FUTEK
ADVANCED SENSOR TECHNOLOGY, INC.

10 Thomas, Irvine, CA 92618 USA
Tel: (949) 465-0900
Fax: (949) 465-0905

QIA125/QIA127 SPI Communication Guide

Table of Contents

General Description 3

Pin Configurations and Function Descriptions for QIA125 3

Pin Configurations and Function Descriptions for QIA127 5

QIA125/QIA127 SPI Configuration 6

QIA125/QIA127 Internal Design Algorithm 7

SPI Packet Structure 8

"Continuous Read" Mode 8

Timing Diagrams 9

Packet Structure (Get ADC Data): 9

***DRDY* Period** 9

System Behavior 10

Start-up 10

Wrong CRC Error 10

Wrong CMD Error 10

System Health Error 10

Temperature Error 10

Error Codes 11

Sampling Rate Change 11

Error and Fault Detection 12

Voltage Diode Calculation 12

Temperature Conversion 12

Current Conversion 12

Command-Set List 13

Packet and CRC Examples 17

Firmware Revision 18

Firmware Notes 18

New Features 18

Changes 18

Fixes 18

CRC Calculations and References 18

General Description

The *QIA125/QIA127* is a three-channel (simultaneous sampling) digital controller with USB and SPI outputs. The *QIA125/QIA127* (slave device) can be used to communicate with any master devices through an *SPI* bus.

Pin Configurations and Function Descriptions for QIA125

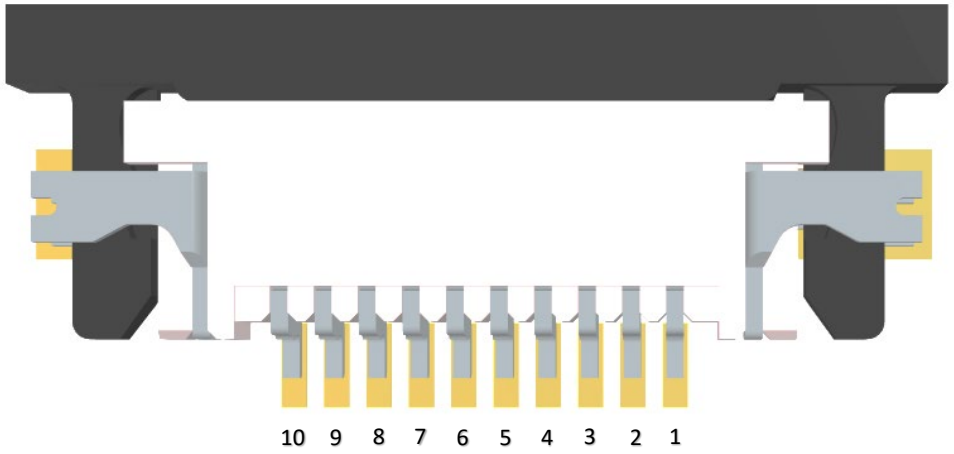


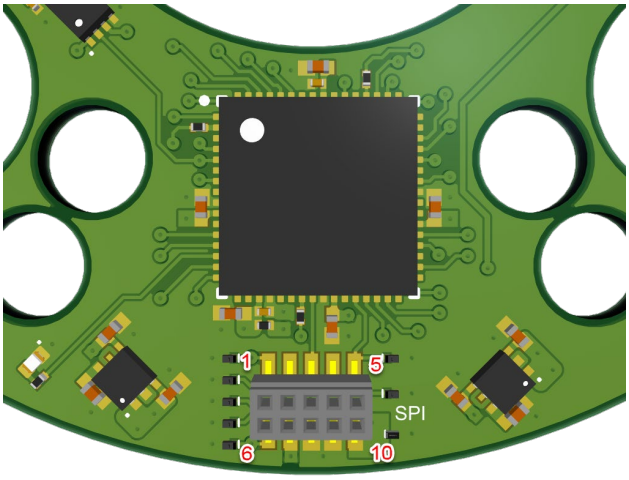
Figure 1a.

Table 1a.

#	Pin	Description
1	***VIN***	Voltage input *5V±10%*
2	***GND***	Ground pins are connected to each other internally
3	***GND***	Ground pins are connected to each other internally
4	***GND***	Ground pins are connected to each other internally
5	***MISO***	Master-In-Slave-Out ***Note:** A weak pull down has been used to eliminate any random spikes.*

6	***MOSI***	Master-Out-Slave-In
7	***SCLK***	Serial clock generated by master
8	$\overline{CS}$	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.
9	$\overline{DRDY}$	Active low $\overline{DRDY}$ pin is used to keep all communication synchronized. It notifies the master device when new data from the sampling system is ready. This ensures that the master 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 master. 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.*
10	***TRIG***	Trigger is an input pin (slave) and output pin (master) dedicated for special applications such as programing for future development.

Pin Configurations and Function Descriptions for QIA127



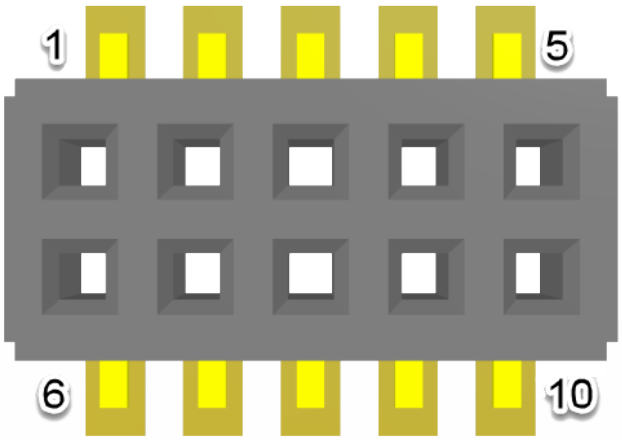


Figure 2b.

Table 1b.

#	Pin	Description
1	***TRIG***	Trigger is an input pin (slave) and output pin (master) dedicated for special applications such as programing for future development.
2	$\overline{DRDY}$	Active low $\overline{DRDY}$ pin is used to keep all communication synchronized. It notifies the master device when new data from the sampling system is ready. This ensures that the master 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 master. 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.*
3	$\overline{CS}$	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.
4	***SCLK***	Serial clock generated by master
5	***MOSI***	Master-In-Slave-Out ***Note:** A weak pull down has been used to eliminate any random spikes.*
6	***MISO***	Master-Out-Slave-In

7	***CHASSIS***	Connected to Chassis
8	***GND***	Ground pins are connected to each other internally
9	***GND***	Ground pins are connected to each other internally
10	***VIN***	Voltage input *5V±10%*

QIA125/QIA127 SPI Configuration

Table 2.

Serial Word Length	8-Bit			
SPI Mode	*Mode 0, CPOL = 0, CPHA = 0*			
SCLK Frequency	**Min**	*4 MHz*	**Max**	*8 MHz*
Internal Clock Frequency of MCU	*32 MHz*			
Operation Mode	Slave			

QIA125/QIA127 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 master device's request. $\overline{DRDY}$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 data (ADC1, ADC2 and ADC3) from the highest interrupt priorities
- Slave 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 ***GADC*** command to go to the default state
    - Checks the *CRC16* byte, *CMD* byte, system health status, and board temperature
        - If either the *CRC16* and 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*
            - Loades 12 bytes of data (including the *Error Code* byte, latest *ADC1, ADC2, ADC3* data 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 QIA125/QIA127 via the MOSI 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 the ADC1, ADC2, and ADC3 data.

SPI Packet Structure

The packet structure stays consistent during all transactions and always includes twelve 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 *ADC* data 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.***)

***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.*

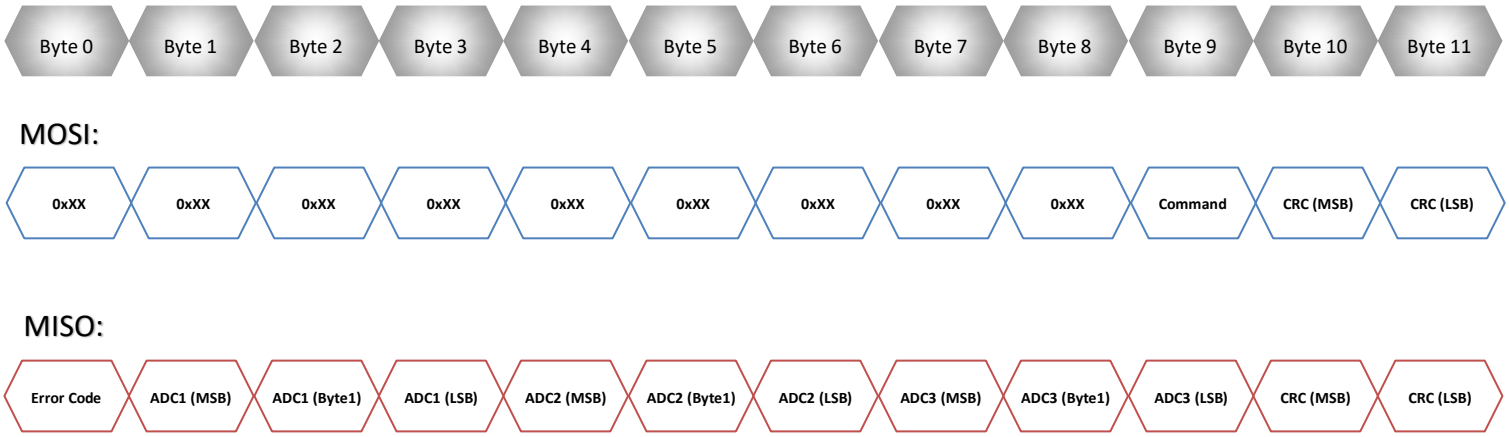


Figure 2.

"Continuous Read" Mode

GADC 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, all three ADC data followed by the CRC bytes.*

Timing Diagrams

Packet Structure (Get ADC Data):

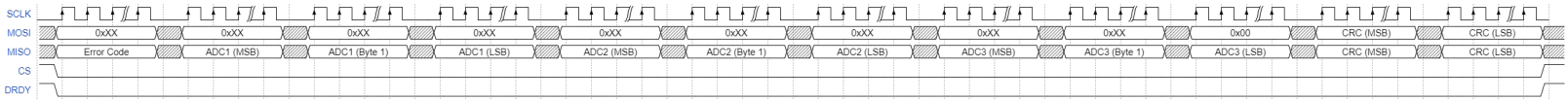


Figure 3.

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

$\overline{DRDY}$ Period

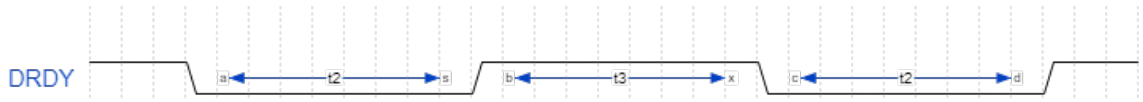


Figure 4.

Table 3.

t_1 (μs)	t_2 (ms)		t_3 (μs)		Description
	Min	Max	Min	Max	
*0 to …**	200	*215*	*80*	*90*	5 SPS
	120	*135*			7 SPS
	90	*100*			10 SPS
	18	*20*			50 SPS
	15	*16.5*			60 SPS
	5.5	7			150 SPS
	3	3.5			300 SPS
	0.90	0.95			960 SPS
	0.32	0.33			2400 SPS
	0.115	0.125			4800 SPS

***Note**: No delay or any delay as long as all 12 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 white LED indicator starts blinking; 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 the system receives a packet with an incorrect *CRC*, it goes to the default state that replies with the first byte of the packet as an Error Code followed by the *ADC* data and *CRC*.

Wrong CMD Error

When the system receives a packet with an undefined command, it goes to the default state that replies with the first byte of the packet as an Error Code followed by the *ADC* data and *CRC*.

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 ($16\ °C - 40\ °C$).

***Note:** Three red LEDs will turn on to indicate the System Health/Temperature error(s).*

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, 0x05 carries the CRC and system health 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 (ms)	SR Code	Sampling Rate
1800	*0x00*	5 SPS
1300	*0x01*	7 SPS
1000	*0x02*	10 SPS
300	*0x03*	50 SPS
200	*0x04*	60 SPS
120	*0x05*	150 SPS
75	*0x06*	300 SPS
30	*0x07*	960 SPS
20	*0x08*	2400 SPS
15	*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 QIA125/QIA127 utilizes an internal 12-bit ADC to provide the system health status and the board temperature reading to the master device. The following formulas can be used to convert the ADC data to board temperature and bridge current.

Voltage Diode Calculation

$$\boldsymbol{V_{diode}}\ (\boldsymbol{mV}) = \frac{ADC\ Data\ (unsigned\ right\ justified)\ \times 3300}{2^{12}}$$

Temperature Conversion

$$\boldsymbol{T_{die}}(^{\circ}\boldsymbol{C}) = \frac{760\ (mV) - V_{diode}\ (mV)}{1.55}$$

Current Conversion

$$\boldsymbol{Current}(\boldsymbol{mA}) = \frac{V_{diode}\ (mV) \times 400}{3000\ \times\ 10.09}$$

Command-Set List

Table 7.

Type	Name	Description	MOSI Line Packet Structure (Master to QIA125/QIA127)												MISO Line Packet Structure (QIA125/QIA127 to Master)											
											CMD	CRC	CRC	Error Code	PLD	PLD	PLD	PLD	PLD	PLD	PLD	PLD	PLD	CRC	CRC
			Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7	Byte8	Byte9	Byte10	Byte11	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7	Byte8	Byte9	Byte10	Byte11
Get	*GADC*	Get ADC Data	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x00*	*MSB*	*LSB*	*See Table 5.*	*ADC1 MSB*	*ADC1 Byte1*	*ADC1 LSB*	*ADC2 MSB*	*ADC2 Byte1*	*ADC2 LSB*	*ADC3 MSB*	*ADC3 Byte1*	*ADC3 LSB*	*MSB*	*LSB*
Get	*GD1CP0*	Get Direction 1 Calibration Point Zero	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x01*	*MSB*	*LSB*	*See Table 5.*	*ADC1 MSB*	*ADC1 Byte1*	*ADC1 LSB*	*ADC2 MSB*	*ADC2 Byte1*	*ADC2 LSB*	*ADC3 MSB*	*ADC3 Byte1*	*ADC3 LSB*	*MSB*	*LSB*
Get	*GD1CP1*	Get Direction 1 Calibration Point One	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x02*	*MSB*	*LSB*	*See Table 5.*	*ADC1 MSB*	*ADC1 Byte1*	*ADC1 LSB*	*ADC2 MSB*	*ADC2 Byte1*	*ADC2 LSB*	*ADC3 MSB*	*ADC3 Byte1*	*ADC3 LSB*	*MSB*	*LSB*
Get	*GD1CP2*	Get Direction 1 Calibration Point Two	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x03*	*MSB*	*LSB*	*See Table 5.*	*ADC1 MSB*	*ADC1 Byte1*	*ADC1 LSB*	*ADC2 MSB*	*ADC2 Byte1*	*ADC2 LSB*	*ADC3 MSB*	*ADC3 Byte1*	*ADC3 LSB*	*MSB*	*LSB*
Get	*GD1CP3*	Get Direction 1 Calibration Point Three	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x04*	*MSB*	*LSB*	*See Table 5.*	*ADC1 MSB*	*ADC1 Byte1*	*ADC1 LSB*	*ADC2 MSB*	*ADC2 Byte1*	*ADC2 LSB*	*ADC3 MSB*	*ADC3 Byte1*	*ADC3 LSB*	*MSB*	*LSB*
Get	*GD1CP4*	Get Direction 1 Calibration Point Four	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x05*	*MSB*	*LSB*	*See Table 5.*	*ADC1 MSB*	*ADC1 Byte1*	*ADC1 LSB*	*ADC2 MSB*	*ADC2 Byte1*	*ADC2 LSB*	*ADC3 MSB*	*ADC3 Byte1*	*ADC3 LSB*	*MSB*	*LSB*

Get	*GD1CP5*	Get Direction 1 Calibration Point Five (+Span)	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x06*	*MSB*	*LSB*	*See Table 5.*	*ADC1 MSB*	*ADC1 Byte1*	*ADC1 LSB*	*ADC2 MSB*	*ADC2 Byte1*	*ADC2 LSB*	*ADC3 MSB*	*ADC3 Byte1*	*ADC3 LSB*	*MSB*	*LSB*
Get	*GD2CP0*	Get Direction 2 Calibration Point Zero	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x07*	*MSB*	*LSB*	*See Table 5.*	*ADC1 MSB*	*ADC1 Byte1*	*ADC1 LSB*	*ADC2 MSB*	*ADC2 Byte1*	*ADC2 LSB*	*ADC3 MSB*	*ADC3 Byte1*	*ADC3 LSB*	*MSB*	*LSB*
Get	*GD2CP1*	Get Direction 2 Calibration Point One	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x08*	*MSB*	*LSB*	*See Table 5.*	*ADC1 MSB*	*ADC1 Byte1*	*ADC1 LSB*	*ADC2 MSB*	*ADC2 Byte1*	*ADC2 LSB*	*ADC3 MSB*	*ADC3 Byte1*	*ADC3 LSB*	*MSB*	*LSB*
Get	*GD2CP2*	Get Direction 2 Calibration Point Two	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x09*	*MSB*	*LSB*	*See Table 5.*	*ADC1 MSB*	*ADC1 Byte1*	*ADC1 LSB*	*ADC2 MSB*	*ADC2 Byte1*	*ADC2 LSB*	*ADC3 MSB*	*ADC3 Byte1*	*ADC3 LSB*	*MSB*	*LSB*
Get	*GD2CP3*	Get Direction 2 Calibration Point Three	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x0A*	*MSB*	*LSB*	*See Table 5.*	*ADC1 MSB*	*ADC1 Byte1*	*ADC1 LSB*	*ADC2 MSB*	*ADC2 Byte1*	*ADC2 LSB*	*ADC3 MSB*	*ADC3 Byte1*	*ADC3 LSB*	*MSB*	*LSB*
Get	*GD2CP4*	Get Direction 2 Calibration Point Four	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x0B*	*MSB*	*LSB*	*See Table 5.*	*ADC1 MSB*	*ADC1 Byte1*	*ADC1 LSB*	*ADC2 MSB*	*ADC2 Byte1*	*ADC2 LSB*	*ADC3 MSB*	*ADC3 Byte1*	*ADC3 LSB*	*MSB*	*LSB*
Get	*GD2CP5*	Get Direction 2 Calibration Point Five (-Span)	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x0C*	*MSB*	*LSB*	*See Table 5.*	*ADC1 MSB*	*ADC1 Byte1*	*ADC1 LSB*	*ADC2 MSB*	*ADC2 Byte1*	*ADC2 LSB*	*ADC3 MSB*	*ADC3 Byte1*	*ADC3 LSB*	*MSB*	*LSB*
Get	*GSSN*	Get Sensor Serial Number	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x0D*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*SSN MSB*	*SSN Byte1*	*SSN LSB*	*MSB*	*LSB*

Get	*GISN*	Get Instrument Serial Number	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x0E*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*ISN MSB*	*ISN Byte1*	*ISN LSB*	*MSB*	*LSB*
Get	*GFRN*	Get Firmware Revision Number	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x0F*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*Major MSB*	*Minor Byte1*	*Patch LSB*	*MSB*	*LSB*
Get	*GDR*	Get Data Rate	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x10*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*SR Code See Table 6.*	*MSB*	*LSB*
Set	*S5SPS*	Set 5 Sample Per Second	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x11*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*MSB*	*LSB*
Set	*S7SPS*	Set 7 Sample Per Second	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x12*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*MSB*	*LSB*
Set	*S10SPS*	Set 10 Sample Per Second	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x13*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*MSB*	*LSB*
Set	*S50SPS*	Set 50 Sample Per Second	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x14*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*MSB*	*LSB*
Set	*S60SPS*	Set 60 Sample Per Second	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x15*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*MSB*	*LSB*
Set	*S150SPS*	Set 150 Sample Per Second	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x16*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*MSB*	*LSB*
Set	*S300SPS*	Set 300 Sample Per Second	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x17*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*MSB*	*LSB*
Set	*S960SPS*	Set 960 Sample Per Second	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x18*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*MSB*	*LSB*

Set	*S2400SPS*	Set 2400 Sample Per Second	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x19*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*MSB*	*LSB*
Set	*S4800SPS*	Set 4800 Sample Per Second	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x20*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*MSB*	*LSB*
Get	*GSHS*	Get System Health Status	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x21*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*Internal ADC MSB*	*Internal ADC Byte1*	*Internal ADC LSB*	*MSB*	*LSB*
Get	*GBT*	Get Board Temperature	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0xXX*	*0x22*	*MSB*	*LSB*	*See Table 5.*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*0x00*	*Internal ADC MSB*	*Internal ADC Byte1*	*Internal ADC LSB*	*MSB*	*LSB*

***Note:** 0xXX = Don't care*

***Note:** All pre-defined responses from each command that is sent on the MOSI line should be expected in the next $\overline{DRDY}$ period.*

Packet and CRC Examples

The following transaction is the response to the ***GSSN*** command (***Get Sensor Serial Number***) that is being clocked out with the ***GADC*** command (***Get ADC Data***):

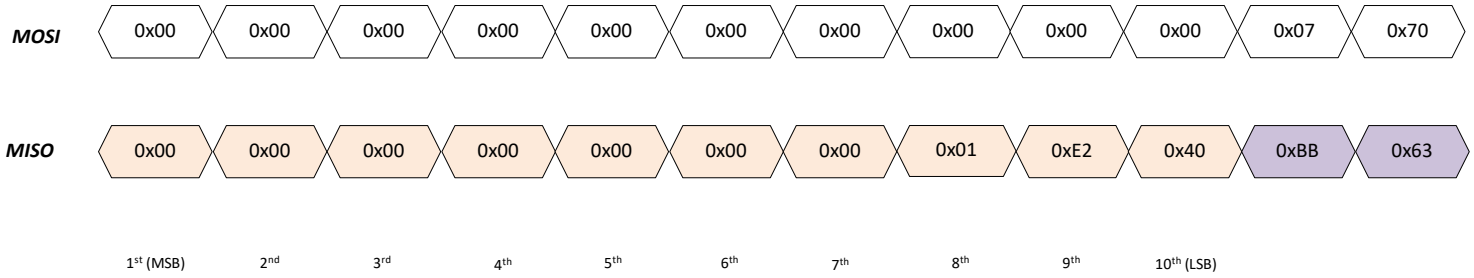


Figure 6.

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:

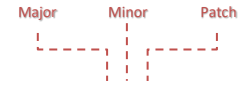
// *CRC* calculation for the MISO transaction, *MSB = 0x00 and LSB = 0x40*

u8 BUFFER[] = {*LSB,…,MSB*} »»»» *u8* BUFFER[] = {*0x40, 0xE2, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00*};

u16 crc16(*0xFFFF*, *BUFFER*, 10);

then function returns *0XBB63*

Firmware Revision



Revision 2.0.3

Firmware Notes

New Features

- N/A

Changes

- N/A

Fixes

- Internal software communication bug fix

CRC Calculations and References

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

https://github.com/torvalds/linux/blob/master/include/linux/crc16.h

https://github.com/torvalds/linux/blob/master/lib/crc16.c