Integrated IR Application Note 1

"Communication & Algorithms"

ISSUE 3

Abstract

The Integrated IR (INIR) Gas Sensor has been designed using the latest technology with an ARM7TDMI core microcontroller and via software design the necessary techniques have been implemented to increase the reliability of the device minimizing probability of faults. The INIR is a user friendly digital gas sensor, designed to decrease the implementation time, hence increasing productivity. The Integrated IR incorporates the necessary electronics and embedded firmware using infrared gas sensing technology with reduced power dissipation. The sensor will sample the raw signals to output a linear, temperature compensated signal proportional to the gas concentration applied. The output signal is available in digital and analogue forms. The INIR provides users with a simple method of incorporating an infrared gas sensor into their gas detection instrument significantly reducing development time and expertise required for implementation. The Integrated IR can also be factory calibrated to allow installation without the need for recalibration in various gases and target concentrations if required.

Introduction

This document gives an overview of the methods used to communicate with the Integrated IR Gas Sensor and specifically concentrates around the Universal Asynchronous Receive/Transmit (UART) communication protocol. In addition to the above this document incorporates examples of communication based in real life application assuming that the INIR Gas Sensor is connected to another Microcontroller via the UART (TX,RX pins).

The core technology used incorporates a very well-known and proven to work NDIR Gas Sensing technique together with a robust, highly reliable ARM7 MCU mounted on a performance printed circuit board. The INIR has been designed as an improved digital version of the 7-pin single gas like the IR12GM_1 gas sensor, which uses a supported filament lamp for additional shock protection. The necessary electronic circuit and embedded firmware will calculate the linearized and temperature compensated concentration via Digital (32-bit) and Analog (14-bits) outputs. The Configuration Unit can be used to set up, calibrate and evaluate the INIR with easy-to-use PC software.

Alternatively, control of the INIR via UART is available for communicating with an external microprocessor as well. The device contains full fault diagnostics, which are sent via the Digital String along with the Temperature output of the sensor and the linearized Concentration in parts per million (ppm).

Summary of Contents

The first chapter accommodates a complete guide on how to communicate with the INIR Gas Sensor. Highly detailed guidelines and summary tables are attached in order to explain the capabilities of the INIR Sensor and the correct way to implement the communication via UART with another microcontroller or a personal computer.

Please find attached in the chapter two few functions in C code to help us understand how we can read and write to the INIR digital Sensor by using low level firmware commands.

In chapter three there is a small tutorial on how to perform the Calibration Procedure via the Evaluation Software. An example and many screenshots are attached for help us understanding of the software's tools as well as the calibration procedure.

In the final chapter find details about pressure compensation and if or when we will need to implement an algorithm accordingly.

Abbreviations & Glossary

Abbreviation	Meaning/Description
USB	Universal Serial Bus
ADC	Analog-to-Digital Converter
DAC	Digital-to-Analog Converter
INIR	Integrated IR Gas Sensor
UART	Universal Asynchronous Transmit Receive Protocol
MCU	Microcontroller Unit
CPU	Central Processing Unit
PC	Personal Computer
DC	Direct Current
AC	Alternating Current
Config	Configuration
Init	Initialization
FLASH/EE	Flash memory, embedded to MCU

Table of Contents

Abstract 1
Introduction 1
Summary of Contents 1
Abbreviations & Glossary 1
Table of Contents 2
Document Revision Record 2
Document Errata Record 2
❶ | Communicate with Integrated IR 3
1.1 Universal Asynchronous Receive Transmit (UART) 3
1.1.1 UART Communication Settings 3
1.1.2 Digital Output String Via UART 4
1.1.3 Commands to the UART 5
1.1.4 Initialization Procedure for INIR 6
1.1.5 Answers from the UART 7
1.2 Example of Communication between Integrated IR & Custom Device ... 9
1.2.1 COMMAND [A], NORMAL Mode 9
1.2.2 COMMAND [B], Enter ENGINERING Mode 9
1.2.3... COMMAND [C], Enter CONFIGURATION Mode 9
1.2.4 COMMAND [D], Reserved 9
1.2.5 COMMAND [E], ZERO Calibration (for NORMAL/ENGINEERING Modes) 10
1.2.6 COMMAND [F], SPAN Calibration (for NORMAL/ENGINERRING Modes) 10
1.2.7 COMMAND [G], OFFSET Calculation (NORMAL or ENGINERRING Modes) 10
1.2.8 COMMAND [H], Reserved 10
1.2.9 COMMAND [I], Read Back Settings (only in CONFIGURATION Mode) 10
1.2.10 COMMAND [J], Load NEW Settings (only in CONFIGURATION Mode) 12
1.2.11 COMMAND [K], RESET to FACTORY Defaults (only in CONFIGURATION Mode) 13
1.2.12 COMMAND [L], TURN "ON" Humidity Algorithm (all Modes) 13
1.2.13 COMMAND [M], TURN "OFF" Humidity Algorithm (all Modes) 13
1.2.14 COMMAND [N], Reserved 13
1.2.15 COMMAND [O], Reserved 13
1.3 Changing Between Low/Mid/High Ranges 13
1.4 Convert Temperature in KELVIN to CELCIUS 13
1.5 Calculate Conc v.v% by Analogue Output 14
1.6 TRANSLATING THE "FAULTS" CODE 15
1.7 Calculating the Cyclic Redundancy Check (CRC) 16
1.7.1 Bitwise compliment operator 17
❷ | Examples Implementation Functions in C 18
❸ | Calibration Procedure 23
Equipment Used 23
Recommended Environmental Conditions 23
Basic Set Up 23
3.1.Calibration Routine using the Eval. Software 24
3.2 Calibrating Different Target Gases 26
❹ | Pressure Compensation Algorithm 27
❺ | Condensation Compensation Algorithm 28
Appendix A - Table of Coefficients 30
Appendix B – INIR Quick Reference Chart 31
Index 32

Document Revision Record

Version	Issue Date DD/MM/YYYY	Main Changes
0.0 iss 0	10/06/2014	Original Draft version.
2.0 iss 1	12/12/2014	Amended DRAFT version to accommodate the new calibration and firmware communications. (Applies for SMIR firmware 4v0&above or INIR firmware 1v0&above).
2.0 iss 2	14/01/2014	Amended DRAFT version to accommodate the changes to the release of the INIR project. (Applies only to INIR 1v0&above).
Issue 3	11/11/2015	Officially issued Application Note.

Document Errata Record

Version	Issue Date DD/MM/YYYY	Errata Sheet Reference
		No Errata Sheet Exists yet.

1 | Communicate with Integrated IR

There are two ways of communicating with the Integrated IR (INIR). The first way is by reading the Analogue Output of the INIR which can be connected to an Analogue to Digital Converter (ADC). The Analogue Output is giving a linearized value of the gas concentration detected, please see "Calculate Conc v.v% by Reading the Analogue Output" for more details. This method of communication is one-way and we can only read the value coming from the INIR, but we cannot send data back. For best resolution of the concentration we must be able to read the Analogue Output with minimum 3 floating points precision.

The second way of communicating with the Integrated IR is by using the Universal Asynchronous Receive Transmit (UART) protocol. The Digital Output String contains the information we need for the calculated concentration, faults, active, reference as well as the internal temperature of the sensor. We can read the UART output either by plugging in the INIR in SGX's Configuration Unit and using the INIR Evaluation Software or by connecting the UART directly to another microcontroller. This specific way of communication is bidirectional, meaning that we do have the opportunity to send commands to the INIR via UART as well.

With this method it is possible to send commands to the INIR and get answers back from the device accordingly, please see "Universal Asynchronous Receive Transmit (UART)" for more details, in page 3.

The software for the device has been designed to increase response times making the whole module faster than most custom made equipment fully customizable. During the data transition precautions have been taken to decrease the change of software failure by using common coding techniques, fuzzy logic algorithms and hardware supervision functions. Last but not least the Real Time Data Processing algorithms used in the module is making the device suitable for Real Time Applications inside tight industrial standards.

1.1 Universal Asynchronous Receive Transmit

The Universal Asynchronous Receive Transmit (UART) is a protocol that many devices especially microcontrollers are using often to communicate with each other or with a computer. The INIR device has been designed to produce a string with all the data that we need in order to implement the device into any gas sensing application. Please see paragraphs below for settings and more details on how to communicate with the INIR via the UART and the format used for the digital output string transmitted.

Whether we want to communicate with another microprocessor or a computer via the serial communication protocol we must have the same settings in both devices otherwise the communication might be impossible or sometimes faulty. When we are calculating the response times in the concentration of gases for the INIR, it is advisable to take into consideration the delay, which we are having from the transmission and data processing. In case of the INIR Evaluation Software that delay is around 1 second, therefore the data that we can see in the Graph are effectively the data transmitted a second ago, overall 2 seconds delay from Real Time environment. If averaging of more than 2 values is chosen read the INIR Datasheet or is recommended to you RE-calculate the response times

1.1.1 UART Communication Settings

At the moment it is advisable to communicate with the UART of the Integrated IR module by using the recommended setting in the Table 1.0 - UART Settings seen below. Obviously the same settings will be valid if you try to communicate with the INIR by using a PC or another microcontroller.

Please see below the settings for communicating with the UART.

Table 1 –UART Communication Default Settings

Baud Rate	38400
Data Bits	8
Parity	None
Stop Bits	2
Handshaking	None

IMPORTANT WARNING!

It is possible of us changing the Baud Rate using the equivalent variable stored into the Flash, but the 38400 is suitable fast enough with minimum errors for most applications. If the Baud Rate accidentally changed into something wrong for example "10" the INIR will not recognise the Baud Rate but it will use the 38400 by "default". If changed into "9600" for example we need to change the settings in our

1.1.2 Digital Output String Via UART

IMPORTANT WARNING!

All the variables are HEX representation of the numbers but are transmitted via the UART as a Digital "String" that means that all the variables are send and read in ASCII format, see functions 1 to 4 for more details.

Every second the INIR uses the UART to send a string to the digital output. Therefore the TXD pin of the UART is transmitting all the information to the PC or another custom device. There are two different states in the transmission, depending on the "Mode" that the INIR is running. There are generally three modes that the MCU can run and all of them can be configured via the UART by sending the appropriate command. Please see the Table 3 - UART Commands for more information on the commands that are available to the Integrated IR. In the same way the INIR can send an answer back so it is possible to have a two-way communication with the Integrated IR device. See the Table 8 - Read Back the Settings from the Integrated IR to understand how the data are being stored to the Microcontroller. We can also see the Table 10 - Answers from the Integrated IR UART, for the possible answers that we can get when sending a command. The UART Communication settings can vary but the INIR can only reach at a maximum of 38400 Baud Rate. The probability of an error in communication protocol of the specific Baud Rate is around 0.015% given by the manufacturer assuming ambient temperature of 25°C, without strong electromagnetic field. Consider using the CRC to minimize or even eliminate communication errors. The basic difference between the NORMAL and ENGINEERING Mode is the digital output representation of the signal processing. Therefore in the ENGINEERING MODE we are sending two more variables that are representing the Active and Reference Peak-to-Peak voltages of the signals accordingly.

The Faults in both cases are extracted and can help us identify critical errors of the Gas Sensor itself. With the help of the CRC implementation in the Integrated IR can be used into a sensing network system able to interact with the environment via sophisticated PC software.

Please see the Table 2 below for more information about the digital output string depending on the different modes available.

Table 2 - Digital Output, UART String

Mode	String
"NORMAL" MODE	[// Start Character 0x000005B (HEX) 0xAAAAAAAA // Gas concentration in PPM (HEX) 0xAAAAAAAA // Faults (HEX) 0xAAAAAAAA // Sensor Temperature (HEX) 0xAAAAAAAA // CRC 0xAAAAAAAA // 1's Compliment of CRC] // End Character 0x000005Du (HEX)
"ENGINEERING" MODE	[// Start Character 0x000005Bu (HEX) 0xAAAAAAAA // Gas concentration in PPM (HEX) 0xAAAAAAAA // Faults (HEX) 0xAAAAAAAA // Sensor Temperature (HEX) 0xAAAAAAAA // Reference 1 sec Average value 0xAAAAAAAA // Active 1 sec Average value 0xAAAAAAAA // CRC 0xAAAAAAAA // 1's Compliment of CRC] // End Character 0x000005D (HEX)
"CONFIGURATION" MODE	The UART (Digital Output) and the DAC (Analogue Output) are disabled but communication is fully functional.
NOTE: Obviously the 0xAAAAAAAA hex number is an example representing the 32-bit numbers that are being transmitted via the UART and is different in every case depending on the actual value that is representing, for example a concentration of 500 PPM will be 0x000001F4 in hex.	

1.1.3 Commands to the UART

The commands that we can send to the UART are specified in the Table 3 below. All the commands must be initialized with the character "["and end with the character "]", using capital block letters all the time. The implementation of the commands on Table 3 can be used to communicate and perform tasks, in both high and low applications. By using a MCU the INIR effectively is a standalone device that can also be implemented in any network based system design. INIR is not a standalone gas sensing instrument, still remains a digital gas sensor and should be handled as a component because is clustered like that.

Table 3 - INIR NEW UART Commands

UART Command	HEX Code	Meaning of the Command	Detailed Description
[A]	[0x41]	**Enter Normal Mode**	Puts INIR into NORMAL MODE. In this Mode we cannot write or read the settings and the Active and Reference signals are not being transmitted via UART. Can Perform Commands: [B], [C], [E], [F], [G], [L], [M]
[B]	[0x42]	**Enter Engineering Mode**	Put the INIR into ENGINEERING MODE. In this mode the Smart IR is calculating everything. The Active and Reference signal values are also exported via UART. Can Perform Commands: [A],[B], [C], [E], [F], [G], [L], [M]
[C]	[0x43]	**Enter Configuration Mode**	The INIR is not calculating anything; the UART and DAC are disabled. Can Perform Commands: [A], [B], [C], [I], [J], [K], [L], [M], [N]
[D]	[0x44]	**NOT USED**	Reserved for future used, no effect.
[E]	[0x45]	**Calibrate New Zero**	Calibrates the new ZERO value. The value will be stored in the Flash/EE memory of the sensor automatically. Can be performed under NORMAL or ENGINEERING modes.
[F]	[0x46]	**Calibrate New High Span**	Calibrate new High SPAN value. The value will be stored in the Flash/EE memory of the sensor automatically. Can be performed in NORMAL or ENGINEERING modes
[G]	[0x47]	**Calculate New Offset**	Calculates an offset value that is being used to improve the precision and linearity in low concentrations. Can be performed in NORMAL / ENGINEERING modes.
[H]	[0x48]	**NOT USED**	Reserved for future used, no effect.
[I]	[0x49]	**Read Back the Settings**	Send this command to the INIR if you want to read the settings back. All the settings are in a hex form representing integer values only.*3, see Table 4.0. Can only be performed under CONFIGURATION mode.
[J]	[0x4A]	**Load new settings to INIR**	Write the new setting to the device, all the settings are in a hex form representing integer values only, see paragraph 1.2.10. Can only be performed under CONFIGURATION mode.
[K]	[0x4B]	**RESET to Factory Default Values**	Reset all the variables into the Factory Defaults and erase all the custom settings that customer have changed like the calibration or the coefficients. Can only be performed under CONFIGURATION mode.
[L]	[0x4C]	**Turn Humidity Algorithm "ON"**	If we send this command the Humidity Compensation Algorithm would be "ON". Can be performed in all the modes. Setting saved into the Flash Memory.
[M]	[0x4D]	**Turn Humidity Algorithm "OFF"**	If we send this command the Humidity Compensation Algorithm would be "OFF". Can be performed in all the modes. Setting saved into the Flash Memory.
[N]	[0x4E]	**Download new coefficient**	Customer can send/update only one coefficient into the MCU. If you want to update the entire coefficients at one go please use command [J] above. Eg:[N0000000300000006] -> [Nvariable_numbervariable_value]
[O]	[0x4F]	**NOT USED**	Reserved for future used, no effect.

NOTEs:
1. All the command must start with ("["= 0x0000005B) and finish with ("]"= 0x0000005D), for the UART to work.
2. The Error in the communication protocol UART and any device running at maximum 38400 Baud rate is 0.015%.
3. see Table 8 – Read Back the Settings from the INIR

1.1.4 Initialization Procedure for INIR

In order to retain the compatibility with any past and future versions of the INIR firmware we are recommending to our customers to always following the procedure below. Please follow the same procedure every time the INIR is powered on or we had a reset for whatever reason.

Initialization Procedure for INIR

1. After each power "ON", send COMMAND [C], put the sensor into Configuration Mode.
2. Send COMMAND [I], read back all the settings of the INIR. This step is recommended so you can make a sanity check, CRC validation and confirm correct operation of the UART settings etc.
3. After the above steps, send COMMAND [B], put the sensor into Engineering Mode.
4. Once the step 3 has been executed the sensor will automatically start sending the data around every second.
5. It is recommended after the above steps to leave the sensor to warm up for the required time specified inside the INIR Datasheet, 45 seconds.
6. From this step onwards the sensor can operate typically and perform any command given etc.

End of Initialization Procedure

In the past there were versions of the firmware and in the future this may also change as well, where the customer doesn't have to implement the initialization procedure for the INIR. Nevertheless from a safety point of view just in case something goes wrong we are strongly recommending to our customers to always implement the functionality check, hence initialization procedure as described in 1.1.4 paragraph. This way each time the instrument will reset the communication will be verified and the sanity check of the coefficients will validate the correct operation of the Flash/EE.

Example of where the Initialization procedure would be extremely useful is in a mining situation where the Instrument keeps triggering and alarm every time is going into a reset because the coefficients are all zero. In this case yes the calculations may not be correct and other fault codes will be appearing in the Error generation routine but it will save much trouble if the instrument manufacturer would have implement the Initialization technique in the first place and found the problem the minute the instrument restarted from a power on reset or other reasons.

Last but not least we should always remember to use the CRC for eliminating any communication errors with the INIR, please read carefully the 1.7 Calculate the CRC for more information on this operation. Using the Initialization Procedure and the CRC validation the overall errors from communication or loss of information will be down to zero.

1.1.5 Answers from the UART

Below we can see the set of Readings coming as an answer from the INIR when we perform the command [I] -> Read Back Readings, under CONFIGURATION mode, see §1.2.9. Every variable is being stored as an integer and customers must keep them in the same format, otherwise the Integrated IR will not function properly. Please see the Table 8 – Read Back the Settings from the Integrated IR and Table 4 for examples on how to convert from the readings to the actual normal values of the data including multipliers needed.

Table 4 - Integrated IR Variables & Multipliers[a,b]

Number	Variable Type	Variable Name	Example Value Stored as Integer	Multiplier	Detailed Description
1	uint32	sensor_type[c]	26	*1	Holding the Gas Sensor Type like eg. INIR-ME100%
2	uint32	gas_type[d]	0.0	*1	Holding the Gas Type eg. Methane, Propane etc…
3	uint32	conc_range	100.0	*10000	Full Scale Target for Analogue Output Calculations
4	float32	high_span_gas_conc	100.0	*10000	High Span gas Cylinder Conc. in %v.v (eg. 20%v.v)
5	float32	low_span_gas_conc	2.0	*10000	Low Span gas Cylinder Conc. In %v.v (eg 2%v.v)
6	float32	a_coeff_low_range[e]	0.2670	*1000000	"a" coefficient for low range of the target gas
7	float32	a_coeff_mid_range	0.0563	*1000000	"a" coefficient for middle range of the target gas
8	float32	a_coeff_high_range	0.0563	*1000000	"a" coefficient for high range of the target gas
9	float32	n_coeff_low_conc	0.7250	*1000000	"n" coefficient for low range of the target gas
10	float32	n_coeff_mid_conc	0.4970	*1000000	"n" coefficient for middle range of the target gas
11	float32	n_coeff_high_conc	0.4970	*1000000	"n" coefficient for high range of the target gas
12	float32	betaneg_coeff_low_range	-0.137	*1000000	"beta" negative coefficient for low range
13	float32	betaneg_coeff_mid_range	-0.137	*1000000	"beta" negative coefficient for middle range
14	float32	betaneg_coeff_high_range	-0.137	*1000000	"beta" negative coefficient for high range
15	float32	betapos_coeff_low_range	-0.106	*1000000	"beta" positive coefficient for low range
16	float32	betapos_coeff_mid_range	-0.106	*1000000	"beta" positive coefficient for middle range
17	float32	betapos_coeff_high_range	-0.106	*1000000	"beta" positive coefficient for high range
18	float32	alphaneg_coeff	0.000235	*1000000	"alpha" negative coefficient
19	float32	alphapos_coeff	0.000363	*1000000	"alpha" positive coefficient
20	uint32	averaging	12	*1	The number of averaging seconds in signal processing
21	uint32	baud_rate	38400	*1	The baud rate to run the UART communication
22	uint32	current_conc_range	0	*1	LOW_CONC = 0 , MID_CONC = 1 and HIGH_CONC = 2
23	uint32	customer_calibration_time	123032	*1	Set time in format: hhmmss, calibration time
24	uint32	customer_calibration_date	240615	*1	Set date in format: DDMMYY, calibration date
	uint32	serial_number	240614001	*1	Serial number of the device, eg. 060614001
	uint32	time_delay_ms	25	*1	Delay in ms of the signal before reaches the peak
	uint32	firmware_version	430	*1	Holding the firmware version and is fixed
	float32	Act_1s_Average_Calibrate	0.001200	*1000000	Calibrated Active 1s second average value
	float32	Ref_1s_Average_Calibrate	0.001200	*1000000	Calibrated Reference 1s second average value
	float32	zero	1.0	*1000000	Calibrated value of Zero, when calibration done
	float32	span	4.0	*1000000	Calibrated value of Span, when calibration done
	float32	offset	0.000	*10000	Offset Concentration in %v.v, precision in LOW Range
	uint32	calibration_temperature	293.1	*10	Calibration temperature stored in (Kelvin * 10)

a Marked with Green are the values that Customer CAN READ, WRITE, CHANGE.
Marked with RED are the values that Customer CAN READ but NOT WRITE/CHANGE, overwriting those values will have no effect.
b All Readings are transmitted as integer numbers, in order to convert the integer into the normal value just divide by the multiplier. eg.
high_span_gas_conc = 100 *10000 = 1000000
Convert high_span_gas_conc = 1000000 / 10000 = 100 32-bit floating number
The INIR Evaluation Software is doing all the calculations automatically.
c The sensor type is represented by specific number look at Table 5 – Number Equivalent for Sensor Type.
d The gas type is represented by specific number look at Table 6 – Number Equivalent for Gas Type.
e For the coefficient table please look at Appendix A – Table of Coefficients.

Please find attached below in the Table 5 the Equivalent Number for the Sensor Type in order to translate the information when you want to read or write the coefficients.

Table 5 - Equivalent Number for Sensor Type

Sensor_type	Number Equivalent	Sensor_type	Number Equivalent	Sensor_type	Number Equivalent
IR11GM	0	IR23BD	15	IR602	30
IR11GM_1	1	IR14BD	16	IR603	31
IR11GJ	2	IR24BD	17	IR604	32
IR11EJ	3	IR31BC	18	Reserved for future	33
IR11EM	4	IR32BC	19	Reserved for future	34
IR12GM	5	IR33BC	20	Reserved for future	35
IR12GM_1	6	IR34BC	21	Reserved for future	36
IR12GJ	7	TWIN_GAS_SENSOR	22	Reserved for future	37
IR12EJ	8	INIR-CD	23		
IR11BD	9	INIRCD_H	24		
IR21BD	10	INIRME_H	25		
IR11BR	11	INIR-ME	26		
IR12BD	12	INIR12PR	27		
IR22BD	13	INIR12BU	28		
IR13BD	14	IR601	29		

Please find below in Table 6, the Equivalent Number used for the Gas Type in the variable position 2 in order to translate the equivalent number when you want to read or write the coefficients.

Table 6 - Equivalent Number for Gas Type

Gas_type	Number Equivalent	Gas_type	Number Equivalent	Gas_type	Number Equivalent
METHANE	0	Reserved	4	Reserved	8
PROPANE	1	Reserved	5	Reserved	9
BUTANE	2	Reserved	6	Reserved	10
CARBON DIOXIDE	3	Reserved	7	Reserved	11

Below in the Table 7 – Answers from the INIR UART we can find attached the possible answers from the INIR via the UART communication port after we have performed a command.

Table 7 - Answers from the INIR UART

UART Answer	HEX Code	Meaning of the Answer	Detailed Description
[AK]	0x00414B00	**ACKNOWLEDGEMENT**	Everything went OK, if we have sent a command that the command has been executed correctly.
[NA]	0x004E4100	**NO ACKNOWLEDGEMENT**	Something went wrong; if we have sent a command or the command has not been executed correctly.
NOTE: All the commands have a starting character a starting ("["= 0x5B00000) and a finishing character ("]"= 0x000005D). So for example [AK] => 0x5B414B5D in HEX.			

1.2 Example of Communication between Integrated IR & Custom Device

The following example for communicating with the INIR Gas Sensor have been created to help customers use the device assuming that is connected with another device via UART, see Figure 1 below.

We would advise our customers to send the commands by sending each 32-bit hex number individually. Sending one string will make the process of debugging difficult and the technical support slower as extra time will be needed to translate the string of data from one system to another. It would be advisable to use a function to send messages via UART for unsigned/signed 32-bit HEX numbers based on the commands, see table 3 and functions 1 & 2. We can identify the received message from INIR based on tables 2, 7 & §1.2.9.

Please read chapter 2 following for examples on how to read and write to the INIR using C code.

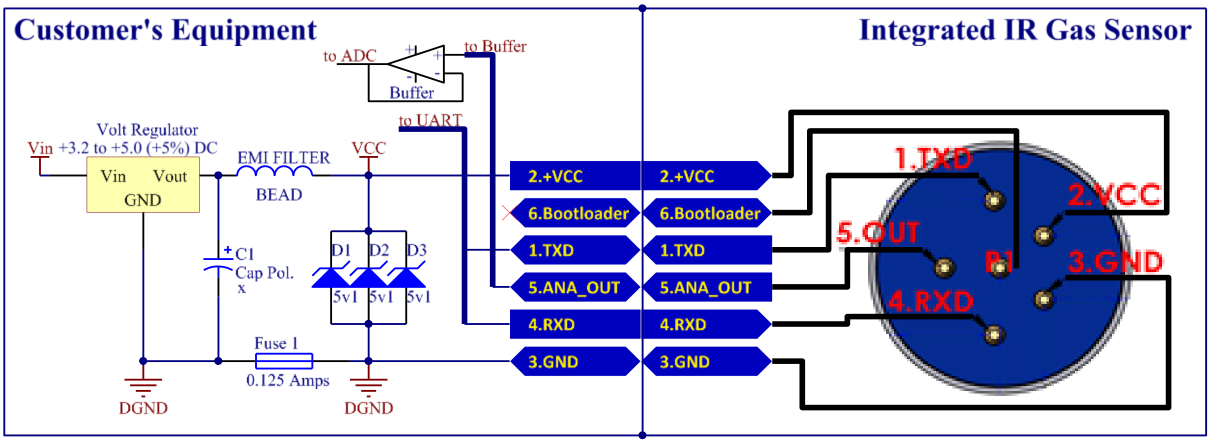


Figure 1 -Schematic Diagram for Communicating with the INIR

1.2.1 COMMAND [A], NORMAL Mode

See below how to enter NORMAL Mode:

COMMAND	HEX	ASCII
Send Com.	0x005B415D	[A]
Get Answer	0x5B414B5D	[AK] , everything OK
Or Answer	0x5B4E415D	[NA], went wrong

When we are in this mode we can only perform the following commands:

[A], [B], [C], [E], [F], [G], [L] and [M].

1.2.2 COMMAND [B], Enter ENGINERING Mode

See example below how to enter ENGINEERING Mode:

COMMAND	HEX	ASCII
Send Com.	0x005B425D	[B]
Get Answer	0x5B414B5D	[AK], everything OK
Or Answer	0x5B4E415D	[NA], went wrong

When we are in this mode we can only perform the following commands:

[A], [B],[C], [E], [F], [G], [L] and [M].

1.2.3 COMMAND [C], Enter CONFIGURATION Mode

Only in Configuration Mode we can change the settings. In this Mode no data are transmitted from the INIR, DAC is disabled. See example below how to enter CONFIGURATION Mode:

COMMAND	HEX	ASCII
Send Com.	0x005B435D	[C]
Get Answer	0x5B414B5D	[AK], everything OK
Or Answer	0x5B4E415D	[NA], went wrong

When we are in this mode we can only perform the following commands:

[A], [B], [C], [I], [J], [K], [L], [M] and [N].

1.2.4 COMMAND [D], Reserved

This command has been reserved for future use; no effect on the Integrated IR (INIR), no function performed if you sent this command to the INIR.

COMMAND	HEX	ASCII
Send Com.	0x005B445D	[D]
Answer	0x5B4E415D	[NA],not executed

1.2.5 COMMAND [E], ZERO Calibration (for NORMAL/ENGINEERING Modes)

In order to perform ZERO Calibration we need to be in the NORMAL or ENGINEERING Mode. Please also read Chapter 3 – New Calibration Routine for more information on how you should perform the ZERO calibration.

Before we execute the process make sure we have completed the following steps:

1. Set INIR in CONFIGURATION Mode
2. Read Back Settings command [I], see §1.2.9
3. Load NEW Settings command [J], see §1.2.10
4. Set INIR in NORMAL or ENGINEERING Mode, see §1.2.1 or §1.2.2.
5. Apply Zero target Gas eg. 100% Nitrogen, wait for the signals to stabilize for at least 30 mins after power on.
6. See below how to perform Zero Calibration

COMMAND	HEX	ASCII
Send Com.	0x005B455D	[E]
Get Answer	0x5B414B5D	[AK], everything OK
Or Answer	0x5B4E415D	[NA], went wrong

1.2.6 COMMAND [F], SPAN Calibration (for NORMAL/ENGINERRING Modes)

In order to perform SPAN we need to be in the NORMAL or ENGINEERING Mode. No need to perform calibration in ranges as the Smart IR is a single range operating Module with automatic switchover between ranges depending on Gas Concentration. Please also read Chapter 2.0 – New Calibration Routine explaining how to perform SPAN calibration.

Before we execute the process make sure we have completed the following steps:

1. We should be already in NORMAL or ENGINEERING Mode, see 1.2.1 or 1.2.2.
2. Make sure we have performed the ZERO Calibration first, see paragraph 1.2.5
3. Apply SPAN target Gas eg. 20% Methane, wait for the signals to stabilize
4. Perform SPAN Calibration:

COMMAND	HEX	ASCII
Send Com.	0x005B465D	[F]
Get Answer	0x5B414B5D	[AK], everything OK
Or Answer	0x5B4E415D	[NA], went wrong

1.2.7 COMMAND [G], OFFSET Calculation (NORMAL or ENGINERRING Modes)

In order to calculate OFFSET we need to be in the NORMAL or ENGINEERING Mode. No need to perform calibration in ranges as the INIR is a single range operating Module with automatic switchover between ranges depending on Gas Concentration. Please read Chapter 2.0 – New Calibration Routine explaining how to perform the OFFSET calibration.

Before we execute the process make sure we have completed the following steps:

1. We should already be in NORMAL or ENGINEERING Mode, see 1.2.1 or 1.2.2.
2. Performs the ZERO,SPAN Calibration first, see paragraph 1.2.5 and 1.2.6.
3. Apply OFFSET target Gas eg. 2% Methane, wait for the signals to stabilize
4. Perform Offset Calculation, see below:

COMMAND	HEX	ASCII
Send Com.	0x005B475D	[G]
Get Answer	0x5B414B5D	[AK], everything OK
Or Answer	0x5B4E415D	[NA], went wrong

1.2.8 COMMAND [H], Reserved

This command has been reserved for future use; no effect on the Smart IR, no function performed if you do sent this command to the MCU.

COMMAND	HEX	ASCII
Send Com.	0x005B485D	[D]
Answer	0x5B4E415D	[NA],not executed

1.2.9 COMMAND [I], Read Back Settings (only in CONFIGURATION Mode)

See example below how to perform commands under the CONFIGURATION Mode:

COMMAND	HEX	ASCII
Send Command	0x005B495D	[I]

Get ANSWER -> **Table 8 – Answers from the Integrated IR via UART**

Note: In this example the values could be different from the ones displayed in the table if the customer has already changed the values or SGX has performed calibration on the INIR. Those values are just an example of how the data transmitted via the UART.

Table 8 - Reading Back Settings in Configuration Mode

Number	HEX Number	Variable Name	Example of Answer in Decimal Format	Divider	Detailed Description
	0x0000005B		[		**Start String Character**
1	0x00000006	sensor_type[c]	=6	/1	Holding the Gas Sensor Type like eg.IR12GM_1
2	0x00000000	gas_type[d]	=0	/1	Holding the Gas Type eg. Methane, Propane etc...
3	0x000F4240	conc_range	=1000000	/1	Full Scale Target for Analogue Output Calculations
4	0x000F4240	high_span_gas_conc	=1000000	/10000	High Span gas Cylinder Conc. in ppm (eg. 20000 ppm)
5	0x00004E20	low_span_gas_conc	=20000	/10000	Low Span gas Cylinder Conc. In ppm (eg 2000 ppm)
6	0x000412F8	a_coeff_low_range[e]	=267000	/1000000	"a" coefficient for low range of the target gas
7	0x0000DBEC	a_coeff_mid_range	=56300	/1000000	"a" coefficient for middle range of the target gas
8	0x0000DBEC	a_coeff_high_range	=56300	/1000000	"a" coefficient for high range of the target gas
9	0x000B1008	n_coeff_low_conc	=725000	/1000000	"n" coefficient for low range of the target gas
10	0x00079568	n_coeff_mid_conc	=497000	/1000000	"n" coefficient for middle range of the target gas
11	0x00079568	n_coeff_high_conc	=497000	/1000000	"n" coefficient for high range of the target gas
12	0xFFFDE8D8	betaneg_coeff_low_range	=-137000	/1000000	"beta" negative coefficient for low range
13	0xFFFDE8D8	betaneg_coeff_mid_range	=-137000	/1000000	"beta" negative coefficient for middle range
14	0xFFFDE8D8	betaneg_coeff_high_range	=-137000	/1000000	"beta" negative coefficient for high range
15	0xFFFE61F0	betapos_coeff_low_range	=-106000	/1000000	"beta" positive coefficient for low range
16	0xFFFE61F0	betapos_coeff_mid_range	=-106000	/1000000	"beta" positive coefficient for middle range
17	0xFFFE61F0	betapos_coeff_high_range	=-106000	/1000000	"beta" positive coefficient for high range
18	0x000000EB	alphaneg_coeff	=235	/1000000	"alpha" negative coefficient
19	0x0000016B	alphapos_coeff	=363	/1000000	"alpha" positive coefficient
20	0x0000000A	averaging	=10	/1	Used for the Active and Reference signal processing
21	0x00009600	baud_rate	=38400	/1	The baud rate to run the UART communication
22	0x00000000	current_conc_range	=0	/1	LOW_CONC = 0 , MID_CONC = 1 and HIGH_CONC = 2
23	0x00019465	customer_calibration_time	=103525	/1	Set time in format: hhmmss, calibration time
24	0x00030FA6	customer_calibration_date	=200614	/1	Set date in format: DDMMYY, calibration date
	0x00000001	serial_number	=1	/1	Serial number of the device, eg. 060614001
	0x00000019	time_delay_ms	=25	/1	Delay in ms of the signal before reaches the peak
	0x00000190	firmware_version	=400	/1	Holding the firmware version and is fixed
	0x000034BC	Act_1s_Average_Calibrate	=13500	/1000000	Calibrated Active 1s second average value
	0x00003458	Ref_1s_Average_Calibrate	=13400	/1000000	Calibrated Reference 1s second average value
	0x0010C8E0	zero	=1100000	/1000000	Calibrated value of Zero, when calibration done
	0x0006DDD0	span	=450000	/1000000	Calibrated value of Span, when calibration done
	0x00000000	offset	=0.000	/10000	Calculated Offset Concentration in ppm, precision in LOW
	0x00000B73	calibration_temperature	=2931	/10	Calibration temperature stored in Kelvin * 10 as an integer
		val_crc	=	/1	CRC
		Inv_crc	=	/1	1's Compliment of CRC
	0x0000005D		]		**End String Character**

a Marked with Green are the values that Customer CAN READ, WRITE, CHANGE.
Marked with RED are the values that Customer CAN READ but NOT WRITE/CHANGE, overwriting those values will have no effect.
b All Readings are transmitted as integer numbers, in order to convert the integer into the normal value just divide by the multiplier. eg.
high_span_gas_conc = 100 *10000 = 1000000
Convert high_span_gas_conc = 1000000 / 10000 = 100 32-bit floating number
The Smart IR Evaluation Software is doing all the calculations automatically.
c The sensor type is represented by specific number look at Table 5.0 – Number Equivalent for Sensor Type.
d The gas type is represented by specific number look at Table 6.0 – Number Equivalent for Gas Type.
e For the coefficient table please look at Appendix A – Table of Coefficients.

NOTE:In this example the values could be different from the ones displayed in the table if the customer has already changed the values or SGX has performed calibration on the INIR. Those values are just an example of how the data transmitted via the UART.

1.2.10 COMMAND [J], Load NEW Settings (only in CONFIGURATION Mode)

COMMAND	HEX	ASCII
Send Command	See below	See below

In order for us to be able to Load NEW Settings to the Smart IR the module must be in the CONFIGURATION MODE.

For that process please complete the following:

1. Perform COMMAND [C], Enter CONFIG. Mode
2. Perform COMMAND [I], Read Back Settings, it is advisable to always get back the existing settings before you attempt to change them.After the above steps now perform COMMAND [J], Load NEW Settings.

SEND SETTINGS -> see Table 9 – Loading New Settings/Coefficients to INIR.

The above commands can be send as a single cascaded string. Please observe below that the order of the variables should be correct and would be exactly the same like sending the commands one by one. There is an example below:

"[J0x000000060x000000000x000F42400x000F4240 ……………………………….. 0x00030FA6**]"**

Table 9 - Loading New Settings/Coefficients to INIR

HEX Number SEND	Variable Name	Example of Answer in Decimal Format	Divider	Detailed Description
0x0000005B		**[**		**Start String Character**
0x0000004A		J		Letter J, representing Command J load new data
0x00000024	sensor_type[c]	=6	/1	Holding the Gas Sensor Type like eg.IR12GM_1
0x00000000	gas_type[d]	=0	/1	Holding the Gas Type eg. Methane, Propane etc…
0x000F4240	conc_range	=100	/10000	Full Scale Target for Analogue Output Calculations
0x000F4240	high_span_gas_conc	=1000000	/10000	High Span gas Cylinder Conc. in ppm (eg. 20000 ppm)
0x00004E20	low_span_gas_conc	=20000	/1000000	Low Span gas Cylinder Conc. In ppm (eg 2000 ppm)
0x000412F8	a_coeff_low_range[e]	=267000	/1000000	"a" coefficient for low range of the target gas
0x0000DBEC	a_coeff_mid_range	=56300	/1000000	"a" coefficient for middle range of the target gas
0x0000DBEC	a_coeff_high_range	=56300	/1000000	"a" coefficient for high range of the target gas
0x000B1008	n_coeff_low_conc	=725000	/1000000	"n" coefficient for low range of the target gas
0x00079568	n_coeff_mid_conc	=497000	/1000000	"n" coefficient for middle range of the target gas
0x00079568	n_coeff_high_conc	=497000	/1000000	"n" coefficient for high range of the target gas
0xFFFDE8D8	betaneg_coeff_low_range	=-137000	/1000000	"beta" negative coefficient for low range
0xFFFDE8D8	betaneg_coeff_mid_range	=-137000	/1000000	"beta" negative coefficient for middle range
0xFFFDE8D8	betaneg_coeff_high_range	=-137000	/1000000	"beta" negative coefficient for high range
0xFFFE61F0	betapos_coeff_low_range	=-106000	/1000000	"beta" positive coefficient for low range
0xFFFE61F0	betapos_coeff_mid_range	=-106000	/1000000	"beta" positive coefficient for middle range
0xFFFE61F0	betapos_coeff_high_range	=-106000	/1000000	"beta" positive coefficient for high range
0x000000EB	alphaneg_coeff	=235	/1000000	"alpha" negative coefficient
0x0000016B	alphapos_coeff	=363	/1000000	"alpha" positive coefficient
0x0000000A	averaging	=10	/1	Used for the Active and Reference signal processing
0x00009600	baud_rate	=38400	/1	The baud rate to run the UART communication
0x00000000	current_conc_range	=0	/1	LOW_CONC = 0 , MID_CONC = 1 and HIGH_CONC = 2
0x00019465	customer_calibration_time	=103525	/1	Set time in format: hhmmss, calibration time
0x00030FA6	customer_calibration_date	=200614	/1	Set date in format: DDMMYY, calibration date
0x0000005D		**]**		**End String Character**

a Marked with Green are the values that Customer CAN READ, WRITE, CHANGE.
b All Readings are transmitted as integer numbers, in order to convert the integer into the normal value just divide by the multiplier. eg.
high_span_gas_conc = 100 *10000 = 1000000
Convert high_span_gas_conc = 1000000 / 10000 = 100 32-bit floating number
The Smart IR Evaluation Software is doing all the calculations automatically.
c The sensor type is represented by specific number look at Table 5.0 – Number Equivalent for Sensor Type.
d The gas type is represented by specific number look at Table 6.0 – Number Equivalent for Gas Type.
e For the coefficient table please look at Appendix A – Table of Coefficients.
NOTE: In this example the values could be different from the ones displayed in the table if the customer has already changed the values or SGX has performed calibration on the INIR. Those values are just an example of how the data transmitted via the UART.

1.2.11 COMMAND [K], RESET to FACTORY Defaults (only in CONFIGURATION Mode)

In order to RESET to Factory Default values we need to be in the CONFIGURATION Mode. The Integrated IR has and internal Repository that is holding the Factory Default values that was calculated during Factory Production/Calibration and cannot be erased unless a "mass erase" command is send to the microprocessor, not used by customers.

Performing this command will also erase the values that we had potentially changed later and also the other calibrations done after initial production.

COMMAND	HEX	ASCII
Send Com.	0x005B4B5D	[K]
Get Answer	0x5B414B5D	[AK],everything OK
Or Answer	0x5B4E415D	[NA], went wrong

WARNING!

Performing this command will erase the custom values changed and also any calibrations we have done after factory production. Proceed with this command carefully and is recommended always to do a valid re-calibration after that command has being executed.

1.2.12 COMMAND [L], TURN "ON" Humidity Algorithm (all Modes)

This command can be performed in all the modes, when is executed this command will switch "ON" the Humidity Compensation Algorithm that will minimize the response of the sensor due to condensation or humidity problems. This algorithm will reduce the recovery time from condensation effects as well.

COMMAND	HEX	ASCII
Send Com.	0x005B4C5D	[L]
Get Answer	0x5B414B5D	[AK], everything OK
Or Answer	0x5B4E415D	[NA], went wrong

1.2.13 COMMAND [M], TURN "OFF" Humidity Algorithm (all Modes)

This command can be performed in every mode, when executed it will switch "OFF" the Humidity Compensation Algorithm. Disabling the algorithm will result to a much higher Concentration response due to condensation or humidity.

COMMAND	HEX	ASCII
Send Com.	0x005B4D5D	[M]
Get Answer	0x5B414B5D	[AK],everything OK
Or Answer	0x5B4E415D	[NA], went wrong

1.2.14 COMMAND [N], Reserved

This command can be performed in CONFIG MODE only, when executed it will send only one coefficient to the MCU.

COMMAND	HEX	ASCII
Send Com.	0x005B4E5D	[Nnumbervalue]
Get Answer	0x5B414B5D	[AK],everything OK
Or Answer	0x5B4E415D	[NA], went wrong

Where **number,** please use the coefficient number, see table 4 or 8. Where **value** please use an appropriate value that you want to load.

1.2.15 COMMAND [O], Reserved

This command has been reserved for future use; no effect on the Integrated IR, no function performed if you do sent this command to the MCU.

1.3 Changing Between Low/Mid/High Concentration Ranges

From Firmware version 4v0 the Integrated IR is using automatic range switchover and the customer doesn't need to send external commands. If you are using the a Integrated IR with older versions like 3v1 or 2v5 please read the AN1 – Smart IR Protocol & Calibration Routine, or download at: http://www.sgxsensortech.com.

1.4 Convert Temperature in KELVIN to CELCIUS

The internal temperature that is extracted with the digital string from the Integrated IR is in the following format : (Kelvin*10)

- If we want to convert that temperature into Kelvin just divide that number by 10, which is the scaling factor.

$$Temp(^{o}K)=\frac{\text{Temp.Reading from INIR}}{10}$$

- If we want to convert the temperature into Celsius then use the following equation:

$$°C = \left[\left(\frac{Value\ in\ Kelvin}{10}\right) - 273.15\right] in\ °C$$

Internal Temperature Readings from INIR always include a small error around

1.5 Calculate Conc v.v% by Reading the Analogue Output

The DAC output can vary from 0 to 2.5 Volts. In order to calculate the concentration from the Volts that we are reading we can use the following equation:

$$\%vol = \left(\frac{DACvolts}{1.25} - 1\right) * Range$$

where $Range = 2, 5, 100$ or whatever is the range in concentration of the sensor (see Appendix A – Table of Coefficients).

In the same way we can convert the %vol to the HEX number that the DAC in using to give as the output by using the following equation:

$$HEX = \left(\frac{\%vol * 2048}{Range}\right) + 2047$$

In the same way we can convert the Volts from the HEX number that the DAC in using to give as the output by using the following equation:

$$DACvolts = \left(\frac{HEX * 1.25}{2048}\right)$$

The maximum current that the DAC can source is 0.5 mA, based on a 5K resistive Load at 2.5Volts. There are some limitations in the concentration, if we are reading DACvolts = 0 then the DAC is stopped or not functioning. The DACvolts = 0.5 means that the Smart IR has a fault.

Anything between the 0.5 and 2.5 can represent the 100% Full Scale with 1.25 being the zero of the scale. The scale representing the DAC output can be found in Figure below.

IMPORTANT!

The precision in the Analogue Output is not as high as the Digital output, therefore is recommended to use at least a 12-bit ADC to read it.

Example 1: If Range = 5: 4 Floating points precision => 0.0024%vol/mV of DACvolts

Example 2: If Range = 100: 3 Floating points precision => 0.08%vol/mV of DACvolts

If Range = 100 : 4 Floating points precision => 0.048%vol/mV of DACvolts

NOTE: If we are using the Analogue Output and we have a fault, the output will be fixed at around 0.5 Volts DC, but there is no way to know the source of the problem. If the DAC is exactly 0 Volts DC then please make sure the connections are correct. If everything seems alright then we have to check if we are accidentally set the INIR into configuration mode.

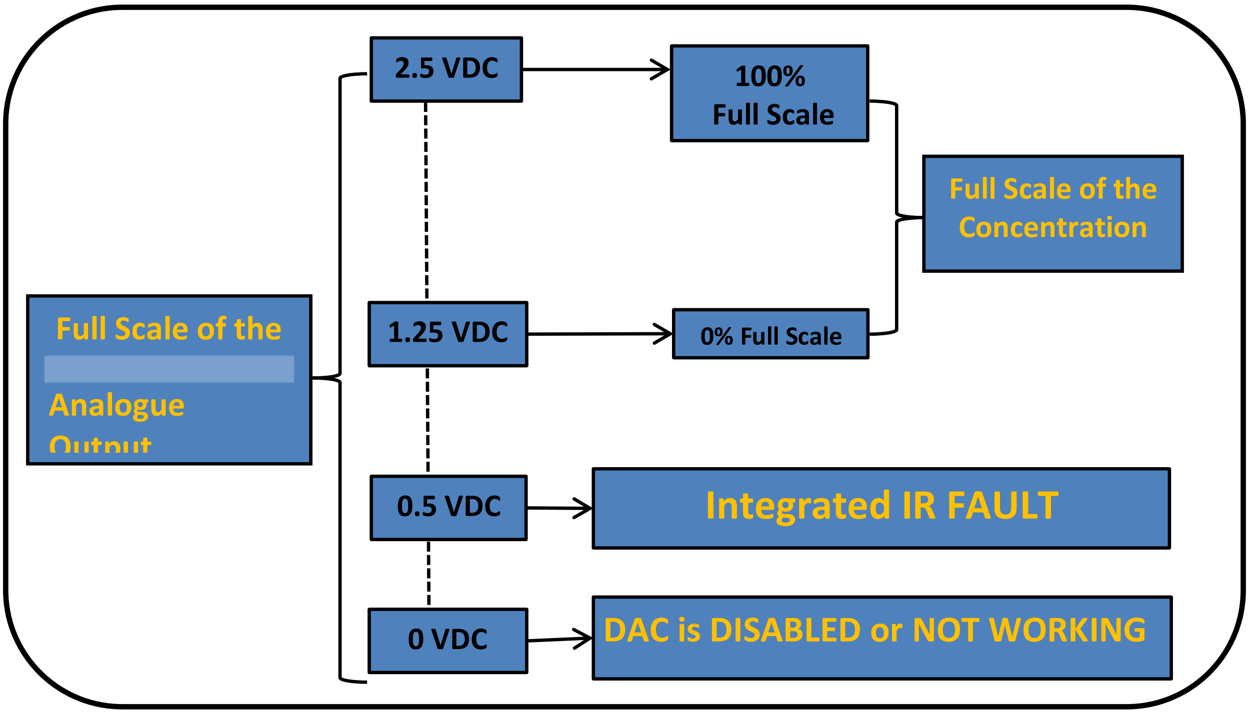


Figure 2 - Full Scale Representing the DAC Output

1.6 TRANSLATING THE "FAULTS" CODE

The Fault is a 32-bit unsigned integer number that is representing all the errors code or hardware related. The Fault is transmitted via UART like the Concentration. Every 4-bits of the Fault variable are representing levels of errors; we can simultaneously monitor 8 different parts of the system. The association between the Fault variable and the different errors can be found below in the Table 9.0 – Integrated IR Faults/Errors Interpretation.

To interpret the Fault, first we need to identify the Assigned Digit that we want to translate then the function/error that is representing from the column Associated Part and finally the Meaning of the error depending on the value of the digit.

Table 10 - Integrated IR Faults/Errors Interpretation

Variable	Associated Part	Assigned DIGIT	Error Meaning	Value UART
Fault	ALL Parts	**7 to 0**	NO ERROR IN ALL PARTS (EVERYTHING OK)*1	0xAAAAAAAA
ENABLED	Gas Sensor	**0**	NO ERROR	0x0000000A
			01-Sensor not Present	0x00000001
			02-Temperature sensor not working **OR** Device Temperature Out of the Operating Range	0x00000002
			03-Active or Reference are weak	0x00000003
			04-First Time Configuration Mode, no settings present	0x00000004
ENABLED	Power Related	**1**	NO ERROR	0x000000A0
			01-Last Reset was because of a Power on Reset	0x00000010
			02-Last Reset was because of a Watchdog Timer	0x00000020
			03-Last Reset was because of a Software Reset	0x00000030
			04-Last Reset was because of an External Pin Interrupt	0x00000040
			05-Not assigned yet	0x00000050
DISABLED	ADC Related	**2**	NO ERROR	0x00000A00
			01-ADC is switched off	0x00000100
			02-Weak Signals Received	0x00000200
			03-Not assigned yet	0x00000300
ENABLED	DAC Related	**3**	NO ERROR	0x0000A000
			01-DAC is switched off	0x00001000
			02-DAC output disable in Configuration mode	0x00002000
ENABLED	UART Related*2	**4**	NO ERROR	0x000A0000
			01-Break Indicator P1.0 set LOW for more than the maximum word length	0x00010000
			02-Framing Error, stop bit was invalid	0x00020000
			03-Parity Error, stop bit was invalid	0x00030000
			04-Overrun Error, data overwrite before being read	0x00040000
DISABLED	TIMERS Related	**5**	NO ERROR	0x00A00000
			01-Timer1 Error	0x00100000
			02-Timer2/Watchdog Error	0x00200000
			03-Not assigned yet	0x00300000
ENABLED	General Error	**6**	NO ERROR	0x0A000000
			01-Over Range of Conc.%v.v Operation > Full Scale	0x01000000
			02-Under Range of Conc.%v.v < 0.07%	0x02000000
ENABLED	MEMORY Related*3	**7**	NO ERROR	0xA0000000
			01-Unable to store Data, to the INIR	0x10000000
			02-Unable to read Data from the INIR	0x20000000
			03-Not assigned yet	0x30000000

*1: It normal to get a Fault = 0xAAAAAA1A because the "1" in the second digit representing the Power on Reset process of the MCU which is a normal operation when we turn on the Integrated IR.
*2: This is the function that will check the UART Status (COMSTA0) register, to produce a fault depending on the previous experience. Obviously if the error is serious we will not be able to receive the message via UART, but we can check it later to find out what caused it.
*3: The Memory will work correctly as long as the MCU is working but it will not be able to store the data in the flash upon reset, if the Error 01 is present. If we cannot read the Memory then check if the Smart IR is in CONFIGUATION MODE.

1.7 Calculating the Cyclic Redundancy Check (CRC), Validate Data

The Cyclic Redundancy Check (CRC) I a simple way to validate data transmitted via any protocol from one side of the communication channel to the other side. Every time we are transmitting data via the Universal Asynchronous Receive Transmit (UART) protocol there is a chance from the physical properties of the channel to have a change of error around 0.0017%. In addition to that percentage error there are other parameters that could implement a loss in information or even synchronization between the INIR and a custom made instrument or even with a PC. Strong Electromagnetic Fields or a very quick reset because of a power failure or a lighting strike could also create a temporary loss of information or communication. For the above reasons and to minimize even completely eradicate the communication errors or loss of information SGX has implement a custom CRC, which is basically a simple value carrying a checksum from all the data transmitted via UART within the 1 second period of time. In functions 1,2 we can also observe an example with C code on how SGX is calculating the CRC. By using the same Algorithm for those transmitted values you should get the sane CRC value, otherwise some of your data are not correct, most likely have been corrupted during transmission. Below we can find the Equation SGX is using to calculate the CRC and the 1s Compliment of the CRC as well.

Equation for CRC Calculation

$$\sum_{i=0}^{n} [\, X_k \gg (8 * i) \,]$$
$$where \;\; n = 1, 2, 3, 4$$

X_k: *Each number we are sending out.*
>> : is the right shift bit operator from the C programming language.

Example 1

Please see Function 3 – Example on how to send messages to the INIR (UART), this function is transmitting some data as the sensor is in Engineering Mode. The CRC check is calculated inside the UART_Send_Uint() and the UART_Send_Int() function 1, 2 in page 16 and 17 accordingly. The val_crc variable is holding the final summation of the CRC transmitted value and the inv_crc is the Bitwise Compliment of CRC.

inv_crc = $\sum_{i=0}^{n} [\, X_k \gg (8 * i) \,]$ $, where\ n =$ 1,2,3,4 and X_k $: Each\ number\ we\ are\ sending$

In case of the Engineering mode, Function 3 we just want to send:

For k=1: $\sum X1 = \sum_{i=0}^{n} [\, 0x000005Bu \gg (8 * i) \,]$
// START Character "[", counts in CRC.

For k=2: $\sum X2 = \sum_{i=0}^{n} [\, (Gas\_Conc\_PPM) \gg (8 * i) \,]$
// Gas Concentration in PPM.

For k=3: $\sum X3 = \sum_{i=0}^{n} [\, (uint32\_t)(Fault) \gg (8 * i) \,]$
// Output the Final Fault code for all the parts, counts in CRC

For k=4: $\sum X4 = \sum_{i=0}^{n} [\, (uint32\_t)(Temp.) \gg (8 * i) \,]$
// Send the Sensor Temperature, counts in CRC.

For k=5: $\sum X5 = \sum_{i=0}^{n} [\, (uint32\_t)(Ref) \gg (8 * i) \,]$
// Send the reference average value of the 1 second, counts in CRC.

For k=6: $\sum X6 = \sum_{i=0}^{n} [\, (uint32\_t)(Act) \gg (8 * i) \,]$
// Send the active average value of the 1 second, counts in CRC.

inv_crc = $\sum X1 + \sum X2 + \sum X3 + \sum X4 + \sum X5 + \sum X6;$
inv_crc = ~val_crc;
// Calculate Bitwise compliment of the CRC

val_crc
// SEND CRC, not contributing to CRC calculations

inv_crc
// SEND BITWISE COMPLIMENT OF CRC, not contributing to CRC.

0x000005Du
// END Character "]", we do not count it in CRC

1.7.1 Bitwise compliment operator

Bitwise compliment operator is an unary operator(works on one operand only). It changes the corresponding bit of the operand to opposite bit, i.e. 0 to 1 and 1 to 0. It is denoted by ~.

35=00100011 (In Binary)

Bitwise complement Operation of 35
~ 00100011
____
11011100 = 220 (In decimal)

<u>Twist in bitwise complement operator in C Programming</u>

Output of ~35 shown by compiler won't be 220, instead it shows -36. For any integer n, bitwise complement of n will be -(n+1). To understand this, you should understand the concept of 2's complement.

<u>2's Complement</u>

Two's complement is the operation on binary numbers which allows number to write it in different form. The 2's complement of number is equal to the complement of number plus 1.

For example:

Decimal	Binary	2's complement
0	00000000	-(11111111+1) = -00000000 = -0(dec)
1	00000001	-(11111110+1) = -11111111 = -256(dec)
12	00001100	-(11110011+1) = -11110100 = -244(dec)
220	11011100	-(00100011+1) = -00100100 = -36(dec)

Note: Overflow is ignored while computing 2's complement.

If we consider the bitwise complement of 35, 220(in decimal) is converted into 2's complement which is -36. Thus, the output shown by computer will be -36 instead of 220.

<u>How is bitwise complement of any number N=-(N+1)?</u>

Bitwise complement of

N= ~N (represented in 2's complement form)

2'complement of ~N= -(~(~N)+1) = -(N+1)

```
#include <stdio.h>
int main()
{
 printf("complement=%d\n",~35);
 printf("complement=%d\n",~-12);
 return 0;
}
```

Results on the Screen:
complement=-36
Output=11

❷ | Examples Implementation Functions in C

Function 1 - Example of Function to Send an Un-Signed Number to the UART

```
//-------------------------------------------------------------------------------------------------        //
// Function: int UART_Send_Uint(uint32_t data, uint8_t no_of_bytes)                                        //
//-------------------------------------------------------------------------------------------------        //
// Details: This is a function that outputs a 32bit unsigned integer to the UART                          //
//-------------------------------------------------------------------------------------------------        //
// Inputs: void                                                                                            //
//-------------------------------------------------------------------------------------------------        //
// Outputs: returns 0 -> everything OK                                                                     //
//-------------------------------------------------------------------------------------------------        //
// Calls: using register of ARM7                                                                           //
//-------------------------------------------------------------------------------------------------        //
int UART_Send_Uint (uint32_t data, uint8_t no_of_bytes)         // START OF UART_Send_Uint()
{
 static unsigned char szTemp[10] = "";                          // Used to store result before printing to UART
 static unsigned char z, i          = 0;
 static unsigned char nLen          = 0;                        // Used to store the length of the string of data to be transmitted
 static uint32_t temp_data          = 0;

 temp_data = data;                                              // Load data to a temporary variable buffer
 for ( z = 0 ; z < 4 ; z++ )                                    // Load Array
 {
  val_crc += (uint8_t)temp_data;                                // Using an 8-bit unsigned buffer to calculate the value of the CRC
  temp_data = temp_data >> 8;                                   // Shift data by 8, we do that 4 times, 4 x 8 = 32 bit number
 }

 sprintf ( (char*)szTemp, "%08x\n\r", data );                   // Convert all the uint_32t data to a string
 nLen = strlen((char*)szTemp);                                  // Get the length of the string to be transmitted

for ( i = 0 ; i < nLen ; i++ )                                  // Loop to send String depending on the Length
 {
 COMTX = szTemp[i];                                             // Send String to the UART
 while ((COMSTA0 & THRE) == 0x00);                              // Wait for Tx buffer empty bit to be set
 }

 return(0);                                                     // return zero value if everything went OK
}                                                               // END OF UART_Send_Uint()
```

Comments/Functionality:

The above function is an example written in C code for sending an un-signed integer via the UART. The user must provide the function with the unsigned integer number and the number of bytes for that number. The transmission is done after the val_crc has been calculated that variable will be updated each time a number is transmitted until the end character ']'.

The whole number will be transmitted via UART character by character using the COMTX register which is the buffer Transmit (TX) register of the UART depending on the MCU you are using and the technology. We are always checking if the Tx buffer is empty before we send the next character to avoid any overflow and loss of information.

Please observe the sprintf ((char*)szTemp, "%08x\n\r", data); line, because is very important, as is converting the 32-Bit data number into a "string" using the hex form of that hex number. That means that the actual hex number will be transmitted as an ASCII string, eg "0x0000001A".

Function 2 Example Function Sending an Integer Number to the UART

```
//-----------------------------------------------------------------------------------------------------------   //
// Function: int UART_Send_Int(int32_t data, uint8_t no_of_bytes)                                             //
//-----------------------------------------------------------------------------------------------------------   //
// Details: This is a function that outputs a 32bit signed integer to the UART                              //
//-----------------------------------------------------------------------------------------------------------   //
// Inputs: void                                                                                                //
//-----------------------------------------------------------------------------------------------------------   //
// Outputs:returns 0 -> everything OK                                                                          //
//-----------------------------------------------------------------------------------------------------------   //
// Calls: using register of ARM7                                                                               //
//-----------------------------------------------------------------------------------------------------------   //
int UART_Send_Int(int32_t data, uint8_t no_of_bytes)            // START OF UART_Send_Int()
{
 static signed char szTemp[10]    = "";
 static signed char i              = 0;
 static signed char nLen           = 0;
 static int32_t temp_data          = 0;

 temp_data = data;                                              // Load data to a temporary variable
 for ( i = 0 ; i < 4 ; i++ )                                    // Load Array
 {
  val_crc += (int8_t)temp_data;                                 // Using an 8-bit unsigned buffer to calculate the value of the CRC
  temp_data = temp_data >> 8;                                   // Shift data by 8, we do that 4 times, 4 x 8 = 32 bit number
 }
 sprintf ( (char*)szTemp, "%08x\n\r",data );                    // Convert all the uint_32t data to a string
 nLen = strlen((char*)szTemp);                                  // Get the length of the string to be transmitted
 for ( i = 0 ; i < nLen ; i++ )
 {
  COMTX = szTemp[i];                                            // Send String to the UART
  while ((COMSTA0 & THRE) == 0x00){}                            // Wait for Tx buffer empty bit to be set
 }
 return(0);                                                     // return zero value if everything went OK
}                                                               // END OF UART_Send_Int()
```

Comments/Functionality:

The above function is an example written in C code for sending a nsigned integer via the UART. The user must provide the function with the signed integer number and the number of bytes for that number. The transmission is done after the val_crc has been calculated that variable will be updated each time a number is transmitted until the end character ‘]’.

The whole number will be transmitted via UART character by character using the COMTX register which is the buffer Transmit (TX) register of the UART depending on the MCU you are using and the technology. We are always checking if the Tx buffer is empty before we send the next character to avoid any overflow and loss of information.

Please observe the sprintf ((char*)szTemp, "%08x\n\r", data); line, because is very important, as is converting the 32-Bit data number into a “string” using the hex form of that hex number. That means that the actual hex number will be transmitted as an ASCII string, i.e. “0x0000001A”.

Function 3 - Example on how to Send Messages to the INIR (UART)

```
//------------------------------------------------------------------------------------------------------------  //
// Function: int  UART_Build_Send_Msg_ENGINEERING(void)                                                         //
//------------------------------------------------------------------------------------------------------------  //
// Details: This is a function builds the entire message string to be                                          //
// sent via the UART when we are in Engineering mode [B]                                                        //
//------------------------------------------------------------------------------------------------------------  //
// Inputs:void                                                                                                  //
//------------------------------------------------------------------------------------------------------------  //
// Outputs:returns 0 -> everything OK                                                                           //
//------------------------------------------------------------------------------------------------------------  //
// Calls: UART_Send_Uint(),UART_Send_Int()                                                                      //
//------------------------------------------------------------------------------------------------------------  //
int  UART_Build_Send_Msg_ENGINEERING(void)                                    // START OF UART_Build_Send_Msg_ENGINEERING()
{
 val_crc = 0;                                                                 // Reset so Start Character not included in CRC
 inv_crc = 0;
 UART_Send_Uint    (             (0x000005Bu                    ), 1);        // START CHARACTER ------ START OF MESSAGE
 UART_Send_Int     (             (Displayed_Gas_Conc_PPM        ), 4);        // Load and Send the CONCENTRATION in PPM
 UART_Send_Uint    ((uint32_t)(Fault_Final                      ), 4);        // Output the Final Fault code for all the parts, see manual
 UART_Send_Uint    ((uint32_t)(Sensor_Temperature               ), 4);        // Send the Sensor Temperature, not ambient
 UART_Send_Uint    ((uint32_t)(Ref_1s_Average*1000000           ), 4);        // Send the reference average value of the previous 1 second
 UART_Send_Uint    ((uint32_t)(Act_1s_Average*1000000           ), 4);        // Send the active average value of the previous 1 second
 inv_crc = ~val_crc;                                                          // Calculate 1's compliment of the CRC
 UART_Send_Uint    (            (val_crc                        ), 2);        // LOAD CRC
 UART_Send_Uint    (            (inv_crc                        ), 2);        // LOAD 1's COMPLIMENT OF CRC
 UART_Send_Uint    (            (0x000005Du                     ), 1);        // END   CHARACTER ------ END OF MESSAGE
 return(0);
}                                                                             // END OF UART_Build_Send_Msg_ENGINEERING()
```

Comments/Functionality:

In the function above we can observe an example in C code used to send data via UART. The function is using the function 1 & 2 to help transmit various unsigned and signed integers numbers via the UART. All messages start with '[' and finish the ']' character.

Noticeable here is the fact that all the numbers are already multiplied by a constant to be transmitted as an integer based on the Table 4 - Integrated IR Variables & Multipliers. All the numbers are being transmitted as a hex representative formatted with 8 digits of each number by 4 bits each equals 32-Bit numbers. Please observe how the CRC is sent to the UART. So on the other side of the communication channel the recipient could re-calculate from the received data the CRC and validate the data received/transmitted. If the CRC we are re-calculating is not exact match even if the data look ok we should reject the whole data and keep last known good data. The data received

Function 4 - Example Function on How to Read the Message from the INIR

```
//----------------------------------------------------------------------          //
//Function:int UART_Determine_Message (void)                                       //
//------------------------------------------------------------------------         //
// Details:This is a function that determines what message has been                //
// sent - ie what is between the '[' and ']'                                        //
//------------------------------------------------------------------------         //
// Inputs:void                                                                      //
//------------------------------------------------------------------------         //
// Outputs:returns 0 -> everything OK                                               //
//------------------------------------------------------------------------         //
// Calls: using register of ARM7                                                    //
//------------------------------------------------------------------------         //
int UART_Determine_Message (void)                    // START OF UART_Determine_Message ()
{
 static uint32_t k;
 static uint32_t counter_1;
 received_data = COMRX;                              // Store incoming data in a temporary variable

if (message_started_flag == FALSE)
 {
  if (received_data == '[')                          // Opening character for the incoming message
  {
   message_started_flag = TRUE;                      // Flag to indicate the message started
   for(k=0;k<max_array_message;k++)
   {
    received_message[k] = 0;                         // Empty the array of its previous contents by setting zero to each element
   }
  }
 }
 else if (message_type_flag == FALSE)
 {
  message_type_flag = TRUE;
  message_type = received_data;
 }
 else
 {
  if (received_data == ']')                          // Closing character for the incoming message
  {
   UART_Handle_Message();                            // The message is finished run the function to handle it
   message_started_flag = FALSE;
   /* Checksum Required for incoming data */
   message_type_flag = FALSE;
   received_message_ptr =&received_message[0];       // Reset the pointer of the message to the beginning of the string
   x = 8;
   counter_1 = 0;                                    // Reset the counter of the message length to zero
  }
  else
  {
   if ((received_data >= 0x30u) && (received_data <= 0x39u))
  {
   received_data -= 0x30u;                           // ASCII code for "0"
  }
  else if ((received_data >= 0x41u)&&(received_data <= 0x46u))
  {
   received_data = hex_equiv[received_data - 0x41u]; // Find which hex equivalent is the data we have received
  }

  received_4_bytes = received_4_bytes << 4;
  received_4_bytes |= received_data;

 if (x == 1)
  {
   *received_message_ptr = received_4_bytes ;
   x=9;
   received_message_ptr++;                           // Increase the pointers position to travel from beginning to the last element
```

```
  counter_1++;                              // Message length is increased each time we finish reading every element
  }
  if (counter_1 == (max_array_message + 1))   // Stops the received message going past the max array size
  {
   UART_Handle_Message();
   message_started_flag      = FALSE;       // Will say the message is unknown and send a [nak]
   message_type_flag         = FALSE;
   received_message_ptr      = &received_message[0];    // Pointer to the beginning of the message string
   counter_1 = 0;
  }
  else
  {
   x--;
  }
 }
}
return(0);
}                                           // END OF UART_Determine_Message ()
```

Comments/Functionality:

The above function can be used as a reference example on how to read a message from the INIR. In the above function we are using that to determine what the customer send us as a command. The code is self-explanatory and is got additional comments to help us with the understanding of the basic functionality. The key factor is the fragmented command or message. Is would be a recommended practice to always check for the existence of the starting '[' and ending ']' characters.

IMPORTANT WARNING!

All the above functions 1 to 4 are just examples on how could someone read, write or send data to the INIR by using a well-known programming language called 'C'. There are many other programming languages for Software or Firmware Engineering and many more compilers that the above functions could or not compile.

SGX Ltd is not responsible for any damage or problems occur during or after implementing the above functions as we provide them or by modifying them. SGX Ltd is willing to provide the customers with technical support regarding Gas Sensor components but will not in any way be involved into designing, modifying or altering instrument's hardware or software or acting as a consultancy agency for designing gas detection instruments.

Our customers must have the first and the last word of what language or MCU they want to use in their application and they are the only ones who can decide what is best for them.

SGX Ltd is not responsible for providing any support other than guidelines or hints included in this document for code development or translate the above code into another programming language as it is impossible to possess expertise for all the programming languages or all the platforms that our customers may be using.

③ | Calibration Procedure

The following procedure can be implemented in the Calibration of the Infrared Gas Sensors of the SGX's IR production line. The sensors that fell in that category can be found into the Table 2.0 - Sensor List Compatible with the Smart IR inside Smir Datasheet or Integrated IR inside the Integrated IR Datasheet in the ordering codes section. The coefficients that we are going to use for doing the calibration are depending on the Sensor and the targeted Gas. Please find the correct coefficients from the Table of Coefficients at the Appendix A – Table of Coefficients or contact SGX Ltd fro more details. The INIR Gas Sensor got custom coefficients so please do not change them if you are recalibrating the sensor.

IMPORTANT WARNING!

It is not advisable to do any kind of Re-Calibrate on a factory calibrated pair other than Re-Zero, as this will void the certificate. Please be aware that different rules apply for every country. SGX's recommendations are here to be used as guidelines and not to supersede or overwrite any standard, law, rule or regulation that applies in any case. The equipment used for performing the calibration should be suitable to ensure the reliability and the repeatability of the linearity.

Equipment Used

1 x Certified Gas Cylinder of Dry 100% Nitrogen (Gas Cylinder for doing the Zero Calibration)
1 x Certified Gas Cylinder of Dry 20.0% CH_4 / Balance Nitrogen (Gas Cylinder of the target Gas)
1 x Certified Gas Cylinder of Dry 2.0% CH4/Balance Nitrogen (Gas Cylinder of the offset Gas if required)
1 x INIR Digital Gas Sensor
3 x Plastic Tubing
3 x Flow Regulators
1 x Gas Cap

Recommended Environmental Conditions

Pressure	100.5kpa -101.5kpa
Temperature	20°C – 25 °C
Relative Humidity	5% or completely Dry
Gas Flow	Maximum 1000 ml/hour
Averaging	12-13 /sec
Warm Up Time Before Calibration	45 to 60 mins

Note: If more accurate Evaluation required please use an Environmental Chamber to keep temperature, humidity, vibration and pressure stable to desired values.

Basic Set Up

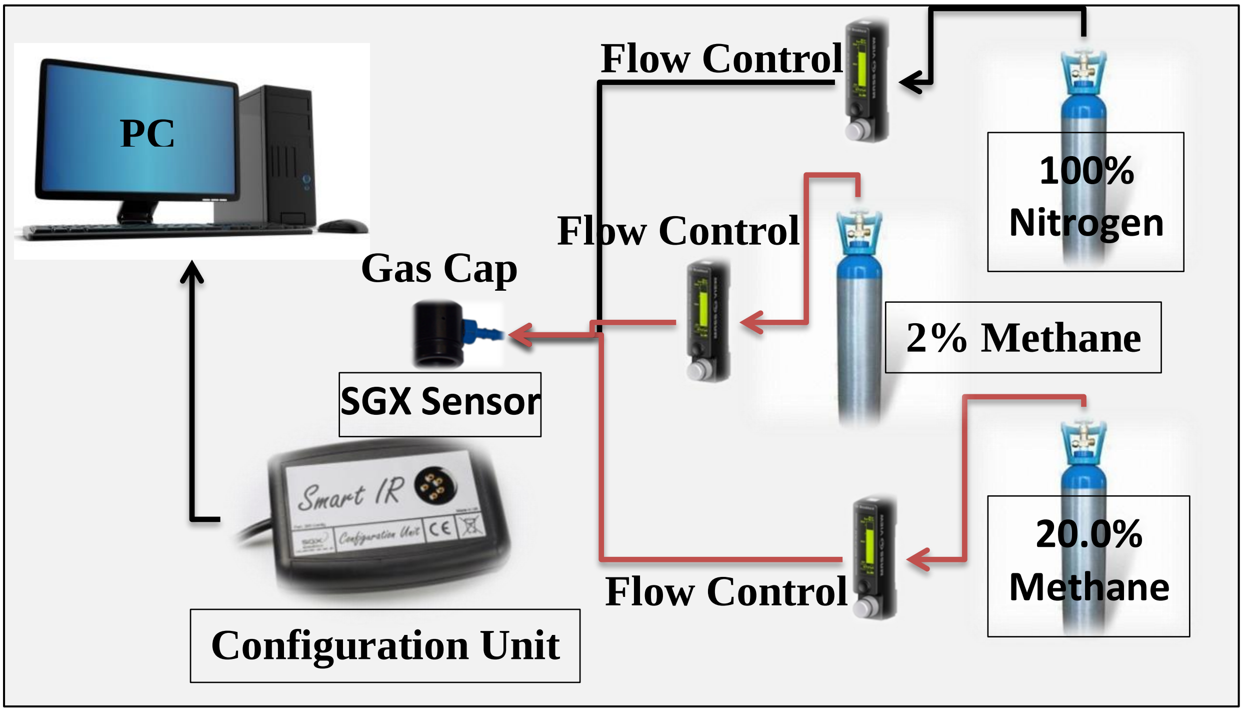
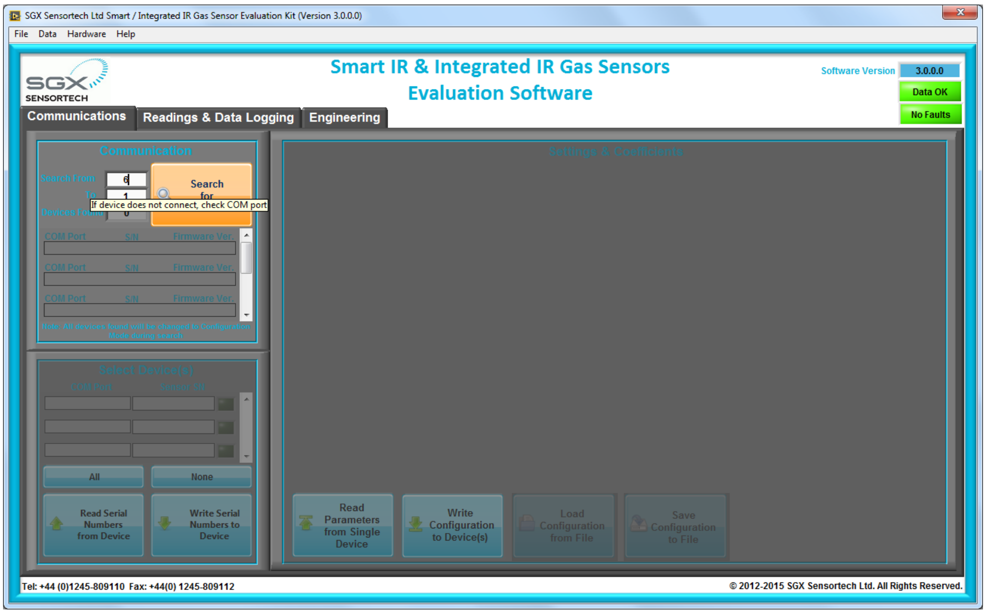


Figure 3 - Basic Calibration Set Up

3.1. Calibration Routine using the Evaluation Software

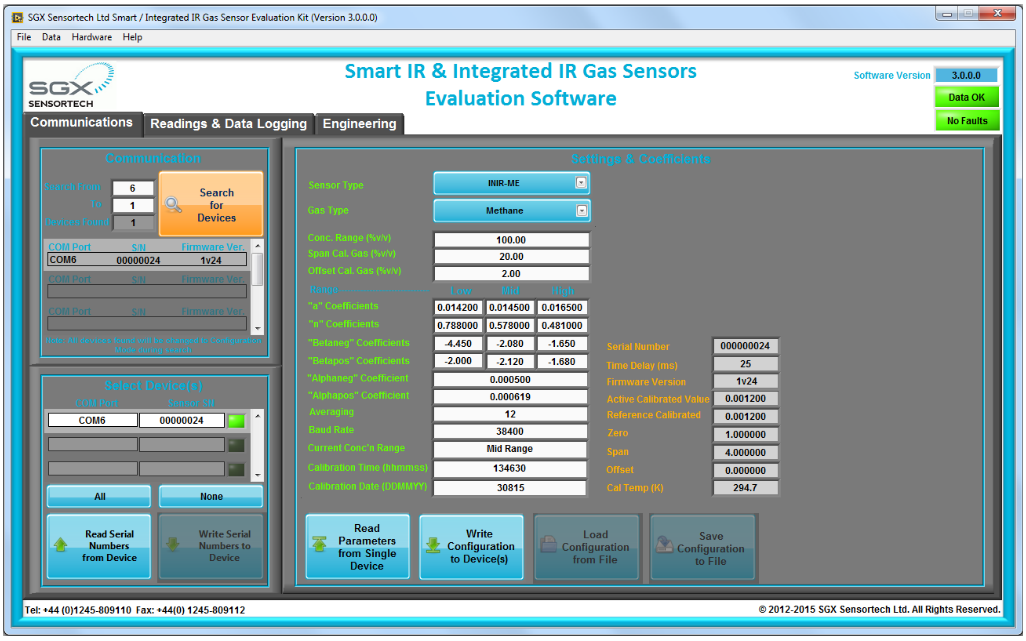
Step 1 (Communicate):

a. Plug the Gas Sensor onto the Integrated IR and Connect the Configuration Unit to the PC
c. Plug the Cap onto the Gas Sensor
d. Run/Open the Integrated IR Evaluation Software version 3.0.0 and above.
e. Go to "**Communications" Tab** -> "**Search for Devices"**, by pressing the Orange Button



Step 2(Configure):

a. Once connection established ->Go to "**Communications" Tab**
c. Set up the **Sensor Type, Target Gas Type etc..**
d. Fill in the correct coefficients for the target Gas for an example please see below:



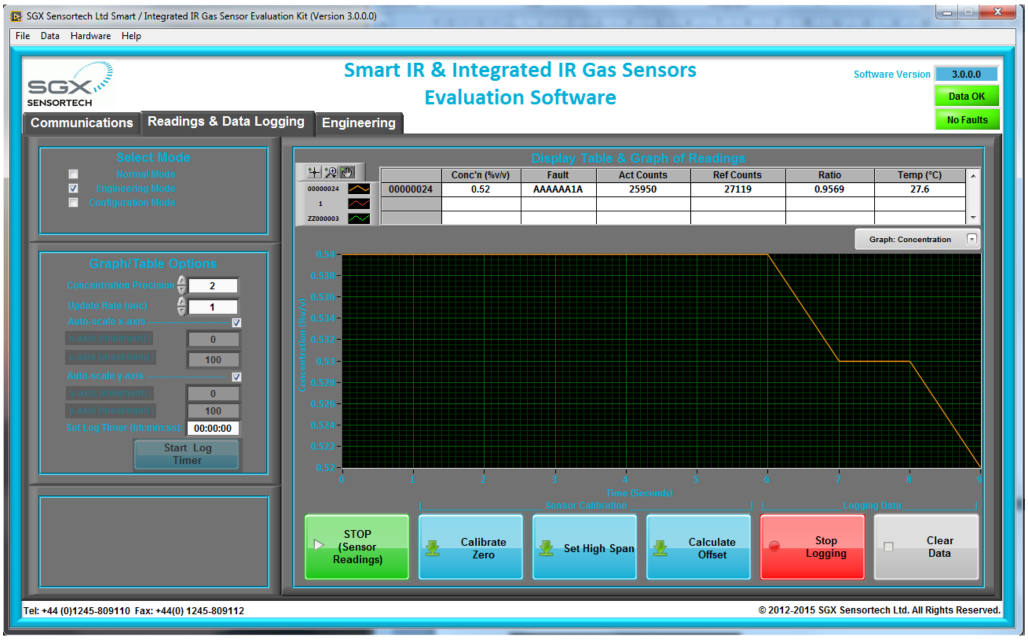
e. Press **Write Configuration to Device(s)** to write the settings to the INIR.
(HINT: You will not be able to rewrite the Orange Variables, only the Green ones).
f. Select **Engineering Mode**
g. Go to "**Readings & Data Logging" Tab**.

Step 3(Warm Up):

a. Connect the 100% Nitrogen, open the Gas Regulator, and use also the flow controller to regulate the gas flow ideally around 500ml/min.

b. **Press Start** for the Software to start reading the Integrated IR, see below:

c. Wait for 45 mins for the Sensor to warm up and reach the Ambient Temperature (ideally 20 °C).



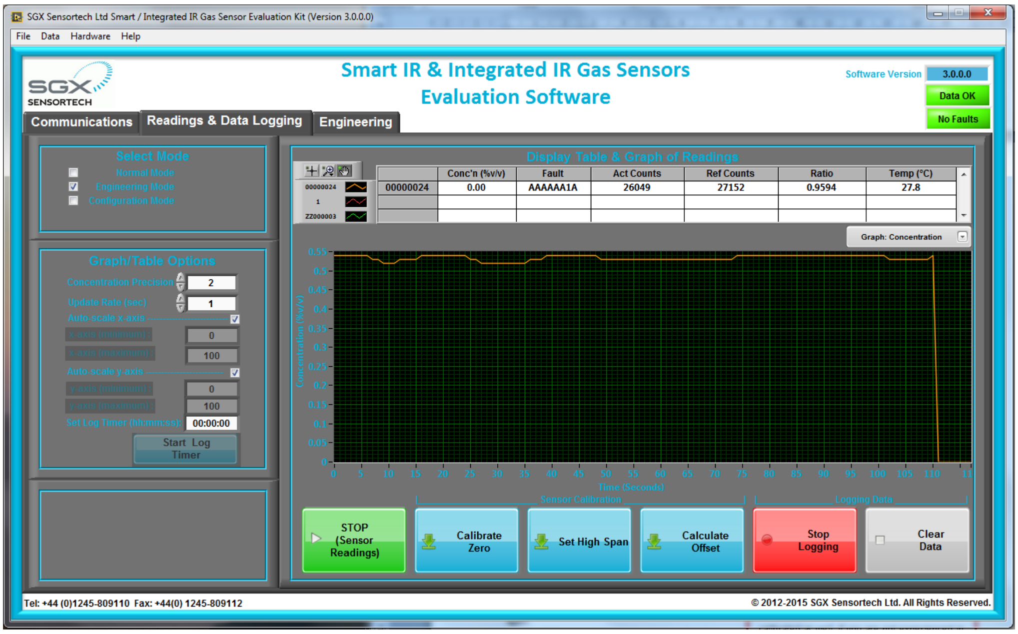
IMPORTANT WARNING!

Try to keep the ambient temperature stable during the Calibration Process, the Zero, High Span Calibration and Offset Calculations should be done in the same Temperature or close enough no more than +/- 1.0°C. Above that limit you will implement an inaccuracy in temperature compensation algorithm

Step 4(Calibrate Zero):

a. When sensor is stable press the **Set Zero Button.**

b. Wait 5 seconds for the reading to be stable.



Step 5(Calibrate Span):

a. Disconnect the Gas Cap of the 100% Nitrogen

b. Connect the Gas Cap of the 20.00% Methane/ Balance Nitrogen

c. Wait for 3 mins.

d. After Signals are stable, press **Set High Span Button**

e. Wait 5 seconds for the signals to be stable again.

f. Disconnect the 20.0% Methane Gas Cap.

g. Connect the Gas Cap of the 100% Nitrogen

h. Wait for 3mins.

HINT: You can skip **Step 6** if you are happy with the linearity and precision below 5% Methane, perform Offset Calculation only if proven your sensor is reading outside +/-6% limits.

Step 6(Calculate Offset):

a. Connect the Gas Cap of the 2.00% Methane/ Balance Nitrogen

b. Wait for 3 mins.

c. After Signals are stable, press **Calculate Offset**

Step 7(Finish):

a. Start doing the Experiments you want, the Integrated IR Gas Sensor is now fully calibrated.

IMPORTANT WARNING!

Every time we change the sensor for maintenance or repair the Calibration procedure should be done again, it is against the law and regulations to use an un-calibrated device in the field. Make sure when you are doing the calibration that the equipment used is calibrated as well. If you are not experienced in the calibration process or you do not have the equipment to do it, SGX Sensortech (IS) Ltd can provide you with a pre-calibrated pair, but still you should be able to calibrate the instrument at regular intervals when required by regional regulations.

For other gases or other sensors the calibration routine would be exactly the same depending on the Integrated IR Firmware. Customers with Integrated IR with a Firmware Version 1v2 or Smart IR Firmware 4v0 and above must be using this document as a calibration guide. Customers that are using Smart IR modules that got older Firmware Versions like for example 3v9 and below customers should be using the Smart IR Application Note 1 that applies for that firmware version.

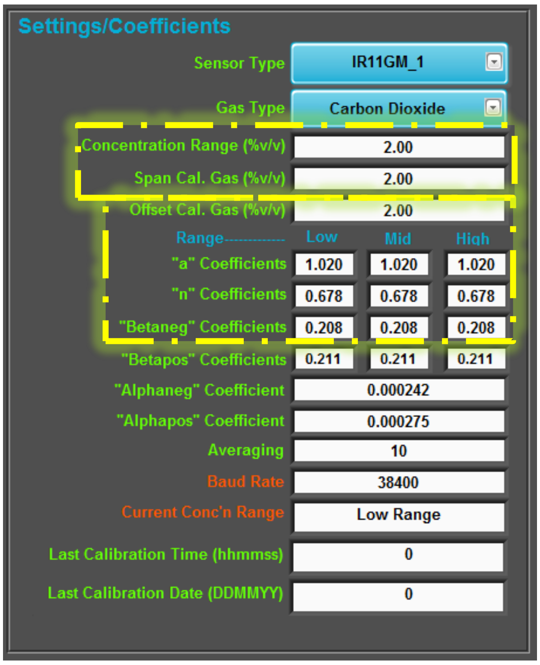
The Evaluation Software version 2.0.0.0 and above has a back compatibility with all the previous firmware versions of the Smart IR and Integrated IR gas sensors. With the new Evaluation software customer can perform testing for both 3v1 and 4v0 firmware versions of the Smart IR, the software will automatically detect the new or old version of the firmware and change the user interface accordingly to match the functionality required.

2.2 Calibrating Different Target Gases

If we intent to do a Carbon Dioxide calibration or any other target gas like Propane or Butane and the full operating range is going to be a very low concentration like 2.0% then we can ignore the offset calculation and only do the Zero and Span calibrations. By default the offset is considered to be zero if we haven't done a calibration before.

The coefficients then for the full 2.0%v.v low range are going to be the same for the Low, Mid and High ranges see an example below, particularly inside the yellow boxes.

As we can observe from the above settings the coefficients for the full low concentration range are the same in the Low, Mid and High indicated equivalent text boxes and the reason behind that is because as we said previously there is going to be only one range low enough to be able to represented by only on set of coefficients accurately with no need to calculate the offset. Please see the Appendix A – Table of Coefficients for more details about the settings.



❹ | Pressure Compensation Algorithm

The Integrated IR (INIR) will need Pressure compensation as any IR Gas Sensor will need. The standard devices like the IR12GM_1 will need pressure compensation especially when we going to extreme operating pressures like 130 or 50 kpa. In this chapter we are going to introduce test data and help our customers to come up with an efficient algorithm for pressure compensation.

Nevertheless we are recommending to always doing a Re-Zero only calibration each time we are changing the conditions of any test or real life measurements. That Re-Zero calibration will minimize any temperature, pressure or humidity Effects as the sensor would now have a stable baseline within the same conditions of the real-time measurements. This Re-Zero technique is something that most of the instrument manufacturers are using but it will not account for any environmental effects like rain, thunderstorms, earthquakes, hurricanes or any relevant extreme environmental effect.

As a starting point we would recommend to our customers to do simple experiments with pressure effects to the INIR.

The best way at the moment if customers want to implement pressure compensation into their instruments is to Contact SGX Ltd Headquarters or our local sales support office.

Pressure Compensation data are pending approval for publication.

Figure 3 - Pressure Compensation Representative Test results

❺ | Condensation Compensation Algorithm

What is Humidity?

Humidity is the amount of water vapor in the air. Water vapor is the gaseous state of water and is invisible. Humidity indicates the likelihood of precipitation, dew, or fog. Higher humidity reduces the effectiveness of sweating in cooling the body by reducing the rate of evaporation of moisture from the skin. This effect is calculated in a heat index table or humidex.

There are three main measurements of humidity: absolute, relative and specific. Absolute humidity is the water content of air at a given temperature expressed in gram per cubic metre. Relative humidity, expressed as a percent, measures the current absolute humidity relative to the maximum (highest point) for that temperature. Specific humidity is a ratio of the water vapor content of the mixture to the total air content on a mass.

The relative humidity (ϕ) of an air-water mixture is defined as the ratio of the partial pressure of water vapor (H_2O) (e_w) in the mixture to the saturated vaporpressure ofwater (e^*_w) at a given temperature. Thus the relative humidity of air is a function of both water content and temperature.

Relative humidity is normally expressed as a percentage and is calculated by using the following equation:

$$\phi = \frac{e_w}{e^*_w} \times 100\%$$

Relative humidity is an important metric used in weather forecasts and reports, as it is an indicator of the likelihood of precipitation, dew, or fog. In hot summer weather, a rise in relative humidity increases the apparent temperature to humans (and other animals) by hindering the evaporation of perspiration from the skin. For example, according to the Heat Index, a relative humidity of 75% at 80.0 °F (26.7 °C) would feel like 83.6 °F ±1.3 °F (28.7 °C ±0.7 °C) at ~44% relative humidity.

What is Condensation?

Condensation is the formation of liquid drops of water from water vapor. It is the process which creates clouds, and so is necessary for rain and snow formation as well. Condensation in the atmosphere usually occurs as a parcel of rising air expands and cools to the point where some of the water vapor molecules clump together faster than they are torn apart from their thermal energy.

A very important part of this process is the release of the latent heat of condensation. This is the heat that was absorbed when the water was originally evaporated from the surface of the Earth, a process which keeps the Earth's surface climate much cooler that it would otherwise be if there were no water. The heat removed from the surface through evaporation is thereby released again higher up in the atmosphere when clouds form.

Another way in which condensation occurs is on hard surfaces, such as during the formation of dew. Water condensing on a glass of ice water, or on the inside of windows during winter, is the result of those glass surfaces' temperature cooling below the dewpoint of the air which is in contact with them.

How INIR perform under Condensation?

The Integrated IR (INIR) Gas Sensor has the relevant Algorithm to minimize the effect that condensing or humidity has into the sensor. The Condensation in mining industry usually comes from two sources, the first one being just sprayed water to minimize the dust clouds and the second being due to hard surfaces effect we have described above.

Irrelevant where the condensation comes from or how it forms IR Gas Sensors will react to that effect the same way they are reacting to the presence of Methane. For that reason sometimes we may get false alarms down the mine and that is why SGX Ltd has implemented the Condensation Algorithm to keep the effect minimum under the first alarm point usually got underground. Eventually the Condensation will evaporate as the sensor warms up and the re-bounce of that effect is within a small period of time.

The above algorithm is recommended to be "ON" in INIR-MEGG% as the effect of Humidity and Condensation only appears to Methane, Propane or Butane detectors, the Carbon Dioxide Gas Sensors from experiments do not appear to correspond to that effect and are not affected the way the Methane Gas Sensors do.

Please read page 29 including results from test done to prove the theory and validate the use of the algorithm under controlled environmental conditions of pressure, temperature. The only variable measurable condition was the condensation that it was produced with a commercial humidifier. The algorithm and off course any firmware or software information or design about related products is proprietary information of SGX Ltd and is protected under United Kingdom, EU or global Copyright Law.

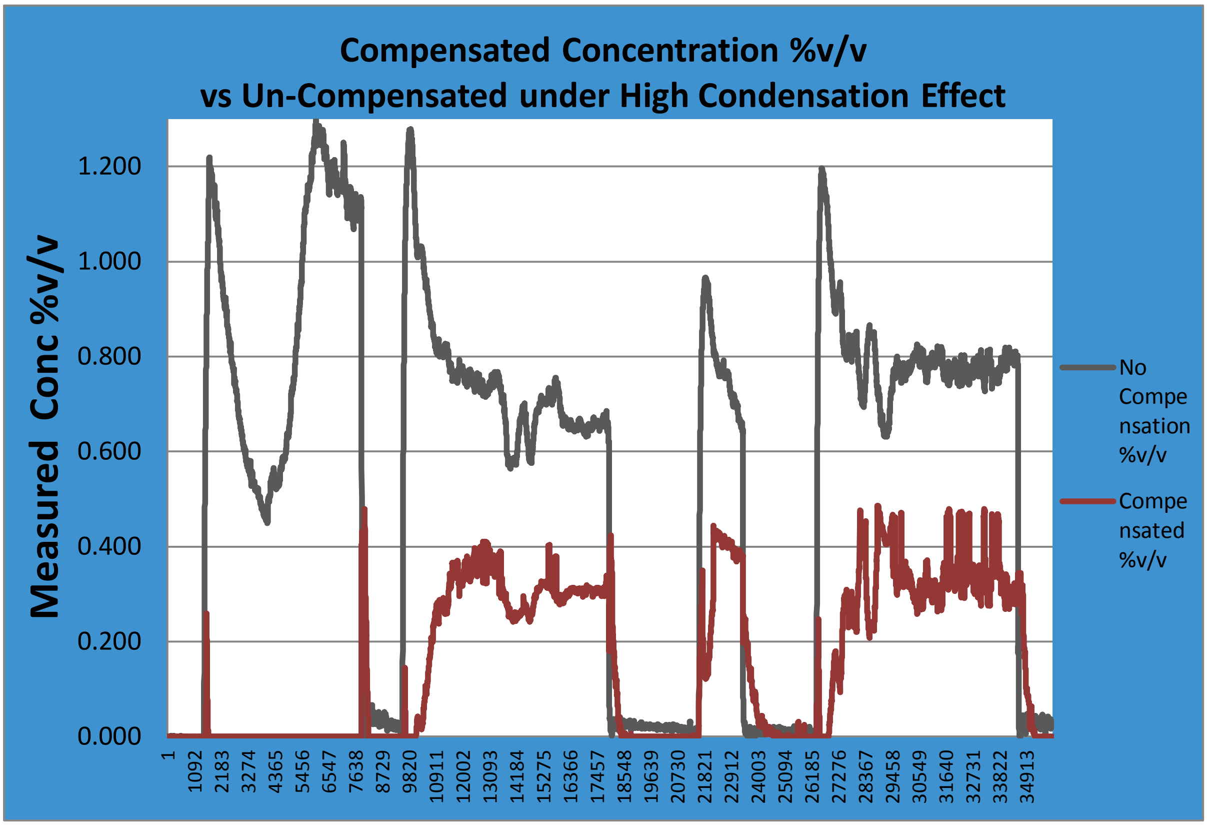


Figure 3 - Condensation Compensation Representative Test results

IMPORTANT WARNING!

The Compensation Condensation Algorithm is tool that customers may choose to use it or not. We are strongly recommending to our customers to test extensively the algorithm and the INIR after they have implemented the INIR Gas Sensor into their own instrument and then based on real-life in the field experiments to decide to enable the Algorithm or not.

SGX Ltd has not liability if the specific tool is not used as intended "to minimize the response effect due to condensation forming inside the INIR Gas Sensor".

The specific algorithm is not designed to "cure" the problem and it will not work with the same performance in the presence of Gas and that is because of the physical properties of the Detector and the way that the IR Gas Sensors are designed.

The best solution that SGX Ltd is proposing is to "Turn ON" the algorithm and Turn it "OFF" after a short period of time or Turn it "ON" only during the exposure to water particles is certain for a short period of time as well.

Appendix A - Table of Coefficients

Sensor Type	Gas Type	Range (%v.v)	a Low	a Mid	a High	n Low	n Mid	n High	alphapos	alphaneg	Betapos Low	Betapos Mid	Betapos High	Betaneg Low	Betaneg Mid	Betaneg High
INIR	CH_4 (ME)	0-100%	0.0150	0.0154	0.0174	0.742	0.575	0.483	0.000556	0.00515	-3.48	-1.97	-1.53	-2.53	-2.09	-1.59
	C_3H_8 (PR)	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX
	C_4H_{10} (BU)	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX	Contact SGX
INIR	CO_2 (CD)	0-5%														

NOTE:

1. These coefficients are based upon results measured at SGX Sensortech (IS) Ltd using standard test equipment. These coefficients may vary slightly when using different circuits. It may be required to recalculate some of these coefficients if small inaccuracies are observed during testing (refer to Infrared Sensor Application Note 5 for determination of coefficients).

2. The above coefficients are representing the average typical coefficients that the customers can use for evaluation. SGX Sensortech (IS) Ltd is making sure

Appendix B – INIR Quick Reference Chart

How to Connect INIR in Customer's Instrument	Use Gas Sockets; see INIR Datasheet for Footprint and Dimensions. Precautions and Special Handling Applies see INIR Datasheet. A. Plugging or Unplugging the Sensor while in operation may damage the device beyond repair. Always power down the instrument when performing maintenance. B. Do not drop the Integrated IR on the floor as this could cause damage to the pins or internal components. C. Avoid mechanical force against pins or sockets. Protect from dust and sprayed acidic particules. D. Do not immerse in water or other fluids. E. Do not solder the module directly onto a pcb or wires. Excessive heat could cause damage.
How to Power "ON" INIR	INIR will immediately be powered on when the appropriate and within specs power supply applied into the voltage input (see INIR Datasheet). The Configuration Mode is not putting the INIR in standby or shut down Mode is only stopping the readings ADC, the digital and the analog output.
Initialization Procedure	See page xx in AN1 – Integrated IR Application Note, paragraph
How to Set Mode	See page xx in AN1 – Integrated IR Application Note, Table
How to Send a Command	See page xx in AN1 – Integrated IR Application Note, paragraph
Read the Answers from INIR	See page xx in AN1 – Integrated IR Application Note, paragraph
How to Check CRC	See page xx in AN1 – Integrated IR Application Note, paragraph
Translating the Faults	See page xx in AN1 - Integrated IR Application Note, Table
Safety Instructions	See page xx in INIR Datasheet
Special Conditions	ESD , see page xx in INIR Datasheet
	EMC, see page xx in INIR Datasheet
	High Humidity>85%, see page xx in INIR Datasheet
	Condensation, see page xx in INIR Datasheet
	High Temperature >75^{o}C, see page xx in INIR Datasheet
Troubleshooting The Gas Sensor	Contact SGX Ltd.

Index

A
Abstract, 5
C
Calibrate Span, 25
Calibrate Zero, 25
Calibration Procedure, 22
Commands, 9, 10
Communicate, 23
CONFIGURATION, 11, 14
Configure, 23
Copyright, 2
D
DAC, 20
DACvolts, 20
Digital Output String, 8
E
ENGINERING, 13
Errata, 3, 4
F
Faults/Errors, 21
Firmware, 19
G
Gas Type, 12
I
Introduction, 6
K
KELVIN to CELCIUS, 19
M
Microcontroller, 13
Multipliers[a], 11
N
NORMAL, 13
O
offset, 10, 11, 16, 27
R
Ranges, 19
RESET to FACTORY Def, 18
Revision, 3, 4
S
Sensor Type, 12
SGX Sensortech (IS) Ltd, 2
Smart IR Module, 13
software, 5, 6, 7, 9, 26
U
UART, 6, 7, 13
UART Settings, 8
W
Warm Up, 24

SGX Sensortech (IS) Ltd
2 Hanbury Road,
Chelmsford, Essex
CM13AE
United Kingdom
http://www.sgxsensortech.com

Author : Theodoros Koutsos
Created & Printed in the Essex, United Kingdom by SGX Sensortech (IS) Ltd.