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.

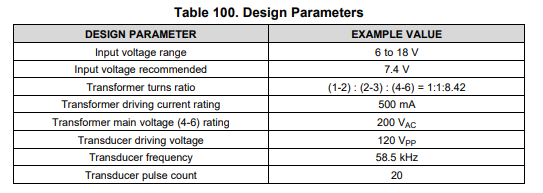
PGA460

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:

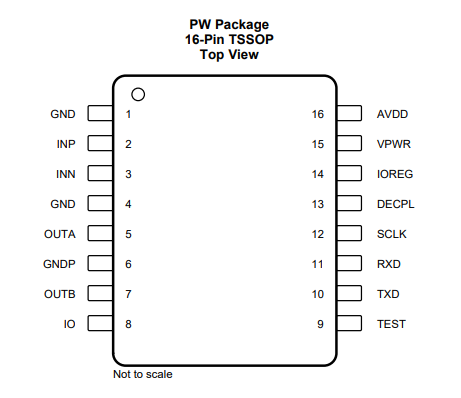
* + Driving multiple sensors with one transducer (Multiplexing)
  + Transmitting distance results back to microcontroller
  + Embedded system to run at rate more than 4Hz

Initialization of PGA460

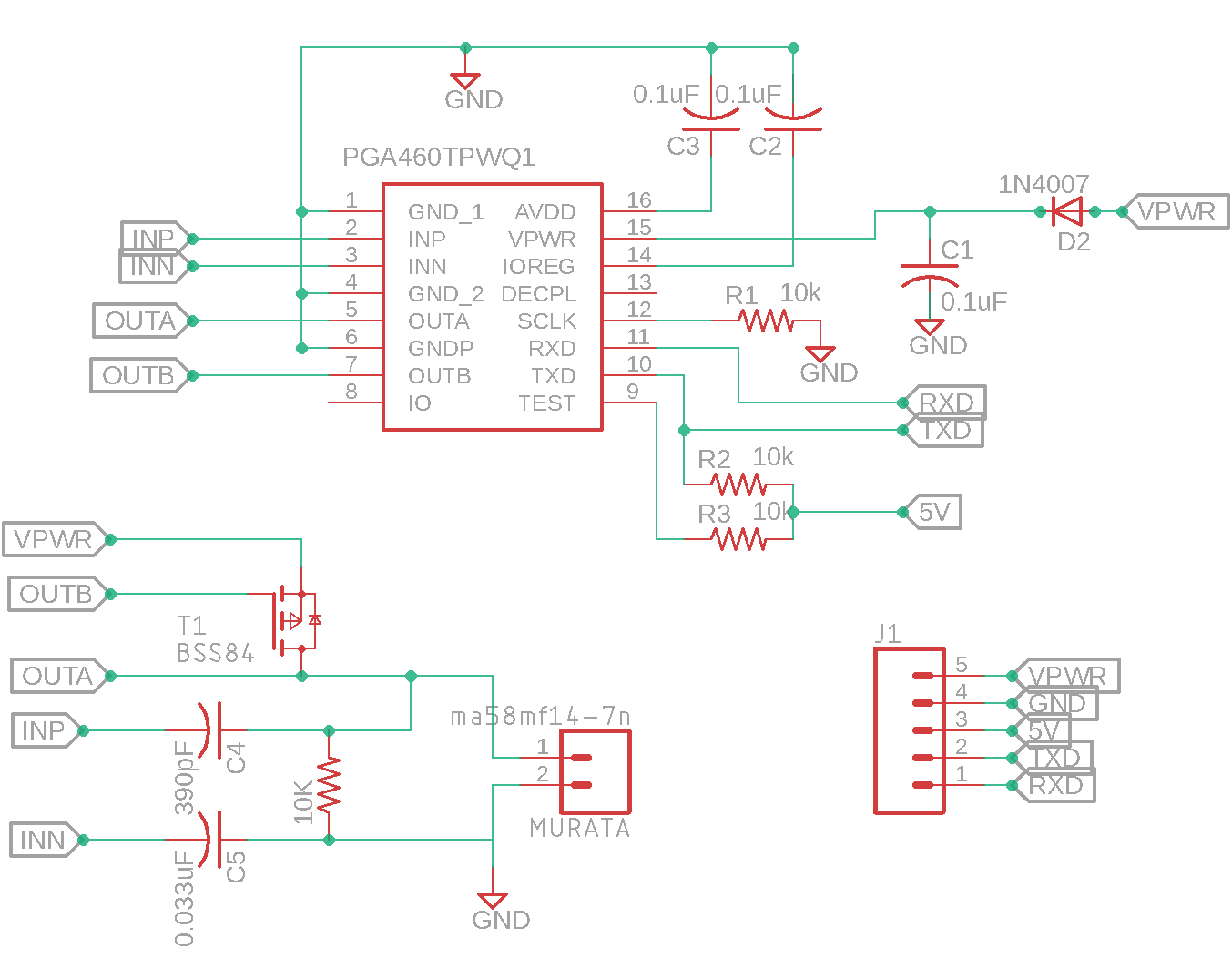
* Register configuration settings for threshold values
* Upon power boot or power cycle, commit configuration settings

Hardware Wiring

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](:%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.

***Setup Instructions***

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)*

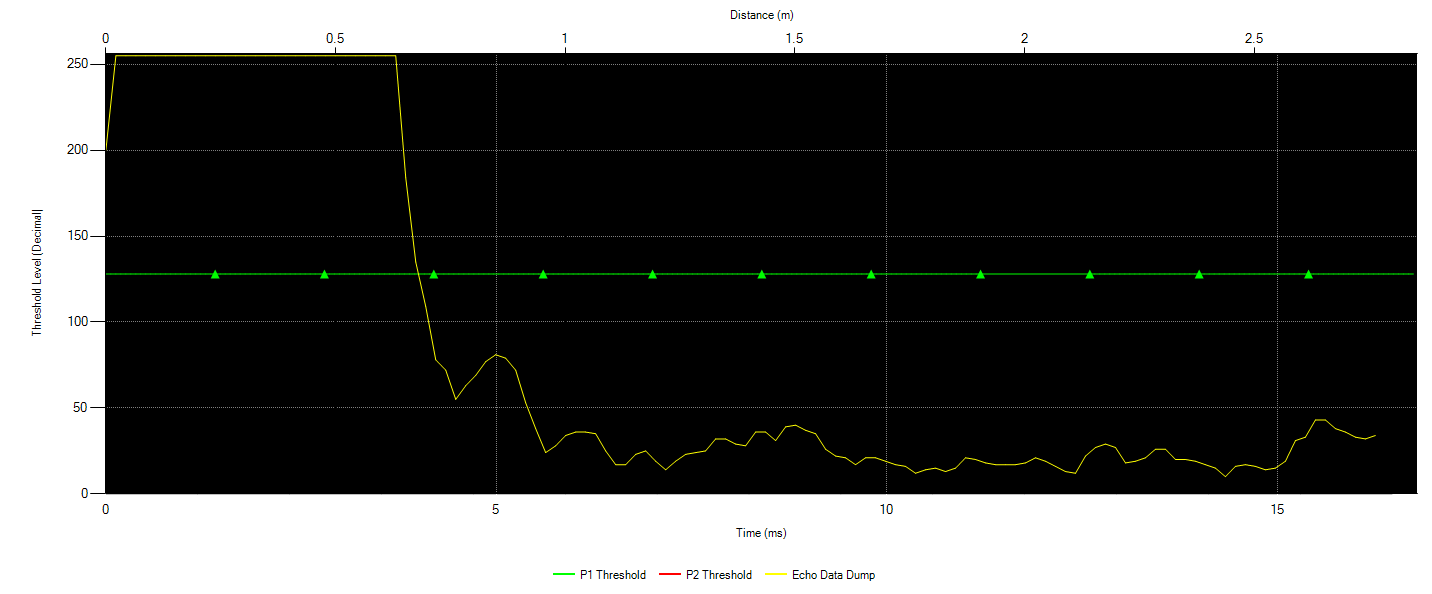
EEPROM is a non-volatile memory that contains parameters under which PGA460 recognizes type of transducer is being used and accordingly carries out its commands. The following parameters affect system diagnostics, burst and listen patterns, and filtering configurations:

* 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.

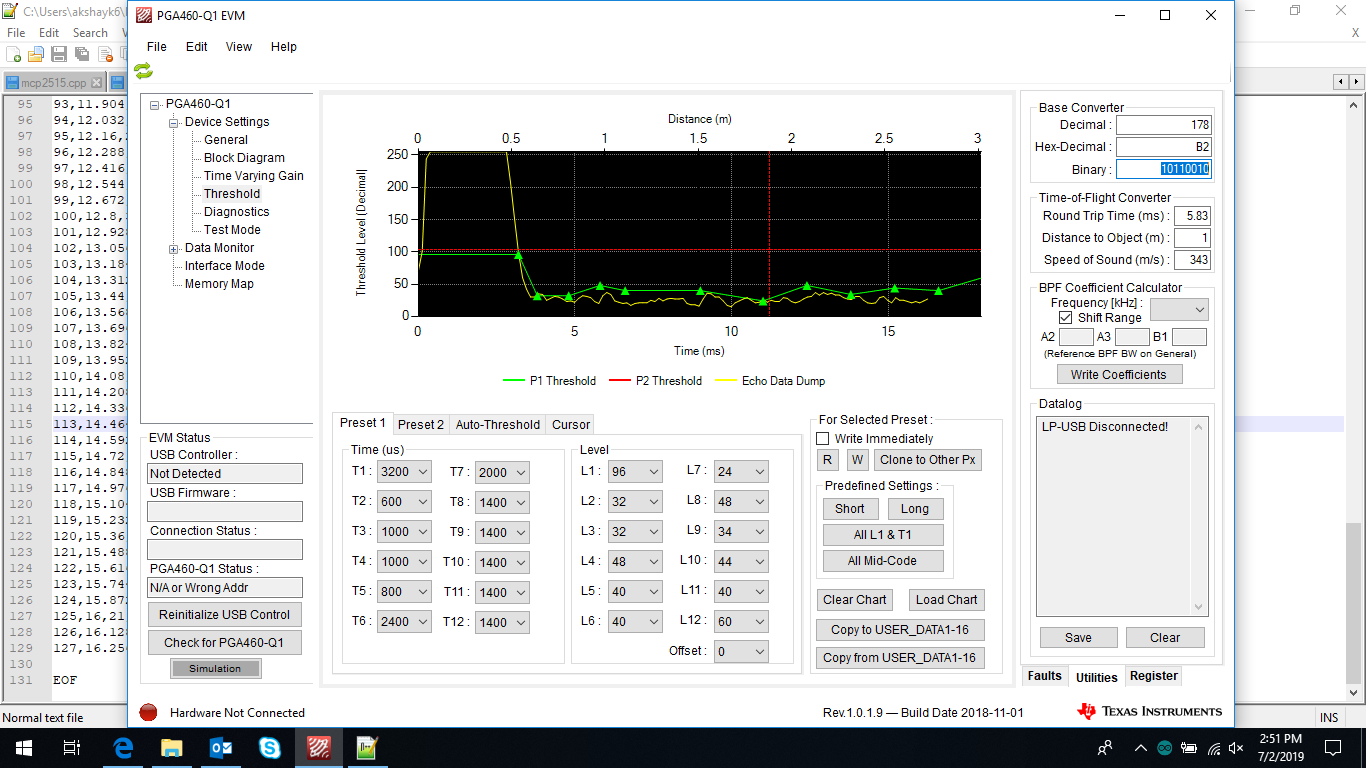
*THRESHOLD PARAMETERS*

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 check to be cleared (default as 0). 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

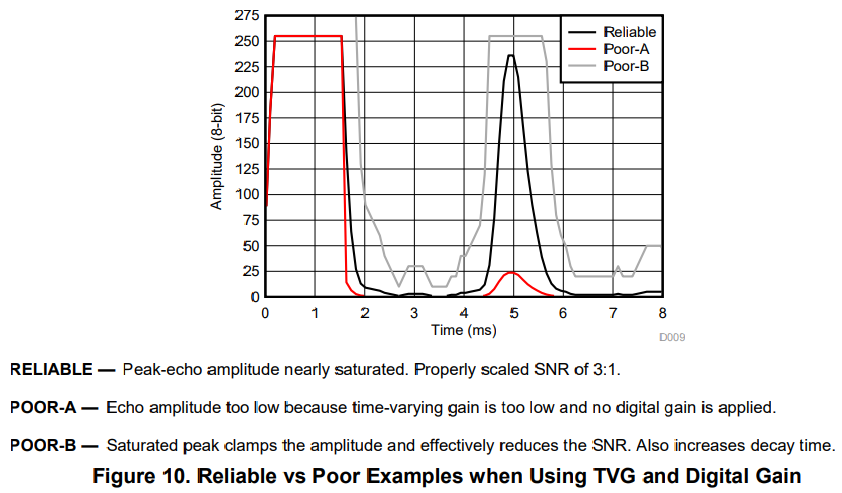


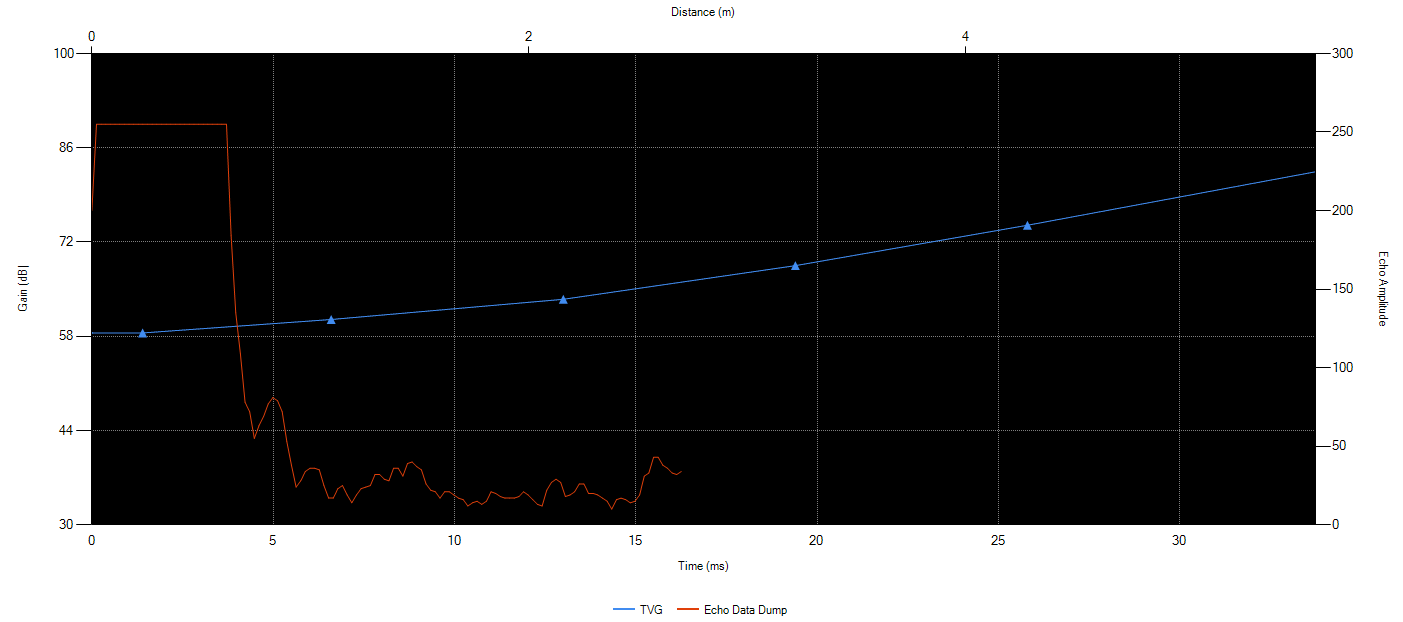
*The graph now shows the threshold values reconfigured to follow the base of echo data dump as closely as possible. As a warning, this level is susceptible to false positives with changing environments, but if the noise-level is known to be constant, this type of configuration will reliably detect objects.*

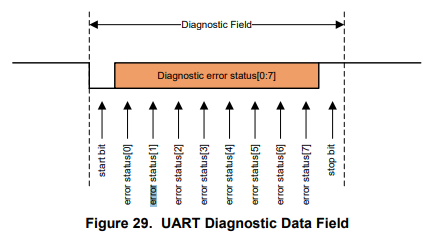
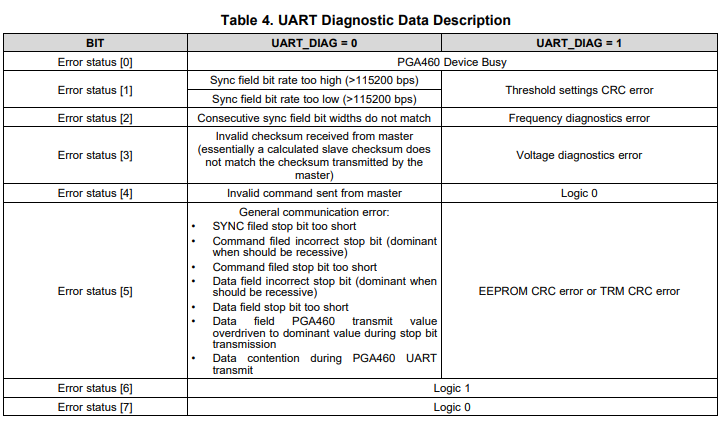
*ANALOG FRONT-END(AFE) and TIME VARYING GAINS(TVG)*

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).

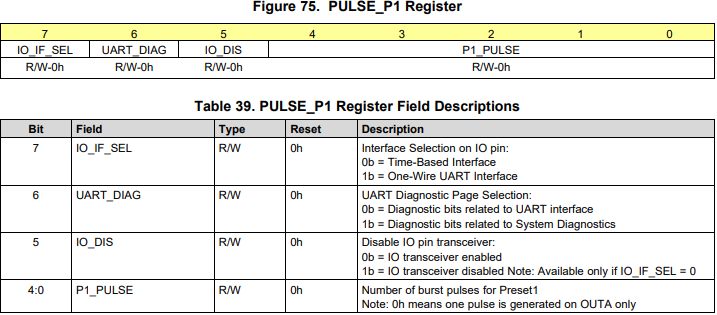
The amplified signals are then passed into a time-varying gain filter (TVG) to allow for uniform amplification of returned echo from different distances. It is suggested that TVG gain should be configured to saturate the peack echo without truncating the peak. In addition, TVG gain should increase non-linearly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





***Diagnostic Field and***

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 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, discrepancy should be recognized when the amount of data transmitted does not correspond with calculated checksum.