SignalCore™

PRESERVING SIGNAL INTEGRITY

SC5405A

1 MHz to 3.9 GHz RF Upconverter

for PXI Express

Operating & Programming Manual

support@signalcore.com

Contents

Important Information

Warranty

This product is warranted against defects in materials and workmanship for a period of three years from the date of shipment. SignalCore will, at its option, repair or replace equipment that proves to be defective during the warranty period. This warranty includes parts and labor.

Before any equipment will be accepted for warranty repair or replacement, a Return Material Authorization (RMA) number must be obtained from a SignalCore customer service representative and clearly marked on the outside of the return package. SignalCore will pay all shipping costs relating to warranty repair or replacement.

SignalCore strives to make the information in this document as accurate as possible. The document has been carefully reviewed for technical and typographic accuracy. In the event that technical or typographical errors exist, SignalCore reserves the right to make changes to subsequent editions of this document without prior notice to possessors of this edition. Please contact SignalCore if errors are suspected. In no event shall SignalCore be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, SIGNALCORE, INCORPORATED MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF SIGNALCORE, INCORPORATED SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. SIGNALCORE, INCORPORATED WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of SignalCore, Incorporated will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against SignalCore, Incorporated must be brought within one year after the cause of action accrues. SignalCore, Incorporated shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow SignalCore, Incorporated's installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright & Trademarks

International Materials Declarations

SignalCore, Incorporated uses a fully RoHS (Restriction of Hazardous Substances) compliant manufacturing process for our products. Therefore, SignalCore hereby declares that its products do not contain restricted materials as defined by European Union Directive 2011/65/EU (EU RoHS) in any amounts higher than limits stated in the directive. This statement is based on the assumption of reliable information and data provided by our component suppliers and may not have been independently verified through other means. For products sold into China, we also comply with the "Administrative Measure on the Control of Pollution Caused by Electronic Information Products" (China RoHS). In the current stage of this legislation, the content of six hazardous materials must be explicitly declared. Each of those materials, and the categorical amount present in our products, are shown below:

組成名稱 **Model Name**	鉛 **Lead (Pb)**	汞 **Mercury (Hg)**	镉 **Cadmium (Cd)**	六价铬 **Hexavalent Chromium (Cr(VI))**	多溴联苯 **Polybrominated biphenyls (PBB)**	多溴二苯醚 **Polybrominated diphenyl ethers (PBDE)**
SC5405A	✓	✓	✓	✓	✓	✓

A ✓ indicates that the hazardous substance contained in all of the homogeneous materials for this product is below the limit requirement in SJ/T11363-2006. An **X** indicates that the particular hazardous substance contained in at least one of the homogeneous materials used for this product is above the limit requirement in SJ/T11363-2006.

CE European Union EMC & Safety Compliance Declaration

The European Conformity (CE) marking is affixed to products with input of 50 - 1,000 VAC or 75 - 1,500 VDC and/or for products which may cause or be affected by electromagnetic disturbance. The CE marking symbolizes conformity of the product with the applicable requirements. CE compliance is a manufacturer's self-declaration allowing products to circulate freely within the European Union (EU). SignalCore products meet the essential requirements of Directives 2014/30/EU (EMC) and 2014/35/EU (product safety), and comply with the relevant standards. Standards for Measurement, Control and Laboratory Equipment include EN 61326-1:2013 and EN 55011:2009 for EMC, and EN 61010-1 for product safety.

Recycling Information

All products sold by SignalCore eventually reach the end of their useful life. SignalCore complies with EU Directive 2012/19/EU regarding Waste Electrical and Electronic Equipment (WEEE).

Warnings Regarding Use of SignalCore Products

(1) PRODUCTS FOR SALE BY SIGNALCORE, INCORPORATED ARE NOT DESIGNED WITH COMPONENTS NOR TESTED FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED “SYSTEM FAILURES”). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE SOLELY RELIANT UPON ANY ONE COMPONENT DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM SIGNALCORE' TESTING PLATFORMS, AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE SIGNALCORE PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY SIGNALCORE, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF SIGNALCORE PRODUCTS WHENEVER SIGNALCORE PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Getting Started

Unpacking

All SignalCore products ship in antistatic packaging (bags) to prevent damage from electrostatic discharge (ESD). Under certain conditions, an ESD event can instantly and permanently damage several of the components found in SignalCore products. Therefore, to avoid damage when handling any SignalCore hardware, you must take the following precautions:

- Ground yourself using a grounding strap or by touching a grounded metal object.
- Touch the antistatic bag to a grounded metal object before removing the hardware from its packaging.
- *Never* touch exposed signal pins. Due to the inherent performance degradation caused by ESD protection circuits in the RF path, the device has minimal ESD protection against direct injection of ESD into the RF signal pins.
- When not in use, store all SignalCore products in their original antistatic bags.

Remove the product from its packaging and inspect it for loose components or any signs of damage. Notify SignalCore immediately if the product appears damaged in any way.

Verifying the Contents of your Shipment

Verify that your SC5405A kit contains the following items:

Quantity	Item
1	SC5405A RF Upconverter for PXI Express
1	Software Installation USB Flash Drive (may be combined with other products onto a single drive)

Setting Up and Configuring the SC5405A

The SC5405A is designed for use in a PXI Express (PXIe) or PXIe hybrid chassis. Chassis manufacturers must provide at least the minimum required per-slot power dissipation cooling capability to be compliant with the PXIe specifications. The SC5405A is designed to be sufficiently cooled in either all-PXIe chassis or PXIe hybrid chassis (chassis with a mix of PXI Express slots and traditional PXI slots). However, certain environmental factors may degrade performance. Inadequate cooling can cause the temperature inside the RF housing to rise above the maximum for this product, leading to improper performance and potentially reducing product lifespan or causing complete product failure. Maintain adequate air space around the chassis at all times and keep the chassis fan filters clean and unobstructed.

Refer to your chassis manufacturer's user manual for proper setup and maintenance of your PXIe or PXIe hybrid chassis. The SC5405A on-board temperature sensor should indicate a rise of no more than 20 °C above ambient temperature under normal operating conditions.

The SC5405A is a PXIe RF upconverter with all user I/O located on the front face of the module as shown in Figure 1. Each I/O location is discussed in further detail below.

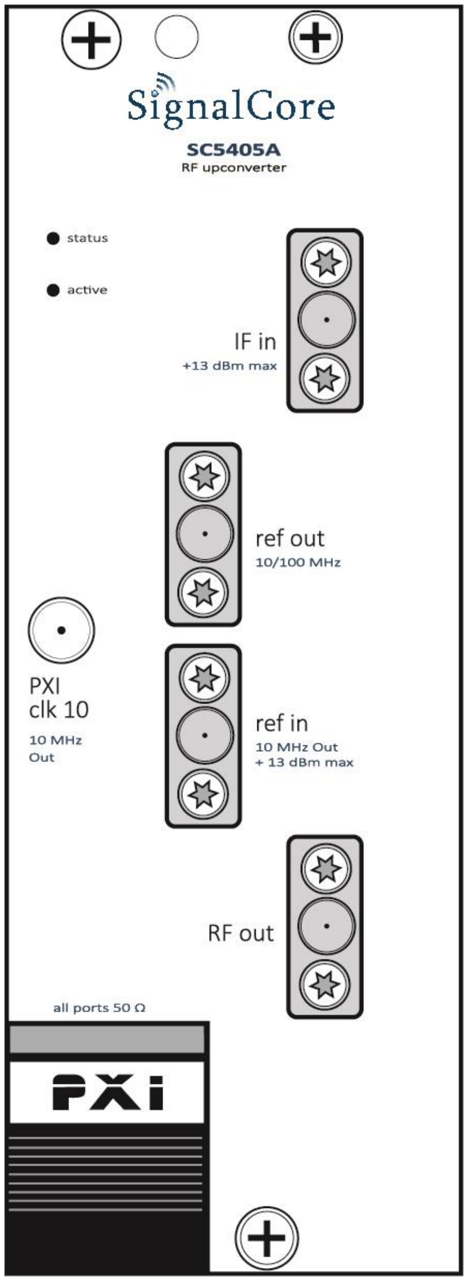


Figure 1. PXI Express front view of the SC5405A.

All signal connections (ports) on the SC5405A are SMA-type, with the exception of the PXI backplane clock connection (MCX connection). Exercise caution when fastening cables to the signal connections. Over-tightening any connection can cause permanent damage to the device.

⚠ *The condition of your system's signal connections can significantly affect measurement accuracy and repeatability. Improperly mated connections or dirty, damaged or worn connectors can degrade measurement performance. Clean out any loose, dry debris from connectors with clean, low-pressure air (available in spray cans from office supply stores).*

⚠ *If deeper cleaning is necessary, use lint-free swabs and isopropyl alcohol to gently clean inside the connector barrel and the external threads. Do not mate connectors until the alcohol has completely evaporated. Excess liquid alcohol trapped inside the connector may take several days to fully evaporate and may degrade measurement performance until fully evaporated.*

Tighten all SMA connections with 3 in-lb min to 5 in-lb max (56 N-cm max)

Signal Connections

RF OUT This port outputs 1 MHz to 3.9 GHz signals from the upconverter. The nominal output impedance is 50 Ω. Maximum output power is +20 dBm.

IF IN This port accepts the 70 MHz IF signal to the upconverter. The nominal input impedance is 50 Ω. Maximum input power is +13 dBm.

REF IN This port accepts an external 10 MHz reference signal, allowing an external source to synchronize the internal reference clock. This port is AC-coupled with a nominal input impedance of 50 Ω. Maximum input power is +13 dBm.

REF OUT This port outputs the internal 10 MHz or 100 MHz reference clock. If the internal reference clock is synchronized to an external reference clock through the 10 MHz "**ref in**" port, this output port will also be synchronized. This port is AC- coupled with a nominal output impedance of 50 Ω.

PXI CLK10 This port outputs the 10 MHz chassis reference signal from the chassis backplane, allowing it to synchronize the internal reference clock. An MCX male to SMA male cable is required (but not supplied) to connect this port to the "**ref in**" port in order to use this reference for synchronization. This port may be enabled or disabled through a software switch to minimize possible clock noise when not in use. This port has a nominal output impedance of 50 Ω and drives 0 dBm into a 50 Ω load.

Indicator LEDs

The SC5405A provides visual indication of important modes. There are two LED indicators on the device. Their behavior under different operating conditions is shown in Table 1.

Table 1. LED indicator states.

LED	Color	Definition
STATUS	Green	"Power good" and all oscillators phase-locked
STATUS	Red	One or more oscillators off lock
STATUS	Off	Power fault

LED	Color	Definition
ACTIVE	Green/Off	Device is open (green) /closed (off) , this indicator is also user programmable (see register map)
ACTIVE	Orange	User initiated standby mode

SC5405A - Theory of Operation

Overview

The SC5405A operates on the principle of heterodyning, a process whereby an incoming RF/IF signal is mixed with specific oscillator frequencies in stages, producing both sum and difference frequency products. At each stage, the summed frequency product (or image) is removed through low-pass filtering, allowing the difference frequency product to continue through the signal path. Repeating this process several times using carefully selected local oscillators (LOs) and well-designed band-pass filtering, the original signal is translated, or “up-converted” in frequency from low IF to higher RF. Using a tunable LO as the final mixing oscillator allows the upconverter to translate a fixed IF input frequency to a broad range of RF frequencies.

The SC5405A is a three-stage superheterodyne upconverter that delivers superior image rejection over single stage conversion and offers both high signal-to-noise dynamic range and high spurious-free dynamic range. The RF output ranges from 1 MHz to 3.9 GHz and the IF input is fixed at 70 MHz. When the output frequency is lower than the intermediate frequency, the device technically behaves as a downconverter. The converted spectrum polarity may be inverted or non-inverted by programming the device accordingly.

Fundamentally, each conversion stage consists of a frequency mixer that mixes two input signals and produces a wanted third. The *wanted* third component is selected via a frequency filter among other signals generated in the mixing process. The three primary components of the signals in each conversion mixer are commonly known as the local oscillator (LO), radio frequency (RF), and the intermediate frequency (IF) as shown in Figure 2.

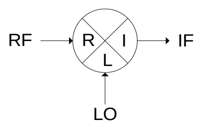


Figure 2. Frequency conversion stage using a mixer.

Where R represents the RF component, L represents the LO component, and I represents the IF component. The LO is resident in the upconverter and is either frequency tunable or fixed in frequency depending on the stage.

The final IF stage (IF1) is a down-conversion stage – the RF signals are down converted from this IF, and its frequency is higher than the highest output RF frequency specified. The second and third stages successively convert the 70 MHz IF up to the final IF1, which is at 4675 MHz. Having a high final IF allows the upconverter to suppress internal IF frequencies from appearing at the RF output port.

The SC5405A exhibits very low phase noise of -107 dBc/Hz at 10 kHz offset on a 1 GHz RF carrier with a typical noise floor of -150 dBm/Hz. The noise floor can be further reduced below -165 dBm/Hz by enabling the internal preamplifier. With gain control between -100 dB to +30 dB, capable of generating RF signals with levels between -100 dBm to +15 dBm with 0 dBm at the IF port. The excellent close-in spurious free dynamic range is achieved using low noise linear amplifiers, low loss mixers, and high performance solid state attenuators. State-of-the-art solid state attenuators have improved linearity over earlier designs. Their attenuation level changes settle under a microsecond, and for applications that involve frequent range changing, they offer a vastly superior lifetime over mechanical attenuators.

The real-time bandwidth is shaped primarily by the surface acoustic wave (SAW) filter at the front IF (IF3). Filters in the final (IF1) and second IF stages are not as selective as the final IF filter but they ensure good isolation between local oscillators (LO). Keeping each LO isolated helps to suppress unwanted spurious signals.

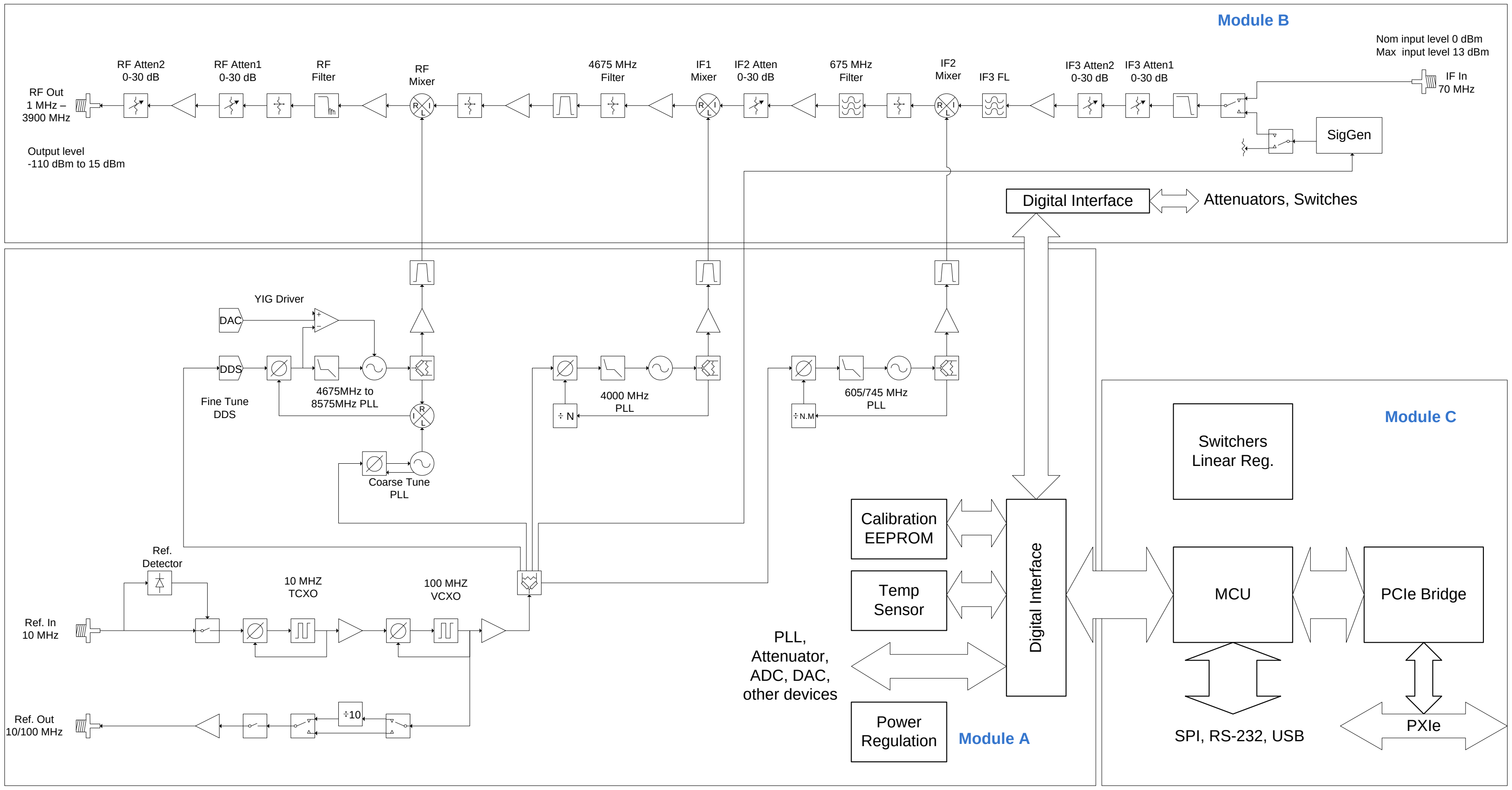
Frequency accuracy is provided by an onboard 10 MHz temperature compensated crystal oscillator (TCXO) which can be phase-locked to an external reference source if required, and it is recommended to do so in applications that may require a more stable and accurate base reference.

Signal Path Description

Figure 3 depicts an overall block diagram of the SC5405A. Starting from the upper right, the IF3 input of the SC5405A is AC coupled, followed by a switch set by default to switch in the IF port, but it will switch to the signal generator path when the device enables internal signal generation. When internal signal generation is not selected, the signal generator is powered down to prevent possible leakages into the IF path. It is strongly recommended that the IF source be disconnected when internal generation is selected to prevent external leakages. Furthermore, the internal switch does not terminate the external port to a load, possibly causing unwanted signal reflections.

Following the switches are two digital step attenuators that are cascaded together to form a 60 dB attenuation range with 1 dB step increment. These attenuators control the signal level to the IF2 mixer, determining the linearity and noise dynamic ranges of the device. For better linearity the input levels should be lower, and to improve the signal-to-noise ratio (SNR) the levels should be higher. For a compromise, the level at the mixer should be about -16 dBm to -18 dBm, so for a 0 dBm IF input the IF3 attenuator(s) should be set to 16 – 18 dB attenuation respectively.

After the attenuators, the IF3 filter shapes the bandwidth of the device. It is the most selective of all the IF stages. This filter has significant loss so a linear LNA is required to boost the signal prior to passing it through the filter. Following the filter is the IF2 mixer, driven by local oscillator LO3. This LO frequency switches between 745 MHz and 605 MHz to convert the 70 MHz IF to 675 MHz, which is the center frequency of IF2. The default LO2 frequency is 745 MHz, which preserves the spectral polarity of the IF at the RF. Selecting 605 MHz will invert the spectral content of the RF with respect to the IF. This could be useful for some systems, especially those where the IF is digitally sampled up and is spectrally inverted.



3.9 GHz Three Stage Upconverter

Figure 3. Block diagram of the SC5405A.

The second IF (IF2) is filtered to remove LO leakages and images and is well-conditioned by the use of linear LNA to keep the signal levels at an optimum so that SNR is not lost. There is another digital step attenuator in this IF to balance device to device gain variation. It is also used when a large amount of attenuation is required by the device to produce extremely low level signals such as -100 dBm. IF2 is mixed with LO2 at the IF1 mixer to obtain the IF1 frequency of 4675 MHz, which is in turn "downconverted" by LO1 and the RF mixer to obtain the RF frequency range of 1 MHz to 3.9 GHz.

The RF mixer, due to its non-ideal nature, leaks IF1 and LO1 signals through to the RF port. So at the RF port of the mixer, all components as well as their harmonics are present. Although IF1 and LO1 signals are relatively low compared to the RF component and well filtered at the output port, their harmonics could intermodulate with each other and the RF to produce unwanted spurious products inside the RF band. It is important to keep IF1 at the RF mixer low enough but not too low, so that these spurious products are kept to at least -65 dBc below the RF level. The API function **sc5405a_CalcAutoAttenuation** has an input parameter call "perfSelect" (short for performance selection) that computes the attenuation requirement for all the attenuators to meet the performance selected. For best signal-to-noise ratio, the IF at the RF mixer is high, causing larger in-band spurs but greatly improving the close-in signal to noise power.

Following the RF mixer is an elliptic low-pass filter to suppress both IF1 and LO1 leakages from exiting at high levels out of the RF port. The final two RF attenuators are used to control the RF output levels.

Local Oscillator Description

The signal path circuit is separate from the local oscillator generation circuits to maximize isolation between the RF/IF signals and the local oscillators, except for the LO injection paths into the mixers. Although both circuits reside within the same module, well-designed shielding and circuit layouts ensure leakages between them are kept to a minimum.

The first local oscillator, LO1, is an agile, tunable phase-lock synthesizer. The synthesizer tunes from 4675 MHz to 8575 MHz, a tuning range of 3900 MHz. The minimum step size is 1 Hz and is accomplished through a multiple phase-locked loop and DDS hybrid architecture. The use of a hybrid tuning architecture is important for improved phase noise and improved close-in phase spurious responses. Operating LO1 at such high frequencies internally to obtain a 1 MHz to 3.9 GHz RF range requires that the phase noise at these frequencies is sufficiently low so that the converted RF signal phase noise is not degraded significantly. For example, to produce a 100 MHz RF signal, LO1 is tuned to 4775 MHz, which is about 48 times higher in frequency than the output frequency. To further ensure phase noise remains low farther away from the carrier, especially at 100 kHz and 1 MHz offsets, a YIG oscillator is used. It is important to realize that having a phase noise "plateau" out to several tens of MHz, which is a common phenomenon with VCO-based synthesizers, is not acceptable for many applications.

Another reason for a hybrid tuning architecture is to reduce the phase spurs associated with phase-locked loops. A simple fractional PLL may provide resolution to 1 Hz, but it cannot provide 1 Hz frequency tuning steps with low fractional phase spurs. By using two DDS circuits to provide the 1 Hz tuning steps and mathematically ensuring that DDS-generated spurs are suppressed within the architecture, LO1 is made to fine tune to exact frequencies, that is, the frequency synthesized is an exact integer multiple or division of the reference signal.

The second local oscillator, LO2, is fixed at 4.0 GHz, synthesized using an integer PLL and a fixed narrow tune VCO with very low phase noise. The typical raw phase noise of the second stage oscillator is less than -150 dBc/Hz @ 1 MHz offset. LO2 phase noise contribution to the overall phase noise of the device is less than 1 dB. LO1 dominates the phase noise of the device.

The third local oscillator, LO3, is synthesized using a fractional PLL and has phase noise lower than both LO1 and LO2. LO3 is switchable between two frequencies: 605 MHz and 745 MHz, the latter frequency being the default. Both of these frequencies will set the IF center frequency at 70 MHz. However, at the default LO3 frequency the final RF output spectral polarity is the same as that of the input IF, whereas the 605 MHz frequency will create an inverted RF spectrum. If LO3 is set to 605 MHz by calling the **sc5405a_SetIfInversion** function or register, the RF output spectral content will be inverted with respect to the input IF spectrum. See Figure 4 for a graphical representation of this process.

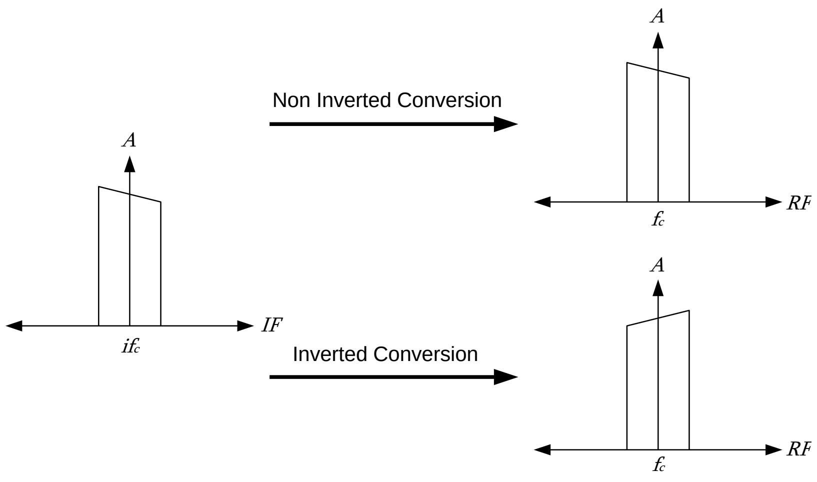


Figure 4. Graphical representation of RF inversion.

All local oscillators are phase-locked to an internal 100 MHz voltage controlled crystal oscillator (VCXO), which sets their close-in phase noise performance. The 100 MHz VCXO is in turn phase-locked to the internal 10 MHz TCXO for frequency accuracy and stability. For better frequency accuracy and stability than the TCXO onboard the SC5405A, or for frequency synchronization, the user can programmatically set the device to phase lock the TCXO to an external 10 MHz reference source by programming the REFERENCE_SETTING register. It is important to note that the TCXO will only attempt to lock to an external source if one is detected. A typical external reference source minimum level of -10 dBm is required for detection to be successful. A reference source level of 0 dBm to +3 dBm is recommended for normal operation. The reference source is feed into the device through the **“ref in”** port. The device can also export a copy of its internal reference through the **“ref out”** port. The output reference frequency is selectable for either 10 MHz or 100 MHz output. By default, routing of the reference signal to the “**ref out**” port is disabled. It can be enabled by programming the REFERENCE_SETTING register. This reference frequency is sourced from the internal 100 MHz OCXO, and the default output selection is 10 MHz, which is divided down from the 100 MHz VCXO. The output reference level is typically +3 dBm.

Frequency Tuning Modes

Tuning of the SC5405A superheterodyne upconverter is accomplished through the tuning of LO1. LO1 has two sets of control parameters that can be explored to optimize the device for any particular application. The first set of parameters, TUNE SPEED, sets the tuning and phase lock time as the frequency is changed. TUNE SPEED consists of two modes - Fast Tune and Normal; both of these modes directly affect the way the YIG oscillator is configured. Fast Tune mode deactivates a noise suppression capacitor across the tuning coil of the YIG oscillator. Doing so increases the rate of current flow through the coil, correspondingly increasing the rate of frequency change. In Normal mode the capacitor is activated, slowing down the rate of frequency change. The advantage of activating the capacitor is that it shunts the noise developed across the coil, decreasing close-in phase noise.

The other set of control parameters, FINE TUNE, sets the tuning resolution of the device. There are three modes: 1 MHz, 25 kHz, and 1 Hz tuning step sizes. The first two modes use only fractional phase detectors to tune the frequency of the LO1 synthesizer, while the third mode enables the DDS to provide 1 Hz resolution. The PLL-only modes (1 MHz and 25 kHz) provide the ability to realize exact frequencies with tuning as fine as 25 kHz. Use of these modes offers several advantages - lower phase spurs and less computational burden to set a new frequency. These modes have the lowest phase spurious signals, below the levels published in the product specification. The DDS mode also tunes to exact frequencies; however, it requires many more computing cycles and additional register-level writes in order to set a new frequency. Comparing times, the device requires up to 175 microseconds to compute and change to a new frequency in PLL only modes but requires up to 350 microseconds in the DDS tuning mode. At first glance it may seem that these differences would directly impact frequency tuning times. However, tuning times are predominantly set by the physical parameters of the YIG oscillator. Computation and register writes typically account for less than 25% of the total tune time of a 10 MHz step change in frequency.

It is important to note that although the synthesized frequencies are exact frequencies, there are observable random phase drifts in the up converted signals. These drifts are due to PLL non-idealities rather than a frequency error in the DDS tuning circuit. Having exact frequency synthesis is important for many applications. Published phase noise and spurs specifications are based on the 1 Hz (DDS) mode.

Setting the SC5405A to Achieve Best Dynamic Range

The SC5405A is designed to achieve high dynamic range. It is designed as a receiver for signal analyzers, which require the ability to handle larger signals well. The design ensures the SFDR dynamic range specification is met when the IF signal is at 0 dBm, 18 dB of IF3 attenuation, and the RF level is at 0 dBm.

For applications where the SNR must be maximized, such as an optimal sine tone, little to no IF3 attenuation should be applied, setting most of the attenuation in the RF attenuators. The IF2 attenuator should be adjusted to lower in-band intermodulation spurs.

The SC5405A is designed for a nominal input IF level of 0 dBm. Depending on the application, the user will need to set the appropriate gain of the device (via attenuation), and the output level, to suit the particular application. For broadband signals, where the absolute noise power is relatively high, the upconverter should be adjusted for improvement in the SNR; that is, decrease IF3 attenuation and increase RF attenuation as necessary to adjust for the desired output level. The optimal setting occurs when the total in-band noise power is at the same level as the IMD products.

For applications that require better linearity to improve close-in inter-modulated products, the IF3 attenuation should be increased. The user should adjust these IF attenuators carefully so that the noise does not dominate.

SignalCore provides a simulation tool that mimics the behavior of the SC5405A. The user may run the simulator to get an understanding of what the parameters need to be set to on the upconverter to achieve certain performance. Additionally, the function **sc5405a_CalcAutoAttenuation** helps the user obtain the necessary attenuator parameters to set up the device for the best compromise of linearity and noise performance for a given set of input and output parameters.

Operating the SC5405A Outside Normal Range

The SC5405A is capable of tuning below 1 MHz and above 3.9 GHz. These frequencies lie outside of the specification range and performance will be degraded if operated in these outer margins. For some applications, the reduced dynamic range or elevated spurious levels in these ranges may not pose an application concern. The lowest tunable frequency is 0 MHz (DC). However, for input frequencies below 1 MHz, the AC coupling capacitors in the circuit limit/attenuate the signal significantly. On the upper end of the spectrum, the input low-pass filter will attenuate the signal rapidly as the frequency increases above 3.9 GHz. Calibration stored on the device EEPROM does not account for these out of range frequencies, so applying any correction using the stored calibration is not valid.

SC5405A Programming Interface

Device Drivers

The SC5405A is programmatically controlled by writing to its set of configuration registers, and its status read back through its set of query registers. The user may choose to program directly at the register level or through the API library functions provided. These API library functions are wrapper functions of the registers that simplify the task of configuring the register bytes. The register specifics are covered in the next section. Writing to and reading from the device at the register level through the API involves calls to the **sc5405a_RegWrite** and **sc5405a_RegRead** functions respectively.

For Microsoft Windows™ operating systems, The SC5405A API is provided as a dynamic linked library, *sc5405a.dll*. This API uses NI-VISA™ to communicate with the device. Inclusion of the NI-VISA driver is required for code development in programming languages such C, C++, or LabVIEW™. For LabVIEW™ support, an additional LabVIEW API, *sc5405a.llb*, is also provided. The functions in the LabVIEW API are primarily LabVIEW VI wrappers to the standard API functions. NI-VISA™ is available from National Instruments Corporation (www.ni.com).

For other operating systems or VISA implementations such as Agilent VISA, users will need to access the device through their own proprietary PXIe driver. The VISA-based driver code is available to our customers by request. This code can be compiled with Agilent VISA with minimal or no code change. Should the user require assistance in writing an appropriate API other than that provided, please contact SignalCore for additional example code and hardware details.

Using the Application Programming Interface (API)

The SC5405A API library functions make it easy for the user to communicate with the device. Using the API removes the need to understand register-level details - their configuration, address, data format, etc. For example, to obtain the device temperature the user simply calls the function **sc5405a_GetDeviceTemperature**, or calls **sc5405a_SetFrequency** to set the device frequency. The software API is covered in detail in the "Software API Library Functions" section.

Setting the SC5405A - Writing To Configuration Registers

Configuration Registers

The users may write the configuration registers (write only) directly by calling the **sc5405a_RegWrite** function. The following table lists the register address (command) and the effective bytes of command data. Data must be formatted into an unsigned integer of 32 bits prior to passing it to the function. As an example, to write the byte 0xEE into address 0xA8BB of the user EEPROM, the user would call the **sc5405a_RegWrite** as follows:

sc5405a_RegWrite(deviceHandle, 0x23, 0x00A8BBEE)

Table 2: Configuration registers.

Register (Address)	Data Bytes	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Default
INITIALIZE (0x01)	[7:0]	Open	Open	Open	Open	Open	Open	Open	Mode	0x00
SET_SYSTEM_ACTIVE (0x02)	[7:0]	Open	Open	Open	Open	Open	Open	Open	Enable SYS LED	0x00
POWER_SHUT_DOWN (0x05)	[7:0]	Open	Open	Open	Open	Open	Open	Open	Enable	0x00
RF_FREQUENCY (0x10)	[7:0]	Frequency Word [7:0]								0x00
	[15:8]	Frequency Word [15:8]								0x00
	[23:16]	Frequency Word [23:16]								0x00
	[31:24]	Frequency Word [31:24]								0x00
ATTENUATOR_SETTING (0x11)	[7:0]	Attenuator Value								0x00
	[15:8]	Attenuator								0x00
RESERVED (0x12)	[7:0]	Open	Open	Open	Open	Open	Open	Open	Enable	0x00
RF_MODE_SETTING (0x13)	[7:0]	Open	Open	Open	Open	Open	Fast Tune Enable	Fine Tune		0x00
IF_FILTER_SELECT (0x15)	[7:0]	Open	Open	Open	Open	Open	Open	Open	Band	0x00
REFERENCE_SETTING (0x16)	[7:0]	Open	Open	Open	Open	Reserv	100 MHz Out Sel	Ref Out Enable	Lock Enable	0x00
REFERENCE_DAC (0x17)	[7:0]	DAC word [7:0]								0x00
	[15:8]	DAC word [15:8]								0x00
IF_INVERT_SETTING (0x1D)	[7:0]	Open	Open	Open	Open	Open	Open	Open	Enable	
WRITE_USER_EEPROM (0x23)	[7:0]	EEPROM DATA [7:0]								
	[15:8]	EEPROM Address [7:0]								
	[23:16]	EEPROM Address [15:8]								
PHASE_SETTING (0x32)	[7:0]	Units Value[7:4]				Tenths Value				0x00
	[15:8]	Open	Open	Units Value [13:8]						0x00
SET_SIGGEN (0x1B)	[7:0]	Open	Open	Open	Open	Open	Open	Open	Enable	0x00

Tuning the RF Frequency

The frequency of the first local oscillator (LO1) is set by writing the RF_FREQUENCY register (0x10). This register requires four data bytes, these data bytes being the bytes comprising an unsigned 32-bit integer. The data bytes contain the frequency tuning word in Hertz. For example, to tune to a frequency of 2.4 GHz, the data word would be d2400000000 in decimal or 0x8F0D1800 in hexadecimal.

Changing the Attenuator Settings

The ATTENUATOR_SETTING (0x11) register has two data bytes needed to set the value of a specific attenuator. The MSB sets the target attenuator, and the least significant byte (LSB) contains the attenuation value. The MSB values and corresponding attenuator locations are as follows:

MSB value	Attenuator
0x00	IF3_ATTEN1
0x01	IF3_ATTEN2
0x02	RF_ATTEN1
0x03	RF_ATTEN2
0x04	IF2_ATTEN

The LSB contains the attenuation value in 1 dB steps for the attenuator specified in the MSB. For example, to set the RF attenuator #1 to 15 dB, the command data would be 0x020F.

Changing the RF Synthesizer Mode

The RF_MODE_SETTING (0x13) register has one data byte that provides two tuning modes for the device: Fast Tune and Fine Tune. By default the Fast Tune mode is disabled (Normal mode). Asserting high bit 3 of the data byte will enable Fast Tune mode. Fast Tune enables the device to achieve faster lock and settling times between frequency changes. The second mode, Fine Tune mode, has three options: 1 MHz (PLL), 25 kHz (PLL), and 1 Hz (DDS). Selection of these options requires setting the first two bits of the data byte to 0, 1, and 2, respectively. See the "Frequency Tuning Modes" section for more information. For example, to set the device for Fast Tune and a 1 Hz tuning step resolution, the command data would be 0x06.

Selecting the IF Filter Path

The IF_FILTER_SELECT (0x15) register has one data byte that selects between two installed IF filters: IF3_FILTER0 and IF3_FILTER1. Setting bit 0 high will select IF_FILTER1. The exact bandwidths of the filters depend on the available installed options and are stored in the device calibration EEPROM.

Setting the Reference Clock Behavior

The REFERENCE_SETTING (0x16) register has one data byte which sets the reference clock behavior of the device. The default state of this register is 0x00, which disables the export of the internal reference clock, and disables phase locking to an external source. Asserting bit 0 high enables the device to lock to an external clock source. However, the device will not attempt to phase lock until it successfully detects the presence of a clock source at the "**ref in**" port. Asserting bit 1 low disables export of the internal clock. Asserting bit 1 high enables the device to export a 10 MHz signal through the "**ref out**" port. Asserting high bit 1 and bit 2 exports a 100 MHz signal.

Adjusting the Reference Clock Accuracy

The frequency precision of the SC5405A's 10 MHz TCXO is set by the device internally. The device writes the factory calibrated value to the reference DAC on power-up. This value is an unsigned 16-bit number stored in the EEPROM (see the calibration EEPROM map). The user may choose to write a different value to the reference DAC by accessing the REFERENCE_DAC (0x17) register. Command data is 16 bits maximum.

Setting Spectral Inversion in the RF

The default RF spectral polarity is the same as that of the IF input. However, should there be a need to invert the RF spectrum with respect to the IF spectrum, the register IF_INVERT_SETTING (0x1D) is used. This register contains one data byte. Setting bit 0 high will enable inversion.

Storing Data into the User EEPROM Space

There is an on-board 16k byte EEPROM available to the user to store user data information such as user calibration, settings, etc. Writing to the user accessible EEPROM space is accomplished through the register WRITE_USER_EEPROM (0x23). This register has three data bytes: bytes 2 and 1 contain the address of the EEPROM; byte 0 is the byte value to be written. **NOTE:** The user must add a 5 millisecond delay between consecutive writes. There is no delay required in read mode. For example, to write 123 to address 1234 of the user EEPROM the command data would be 0x04D27B.

Setting the Phase of the RF Signal

When the device is tuned to a fixed RF frequency, some applications may need to change the phase of the up converted signal for various reasons. The RF output phase can be programmatically adjusted in 0.1 degree increments from 0 to 360 degrees. Changing the phase is accessed through the register PHASE_SETTING (0x32), which has two data bytes. The first 4 bits contain the tenths value, while bits [13:4] hold the units value. The command data to adjust for 45.5 deg phase shift in the signal would be 0x02D5.

Querying the SC5405A: Writing To Request Registers

The request set of registers shown below inTable **3** are used to retrieve data from the device. They are accessed by calling the **sc5405a_RegRead** function. The parameters this function passes are listed in the "Software API Library Functions" section, which is repeated here for convenience.

sc5405a_RegRead(unsigned int *deviceHandle, unsigned char commandByte, unsigned int instructWord, unsigned int *receivedWord)

Table 3: Query registers.

Register (Address)	**Data Byte**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**	**Default**
FETCH_DEVICE_STATUS (0x18)	[7:0]	Open	Open	Open	Open	Open	Open	Open	TRUE	0x00
Read Byte 1	[7:0]	Device status [7:0]								
Read Byte 0	[15:8]	Device status [15:8]								
FETCH_TEMPERATURE (0x19)	[7:0]	Open	Open	Open	Open	Open	Open	Open	Open	0x00
Read Byte 1		Temp data [7:0]								
Read Byte 0		Open	Open	Sign	Temp data [12:8]					
FETCH_CAL_EEPROM (0x20)	[7:0]	EEPROM Address [7:0]								0x00
	[15:8]	EEPROM Address [15:8]								0x00
Read Byte 1		Cal EEPROM byte data								
Read Byte 0		Open	Open	Open	Open	Open	Open	Open	Open	
FETCH_USER_EEPROM (0x22)	[7:0]	EEPROM Address [7:0]								0x00
	[15:8]	EEPROM Address [15:8]								0x00
Read Byte 1		User EEPROM byte data								
Read Byte 0		Open	Open	Open	Open	Open	Open	Open	Open	

Reading the Device Status

To obtain the device status, write request register FETCH_DEVICE_STATUS (0x18) with 0x01 for the instructWord, and data is returned via the receivedWord pointer. The returned data are summarized in Table 4. It is important to note that the first local oscillator has three phase detectors in the synthesizer, so all three phase detectors must be ANDed to indicate the proper phase-locked status. The three bits that indicate the status of the three phase detectors are [13], [10], and [9].

Table 4. Description of the status data bits.

Bit	Description
[15]	10 MHz TCXO PLL lock status
[14]	100 MHz VCXO PLL lock status
[13]	LO1 PLL Main lock status
[12]	LO2 PLL lock status
[11]	LO3 PLL lock status
[10]	LO1 PLL1 lock status
[9]	LO1 PLL2 lock status
[8]	Reserved
[7]	External reference detected
[6]	Reference output enabled
[5]	Reference lock enabled
[4]	IF3_FILTER1 selected
[3]	Reserved
[2]	Device standby enabled
[1]	PXI 10 enable
[0]	Reserved

Reading the Device Temperature

To obtain temperature data, write request register GET_TEMPERATURE (0x19) with 0x00 for the instructWord. Temperature data is returned via the receivedWord pointer. Once data is received, the least two bytes of data need to be processed to correctly represent the data in temperature units of degrees Celsius. Data is returned in the first 14 bits [13:0]. Bit [13] is the polarity bit indicating whether it is a positive (0x0) or negative (0x1) value. The temperature value represented in the raw data is contained in the next 13 bits [12:0]. To obtain the temperature ADC code, the raw data should be masked (logically ANDed) with 0x1FFF, and the polarity should be masked with 0x2000. The conversion from 12 bit ADC code to an actual temperature reading in degrees Celsius is shown below:

Positive Temperature	=	ADC code / 32
Negative Temperature	=	(ADC code – 8192) / 32

It is recommended to avoid reading the temperature too frequently, especially once the SC5405A temperature has stabilized. The temperature sensor is a serial device located inside the RF module. Therefore, like any other serial device, reading the temperature sensor requires sending serial clock and data commands from the processor. The process of sending clock pulses on the serial transfer line may cause unwanted spurs on the RF signal as the serial clock potentially modulates the local oscillators. Furthermore, once the SC5405A stabilizes in temperature, repeated readings will likely differ by as little as 0.25 °C over extended periods of time. Given that the gain-to-temperature coefficient is on the order of 0.06 dB/°C, gain changes between readings will be negligible.

Reading the Calibration EEPROM

To read a single byte from an address in the device EEPROM write the FETCH_CAL_EEPROM register with the address for the instructWord, and data is returned via receivedWord. The byte data is contained in the least byte of the 32 bit unsigned integer so data must be typed-casted back to an unsigned character byte. The EEPROM maximum address is 0x3FFF. Reading above this address will cause the device to retrieve data starting from the lowest addresses. For example, addressing 0x4000 will return data stored in address location 0x0000. The calibration EEPROM map is discussed in detail in the **"**Working With Calibration Data**"** section.

All calibration data, whether floats or unsigned 32-bit integers, are stored as flattened unsigned 32-bit words. Each data point is comprised of four unsigned bytes, so data must be read back in multiples of four bytes, with the *least significant byte stored in the lower address*. After the data are read back, they need to be un-flattened back to their original type. Since the four bytes constitutes the four bytes of an unsigned 32-bit integer, converting (un-flattening) to an unsigned value simply involves concatenation of the bytes through bit shifting. To convert to floating point representation is a little more involved. First, convert the four bytes into an unsigned 32-bit integer value, and then (in C/C++, etc.) type-cast a float pointer to the address of the value. In C/C++, the code would be float Y = *(float *)&X, where X has been converted earlier to an unsigned integer.

An example written in C code would look something like the following:

```c
byte_value[4]; // read in earlier
unsigned int uint32_value;
float float32_value;

int count  =  0;
while (count  <  4) {
       uint32_value = unit32_value | (byte_value[count] << (count*8));
       count++;
}

float32_value = *(float *)&uint32_value;
```

Reading the User EEPROM

Once data has been written to the user EEPROM, it can be retrieved using the process outlined above for reading calibration data by calling the FETCH_USER_EEPROM registers.

Working With Calibration Data

The device EEPROM on board has capacity for 16k bytes of data. The EEPROM stores both device information and calibration data, which the user may choose to use to correct for conversion gain. Users are not required to use the onboard calibration to compensate for the gain errors associated with temperature, attenuator settings, frequency, pass-band ripple, and filter path selection. Alternatively, users can perform their own system calibration to remove these errors if the unit is integrated into a larger system whose external factors affect the gain significantly. Furthermore, the calibration data provided are raw measured data, and it is at the user's discretion to decide on the appropriate methods of applying the calibration. For example, the user may choose to fit the measured data to a polynomial and use the polynomial coefficients to compute the necessary correction. Alternatively, the user may choose to perform correction through the use of interpolation. The methods outlined in this section only serve as guides on how to use the calibration data for correction, and these are the methods used by SignalCore in deriving published specifications that indicate the use of calibration. The function **sc5405a_CalcGain** utilizes the methods outlined here and may be used to compute the device gain for any particular setting of the device.

Table 5. EEPROM memory map of device attributes and raw calibration data.

EEPROM ADD (HEX)	# DATA POINTS	# BYTES	TYPE	Array matrix	DESCRIPTION
0X00	1	4	U32	[1x1]	Manufacturing Information
0X04	1	4	U32	[1x1]	Product Serial Number
0X08	1	4	U32	[1x1]	RF Module Number
0X0C	1	4	U32	[1x1]	Product Manufacture Date
0x10	1	4	U32	[1x1]	Last Calibration Date
0X2C	1	4	F32	[1x1]	Firmware Revision
0x30	1	4	F32	[1x1]	LO Hardware Revision
0x34	1	4	F32	[1x1]	SC Hardware Revision
0x50	1	4	F32	[1x1]	Calibration Temperature
0x54	1	4	U32	[1x1]	TCXO DAC Value
0x58	2	8	F32	[1x2]	Reserved
0x64	72	288	F32	[3x24]	YIG Calibration (Reserved)
0x184	1	4	F32	[1x1]	IF Filter 0 Bandwidth
0x188	1	4	F32	[1x1]	IF Filter 1 Bandwidth
0x1A0	24	96	F32	[3x8]	Temperature Coefficients
0x204	153	612	F32	[3x51]	IF3_FILTER0 Response Calibration
0x46C	153	612	F32	[3x51]	IF3_FILTER1 Response Calibration
0x788	1	4	F32	[1x1]	IF Invert Gain
0x78C	1	4	F32	[1x1]	Reserved
0x790	1	4	F32	[1x1]	IF3_FILTER1 Gain
0x798	90	360	F32	[3x30]	IF Attenuator Calibration
0x9F8	1650	6600	R32	[33x50]	RF calibration

Table 5 lists the calibration EEPROM map of the SC5405A, indicating how and where board information and calibration data are stored. Since there are only 16k bytes on the EEPROM, SignalCore recommends that all data be read into host memory on initialization of the device and parsed for further mathematical manipulation. Having it available on host memory at all times during an application will greatly increase the speed of data manipulation. Another recommendation is to store the data to a file and have the application read the file rather than the EEPROM to retrieve data on each execution because of the relative slow EEPROM read rate. The **sc5405a_ConvertRawCalData** function is helpful to convert EEPROM data to their original format and types.

EEPROM Data Content

The following list describes the data contents of the EEPROM in detail. All addresses shown are the starting offset positions in the EEPROM and are the starting address for a block of data.

Manufacturing Information *(0x00).* This is an unsigned integer value that contains information used by the factory for production purposes.

Product Serial Number *(0x04).* This is an unsigned integer value that contains the SC5405A serial number. It is unique for every product produced. It is used for the purpose of tracking the history of the product.

RF Module Serial Number *(0x08).* This is the serial number of the shielded RF metal enclosure containing the analog and RF circuitry. All calibration data are stored on the EEPROM within the enclosure. Calibration data are written to this EEPROM at the factory are tracked using the RF module serial number for the SC5405A.

Product Manufacture Date *(0x0C).* This is an unsigned integer: byte 3 is the Year, byte 2 is the Month, byte 1 is the day of the month, and byte 0 is the hour of the day.

Last Calibration Date *(0x10).* This is an unsigned integer: byte 3 is the Year, byte 2 is the Month, byte 1 is the day of the month, and byte 0 is the hour of the day.

Firmware Revision *(0x2C).* This is a float 32 value containing the firmware revision.

LO Hardware Revision *(0x30).* This is a float 32 value containing the local oscillator hardware revision.

SC Hardware Revision *(0x34).* This is a float 32 value containing the signal chain hardware revision.

Calibration Temperature *(0x50).* This is a float 32 value containing the temperature at which the device was calibrated.

TCXO DAC Value *(0x54).* This is an unsigned integer containing the value for the reference DAC to adjust the precision of the temperature-compensated crystal oscillator (TCXO).

YIG Calibration *(0x64).* Data is reserved for device use.

IF Filter Bandwidths *(0x184, 0x188).* These two float 32 data points contain the filter bandwidths of IF3_FILTER0 and IF3_FILTER1, respectively. These are only available if the product contains non-standard filters, different from those provided with the base product.

Gain / Temperature Coefficients *(0x1A0).* This is a 3x8 float matrix, where data is concatenated by rows, that is, data is read back row by row. These coefficients, derived during calibration, are needed to compute for gain as a function of temperature. They are 2nd order polynomial coefficients and are measured over eight different frequencies. See “Gain Correction” section for more information on gain correction factors.

Table 6 is an example of the coefficient data and their format. Variables a_1 and a_2 are the first and second order coefficients.

Table 6. An example of gain-temperature coefficients data and format.

Frequency (MHz)	50.0	250	500	1000	1500	2500	2800	3800
a_1	-0.04500	-0.04800	-0.05600	-0.05000	-0.04500	-0.04800	-0.05600	-0.05000
a_2	-0.00038	-0.00035	-0.00029	-0.00038	-0.00038	-0.00035	-0.00029	-0.00038

IF3_FILTER0 Response Calibration *(0x204).* This is a 3x51 float matrix, and data is read back row by row. This set of data measures pass-band amplitude variation with respect to the center IF, and phase deviation from linear phase of filter IF3_FILTER0. There are a total of 51 offset frequency points from the center IF frequency measured inside the bandwidth of the filter. Table 7 is an example of the data and format.

IF3_FILTER1 Response Calibration *(0x46C).* This is a 3x51 float matrix, and data is read back row by row. This set of data measures pass-band amplitude variation with respect to the center IF, and phase deviation from linear phase of filter IF3_FILTER1. There are a total of 51 offset frequency points from the center IF frequency measured inside the bandwidth of the filter. Table 7 is an example of the data and format.

Table 7. Relative IF gain and phase response calibration and format.

Frequency Offset (MHz)	-12	-11.5	. . .	-5	. . .	-.5	0	.5	. . .	5	. . .	11.5	12
Gain Error	-58.2	-20.6	. . .	0.9	. . .	0.2	0.0	0.3	. . .	-0.2	. . .	-12.8	-30.4
Phase Error(radians)	.286	.138	. . .	.053	. . .	.012	0	.011	. . .	. . .	. . .	-.129	-.134

IF Invert Gain Correction *(0x788).* This is a float that contains the change in IF gain when the device is switched to invert the IF spectrum. The default gain in the IF is the non-inverted mode gain.

IF3_Filter1 Gain Correction *(0x790).* This is a float that contains the change in IF gain when the device is switched to the IF3_FILTER1 path. The default gain in the IF is the IF3_FILTER0 path gain.

IF Attenuator Calibration *(0x798).* This is a 3x30 float matrix, containing the calibrated attenuation values of the three IF attenuators. Data is read in row by row. Each attenuator has 30 attenuation steps, and each row correspond to one attenuator. The first row is the attenuation values of IF3_ATTEN1, the second row contains the values or IF3_ATTEN2, and the third row contains the values of IF2_ATTEN. Table 8 is an example of the data and its format. Since the IF bandwidth is typically less than 40 MHz wide and centered at a fixed frequency, it is sufficient to perform the calibration at the center IF as attenuation variation is insignificant over its range.

Table 8. An example of IF attenuation calibration.

IF3_Attenuator 1	0.973	1.927	2.948	3.912	.	.	.	.	28.630	29.634
IF3_Attenuator2	0.989	1.970	2.998	3.989					28.890	29.874
IF2_ Attenuator	0.995	2.028	3.038	4.023	.	.	.	.	28.868	29.854

RF calibration *(0x9F8).* This is a 63x50 float matrix. **Table 9** is an example of the data and format for the RF calibration data. RF calibration contains data for preamplifier gain, gain with zero RF and IF attenuation, and attenuation values of the RF attenuator for fifty frequency points that span the operational frequency range of the SC5405A. Data is read in concatenation of rows. For example, all the frequency values are read in first, then the preamplifier gain values, followed by the zero attenuation-setting gain, etc. There are a total of 12600 values read that must be read from the EEPROM to form the full set of calibration.

Table 9. Example of the RF calibration data and its format.

Frequency (MHz)	3	5	10	.	500	.	.	.	3875	3900
Preamp Gain	20.564	20.643	20.456	.	20.003	.	.	.	19.654	19.231
Gain	33.223	33.423	33.213	.	33.102	.	.	.	29.980	29.450
RF_ATTEN1 1 dB	0.988	0.955	1.093	.	0.973	.	.	.	1.008	0.995
RF _ATTEN1 2 dB	1.921	2.001	2.045	.	2.056	.	.	.	1.932	2.051
.	.	.	.	.	.	.	.	.	.	.
RF_ATTEN1 30 dB	29.645	29.854	30.065	.	29.588	.	.	.	29.260	29.572
RF_ATTEN2 1 dB	0.988	0.955	1.093	.	0.973	.	.	.	1.008	0.995
.	.	.	.	.	.	.	.	.	.	.
RF_ATTEN2 30 dB	29.645	29.854	30.065	.	29.588	.	.	.	29.260	29.572

Frequency Correction

On power-up, the SC5405A automatically applies the calibration value to the on-board reference DAC that controls the TCXO, which is the primary frequency reference of the device. The user may choose to reprogram the DAC with the 16 bit code obtained from the EEPROM at starting address 0x54, or with another value, by writing the REFERENCE_DAC register (0x17).

Gain Correction

The SC5405A has six dynamic variables that affect its gain, namely, IF attenuator settings, RF attenuator setting, filter path, inversion gain, input frequency, and temperature. Correction of gain needs to take into account five main factors. As noted in the "EEPROM Data Content" section, the pre-amplifier gain, through gain (no attenuation, no pre-amplification), RF attenuation, and gain-over-temperature variation are calibrated over the span of the SC5405A frequency range. These are the frequency dependent parameters that are combined with the IF attenuation to make the total gain calculation.

Let us start by writing the gain equation with no dependence on temperature or frequency. We get the following equation.

$$G_{dev} = G_{preamp} + G + \Delta G_{fil} + \Delta G_{inv} - A_{rf} - A_{if} \qquad \text{Equation 1}$$

G_{dev} is the total gain of the device, G_{preamp} is the gain of the preamplifier, G is the through gain, A_{rf} is the attenuation of the RF attenuator, ΔG_{fil} is the gain change of IF3_FILTER1 path, ΔG_{inv} is the gain change in spectral inversion mode, and A_{if} is the attenuation of the the IF attenuators. If the preamplifier is off, no inversion, default filter path, and no attenuation applied then $G_{dev} = G$.

Writing Equation 1 with dependency on temperature we add on the temperature dependent gain factor $\Delta G(T)$ and obtain the following:

$$G_{dev}(T) = G_{preamp}(T_0) + G(T_0) + \Delta G_{fil}(T_0) + \Delta G_{inv}(T_0) - A_{rf}(T_0) - A_{if}(T_0 + \Delta G(T) \qquad \text{Equation 2}$$

where T is the temperature of the device and T_0 is the fixed temperature at which calibration was performed. The "Reading the Device Temperature" section provides information on how unsigned raw temperature data is converted to Celsius, a floating point type. Taking the frequency dependence of the measured parameters into consideration, Equation 2 may be rewritten as

$$G_{dev}(f,T) = G_{preamp}(f,T_0) + G(f,T_0) - A_{rf}(f,T_0) + \Delta G(f,T) + \Delta G_{fil}(T_0) + \Delta G_{inv}(T_0) - A_{if}(T_0) \qquad \text{Equation 3}$$

Note that the IF attenuation values, $A_{if}(T_0)$ do not need to be frequency dependent as discussed earlier. Using the IF attenuator calibration is as simple as substituting the intended value with the calibrated value. From Table 8, one would use 29.854 dB for an intended 30 dB attenuation. The other two variables, $\Delta G_{fil}(T_0)$ and $\Delta G_{inv}(T_0)$, are also frequency independent as they are only referred to at the center of the IF band, and their values are simply summed in the total gain equation. Only those parameters that depend on frequency and/or temperature are treated below.

To obtain calibrated gain values from the parameters that are a function of frequency, interpolation is required to provide the best estimated values. A natural cubic Spline interpolation is suggested for $G_{preamp}(f,T_0)$, and $A_{rf}(f,T_0)$. The important input parameters for a cubic spline interpolation are the two arrays $[X]$ and $[Y]$, and an arbitrary point x. The output of the interpolation is some interpolated value y based on the inputs. $[X]$ is the set of independent values, $[Y]$ is the set of $[X]$ dependent values, and x is an arbitrary independent value to obtain the interpolated value y. For example, Table 10 lists the input and output parameters to obtain the gain $G(f = 1000\ MHz, T_0)$.

Table 10. Parameters for a spline interpolation.

Frequency (MHz); $[X]$	3	5	19	…	…	950	1050	…	3875	3900
Measured Gain; $[Y]$	33.223	33.423	33.213	…	…	32.652	32.482	…	29.980	29.450
$f; x$	1000									
$G(f,T_0); y$	32.453									

From experience, having a large [X] and [Y] array of points does not necessarily provide the best interpolated value due to the nature of trying to fit a function over many points and over many octaves of frequency. Better results are obtained from a set of localized calibrated points around the point of interest. The function **sc5405a_CalcGain** uses six localized [X] points to compute the interpolated point. Using localized points, the example on Table 10 is re-tabulated in

Table 11. Similarly, frequency dependent preamplifier gain and RF attenuation may be derived.

Table 11. Localized parameters to a spline interpolation.

Frequency (MHz); $[X]$	850	900	950	1050	1100	1150
Gain; $[Y]$	32.681	32.673	32.652	32.482	32.419	32.418
$f; x$	1000					
$G(f, T_0); y$	32.532					

To find the change in gain with respect to change in temperature involves a couple of steps. First, determine the array values of $[\Delta G(f_i, T)]$, where f_i is the frequency point at which a measurement was made. Second, use interpolation to determine the $\Delta G(f, T)$ at some frequency f. Again, natural cubic spline interpolation is recommended in the second step. Let us outline a method to determine $\Delta G(f_i, T)$ at frequency f_i; there are a total of 8 frequency points for this calibration.

The calibration values retrieved from the EEPROM are second order polynomial coefficients fitted to measured data. Writing the general form of the gain function using coefficients, we have:

$$G(f_i, T) = a_0(f_i) + a_1(f_i)T + a_2(f_i)T^2 \qquad \text{Equation 4}$$

Here $a_j(f_i)$ is the j^{th} order coefficient measured some frequency f_i. The gain deviation at temperature T from the gain measured at the calibration temperature T_0 can be written as:

$$\begin{aligned}\Delta G(f_i, T) &= G(f_i, T) - G(f_i, T_0) \\ &= a_1(f_i)[T - T_0] + a_2(f_i)\left[T^2 - {T_0}^2\right]\end{aligned} \qquad \text{Equation 5}$$

Using Equation 5 and the temperature coefficients of

Table 6, we obtain the following:

Table 12. Calculated gain changes at the measured frequency points.

Frequency (MHz) (f_i)	50.0	250	500	1000	1500	2500	2800	3800
$\Delta G(f_i, T = 45^O)$ dB	-0.340	-0.343	-0.351	-0.348	-0.354	-0.351	-0.355	-0.357

After determining the set of gain deviations at some temperature T, we apply spline interpolation to the set of $\Delta G(f_i, T)$ values to obtain $\Delta G(f, T)$, change in gain with respect to both temperature and frequency. Using Table 12 and the convention developed here, the Spline parameters are $[X] = [f_i]$, $[Y] = [\Delta G(f_i, T)]$, $x = f$, and $y = \Delta G(f, T)$.

IF Response Correction

The gain correction procedure outlined above only applies to a signal that is centered in the 70 MHz IF band. The device's Fine-Tune mode (1 Hz) is able to place any RF signal at the center of the IF, so for narrow bandwidth signals (typically less than a MHz), applying the center IF gain correction and assuming no deviation from linear phase is sufficient. However, for a large bandwidth signal that spans several MHz, it is important to apply gain and phase correction to the offset frequencies; those that are offset from the center IF. Although SignalCore performs calibration of the amplitude and phase over the bandwidth of the IF filters (available on the device calibration EEPROM), it is recommended that the user perform in-situ system equalization for digital broadband applications for improved performance.

Measured IF gain and phase error response is available for both filter paths; the user simply needs to properly select the path of interest. The measurement is made using a vector network analyzer in the frequency domain, covered by fifty-one evenly spaced frequency points. The amplitude (gain) error values are measured with respect to the center frequency and are given in decibels, while the phase error values are in radians. The phase errors are deviations from linear phase. Each set of calibrated points consists of a 3x51 floating point array (see Table 7 as an example). There are several ways to apply the frequency domain calibration:

1. Determine a fitted polynomial function for the amplitude error (gain) and multiply this function with the uncorrected amplitude spectrum. Add the two values if dealing in decibels. Additionally, determine a fitted polynomial function for the phase error and add values derived from this function with the uncorrected phase. To derive, let $X(e^{i\omega})$ be the measured uncorrected value, $E(e^{j\omega})$ be the fitted polynomial to the calibrated error values, and $Y(e^{j\omega})$ be the corrected measured value. Also let $Arg[\cdot]$ denote the principle value of the phase of the above terms, and we can relate all the terms as

$$Y\left(e^{j\omega}\right) = E\left(e^{j\omega}\right)X\left(e^{j\omega}\right)$$
$$\left|Y(e^{j\omega})\right|e^{jArg[Y(e^{j\omega})} = \left|E(e^{j\omega})\right|\left|X(e^{j\omega})\right|e^{j(Arg\left[Y\left(e^{j\omega}\right)\right]+Arg\left[Y\left(e^{j\omega}\right)\right])}$$

   From the above equation, we see that the magnitude terms are multiplied and the phase terms added. In the discrete sense (digitized) for every frequency value, ω_i, we apply the above equation to correct for the non-ideality of the IF filter.

2. The other method finds the magnitude and error points through interpolation methods such as Spline, then multiplying the error magnitude with the uncorrected magnitude and adding the error and uncorrected phases. This is similar to method 1, but instead of using a fitted function to obtain the error values, interpolation is used. Interpolation is generally a slower process. This is the method implemented in the library function **sc5405a_CalcIfResponseCorrection**.

Software API Library Functions

SignalCore's philosophy is to provide products to our customers whose lower hardware functions are easily accessible. Having the ability to access the registers directly is a necessity for those who wish to use direct low-level control of frequency and gain settings. However, others may wish for simpler product integration by using higher level function libraries and by not programming registers directly. The functions listed below comprise the function set of the dynamic-linked library (Windows operating systems) and shared library (Linux operating system) versions of the SC5405A API. The LabVIEW palette library differs slightly due to the unique requirements of the LabVIEW programming environment (e.g., LabVIEW already provides standard math functions for curve fitting with spline interpolation).

- **sc5405a_ListResources**
- **sc5405a_OpenDevice**
- **sc5405a_CloseDevice**
- **sc5405a_RegWrite**
- **sc5405a_RegRead**
- **sc5405a_InitDevice**
- **sc5405a_SetStandby**
- **sc5405a_SetFrequency**
- **sc5405a_SetSignalChain**
- **sc5405a_SetSynthesizerMode**
- **sc5405a_SetIfFilterPath** ***(not available on the SC5405A)***
- **sc5405a_SetReferenceClock**
- **sc5405a_SetReferenceDac**
- **sc5405a_SetIfInversion**
- **sc5405a_WriteUserEeprom**
- **sc5405a_SetSignalPhase**
- **sc5405a_SetSpurMode**
- **sc5405a_SetSigGenEnable**
- **sc5405a_GetDeviceStatus**
- **sc5405a_GetTemperature**
- **sc5405a_ReadCalEeprom**
- **sc5405a_ReadUserEeprom**
- **sc5405a_ReadUserEepromBulk**
- **sc5405a_GetRawCalData**
- **sc5405a_GetCalData**
- **sc5405a_CalcAutoAttenuation**
- **sc5405a_CalcGain**
- **sc5405a_CalcIfResponseCorrection**
- **sc5405a_ConvertRawCalData**
- **sc5405a_ConvertRawTempData**
- **sc5405a_Spline**
- **sc5405a_SplineInterp**

Each of these functions is described in more detail on the following pages. To program in C/C++, SignalCore defines the following constants and types which are contained in the C header file, *sc5405a.h.* These constants and types are useful not only as an include file for developing applications using the SC5405A libraries, but also for writing device drivers independent of those provided by SignalCore.

Constants Definitions

```
/* 2-D parameters for storing calibration data */
#define RFCALPARAM                          63    // rows of caldata
#define RFCALFREQ                           50    // frequency points
#define IFATTENUATOR                         3    // total number of IF attenuators
#define IFATTENCALVALUE                     30    // attenuation steps 1-30 dB
#define IFRESPONSEPARAM                      3    // frequency, amplitude, phase
#define IFRESPONSEFREQ                      51    // frequency points over the bandwidth
#define TEMPCOPARAM                          3    // frequency, coeff(1) , coeff(2)
#define TEMPCOFREQ                           8    // frequency points

/* Attenuator assignment */
#define IF3ATTENUATOR1                       0
#define IF3ATTENUATOR2                       1
#define RFATTENUATOR1                        2
#define RFATTENUATOR2                        3
#define IF2ATTENUATOR                        4

#define CALEEPROMSIZE                    15168
#define USEREEPROMSIZE                   16384

/* Tune mode parameters */
#define FASTTUNEENABLE                       1
#define DISABLEDFINEMODE                     0    // 1 MHz tuning steps, PLL implementation
#define PLLFINEMODE                          1    // 25 KHz tuning steps, PLL implementation
#define DDSFINEMODE                          2    // 1 Hz tuning steps, DDS implementation

//Performance selections used in CalculateAutoAttenuation to set the attenuators
#define OPTIMIZEDLINEARITYANDNOISE           0
#define BETTERLINEARITY                      1
#define BETTERSNR                            2
#define OPTIMALSNR                           3

/* Error set */
#define SUCCESS                              0
#define DEVICEERROR                         -1
#define TRANSFERERERROR                     -2
#define INPUTNULL                           -3
#define COMMERROR                           -4
#define INPUTNOTALLOC                       -5
#define EEPROMOUTBOUNDS                     -6
#define INVALIDARGUMENT                     -7
#define INPUTOUTRANGE                       -8
```

```
#define NOREFWHENLOCK                              -9
#define NORESOURCEFOUND                            -10
#define INVALIDCOMMAND                             -11
```

Type Definitions

```
typedef         unsigned char   bool;

typedef struct deviceAttribute_t
{
        unsigned int productSerialNumber;
        unsigned int rfModuleSerialNumber;
        float firmwareRevision;
        float loHardwareRevision;
        float scHardwareRevision;
        unsigned int *calDate; //size of 4 year,month,day,hour
        unsigned int *manDate; //size of 4 year,month,day,hour
}       deviceAttribute_t;

typedef struct calibrationData_t
{
        float **rfCal;    // RF gain calibration
        float **ifAttenCal; // IF attenuators calibration
        float **ifFil0ResponseCal; // IF filter 0 response calibration
        float **ifFil1ResponseCal; // IF filter 1 response calibration
        float **tempCoeff; // temperature coefficients
        float rfCalTemp; // temperature T0 at which calibration was done
        float ifFilter1GainError; // Gain error when switched to filter 1 path
        float ifFilter0Bw; // filter 0 BW in MHz
        float ifFilter1Bw; // filter 1 BW in MHz
        float invertGainError;  // gain error when spectral inversion enabled
        unsigned int tcxoDac;  // The TCXO dac value at T0
}       calibrationData_t;

typedef struct attenuator_t
{
        unsigned int if3Atten2Value;
        unsigned int if3Atten1Value;
        unsigned int rfAttenValue;
        unsigned int if1AttenValue;
}       attenuator_t;

typedef struct deviceStatus_t
{
        bool tcxoPllLock;
```

```
            bool vcxoPllLock;
            bool lo1Pll3Lock;
            bool lo2PllLock;
            bool lo3PllLock;
            bool lo1Pll1Lock;
            bool lo1Pll2Lock;
            bool sigGenPllLock;
            bool extRefDetected;
            bool refClkOutEnable;
            bool extRefLockEnable;
            bool ifBandSelect;
            bool preampEnable;
            bool standbyEnable;
            bool pxiClkEnable;
            bool sigGenEnable;
}           deviceStatus_t;

typedef struct ifResponseCorrect_t
{
            float ampCorrect;
            float phaseCorrect;
}           ifResponseCorrect_t;
```

Function Definitions and Usage

The functions listed below are found in the *sc5405a.dll* dynamic linked library These functions are also provided in the SC5405A LabVIEW palette, except in cases where an existing native function already exists to perform the same or similar task. The LabVIEW functions contain context help (Ctrl+H) to provide further clarification of each function.

Function:	**sc5405a_ListResources**	
Definition:	**int sc5405a_ListResources(char **visaResource, unsigned int *size)**	
Return:	Error code	
Output:	char **visaResources	(pointer list to device resources)
	int size	(number of devices found)
Description:	**sc5405a_ListResources** searches for SignalCore SC5405A devices connected to the host computer and returns an array containing their resource IDs. The user can use this information to open the device(s). See **sc5405a_OpenDevice** function for information on how to open a device.	

Function:	**sc5405a_OpenDevice**
Definition:	**int sc5405a_OpenDevice(char *visaResource, unsigned int *deviceHandle)**
Return:	Error code
Output:	char *visaResource (char pointer to the resource ID)
	unsigned int deviceHandle (device handle return)
Description:	**sc5405a_OpenDevice** opens the device and turns on the front panel "active" LED if successful. This function returns a handle to the device for other function calls.

Function:	**sc5405a_CloseDevice**
Definition:	**int sc5405a_CloseDevice(unsigned int *deviceHandle)**
Return:	The status of the function
Input:	unsigned int *deviceHandle (handle to the device to be closed)
Description:	**sc5405a_CloseDevice** closes the device associated with the device handle and turns off the "active" LED of the front panel if it is successful.
Example:	To exercise the functions that open and close the device:

```
// Declaring
char ** visaResource;
unsigned int      deviceHandle;
int         devicesFound;
int         i, status;
// Allocate memory for possible 100 devices with 1000 char description
char **visaResource = (char**)malloc(sizeof(char*)* MAXDEVICES);
for(i = 0; i< MAXDEVICES; i++)  visaResource[i] = (char*)malloc(sizeof(char)*1000);

status = sc5405a_ListResources(visaResource, &devicesFound);
            if (devicesFound == 0)
            {
                        for (i = 0; i< MAXDEVICES;i++) free(visaResource[i]);
                        free(visaResource);
            }

printf("There are %d SignalCore SC5405A devices found. \n", devicesFound);

for (i = 0;i<devicesFound;i++)  printf("%d.  %s \n", i + 1, visaResource[i]);

status = sc5405a_OpenDevice(visaResource[0], deviceHandle); // get  handle to the first listed device

// Free memory
for(i = 0; i< MAXDEVICES;i++) free(visaResource[i]);
free(visaResource);
//
// Do something with the device
//
//Close the device
int status = sc5405a_CloseDevice(deviceHandle);
```

Function:	**sc5405a_RegWrite**
Definition:	**int sc5405a_RegWrite(unsigned int *deviceHandle, unsigned char commandByte, unsigned int instructWord)**
Return:	The status of the function
Input:	unsigned int *deviceHandle (handle to the opened device)
	unsigned char commandByte (the address byte of the register to write to)
	unsigned int instructWord (the data for the register)

Description: **sc5405a_RegWrite** writes the instructWord data to the register specified by the commandByte. See the register map in The users may write the configuration registers (write only) directly by calling the **sc5405a_RegWrite** function. The **following table lists the register address (**command) and the effective bytes of command data. Data must be formatted into an unsigned integer of 32 bits prior to passing it to the function. As an example, to write the byte 0xEE into address 0xA8BB of the user EEPROM, the user would call the sc5405a_RegWrite as follows:

sc5405a_ RegWrite(deviceHandle, 0x23, **0x00A8BBEE)**

Table 2 for more information. This function should rarely be used.

Example: To set the RF attenuator value to 10 dB:

```
int status = sc5405a_RegWrite(deviceHandle, 0x11, 0x020A);
```

Function:	**sc5405a_RegRead**
Definition:	**int sc5405a_RegRead(unsigned int *deviceHandle, unsigned char commandByte, unsigned int instructWord, unsigned int *receivedWord)**
Return:	The status of the function
Input:	unsigned int *deviceHandle (handle to the opened device)
	unsigned char commandByte (the address byte of the register to write to)
	unsigned int instructWord (the data for the register)
	unsigned int *receivedWord (data to be received)

Description: **sc5405a_RegRead** reads the data requested by the instructWord data to the register specified by the commandByte. See the register map in Table 3 for more information.

Example: To read the status of the device:

```
unsigned int deviceStatus;

int status = sc5405a_RegRead(deviceHandle,0x18,0x01,&deviceStatus);
```

Function:	**sc5405a_InitDevice**
Definition:	**int sc5405a_InitDevice(unsigned int *deviceHandle, bool Mode)**
Return:	The status of the function

Input:	unsigned int *deviceHandle	(handle to the opened device)
	bool Mode	(0 or 1)

Description: **sc5405a_InitDevice** initializes the device to power up state if Mode = 0, or reprograms all device components again to the current programmed state if Mode = 1.

Function: **sc5405a_SetStandby**

Definition: **int sc5405a_SetStandby(unsigned int *deviceHandle, bool standbyStatus)**

Return: The status of the function

Input:	unsigned int *deviceHandle	(handle to the opened device)
	bool standbyStatus	(set to true (1) to set device in standby mode)

Description: **sc5405a_SetStandby** puts the device in standby mode where the power to the analog circuits is disabled, conserving power.

Function: **sc5405a_SetFrequency**

Definition: **int sc5405a_SetFrequency(unsigned int *deviceHandle, unsigned int frequency)**

Return: The status of the function

Input:	unsigned int *deviceHandle	(handle to the opened device)
	unsigned int frequency	(frequency in Hz)

Description: **sc5405a_SetFrequency** sets the RF frequency.

Function: **sc5405a_SetAttenuator**

Definition: **int sc5405a_SetAttenuator(unsigned int *deviceHandle, unsigned int attenValue, unsigned int attenuator)**

Return: The status of the function

Input:	unsigned int *deviceHandle	(handle to the opened device)
	unsigned int attenValue	(the value assigned to the attenuator)
	unsigned int attenuator	(the designated attenuator, see header file)

Description: **sc5405a_SetAttenuator** sets the value of the designated attenuator.

Function: **sc5405a_SetSignalChain**

Definition: **int sc5405a_SetSignalChain(unsigned int *deviceHandle, attenuator_t *atten)**

Return: The status of the function

Input:	unsigned int *deviceHandle	(handle to the opened device)
	attenuator_t atten	(values to the attenuators contained in type)

Description: **sc5405a_SetSignalChain** sets all the attenuators and the preamp state. This simplifies the programming flow when used with **sc_5405a_CalcAutoAttenuation**, which returns the attenuator_t structure.

Example: Define attenuator_t atten and use it in the function:

```
attenuator_t *atten;
bool preamp = 0;
atten = (attenuator_t*)malloc(sizeof(attenuator_t)); //casting may not be necessary
// set for rfLevel = 0 dBm, mixerLevel = -20, ifLevel = 0 dBm
atten->if3Atten2Value = 8;
atten->if3Atten1Value = 2;
atten->rfAttenValue = 20;
atten->if1AttenValue = 0;

int status = sc5405a_SetSignalChain(deviceHandle, atten);
free(atten);
```

Function: **sc5405a_SetSynthesizerMode**

Definition: **int sc5405a_SetSynthesizerMode(unsigned int *deviceHandle, bool fastTuneEnable, unsigned int fineTuneMode)**

Return: The status of the function

Input:	unsigned int *deviceHandle	(handle to the opened device)
	bool fastTuneEnable	(enable/disable faster frequency stepping)
	unsigned int fineTuneMode	(selection of 1 MHz, 25 kHz, 1 Hz step resolution)

Description: **sc5405a_SetSynthesizerMode** enables/disables fast tuning, and sets the step resolution of the upconverter.

Function: **sc5405a_SetReferenceClock**

Definition: **int sc5405a_SetReferenceClock(unsigned int *deviceHandle, bool lockExtEnable, bool RefOutEnable, bool Clk100Enable)**

Return: The status of the function

Input:	unsigned int *deviceHandle	(handle to the opened device)
	bool lockExtEnable	(enables phase locking to an external source)
	bool RefOutEnable	(enables the clock to driven out the REF OUT port)
	bool Clk100Enable	(changes REF OUT between 10MHz to 100 MHz)

Description: **sc5405a_SetReferenceClock** configures the reference clock behavior of the device**.**

Function:	**sc5405a_SetReferenceDac**	
Definition:	**int sc5405a_SetReferenceDac(unsigned int *deviceHandle, unsigned int dacValue)**	
Return:	The status of the function	
Input:	unsigned int *deviceHandle	(handle to the opened device)
	unsigned int dacValue	(16-bit value for the reference DAC)
Description:	**sc5405a_SetReferenceDac** set the value of the DAC that tunes the internal reference TXCO. The user may choose to override the value stored in memory to improve frequency accuracy.	

Function:	**sc5405a_SetIfInversion**	
Definition:	**int sc5405a_SetIfInversion(unsigned int *deviceHandle, bool ifInvertEnable)**	
Return:	The status of the function	
Input:	unsigned int *deviceHandle	(handle to the opened device)
	Bool ifInvertEnable	(enable spectral inversion)
Description:	**sc5405a_SetIfInversion** enables the upconverted signal to be spectrally inverted with respect the IF input. This may be beneficial for some applications.	

Function:	**sc5405a_SetSignalPhase**	
Definition:	**int sc5405a_SetSignalPhase(unsigned int *deviceHandle, float phase)**	
Return:	The status of the function	
Input:	unsigned int *deviceHandle	(handle to the opened device)
	float phase	(phase in degrees, 0-360 deg, 0.1 resolution)
Description:	**sc5405a_SetSignalPhase** increases the phase of the signal by the amount specified.	

Function:	**sc5405a_SetSpurMode**	
Definition:	**int sc5405a_SetSpurMode(unsigned int *deviceHandle, bool phase)**	
Return:	The status of the function	
Input:	unsigned int *deviceHandle	(handle to the opened device)
	bool phase	(modes 0 or 1)
Description:	**sc5405a_SetSpurMode** when set to 1 changes the LO frequency scheme internally to move a leakage spur at center band to an offset of 1.25 MHz. Under normal mode 0, this spur is typically < -120 dBm and by turning on mode 1 improves this in most devices.	

Function:	**sc5405a_SetSigGenEnable**	
Definition:	**int sc5405a_ SeSigGenEnable (unsigned int *deviceHandle, bool sigGenEnable)**	
Return:	The status of the function	
Input:	unsigned int *deviceHandle	(handle to the opened device)
	bool sigGenEnable	(1 to enable, default 0)
Description:	**sc5405a_ SetSigGenEnable** when set to 1 enables an internal 70 MHz signal generator to be switched into the input IF path, while disabling any external input. The upconverter is turned into a CW signal generator, not requiring any external IF sources.	

Function:	**sc5405a_WriteUserEeprom**	
Definition:	**int sc5405a_WriteUserEeprom(unsigned int *deviceHandle, unsigned int memAdd, unsigned char byteData)**	
Return:	The status of the function	
Input:	unsigned int *deviceHandle	(handle to the opened device)
	unsigned int memAdd	(memory address to write to)
	unsigned char byteData	(byte to be written to the address)
Description:	**sc5405a_WriteUserEeprom** writes one byte of data to the memory address specified.	

Function:	**sc5405a_GetDeviceStatus**	
Definition:	**int sc5405a_GetDeviceStatus(unsigned int *deviceHandle, deviceStatus_t *deviceStatus)**	
Return:	The status of the function	
Input:	unsigned int *deviceHandle	(handle to the opened device)
Output:	deviceStatus *deviceStatus	(outputs the status of the device such as PLL lock)
Description:	**sc5405a_GetDeviceStatus** retrieves the status of the device such as LO phase-lock status, and current device settings.	
Example:	Code showing how to use this function:	

```
deviceStatus_t *devStatus;
devStatus = (deviceStatus_t*)malloc(sizeof(deviceStatus_t));

int status = sc5405a_GetDeviceStatus(deviceHandle, devStatus);

if(devStatus->vcxoPllLock)
printf("The 100 MHz is phase-locked \n");
else
printf("The 100 MHz is not phase-locked \n");

free(deviceStatus);
```

Function:	**sc5405a_GetTemperature**	
Definition:	**int sc5405a_GetTemperature(unsigned int *deviceHandle, float *temperature)**	
Return:	The status of the function	
Input:	unsigned int *deviceHandle	(handle to the opened device)
Output:	float *temperature	(temperature in degrees C)
Description:	**sc5405a_GetTemperature** retrieves the internal temperature of the device.	

Function:	**sc5405a_ReadCalEeprom**	
Definition:	**int sc5405a_ReadCalEeprom(unsigned int *deviceHandle, unsigned int memAdd, unsigned char *byteData)**	
Return:	The status of the function	
Input:	unsigned int *deviceHandle	(handle to the opened device)
	unsigned int memAdd	(EEPROM memory address)
Output:	unsigned char *byteData	(the read byte data)
Description:	**sc5405a_ReadCalEeprom** reads back a byte from the memory address of the calibration EEPROM.	

Function:	**sc5405a_ReadUserEeprom**	
Definition:	**int sc5405a_ReadUserEeprom(unsigned int *deviceHandle, unsigned int memAdd, unsigned char *byteData)**	
Return:	The status of the function	
Input:	unsigned int *deviceHandle	(handle to the opened device)
	unsigned int memAdd	(EEPROM memory address)
Output:	unsigned char *byteData	(the read byte data)
Description:	**sc5405a_ReadUserEeprom** reads back a byte from the memory address of the user EEPROM.	

Function:	**sc5405a_ReadUserEepromBulk**	
Definition:	**int sc5405a_ReadUserEepromBulk(unsigned int *deviceHandle, unsigned int startMemAdd, unsigned char *byteDataArray)**	
Return:	The status of the function	
Input:	unsigned int *deviceHandle	(handle to the opened device)
	unsigned int memAdd	(EEPROM start memory address)
Output:	unsigned char *byteData	(the read 64 bytes of data)
Description:	**sc5405a_ReadUserEepromBulk** reads back 64 bytes beginning at the start memory address of the user EEPROM.	

Example: Code to read back 512 bytes of data starting at address 1024 into eepromData:

```
unsigned char eepromData = (unsigned char*)malloc(512);
unsigned char *bufferIn = (unsigned char*)malloc(64);
int i = 0;
int bufferCount = 0;
unsigned int add = 1024;

while(bufferCount < 8){
        int status = sc5405a_ReadUserEepromBulk(deviceHandle, add+bufferCount*64, bufferIn);
        for (i = 0; i < 64; i++) eepromData[i + bufferCount*64] = bufferIn[i];
        bufferCount ++;
}

free(bufferIn);
```

Function: **sc5405a_GetRawCalData**

Definition: **int sc5405a_GetRawCalData(unsigned int *deviceHandle, unsigned char *rawCalDataArray)**

Return: The status of the function

Input: unsigned int *deviceHandle (handle to the opened device)

Output: unsigned char *rawCalDataArray (the entire calibration EEPROM contents)

Description: **sc5405a_GetRawCalData** reads the entire calibration EEPROM into rawCalDataArray. The array must be allocated for at least 15168 bytes.

Example: See code block example for **sc5405a_ConvertRawCalData**.

Function: **sc5405a_GetCalData**

Definition: **int sc5405a_GetCalData(unsigned int *deviceHandle, deviceAttribute_t *deviceAttributes, calibrationData_t *calData)**

Return: The status of the function

Input: unsigned int *deviceHandle (handle to the opened device)

Output: deviceAttribute_t *deviceAttributes (device attributes)

calibrationData_t *calData (structured calibration data)

Description: **sc5405a_GetCalData** returns the device attributes such as serial number, calibration date, and also structured calibration data used for gain calculation and correction.

Example: See code block example for **sc5405a_ConvertRawCalData**.

Function:	**sc5405a_ConvertRawCalData**	
Definition:	**int sc5405a_ConvertRawCalData(unsigned char *rawCalData, deviceAttribute_t *deviceAttributes, calibrationData_t *calData)**	
Input:	unsigned char *rawCalData	(entire cal EEPROM data in raw byte format)
Output:	calibrationData_t *calibrationData	(current calibration data)
	deviceAttribute_t *deviceAttributes	(device attributes)

Description: **sc5405a_ConvertRawCalData** organizes/decodes the entire 15168 bytes of raw calibration data read from the EEPROM and returns two data formats - deviceAttribute_t and calibrationData_t. The array rawCalData must contain valid calibration data, obtained by reading the EEPROM, and rawCalData, calData and deviceAttributes must have memory allocated.

Example: Allocating memory for the input and output parameters in C and calling the function. Similarly, allocated memory must be de-allocated when no longer used or when the program quits.

```
//Declaring
calibrationData_t* calData;
deviceAttribute_t* devAttr;
unsigned char *rawCal;

//allocate memory for raw calibration
rawCal = (unsigned char*)malloc(sizeof(char)*CALEEPROMSIZE);

// Allocate memory; the user may use malloc() instead of calloc()
devAttr->calDate = (unsigned int*)calloc(4,sizeof(unsigned int));
devAttr->manDate = (unsigned int*)calloc(4,sizeof(unsigned int));

calData->rfCal = (float**)calloc(RFCALPARAM,sizeof(float*));
for(i = 0;i<RFCALPARAM;i++)
        calData->rfCal[i] = (float*)calloc(RFCALFREQ,sizeof(float));

calData->ifAttenCal = (float**)calloc(IFATTENUATOR,sizeof(float*));
for(i = 0;i<IFATTENUATOR;i++)
        calData->ifAttenCal[i] = (float*)calloc(IFATTENCALVALUE,sizeof(float));

calData->ifFil0ResponseCal = (float**)calloc(IFRESPONSEPARAM,sizeof(float*));
for(i = 0;i<IFRESPONSEPARAM;i++)
        calData->ifFil0ResponseCal[i] = (float*)calloc(IFRESPONSEFREQ,sizeof(float));

calData->ifFil1ResponseCal = (float**)calloc(IFRESPONSEPARAM,sizeof(float*));
for(i = 0;i<IFRESPONSEPARAM;i++)
        calData->ifFil1ResponseCal[i] = (float*)calloc(IFRESPONSEFREQ,sizeof(float));

calData->tempCoeff = (float**)calloc(TEMPCOPARAM,sizeof(float*));
for(i = 0;i<TEMPCOPARAM;i++)
        calData->tempCoeff[i] = (float*)calloc(TEMPCOFREQ,sizeof(float));

//read in raw calibration
int status = sc5405a_GetRawCal(deviceHandle, rawCal);
//Calling the function to structure the calibration data
status = sc5405a_convertRawCalData(rawCal, devAttr, calData);

//alternatively instead of calling the above 2 functions
status = sc5405a_GetCalData(deviceHandle, devAttr, calData);
```

Function:	**sc5405a_CalcAutoAttenuation**	
Definition:	**int sc5405a_CalcAutoAttenuation(unsigned int frequency, float inputRfLevel, float inputMixerLevel, bool preampEnable, float nominalIfOutLevel, bool if3Filter1Enable, float temperature, calibrationData_t *calData, attenuator_t *attenuator)**	
Input:	unsigned int frequency	(input RF frequency in Hz)
	float inputRfLevel	(input RF level in dB)
	float inputMixerLevel	(input mixer level in dB)
	bool preampEnable	(preamplifier enabled)
	float nominalIfOutLevel	(nominal IF out level in dB)
	bool if3Filter1Enable	(Enable Filter 1 path in IF3)
	float temperature	(current device temperature)
	calibrationData_t * calData	(structured calibration data for the device)
Output:	attenuator_t *attenuator	(attenuation settings for RF, IF1, and final IF3 attenuators)

Description: **sc5405a_CalcAutoAttenuation** returns the set of attenuation settings for all the attenuators that will configure the SC5405A for best dynamic range operation based on user input parameters such as frequency, mixer level, etc. The values are calculated to maintain a good balance between the signal-to-noise dynamic range and the linearity dynamic range. The SC5405A upconverter is designed for best balanced dynamic range with -20 dBm power at the input mixer and 0 dBm nominal power at the RF output port. Each attenuator must have memory allocated before calling this function. **sc5405a_ConvertRawCalData** must be called or valid structure calibration data must be entered before calling this function.

Example: Code showing how to properly use this function:

```
//Declaring
Attenuator_t *atten;
float deviceTemp;

//call a function to return the device temperature
function_to_sc5405a_GetTemperature( &deviceTemp);

unsigned int rfFreq = 1000000000;  // 1.0 GHz
float rfLevel = 0;  // expecting a 0 dBm input signal
float mixerLevel = -20;  //set the mixer level requirement
bool preamp = 0;          // since input level is 0 dB, no need for a preamp
float ifLevel = 0; // to obtain a level clost to 0 dBm at the IF
bool filterPath = 0; // use the default filter path in the IF
float temp = deviceTemp;

//Calling the function
int status = sc5405a_CalcAutoAttenuation(rfFreq, rfLevel,mixerLevel,preamp,ifLevel,filterPath,temp,
atten);
```

Function: **sc5405a_CalcGain**

Definition: **int sc5405a_CalcGain(unsigned int frequency, bool preampEnable, bool ifInvertEnable, bool if3Fil1Enable, attenuator_t *atten, float temperature, calibrationData_t *calData, float *conversionGain)**

Input:	unsigned int frequency	(input RF frequency in Hz)
	bool preampEnable	(preamplifier enabled)
	bool ifInvertEnable	(enable IF spectral inversion)
	attenuator_t *atten	(attenuation settings)
	float temperature	(temperature value of the device in degrees Celsius)
	calibrationData_t *calData	(calibration data for the device)
Output:	float *conversionGain	(calculated calibrate conversion gain for current settings)

Description: **sc5405a_CalcGain** calculates the calibrated gain based on the current user settings.

Function:	**sc5405a_CalcIfResponseCorrection**
Definition:	**int sc5405a_CalcIfResponseCorrection(float *offsetFrequencies, unsigned int nPoints, bool ifFil1PathEnable, calibrationData_t *calData, ifResponseCorrect_t *correctedIfResponse)**
Input:	float *offsetFrequencies (floating point number 1-D array)
	unsigned int nPoints (number of points in the 1-D array)
	bool ifFil1PathEnable (IF filter 1 path enable)
	calibrationData_t *calibrationData (calibration data for the device)
Output:	float *correctedIfResponse
Description:	**sc5405a_CalculateIfResponseCorrection** determines the IF correct response for the set of IF offset frequencies. These offset frequencies may be the frequency components of an FFT of the acquired data being offset from its center frequency. To obtain the offset frequencies, one can simply subtract the frequencies from the IF center frequency. For example, if a digitizer sampling at 100 MHz is used to digitize the 70 MHz IF signal with bandwidth of 3 MHz, the center of the digitized signal is 30 MHz +/- 1.5 MHz. After performing digital spectral inversion and performing an FFT, take the subset of frequency components from 28.5 MHz to 31.5 MHz and subtract 30 MHz to obtain the offset frequencies of -1.5 MHz to 1.5 MHz. Use this set of offset frequencies to compute the gain and phase corrections to be applied to the original signal spectrum at 28.5 MHz to 31.5 MHz. This algorithm may not be sufficient for computation of broadband signals due to the lack of computation speed and correction accuracy. The calibration stored does not account for in-band phase and amplitude variations due to temperature and these variations may cause sufficient errors, especially in broadband digital signals. The user should apply in situ equalization to correct for the in-band amplitude and phase errors.

Function:	**sc5405a_ConvertRawTempData**
Definition:	**int sc5405a_ConvertRawTempData(unsigned int rawTempData, float *temperature)**
Input:	unsigned int rawTempData (16-bit rawTempData stored in a 32-bit unsigned int)
Output:	float *temperature (temperature value of the device in degrees Celsius)
Description:	**sc5405a_ConvertRawTempData** converts the rawTempData variable into a floating point number.

Function:	**sc5405a_Spline**
Definition:	**int sc5405a_Spline(double *xArray, double *yArray, int nPoints, double firstBoundary, double secondBoundary, double *yInterpolant)**
Input:	double *xArray (the set of independent values)
	double *yArray (the set of x-dependent function values; size is the same as xArray)
	int nPoints (the number of points in xArray)
	double firstBoundary (the second derivative of the first point in the set)
	double secondBoundary (the second derivative of the last point in the set)
Output:	double *yInterpolant (the return set of interpolants)
Description:	**sc5405a_Spline** returns the spline interpolants of the input parameters.

Function:	**sc5405a_SplineInterp**
Definition:	**int sc5405a_SplineInterp(double *xArray, double *yArray, double *yInterpolant, double nPoints, double x, double *interpolatedYValue)**
Input:	double *xArray (the set of independent values)
	double *yArray (the set of x-dependent function values; size is the same as xArray)
	double *yInterpolant (the returned interpolant from spline())
	int nPoints (the numbers of points in xArray)
	double x (the value at which interpolation is performed)
Output:	double *interpolatedYValue (the corresponding interpolated value at x)
Description:	**sc5405a_SplineInterp** returns the spline interpolated value.

CALIBRATION & MAINTENANCE

The SC5405A is factory calibrated and ships with a certificate of calibration. SignalCore strongly recommends that the SC5405A be returned for factory calibration every 12 months or whenever a problem is suspected. The specific calibration interval is left to the end user and is dependent upon the accuracy required for a particular application.

SC5405A calibration data is stored in the RF module (metal housing). Therefore, changing or replacing interface adapters will not affect unit calibration. However, SignalCore maintains a calibration data archive of all units shipped. Archiving this data is important should a customer need to reload calibration data into their device for any reason. SignalCore also uses the archived data for comparative analysis when units are returned for calibration.

Should any customer need to reload calibration data for their SC5405A, SignalCore offers free support through support@signalcore.com. SignalCore will provide a copy of the archived calibration data along with instructions on how to upload the file to the SC5405A.

The SC5405A requires no scheduled preventative maintenance other than maintaining clean, reliable connections to the device as mentioned in the "*Getting Started*" section of this manual. There are no serviceable parts or hardware adjustments that can be made by the end user.

Revision Notes

Rev 1.0.0	Initial release.
Rev 1.1.0	Revised images and connectivity references to reflect new I/O interface, updated EN and IEC testing standards and EU Directive references.
Rev 2.0.0	Removed Appendix A (specifications are now available in separate datasheet document).
Rev 2.1.0	Updated a few grammatical errors; added new front interface drawing; updated title page picture.
Rev 2.1.1	Corrected torque range