10 Thomas, Irvine, CA 92618 USA
Email: futek@futek.com

QIA135 SPI Communication Guide

Table of Contents

General Description

The *QIA135* is a six-channel digital controller with USB/UART and SPI outputs. The *QIA135* (Client device) can be used to communicate with any Host devices through an *SPI* bus.

Pin Configuration and Functional Description for *QIA135*

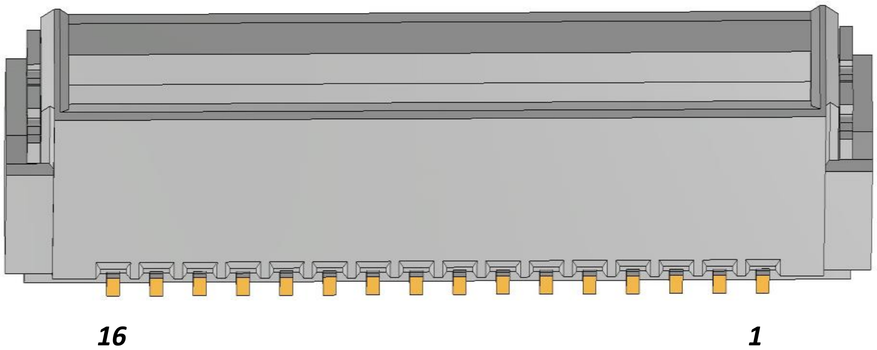

Figure 1

Table 1

	Pin	Description
1	***VIN***	Voltage input *5V±10%*
2	***GND***	Ground pins are connected to each other internally
3	***SCLK***	Serial clock generated by Host
4	***HICO***	Host-In-Client-Out
5	***HOCI***	Host-Out-Client-In
6	$\overline{CS}$	Active low chip-select. Do not drive the $\overline{CS}$ line low until the device has booted up completely. Also ensure that the $\overline{CS}$ line is not driven low unless the $\overline{DRDY}$ is low.
7	$\overline{DRDY}$	Active low $\overline{DRDY}$ pin is used to keep all communication synchronized. It notifies the Host device when new data from the sampling system is ready. This ensures that the Host is always collecting the latest data. When the $\overline{DRDY}$ pin goes low, it indicates that the data is ready to be clocked out. This pin can be used to externally interrupt the Host. The pin returns high when the system is in a conversion state and returns low once new data is ready.

		***Note:** *The pin does not return high once data is read—it will only return high once the system enters a conversion state.*
8	***GND***	Ground pins are connected to each other internally
9	***UART-RX***	Client RX
10	***UART-TX***	Client TX
11	***GND***	Ground pins are connected to each other internally
12	***TRIG***	Trigger is an input pin (Client) and output pin (Host) dedicated for special applications such as programing for future development.
13	***ICSP MCLR***	Programming Pin
14	***ICSP VOLTAGE (3.3V)***	Programming Pin
15	***ICSP DAT***	Programming Pin
16	***ICSP CLK***	Programming Pin

QIA135 SPI Configuration

Table 2.

Serial Word Length	8-Bit			
SPI Mode	*Mode 0 (CPOL = 0, CPHA = 0)*			
SCLK Frequency	**Min**	*2 MHz*	**Max**	*12.5 MHz*
Internal Clock Frequency of MCU	*128 MHz*			
Operation Mode	Client			
Voltage Level	3.3 VDC			

QIA135 Internal Design Algorithm

When the $\overline{DRDY}$ pin goes high, it means the device is in the process of A/D conversion, calculating the CRC16 (See ***CRC Calculations and References***) and generating the packet that needs to be sent per the Host device's request. The $\overline{DRDY}$ pin goes low as soon as it fills out the SPI TX buffer. The following algorithm is being executed while $\overline{DRDY}$ is high:

- Receives the latest ADC data from the highest interrupt priority
- Client Service Function
  - Keeps reading the *RX FIFO* until it is empty
  - Saves all the bytes in a software buffer
  - If the buffer is empty, creates a mock-up default command to go to the default state
  - Checks the *CRC16* byte, *CMD* byte, system health and board temperature
    - If either the *CRC16* or the *CMD are* incorrect
      - Goes to the default state
    - Else
      - Replies with the corresponding packet (See ***Table 7***.)
    - Default State:
      - Restarts the *SPI* module
      - Calculates the *CRC16*
      - Loads 7 bytes of data (including the *Error Code* byte, zeros for payloads and the *CRC16* bytes) into the *TX FIFO* buffer

- $\overline{DRDY}$ goes low

It is important to note that when a packet is clocked into the QIA135 via the HOCI line, the response to that packet must be clocked out in the very next $\overline{DRDY}$ period. If it is not clocked out in the next $\overline{DRDY}$ period, the response will be lost, and the system will go back to clocking out zeros.

SPI Packet Structure

The packet structure stays consistent during all transactions and always includes seven bytes of data for both receiving and transmitting. The first byte (Byte 0) is dedicated for the *Error Code* (See ***Table 5.***); in other words, if the system receives a packet with a wrong *CRC* or an undefined *CMD*, it will return an *Error Code* in the first byte followed by zeros for payloads and the *CRC* for the entire packet. The first byte of the packet (*Error Code*) is used to acknowledge the command response when the packet is sent properly. Refer to the Command Set List (See ***Table 7.***)

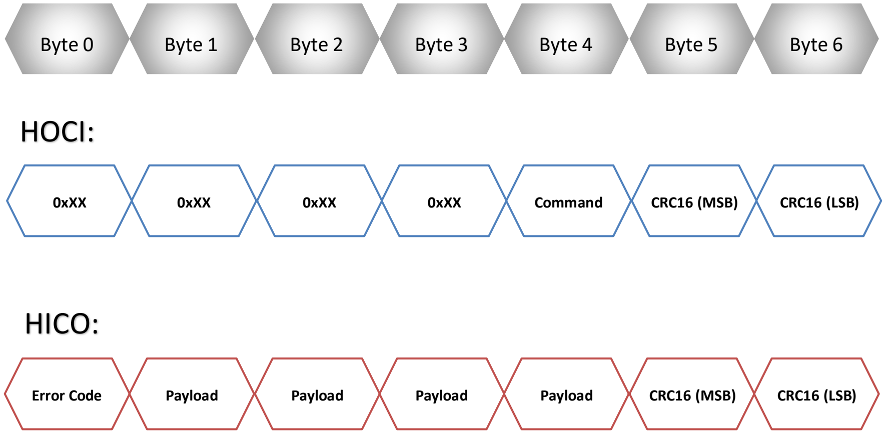

Figure 2.

****Note**: Each word (8-bits) can be clocked out with or without delay, but the entire transaction must be completed within a single $\overline{DRDY}$ period.*

"Continuous Read" Mode

GADCx command may be sent for each $\overline{DRDY}$ period to continuously get the *ADC* data.

****Note:** If the CRC bytes or the CMD bytes are incorrect, the device still fills out the buffer with the Error Code, zeros for payload followed by the CRC16.*

Timing Diagrams

Packet Structure (Get ADCx Data):

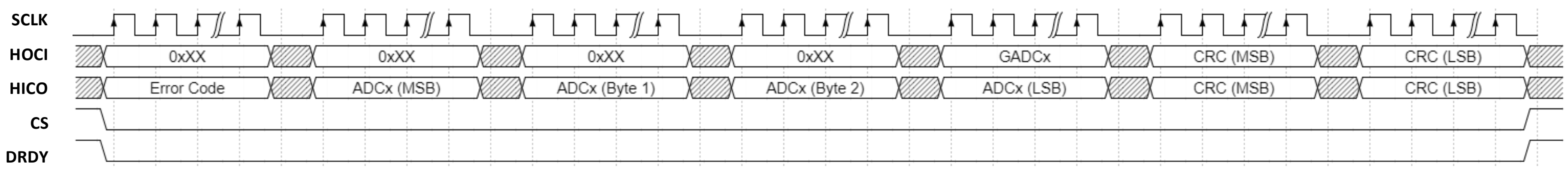

Figure 3.

***Note:** Each clock in **Figure 3.** represents 8-bits.*

$\overline{DRDY}$ Period

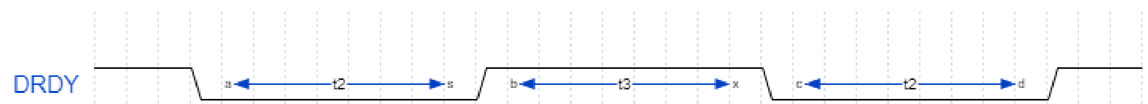

Figure 4.

Table 3.

<table>
<tr><th rowspan="2">t_1 (μs)</th><th rowspan="2">t_2 (ms)</th><th colspan="2">t_3 (μs)</th><th rowspan="2">Description</th></tr>
<tr><th>Min</th><th>Max</th></tr>
<tr><td rowspan="10">0 to …*</td><td>210</td><td rowspan="10">50</td><td rowspan="10">80</td><td>5 SPS</td></tr>
<tr><td>130</td><td>7 SPS</td></tr>
<tr><td>98</td><td>10 SPS</td></tr>
<tr><td>19.6</td><td>50 SPS</td></tr>
<tr><td>16.4</td><td>60 SPS</td></tr>
<tr><td>6.5</td><td>150 SPS</td></tr>
<tr><td>3.2</td><td>300 SPS</td></tr>
<tr><td>0.96</td><td>1000 SPS</td></tr>
<tr><td>0.34</td><td>2400 SPS</td></tr>
<tr><td>0.14</td><td>4800 SPS</td></tr>
</table>

***Note:** No delay or any delay as long as all 15 bytes are clocked out prior to $\overline{DRDY}$ going high. (See t_2)*

System Behavior

Start-up

When the system powers *ON*, it starts reading data from the *EEPROM* and the green solid LED lights up; this represents normal operation mode.

***Note:** Do not drive the $\overline{CS}$ line low until the device has booted up completely. Also ensure that the $\overline{CS}$ line is not driven low unless the $\overline{DRDY}$ is low. The $\overline{DRDY}$ line goes low as soon as the first data is ready to be clocked out.*

Wrong CRC Error

When bit "0" of the error code is set (See ***Table 4***.), it means a that the system has received a packet with an incorrect *CRC*, the system will then go to the default state that replies with the first byte of the packet as an Error Code followed by zeros for payload and *CRC16*.

Wrong CMD Error

When bit "1" of the error code is set (See ***Table 4***.), it means a that the system has received a packet with an undefined command, the system will then go to the default state that replies with the first byte of the packet as an Error Code followed by zeros for payload and *CRC16*.

System Health Error

When bit "2" of the error code is set (See ***Table 4***.), it means one of the followings has happened:

- At least one of the channels has been disconnected (open circuit)
- At least one of the channels has been shorted ($+EXC$ to GND)

Temperature Error

When bit "3" of the error code is set (See ***Table 4***.), it means that the board temperature is out of the defined range ($-30\ °C - 80\ °C$).

***Note:** A red LED will turn on to indicate the temperature error.*

Error Codes

The following table indicates the bit allocations for the error status:

Table 4.

Error Code Byte							
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
0	0	0	0	Board Temperature Reading Error Status	System Health Error Status	Command Error Status	CRC Error Status

Table 5.

Error Code Bit Array	
Error Type	**Command**
No Error	0 0 0 0 0 0 0 0
CRC Error	0 0 0 0 0 0 0 **1**
Command Error	0 0 0 0 0 0 **1** 0
System Health Error	0 0 0 0 0 **1** 0 0
Board Temperature Error	0 0 0 0 **1** 0 0 0

****Note**: Error code byte can contain the combination of multiple errors. For example, 0x09 carries the CRC and board temperature reading errors.*

Sampling Rate Change

When a sampling rate change is requested, it will take certain amount of time (depends on the requested sampling rate) to see the change in the $\overline{DRDY}$ period. *(See* ***Table 6.****)*

Table 6.

Approximate Data Rate Change Timing (s)	SR Code	Sampling Rate
2	0x00	5 SPS
1.4	0x01	7 SPS
1	0x02	10 SPS
0.3	0x03	50 SPS
0.18	0x04	60 SPS
0.08	0x05	150 SPS
0.04	0x06	300 SPS
0.015	0x07	1000 SPS
0.005	0x08	2400 SPS
0.003	0x09	4800 SPS

****Note**: If the requested sampling rate is the same as the current sampling rate, the device replies to the command and does not apply changes to the settings.*

Error and Fault Detection

The *QIA135* utilizes a secondary 24-bit ADC to provide board temperature reading to the Host device using PT1000. The same 24-bit ADC is used to monitor the current draw of the bridge, and the excitation voltage. The following formulas can be used to convert the ADC data to **Current**, **Voltage** and **Temperature** respectively.

RTD Temperature Conversion

$$Excitation\ Current(A) = \left(\frac{(ADC_{DEC} - 8388607) * \left(\frac{2.5}{8388607}\right)}{4}\right) * \frac{1}{1000}$$

$$Rt(\Omega) = \frac{(ADC_{DEC} -\ 8388607) \times 2.5}{8388607 \times 4 \times (Excitation\ Current)}$$

$$T_{RTD}(°C) = \frac{-1000 \times 3.9083 \times 10^{-3} + \sqrt{1000^2 \times (3.9083 \times 10^{-3})^2 - 4 \times 1000 \times -5.7750 \times 10^{-7} \times (1000 - Rt)}}{2 \times 1000 \times -5.7750 \times 10^{-7}}$$

Current Conversion

$$I_{LIM}(mA) = \frac{(ADC_{DEC} -\ 8388607) \times 2.5 \times 1000 \times 400}{8388607 \times 8\ \ \times 3000}$$

Excitation Voltage Conversion

$$EXC(V) = \frac{(ADC_{DEC} -\ 8388607) \times 2.5 \times 3}{8388607 \times 2 \times 0.6}$$

Command-Set List

Table 7.

Type	Name	Description	HOCI Line Packet Structure (Host to QIA135)							HICO Line Packet Structure (QIA135 to Host)						
			N/A	N/A	N/A	N/A	CMD	CRC16	CRC16	Error Code	Payload	Payload	Payload	Payload	CRC16	CRC16
			Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Get	*GADC0*	Get ADC0 Data	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x01*	*MSB*	*LSB*	*See Table 5.*	*ADC0 MSB*	*ADC0 Byte 1*	*ADC0 Byte 2*	*ADC0 LSB*	*MSB*	*LSB*
Get	*GADC1*	Get ADC1 Data	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x02*	*MSB*	*LSB*	*See Table 5.*	*ADC1 MSB*	*ADC1 Byte 1*	*ADC1 Byte 2*	*ADC0 LSB*	*MSB*	*LSB*
Get	*GADC2*	Get ADC2 Data	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x03*	*MSB*	*LSB*	*See Table 5.*	*ADC2 MSB*	*ADC2 Byte 1*	*ADC2 Byte 2*	*ADC2 LSB*	*MSB*	*LSB*
Get	*GADC3*	Get ADC3 Data	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x04*	*MSB*	*LSB*	*See Table 5.*	*ADC3 MSB*	*ADC3 Byte 1*	*ADC3 Byte 2*	*ADC3 LSB*	*MSB*	*LSB*
Get	*GADC4*	Get ADC4 Data	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x05*	*MSB*	*LSB*	*See Table 5.*	*ADC4 MSB*	*ADC4 Byte 1*	*ADC4 Byte 2*	*ADC4 LSB*	*MSB*	*LSB*
Get	*GADC5*	Get ADC5 Data	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x06*	*MSB*	*LSB*	*See Table 5.*	*ADC5 MSB*	*ADC5 Byte 1*	*ADC5 Byte 2*	*ADC5 LSB*	*MSB*	*LSB*
Get	*GSSN*	Get Sensor Serial Number	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x07*	*MSB*	*LSB*	*See Table 5.*	*SSN MSB*	*SSN Byte 1*	*SSN Byte 2*	*SSN LSB*	*MSB*	*LSB*
Get	*GISN*	Get Instrument Serial Number	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x08*	*MSB*	*LSB*	*See Table 5.*	*ISN MSB*	*ISN Byte 1*	*ISN Byte2*	*ISN LSB*	*MSB*	*LSB*
Get	*GFRN*	Get Firmware Revision Number	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x09*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*Major*	*Minor*	*Patch*	*MSB*	*LSB*

Get	*GDR*	Get Data Rate	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x0A*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*See Table 6.*	*MSB*	*LSB*
Set	*S5SPS*	Set 5 Sample Per Second	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x0B*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*MSB*	*LSB*
Set	*S7SPS*	Set 7 Sample Per Second	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x0C*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*MSB*	*LSB*
Set	*S10SPS*	Set 10 Sample Per Second	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x0D*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*MSB*	*LSB*
Set	*S50SPS*	Set 50 Sample Per Second	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x0E*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*MSB*	*LSB*
Set	*S60SPS*	Set 60 Sample Per Second	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x0F*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*MSB*	*LSB*
Set	*S150SPS*	Set 150 Sample Per Second	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x10*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*MSB*	*LSB*
Set	*S300SPS*	Set 300 Sample Per Second	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x11*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*MSB*	*LSB*
Set	*S1000SPS*	Set 1000 Sample Per Second	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x12*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*MSB*	*LSB*
Set	*S2400SPS*	Set 2400 Sample Per Second	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x13*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*MSB*	*LSB*
Set	*S4800SPS*	Set 4800 Sample Per Second	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x14*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*MSB*	*LSB*
Get	*GSHS*	Get System Health Status	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x15*	*MSB*	*LSB*	*See Table 5.*	*Secondary ADC MSB*	*Secondary ADC Byte 1*	*Secondary ADC Byte 2*	*Secondary ADC LSB*	*MSB*	*LSB*
Get	*GBT*	Get Board Temperature	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x16*	*MSB*	*LSB*	*See Table 5.*	*Secondary ADC MSB*	*Secondary ADC Byte 1*	*Secondary ADC Byte 2*	*Secondary ADC LSB*	*MSB*	*LSB*
Get	*GEXCV*	Get Excitation Voltage	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x17*	*MSB*	*LSB*	*See Table 5.*	*Secondary ADC MSB*	*Secondary ADC Byte 1*	*Secondary ADC Byte 2*	*Secondary ADC LSB*	*MSB*	*LSB*

Get	*GBTE**	Get Board Temperature Excitation	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x1B*	*MSB*	*LSB*	*See **Table 5.***	*Secondary ADC MSB*	*Secondary ADC Byte 1*	*Secondary ADC Byte 2*	*Secondary ADC LSB*	*MSB*	*LSB*

***Note:** 0xXX = Don’t care.*
***Note:** All pre-defined responses from each command that is sent on the HOCI line should be expected in the next $\overline{DRDY}$ period.*
***Note:** For the purpose of simplification, the GBTE command can be transmitted singularly during the initialization phase. However, for enhanced precision in temperature date retrieval, it is recommended to use this command each time the RTD conversation is computed.*

Packet and CRC Example

The two following transactions are the request for the ***GSSN*** command (***Get Sensor Serial Number***) and its response respectively. The first HICO transaction includes the pre buffered zeros and the second HICO transaction includes the response of the ***GSSN*** command which is being clocked out with ***GADC5*** (***Get ADC5 Data***) command in this example.

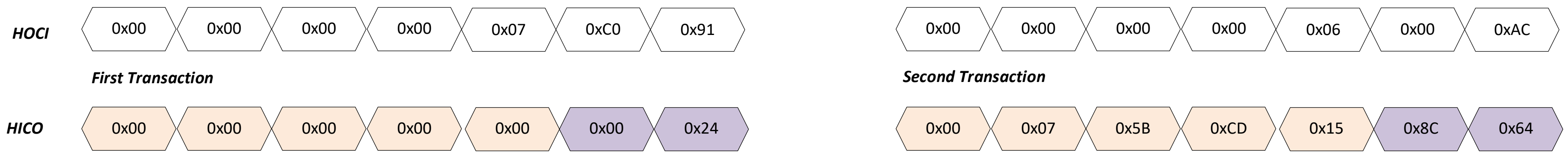

Figure 5.

u16 crc16(u16 crc, const u8 *buffer, size_t len); function (See *CRC Calculations and References*) has been used as a reference to calculate the CRC for the example above:

// *CRC* calculation for the first HICO transaction, *MSB = 0x00 and LSB = 0x00*
u8 BUFFER[] = {*LSB,…,MSB*} »»» *u8* BUFFER[] = {*0x00, 0x00, 0x00, 0x00, 0x00*};
u16 crc16(*0xFFFF*, *BUFFER*, 13);
then function returns ***0x0024***

// *CRC* calculation for the second HICO transaction, *MSB = 0x00 and LSB = 0x15*
u8 BUFFER[] = {*LSB,…,MSB*} »»» *u8* BUFFER[] = {*0x15, 0xCD, 0x5B, 0x07, 0x00*};
u16 crc16(*0xFFFF*, *BUFFER*, 13);
then function returns ***0x8C64***

ADC Data Conversion

The calibration and conversion are done in the firmware therefore the following function could be used to convert Hex to Float.

Hex to Float Conversion Example

```
void uiRdArray(uint8_t *arr, uint8_t *pld, uint8_t size, uint8_t pldStrtIndx,
uint8_t arrStrtIndx){
uint8_t pldIndx = pldStrtIndx; // Pld array index
uint8_t arrIndx = arrStrtIndx; // Arr array index
while(size--){ // Read array
arr[arrIndx++] = pld[pldIndx++];
}//end while
}//end uiRdArray()


void uiRdFloat(float *var, uint8_t *pld, uint8_t strtIndx){
uint8_t tempArray [uiMaxPldSize()]; // Temp array to hold float bytes
uiRdArray(tempArray, pld, 4, strtIndx, 0); // Read float bytes from payload
memcpy(var, &tempArray, fltNumSize()); // Array to float
}//end uiRdFloat()
```

The following link could also be used as a reference for test and verification purposes.

https://gregstoll.com/~gregstoll/floattohex/

Note: "Swap Endianness" needs to be selected.

Current Conversion Example

Example: Received hex value 0x00AF852A from GSHS command.

Hex to decimal: *0x00AF852A* -> 11502890

$$I_{LIM} = \frac{(11502890 - 8388607) \times 2.5 \times 1000 \times 400}{8388607 \times 8 \times 3000} = 15.4688\ mA$$

Excitation Voltage Conversion Example

Example: Received hex value 0x00DDFC23 from GEXCV command.

Hex to decimal: *0x00DDFC23* -> 14548003

$$EXC = \frac{(14548003 - 8388607) \times 2.5 \times 3}{8388607 \times 2 \times 0.6} = 4.5891\ V$$

RTD Temperature Conversion Example

Example: Received hex value 0x00947AF5 from GBTE command.

Hex to decimal: *0x00947AF5* -> 9730805

$$Excitation\ Current = \left(\frac{(9730805 - 8388607) * \left(\frac{2.5}{8388607}\right)}{4}\right) * \frac{1}{1000} = 0.0001\ A$$

Example: Received hex value 0x00966A49 from GBT command.

Hex to decimal: *0x00966A49* -> 9857609

$$Rt = \frac{(9857609 - 8388607) \times 2.5}{8388607 \times 4 \times 0.0001} = 1094.5\ \Omega$$

$$T_{RTD} = \frac{-1000 \times 3.9083 \times 10^{-3} + \sqrt{1000^2 \times (3.9083 \times 10^{-3})^2 - 4 \times 1000 \times -5.7750 \times 10^{-7} \times (1000 - 1094)}}{2 \times 1000 \times -5.7750 \times 10^{-7}} = 24.27\ °C$$

Firmware Revision QIA135

APPLICATION		
Revision	v2.0.1	
Release Date	9/11/2024	
Hardware Compatibility	REV000	
Notes	New Features • Initial Release	
	Changes • Initial Release	
	Fixes • Initial Release	

CRC Calculations and References

The *CRC16* calculation has been implemented (full duplex) per the links below.

https://github.com/futekast/CRC16_ARC