition, TVG gain should increase non-linearly to correspond with attenuation of sound. PGA460 GUI EVM can be used to import echo data dump and visually help adjust TVG levels accordingly.





***GetDistance Script***

The sequence of which to extract distance (m) from PGA460 is:

1. Burst and Listen
2. Pull data measurement
3. Compute distance from data measurement

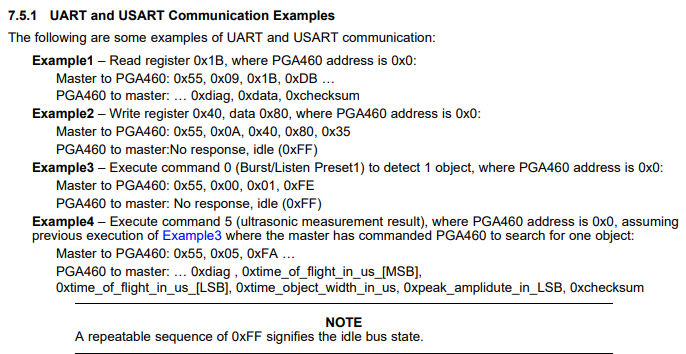
BURST AND LISTEN command, depending on Preset 1 or Preset 2 mode, will issue a corresponding number of bursts (PULSE\_1 and PULSE\_2), followed by an uninterrupted recording period (REC\_LENGTH) for reflectecd signals. See ultrasonicCmd function in Energia library for further details.

PULL DATA MEASUREMENT command will collect the filtered signals and store them in an array called ultraMeasResult, capable of storing information of up to eight detections from a single burst and listen. See pullUltrasonicMeas function in Energia library for further details.

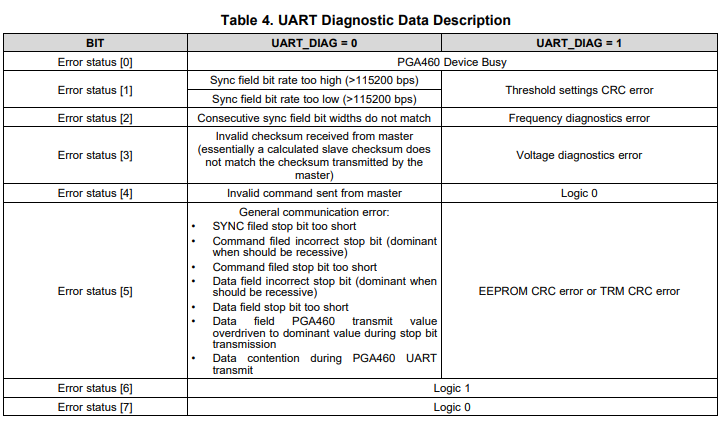
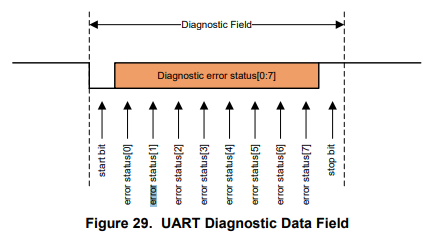
COMPUTE DISTANCE command will use raw data in ultraMeasResult array to return a distance in meter. For each object detection, there is a set of 5 bytes, describing distance, width, and peak amplitude. Since we are only concerned with distance, the 2nd and 3rd bytes will be used in time-of-flight calculation. See printUltraMeasResult funciton in Energia library for further details.

*Time-of-flight calculation for distance - PGA460 Software Development Guide 2.4.1*

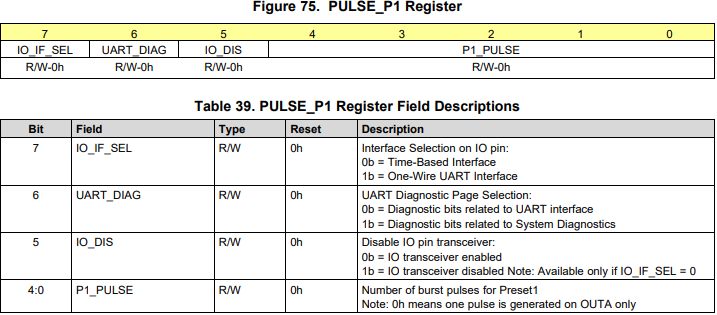
**RELATED USEFUL FUNCTIONS**

***Register Write and Register Read***

Register write and register read are used for committing a value to a single address and checking the value at that address, respectively. Register read has a critical use in verifying appropriate values at specific addresses and resolving errors. Register addresses for all configuration parameters can found in [PGA460 Ultrasonic Signal Processor and Transducer Driver](http://www.ti.com/lit/ds/symlink/pga460.pdf) under Section 7.6 Register Maps. On a hexadecimal level, the value may not describe the register’s complete functions and will require binary conversion to understand its bit-wise functions, which is detailed in the binary mapping of each register. Displayed to the right are examples of using and understanding register write and register read.

***Diagnostic Field***

Every transmission PGA460 sends to microcontroller will start with a diagnostic byte. The diagnostic byte is critical in detecting successful communication and troubleshooting serial communication when transmission fails. The diagnostic byte, if PGA460 is not interrupted by errors, is default to 0x40. In the case that the diagnostic byte is a different value, troubleshooting should start with determining whether UART\_DIAG is 0 or 1. UART\_DIAG bit is part of PULSE\_1 byte, whose register address is 0x1E; the byte value of PULSE\_1 can then be converted to binary to learn of the sixth bit’s value (UART\_DIAG). Knowing what the UART\_DIAG bit is, the diagnostic byte can be converted to binary for deeper understanding of the true problem.



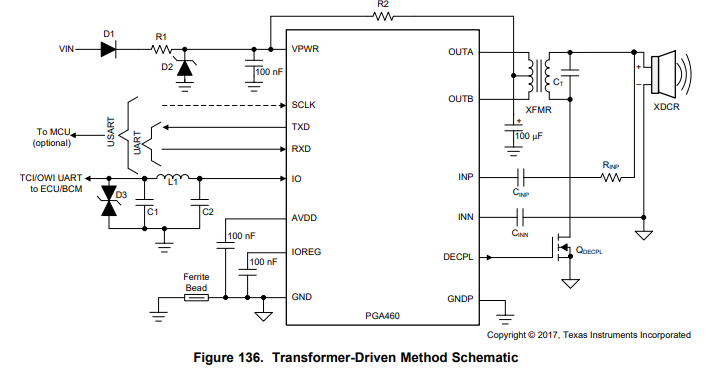
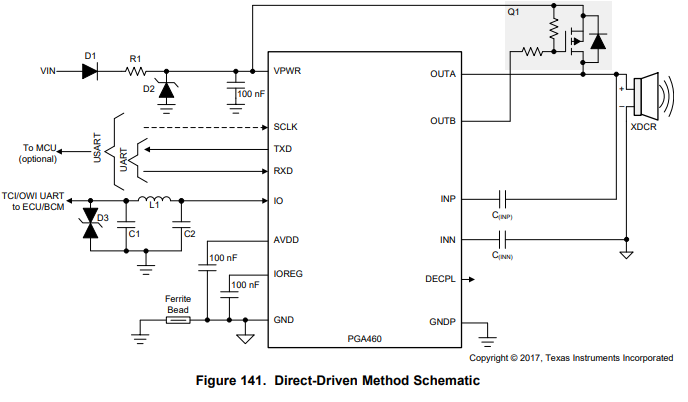
***Checksum Value***

The checksum value is the trailing byte of every transmission between the microcontroller and PGA460 to validate correct data transmission. The checksum value is calculated as an “inverted byte sum with carry operation over all data fields and command field” ([PGA460 Ultrasonic Signal Processor and Transducer Driver](http://www.ti.com/lit/ds/symlink/pga460.pdf), Section 7.3.6.2.1.4) – further investigation of the calculation can be explored in the calcChecksum function in the Energia Library. Depending on the amount of data transmitted per request, it is not expected that microcontroller and PGA460 will generate the same checksum value. In other words, it is NOT an error if the checksum value DOES NOT match up between the microcontroller’s and PGA460’s. Instead, developer should be alert when the amount of data transmitted does not correspond with the calculated checksum.

**TROUBLESHOOTING GUIDE**

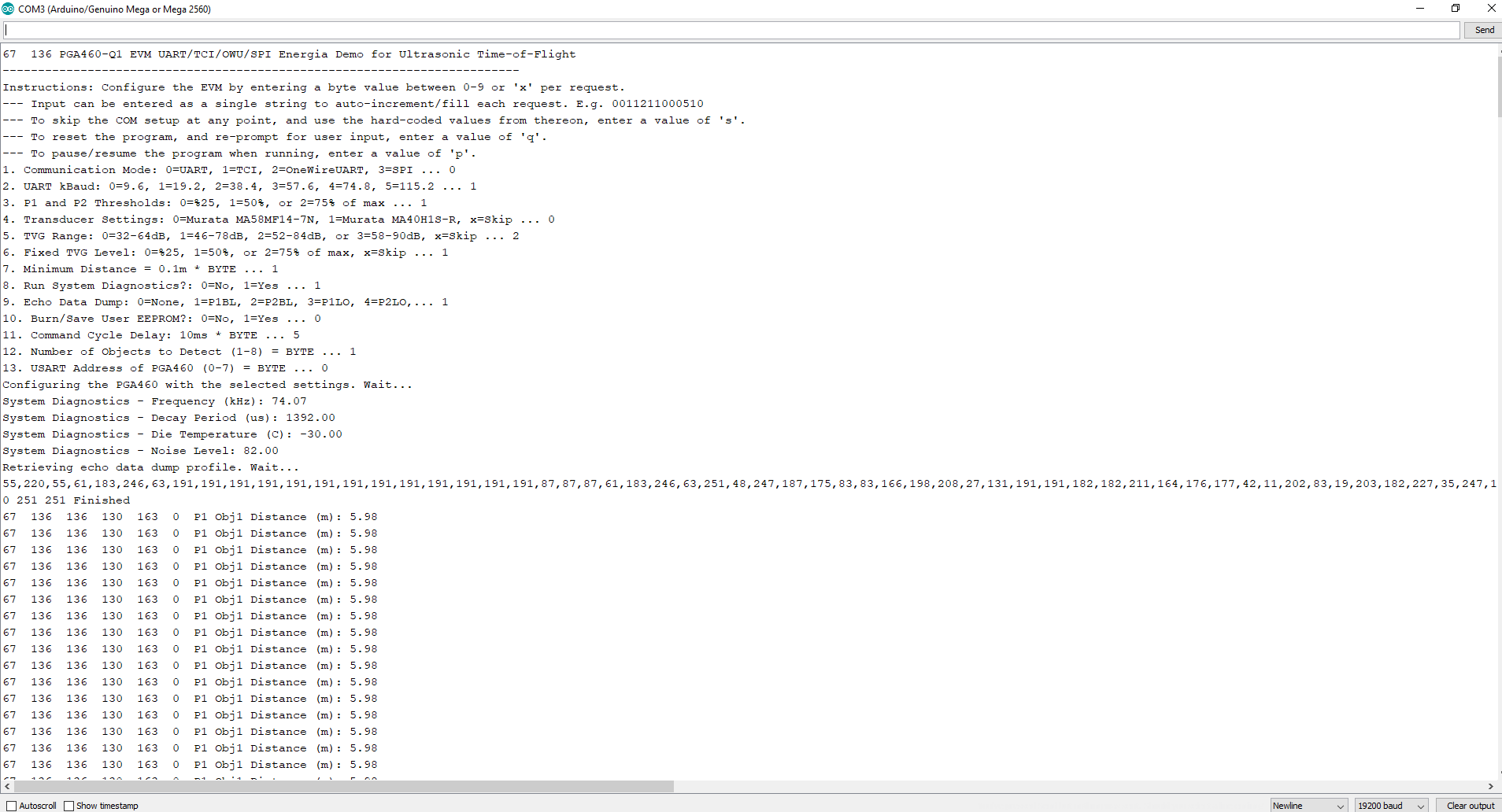
The following guide will describe various issues, observations, and experimentations encountered from working with Arduino Mega 2560, PGA460, and Murata ma58mf14-7n ultrasonic transducer. The majority of the problems have been resolved due a combination of community posts on PGA460 and [personal post](https://e2e.ti.com/support/sensors/f/1023/t/813297?tisearch=e2e-sitesearch&keymatch=%20user%3A411832) to head engineer of TI PGA460 Akeem Whitehead. Listed below are the problems that I have encountered:

1. Unsuccessful communication with PGA460 resulting in constant distance result
2. Increased accuracy with higher voltages
3. Increasing FPS
4. Flickering diagnostic byte
5. Inflexibility in switching serial ports
6. Same distance measurement from multiple transducers
7. UART vs SPI mode

***Unsuccessful communication with PGA460 resulting in constant distance measurement - RESOLVED***

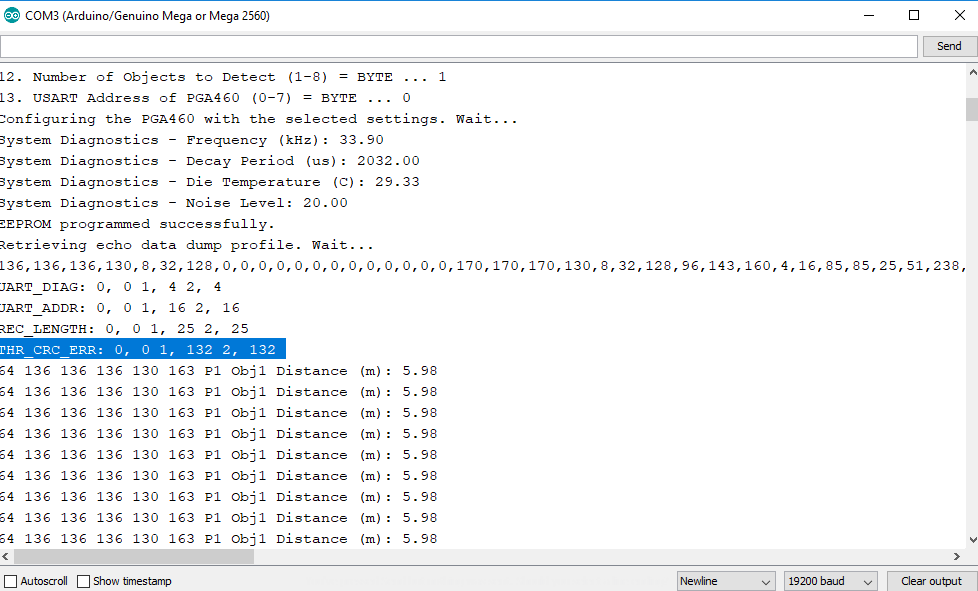
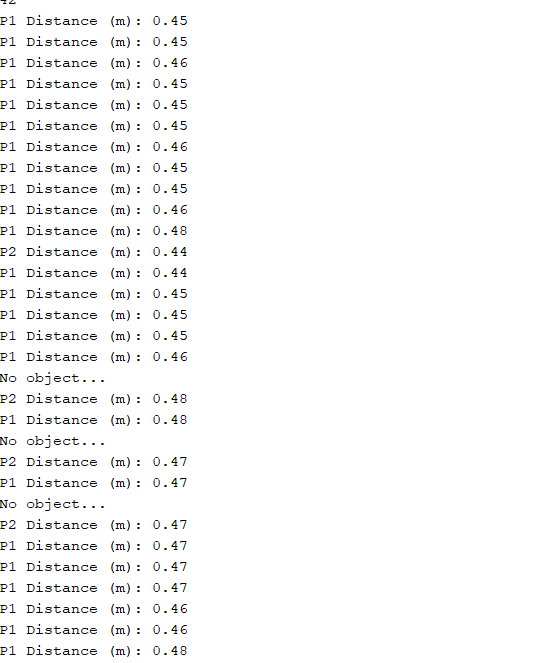
When the project was handed over to me, a transformer-driven schematic (see *29\_SmartSensor\_HW\_PGA ver 0.0.1\_PGA460 ver 0.0.6.sch* in Schematics folder) had already been developed and a PCB board with all related components soldered onto the board. The developed schematics exhibits several flaws, such as the transformer’s secondary coil not grounded and negative terminal of ultrasonic transducer not connected to INN pin, of which I had externally modified to resemble the suggested circuitry in [PGA460 Ultrasonic Signal Processor and Transducer Driver](http://www.ti.com/lit/ds/symlink/pga460.pdf) Section 8.1 without DECPL, IO, and TEST pins connected (optional). SCLK pin is pulled down to GND through a 10k resistor as it is not in use. This was theoretically a working system. I proceeded to compile a personally-developed script from the project’s preceding engineer. The script returned a constant distance and featured non-changing ultrasonic measurement result (UMR) bytes regardless of objects being present. The distance only changed when the power supply was switched on and off, but will remain constant after that.

Further study of [PGA460 Ultrasonic Signal Processor and Transducer Driver](http://www.ti.com/lit/ds/symlink/pga460.pdf)7.3.12showed that PGA460 operates on 3.3V logic while Arduino Mega 2560 operates on a 5V logic. A weak pull-up resistor to 5V on TEST pin will upgrade PGA460 to 5V logic. This feature was added but the same problem persisted. The manufacturer’s code Energia library (presumably working software) was compiled to confirm reliability in hardware, but the problem did not cease. With an oscilloscope, I’d probed OUTA and OUTB pins which showed no indication of bursting. INP and INN pins also showed no signs of receiving signals either. TXD pin was pulled low, while RXD pin was pulled high. All of these observations led me to conclude that the hardware needed to be redesigned on a breadboard setup for easier modifications.

 While the transformer TDK EPCOS B78416A2232A003 was in transit, I decided to experiment with direct-driven method as the components were readily available. Direct-driven circuitry only detects at maximum 1.3 meters, but our application calls for a longer range. I intended with direct-driven method to at least obtain accurate and reliable distance readings and transfer that progress to transformer-driven method later on. In ensuring reliability in software, the Energia library getDistance script is used onwards.

Direct-driven circuitry was built according to the suggestion in [PGA460 Ultrasonic Signal Processor and Transducer Driver](http://www.ti.com/lit/ds/symlink/pga460.pdf) Section 8.1, with BSS84 P-channel transistor. A more careful look at the diagnostic byte showed 67 DEC or 0x43 when it needed to be 64 DEC or 0x40. Analyzing 67 DEC on a binary level showed PGA460 operating on logic 1 and 0 bases simultaneously and therefore communicating the wrong information to the microcontroller. Further research [on community blogs](https://e2e.ti.com/support/sensors/f/1023/t/783905?BOOSTXL-PGA460-DIAG-byte-value-always-0x44-as-first-read-) indicated that the TXD should idle high, but observation from oscilloscope showed TXD idling low, which can occur due to the pin’s tri-state logic nature. Adding a 10k pull-up resistor resolved that incorrect diagnostic byte, which afterwards showed consistently 64 decimal or 0x40.

*Diagnostic byte of 67 DEC indicates both logic 0 and 1 being used.*



*RegisterRead for DEV\_STAT0 results in 132 DEC, which means THR\_CRC\_ERR is not cleared. A value of 128 DEC at DEV\_STAT0 means that THR\_CRC\_ERR is cleared and will enable bursting and therefore detection of objects.*

Even though serial communication was confirmed to be correct, the constant distance measurement did not get resolved. Probing OUTA and OUTB showed no indication of bursting. A note from Akeem Whitehead hinted checking for THR\_CRC\_ERR check to be cleared (set to 0), which would then enable OUTA and OUTB to burst. Afterwards, whenever THR\_CRC\_ERR is not cleared, constant distance measurement problem would occur, and vice versa. In ensuring the THR\_CRC\_ERR is cleared every time, make sure all wires connections to PGA460 are stable and secured and have a Serial.print statement in the function where initiating thresholds is called. Correct distance measurements start to consistently display.

SOLUTIONS: TXD idling high, THR\_CRC\_ERR cleared and set to 0 to enable bursting.

***Increased accuracy with higher voltages – RESOLVED***

Design parameters recommend optimal power supply of 7.4V for PGA460. This voltage is also supplied directly to GATE of BSS84 transistor. At this level, it is observed that the range of the detection is from 0.25m – 0.50 m, with sparse accurate reading after 0.50m. Increasing the voltage to about 12V increase the range of detection to 0.25m – 0.80m, again with sparse accurate readings after 0.80m. When the power supply is increased to 18V, objects can be detected very accurately and consistently from 0.30m – 1m. The increase in accuracy with higher voltages can be explained as more power delivered in each burst causing the reflected echo to have a higher amplitude for easier detection.

SOLUTIONS: For direct-driven model, higher VPWR 🡪 longer range, higher accuracy

***Increasing FPS***

The default GetDistance script from Enegia library set delays that extend for maximum amount of time and numerous if-else statements to accommodate any customization in EEPROM parameters. However, knowing which mode of communication will be used and maximum recording period is, developers can remove the majority of if-else statements and shorten delays to one-fifths of allocated value. Reducing the number of Serial.print statements to a minimum will also result a more accurate diagnostic of cycle frequency. Following these steps, the modified GetDistance\_v10 (in PGA460, 0.3-0.7m, 26Hz folder) script cycles at minimum 20Hz and maximum 40Hz, compared to 4Hz-6Hz of the Energia library’s GetDistance script.

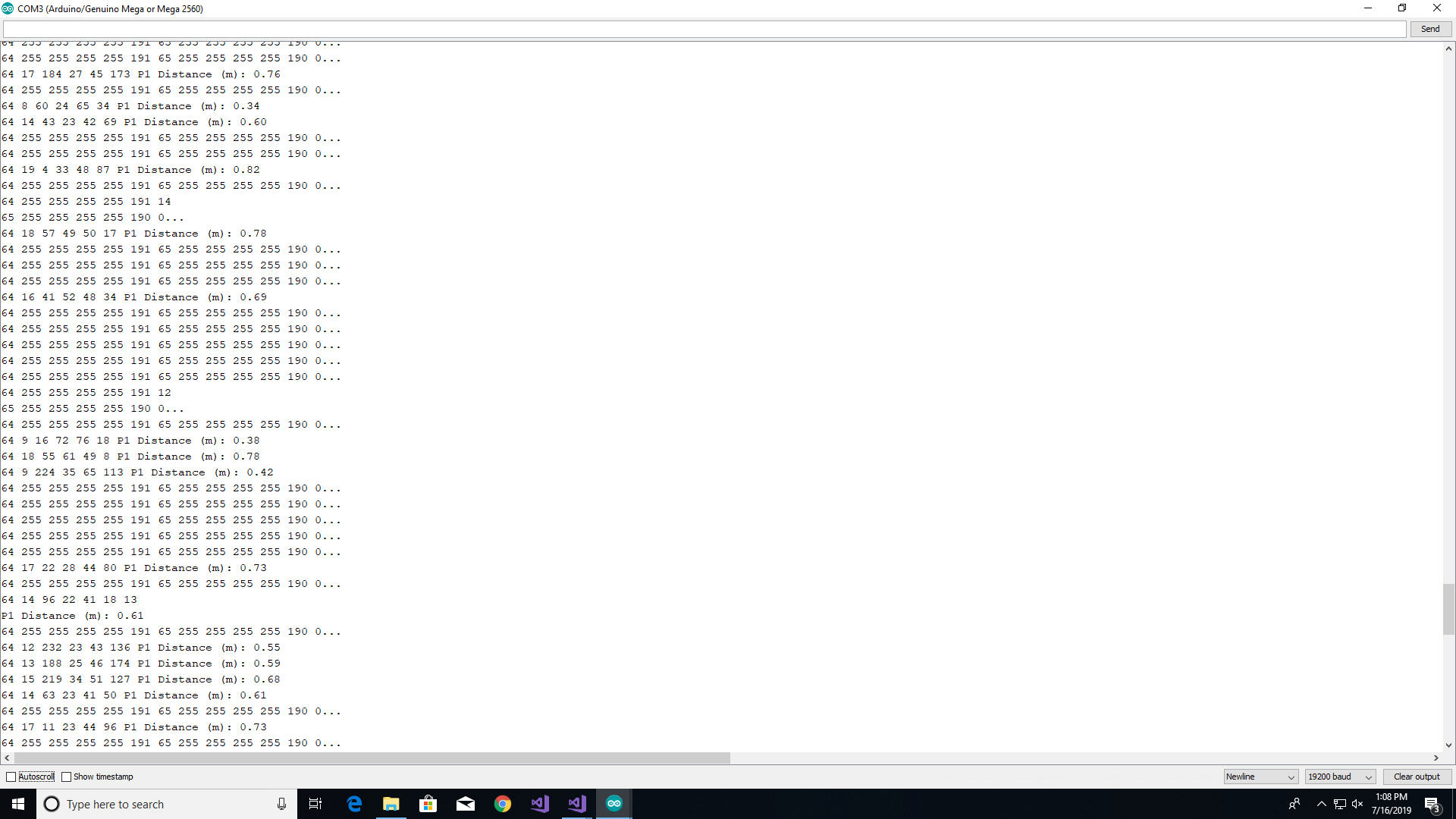
SOLUTIONS: To increase fps, remove unnecessary if-else statements and shorten delays as much as possible

**INSERT IMAGE**

***Flickering diagnostic byte - RESOLVED***

After increasing GetDistance fps to maximum of 40Hz, there were intermittent zero object detection instances in the middle of accurate distance readings chain. I attempted to Serial.print the UMRData bytes again and found the diagnostic byte flickering between 64 and 65 DEC. Retracing my steps in shortening the delays, I discovered the record period, which was 10ms, is long enough for Preset 1, but not long enough for Preset 2. The order of which script runs to obtain a distance reading is to first Burst+Listen for Preset 1, and if no object detected, the script moves on to Burst+Listen for Preset 2. Therefore, the common delay used for both presets caused the diagnostic byte to switch to 65 DEC for Preset 2, which means PGA460 is busy.

SOLUTIONS: Have two different delays, each for Preset 1and Preset 2

******

*Notice diagnostic byte switches to 65 DEC for Preset 2 after being 64 DEC for Preset 1.*

***Inflexibility in switching serial ports - RESOLVED***

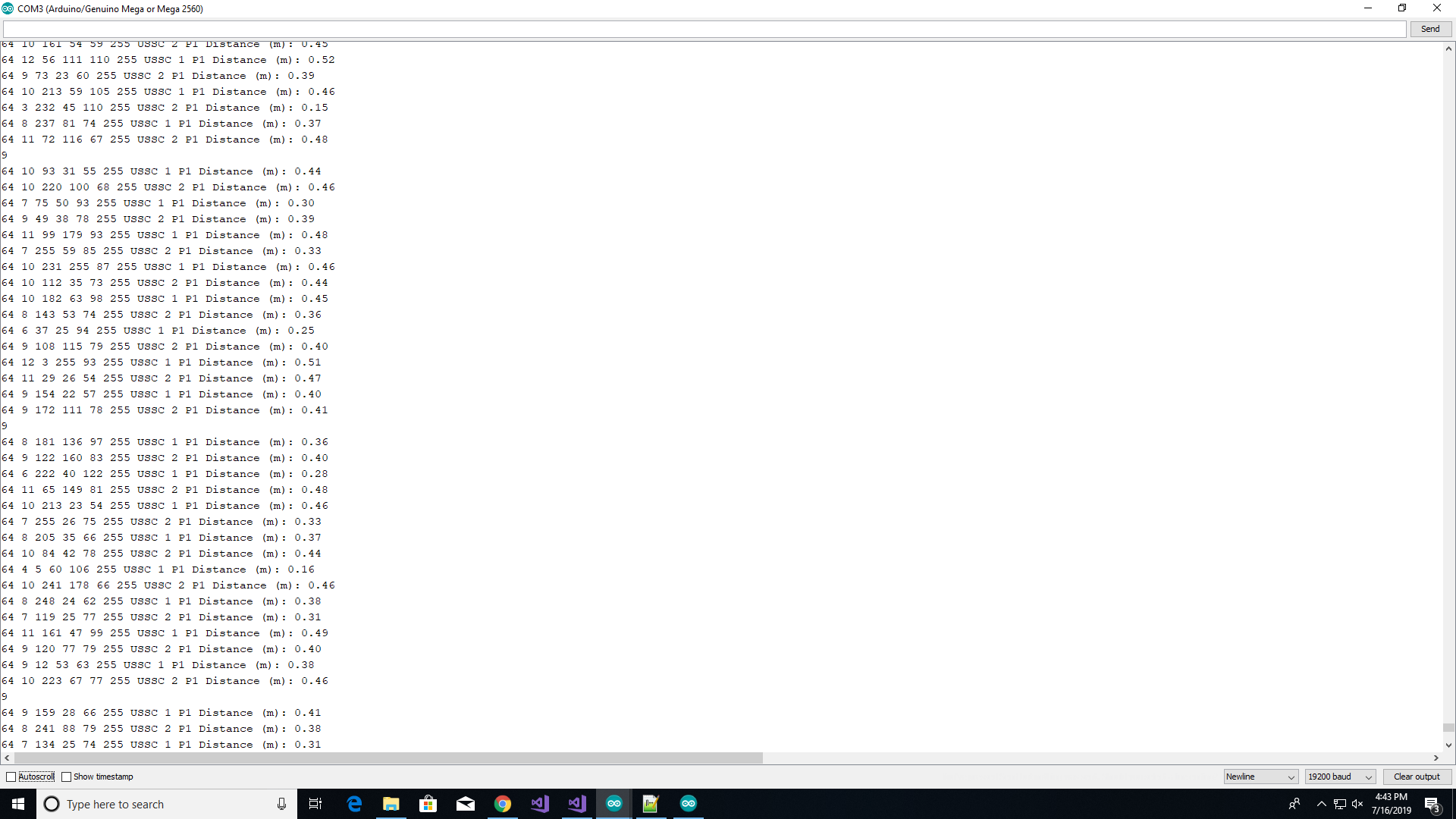
The Energia library predetermines the serial port communication to Serial1 for every transmission from microcontroller to PGA460. This hard-coded HardwareSerial makes it difficult to interface with multiple PGA460s on the same microcontroller. I wanted an option to select which serial port to use and reference this selection in the library code. I then created a global pointer HardwareSerial\* pgaSerial in the Energia library and assign this pointer the serial port I want to use when I first instantiated pga460 object. For more information, review the .ino, .h, and .cpp files in PGA460, 0.3 -0.7m, 26Hz, Flexible Serial folder.

SOLUTIONS: Pass in selected serial port when first instantiated

***Same distance measurement for multiple transducers - RESOLVED***

I had built two direct-driven topography for two pairs of PGA460 and Murata ultrasonic transducer connected to the same microcontroller. Since I wanted to stimulate Burst+Listen on two PGA460s at approximately the same time, I divided up the sequence of extracting distance. I first burst+listen PGA460s consecutively then pulled and printed distance measurement individually. This method resulted in both PGA460s reporting the same UMRData bytes and therefore same distance readings. The cause originated from the UMRData array being a global object and how before pulling distance measurement, the array would be cleared everytime.

SOLUTIONS: Complete cycle of burst+listen, pull, print data for each PGA460 before moving on next PGA460. Disadvantage is shorten fps to 9Hz.



*Two PGA460s reported back different distance readings at 9FPS.*

***SPI mode - IMPENDING***

The final objective of this project is to burst+listen multiple transducers at the same time with minimum fps of 20 Hz. SPI mode on PGA460 allows for broadcast burst+listen command, which will set all PGA460s on the SPI bus to burst+listen at the same time. The major problem of using this method is differentiating the signals An explanation from Akeem Whitehead says differentiation is not possible unless transducers of different frequencies are used.