SignalCore™
PRESERVING SIGNAL INTEGRITY

SC5505A

25 MHz to 6 GHz RF Signal Source

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

SignalCore, Incorporated respects the intellectual property rights of others, and we ask those who use our products to do the same. Our products are protected by copyright and other intellectual property laws. Use of SignalCore products is restricted to applications that do not infringe on the intellectual property rights of others.

International Materials Declarations

SignalCore, Incorporated uses a fully RoHS compliant manufacturing process for our products. Therefore, SignalCore hereby declares that its products do not contain restricted materials as defined by European Union directive 2002/95/EC (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)**
SC5505A	✓	✓	✓	✓	✓	✓

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 2004/108/EC (EMC) and 2006/95/EC (product safety) and comply with the relevant standards. Standards for Measurement, Control and Laboratory Equipment include EN 61326 and EN 55011 for EMC, and EN 61010-1 for product safety.

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 SC5505A kit contains the following items:

Quantity	Item
1	SC5505A Dual Channel RF Signal Source
1	USB Flash Drive Installation Software (may be combined with other products onto a single drive)
1	Getting Started Guide

Setting Up and Configuring the SC5505A

The SC5505A is a designed for use in a 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 SC5505A is designed to be sufficiently cooled in either all-PXIe chassis or PXIe hybrid chassis (PXI Express chassis with 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 SC5505A on-board temperature sensor should indicate a rise of no more than 20 °C above ambient temperature under normal operating conditions.

The SC5505A is a PXIe-based RF signal source with all I/O connections and indicators located on the front face of the module as shown in Figure 1. Each location is discussed in further detail below.

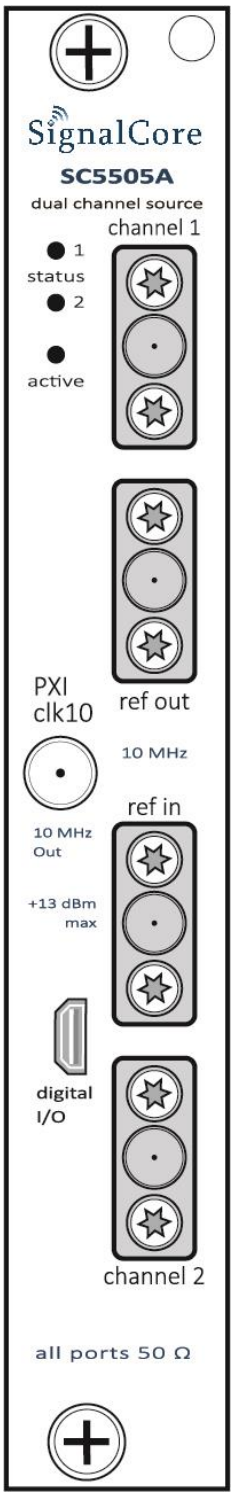

Figure 1. PXIe chassis view of the SC5505A. Module is shown installed in slot 2.

Signal Connections

All signal connections (ports) on the SC5505A are SMA-type. 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 degrade measurement performance until fully evaporated (this may take several days).

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

RF OUT CHANNEL 1	This port outputs the tunable RF signal from channel 1 of the source. The connector is SMA female. The nominal output impedance is 50 Ω.
RF OUT CHANNEL 2	This port outputs the tunable RF signal from channel 2 of the source. The connector is SMA female. The nominal output impedance is 50 Ω.
REF OUT	This port outputs the internal 10 MHz reference clock. The connector is SMA female. This port is AC-coupled with a nominal output impedance of 50 Ω.
REF IN	This port accepts an external 10 MHz reference signal, allowing an external source to synchronize the internal reference clock. The connector is SMA female. This port is AC-coupled with a nominal input impedance of 50 Ω. Maximum input power is +13 dBm.
PXI CLK10	This port outputs the 10 MHz chassis reference signal from the chassis backplane, allowing it to synchronize the internal reference clock. A male-to-male SMA cable is required (nor 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.
DIGITAL I/O	This port is not currently supported for this product.

Indicator LEDs

The SC5505A provides visual indication of important modes. There are two LED indicators on the unit. 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. Contact SignalCore.
ACTIVE	Green/Off	Device is open (green) /closed (off). This indicator is also user programmable (see register map).

Output Amplitude Control

As shown in Figure 2, the SC5505A source architecture at a high level consists of an output amplitude control section and a frequency synthesis section. The amplitude of the signal is controlled through the use of digital step attenuators (DSAs) and a voltage controlled attenuator (VCA). The DSAs provide the coarse-step tuning over a wide range while the VCA provides fine tune correction to the DSAs. The VCA is part of the automatic level control loop (ALC), which additionally consists of an RF amplifier, a power detector, and an integrator. The ALC loop can be closed or open. In the closed loop mode, the power detector outputs a voltage proportional to the power it detects. This voltage is compared to that of the reference ALC DAC voltage, which in turn is set for some calibrated power level. Voltage error between the detector voltage and the ALC DAC voltage drives the integrator output in the direction that will vary the VCA to achieve the desired output power level. When the ALC control loop is opened, the power detector output voltage is grounded, and the integrator is configured as a voltage buffer that drives the ALC DAC voltage to the VCA. In this mode, the ALC DAC voltage directly drives the VCA with voltage levels that correspond to calibrated output power levels.

There are advantages and disadvantages to either of these two amplitude control modes. On one hand, the open loop mode has an advantage over the closed loop mode when close-in carrier amplitude noise is a concern. ALC loops do introduce some level of amplitude modulated noise onto the carrier signal, and these levels may not be acceptable (although they are generally lower than the phase noise). SignalCore offers the option to open the ALC loop to remove any unwanted AM noise that results from closed loop control. Another side effect of the closed loop is that the frequency bandwidth of the ALC loop may slow down amplitude settling. Typically, in order to keep AM noise low and close (in offset frequency) to the signal, the loop bandwidth is also kept low. As a result, the settling time is increased.

On the other hand, a closed loop ALC provides better amplitude control over the entire frequency range. With a temperature-stable ALC DAC, the closed loop will precisely maintain the power at the detector, mitigating errors in the components prior to it in the signal path. Temperature-induced errors in components and abrupt amplitude changes when switching filters in the filter banks contribute to errors in the amplitude of the signal. However, these errors occur before the power detector and are compensated by the feedback loop action. Errors in amplitude are thus confined to the output attenuators, amplifiers, and the loop components. When the loop is opened, amplitude errors resulting from all parts of the amplitude control section as well as the synthesizer section will affect the overall output amplitude accuracy. In particular, when the filters within the filter bank are switched from one bank to another, the signal experiences abrupt discontinuities in its amplitude which the open loop calibration cannot appropriately account for in its correction algorithm.

Setting of the amplitude control components is performed automatically by the system, although it is possible to override the ALC DAC value if needed. In Figure 2, the labels in red indicate parameters or devices which the user can directly control.

Frequency Synthesizer

The synthesizer section of the SC5505A comprises a multiple phase-locked loop architecture whose base frequency reference is a 10 MHz TCXO. The user may choose to phase-lock this base reference to an external source if required. The 100 MHz VCXO is phase-locked to the TCXO for frequency stability. While the TCXO determines the very close-in phase noise, the VCXO phase noise determines the system phase noise in the frequency offset range of approximately 1 kHz to 30 kHz. The 100 MHz VCXO provides the reference signal to the main RF signal synthesizer, which is comprised of three phase-locked loops (PLLs) and a direct digital synthesis (DDS) oscillator. The “fine” PLL provides a tuning resolution of a few mHz over a narrow frequency range, while the “coarse” PLL tunes in steps of a few MHz over several GHz of range. The “main” or summing PLL combines the signals of the “coarse” and “fine” loops into one broad tuning signal with fine tuning capability.

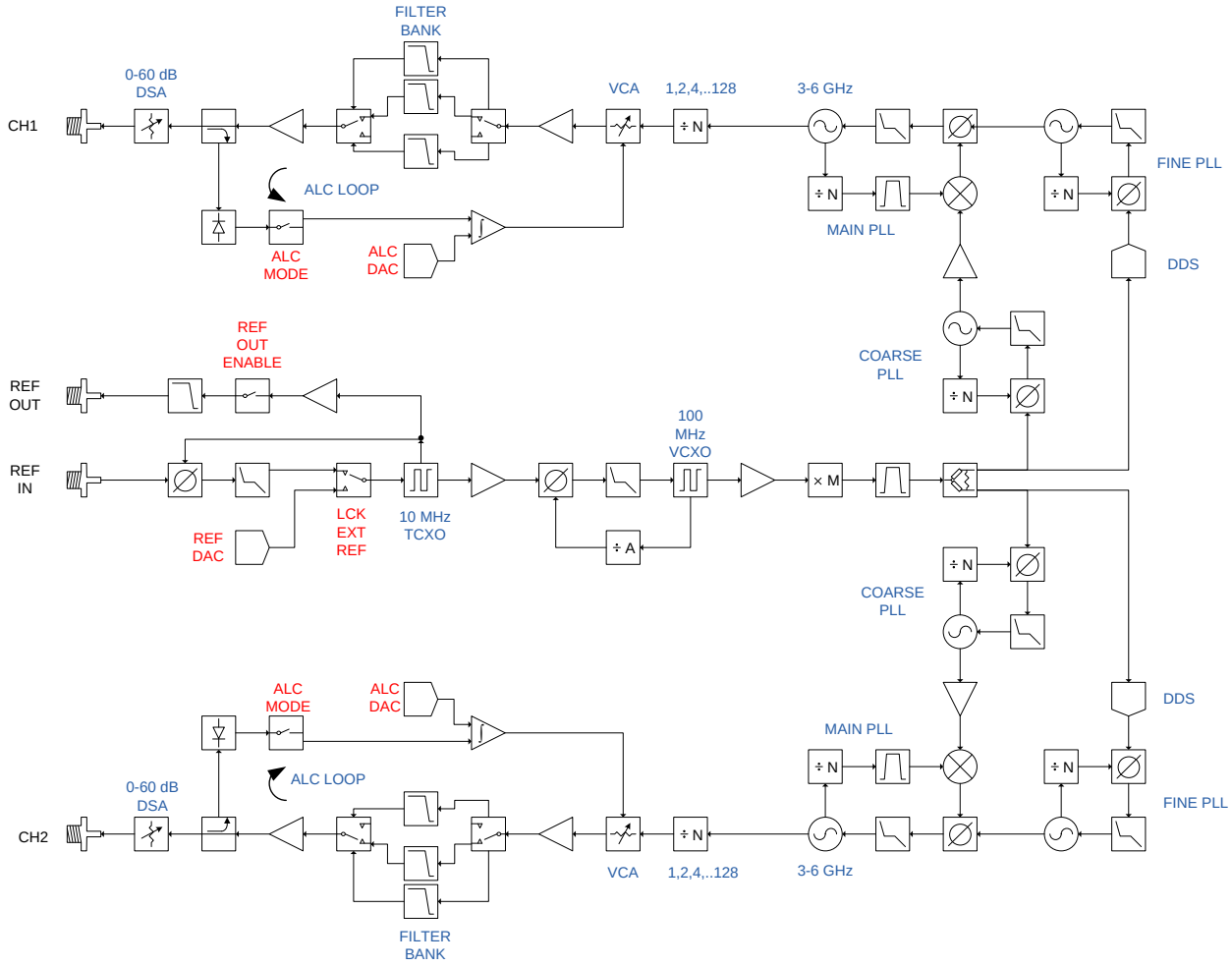

Figure 2. Simplified block diagram of the SC5505A dual channel RF signal source.

Using this multiple loop approach produces signals with low phase noise and low phase spurs, high levels of which exist in single loop architectures such as single fractional-N PLLs. Although a single fractional-N type PLL may provide fine resolution, its large fractional spurs may make it unusable at certain parts of the band - especially at frequency regions close to the integer boundaries. A multiple loop architecture allows for fine tuning with extremely low phase spurs.

Reference Clock Control

As mentioned above, the primary clock reference for the SC5505A is an onboard 10 MHz TCXO. Should the user require better frequency stability and/or accuracy, this TCXO can be programmed to phase-lock to an external source such as an OCXO or rubidium clock. The device can also be programmed to export its internal 10 MHz clock. To adjust the accuracy of the TCXO as needed (e.g., to correct for long-term accuracy drift), the user may vary the reference DAC voltage by writing the **REFERENCE_DAC** register.

Additionally, the SC5505A routes the 10 MHz PXI chassis backplane clock to the front panel of the device. By cabling between the “PXI clk10” port and the “ref in” port, the user may enable the 10 MHz PXI chassis clock as a phase-locked reference for the SC5505A. A word of caution about the PXI chassis clock - the 10 MHz backplane clock is not considered to be a either a stable or accurate clock and could potentially have frequency errors in excess of +/- 40 ppm, significantly outside the locking range capability of the SC5505A.

Harmonics and Range Operation

The SC5505A’s guaranteed operating frequency range is 25 MHz to 6.0 GHz. At low frequencies of operation (200 MHz and lower), the harmonics of the signal can potentially be observed as high as -10 dBc at 0 dBm output. This is due to the limited space available for additional filtering in these ranges. At lower frequencies, the large physical size of appropriate filters makes it impossible to accommodate them within the compact form of this device. Furthermore, as the low frequencies are synthesized through frequency dividers, their output waveforms become more “square” than sinusoidal, giving rise to higher odd-order harmonics.

The device is specified to a maximum calibrated level of +10 dBm, although the maximum calibrated output is greater than that in most regions of the spectrum. The accuracy degrades as the amplitude approaches the compression point due to the linear approximation in the correction algorithm. As a general rule, however, the lower the tuned frequency, the higher the achievable output power.

Channel Standby and RF Enable

The SC5505A has independent standby and output enable features for both channels. The user may wish to place one or both of the channels into standby mode to reduce power consumption and thus lower the operating temperature of the device under the same environmental conditions. Putting one channel into standby also eliminates its signal generation, and as a result the remaining operational channel does not receive any cross-signal contamination, especially when generating low level signals. Although the device has the option to independently disable the RF output of one channel or the other, this feature does not shut down the internal synthesizer, so there still exists the possibility of contaminating signals. They are usually very low but might not be tolerated in some applications. Taking either channel out of standby requires the device to wait for the power rails to settle and all internal components to be reprogrammed, usually on the order of one second.

Disabling the RF output moves the frequency to some very low value so that the step attenuators and the voltage controlled attenuator have the most effective attenuation. Combined, this will push the

signal level below -100 dBm. Enabling the RF output is nearly instantaneous as all components remain active even when the RF output is disabled.

Default Startup Mode

The factory power-up state for the device is detailed in Table 2. The default state can be changed to the current state of either channel programmatically, allowing the user to power up the device in the last saved state without having to reprogram.

Table 2. Factory default power-up state.

	CH1	CH2
Frequency	2.0 GHz	2.4 GHz
Power	0.00 dBm	0.00 dBm
RF Output	Enabled	Enabled
ALC Mode	Closed Loop	Closed Loop
Standby	Disabled	Disabled
Auto Level	Enabled	Enabled
Ref Out	Disabled	
Ext Ref Lock	Disabled	

SC5505A Programming Interface

Device Drivers

The SC5505A is programmed by writing to its set of configuration registers, and its status is read back through its set of query registers. The user may choose to program directly at 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 of 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 **sc5505a_RegWrite** and **sc5505a_RegRead** functions respectively.

For Microsoft Windows™ operating systems, The SC5505A API is provided as a dynamic linked library, *sc5505a.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, *sc5505a.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 SC5505A 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., and the additional layer between the PXIe bridge and the onboard microcontroller that must be configured prior to writing the device registers. Using the API, commands to control the device are greatly simplified. For example, to obtain the device temperature, the user simply calls the function **sc5505A_GetDeviceTemperature** or **sc5505A_SetFrequency** to tune the frequency. The software API is covered in detail in the “Software API Library Functions” section.

SETTING THE SC5505A: WRITING TO CONFIGURATION REGISTERS

Configuration Registers

Users may write the configuration registers (write only) directly by calling the **sc5505a_RegWrite** function. The syntax for this function is **sc5505a_RegWrite(deviceHandle, registerCommand, instructWord)**. The **instructWord** takes a 64 bit-word. However, it will only send the required number of bytes to the device. Table 3 summarizes the register addresses (commands) and the effective bytes of command data.

Table 3. Configuration registers.

Register Name	Register Address	Serial Range	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INITIALIZE	0x01	[7:0]	Open	Open	Open	Open	Open	Open	Open	Mode
SET_SYSTEM_ACTIVE	0x02	[7:0]	Open	Open	Open	Open	Open	Open	Open	Enable "active" LED
RF_FREQUENCY	0x10	[7:0]	Frequency Word (MHz) [7:0]							
		[15:8]	Frequency Word (MHz) [15:8]							
		[23:16]	Frequency Word (MHz) [23:16]							
		[31:24]	Frequency Word (MHz) [31:24]							
		[39:32]	Frequency Word (MHz) [39:32]							
		[47:40]	Open	Open	Open	Open	Open	Open	Open	Channel
RF_POWER	0x11	[7:0]	RF Power Word [7:0]							
		[15:8]	Sign Bit	RF Power Word [14:8]						
		[23:9]								Channel
RF_ENABLE	0x12	[7:0]	Open	Open	Open	Open	Open	Open	Channel	Mode
AUTO_PWR_DISABLE	0x13	[7:0]	Open	Open	Open	Open	Open	Open	Channel	Mode
RF_ALC_MODE	0x14	[7:0]	Open	Open	Open	Open	Open	Open	Channel	Mode
POWER_STANDBY	0x15	[7:0]	Open	Open	Open	Open	Open	Open	Channel	Mode
REFERENCE_SETTING	0x16	[7:0]	Open	Open	Open	Open	Open	PXI Clk 10 Enable	Ref Out Enable	Lock Enable
USER_EEPROM_WRITE	0x1B	[7:0]	Data [7:0]							
		[15:8]	EEPROM Address [7:0]							
		[23:16]	EEPROM Address [15:8]							
REFERENCE_DAC	0x1D	[7:0]	DAC word [7:0]							
		[15:8]	Open	Open	DAC word [13:8]					
START_UP_STATE	0x23	[7:0]	Open	Open	Open	Open	Open	Open	Open	Open
ALC_DAC_VALUE_SET	0x24	[7:0]	ALC DAC Word [7:0]							
		[15:8]	ALC DAC Word [13:8]							
		[23:16]	Open	Open	Open	Open	Open	Open	Open	Channel

Initializing the Device

INITIALIZE (0x01) - Writing 0x00 to this register will reset the device to the default power-on state. Writing 0x01 will reset the device but leave it in the current state. The user has the ability to define the default startup state by writing to the **START UP STATE (0x23)** register, described later in this section.

Setting the System Active LED

SET_SYSTEM_ACTIVE (0x02) - This register turns on the front panel "active" LED with a write of 0x01 or turns off the LED with a write of 0x00. This register is generally written when the device driver opens or closes the device.

Setting the RF Frequency

RF_FREQUENCY (0x10) - This register sets the RF frequency for each channel. Data is sent as a 40 bit word with the LSB in Hz.

Setting the RF Power

RF_POWER (0x11) - This register sets the RF power level for each channel. The LSB is 1/100th of a dB and absolute magnitude is carried in the first 15 bits, starting with bit 0. The sign bit is indicated on bit 15. Setting bit 15 high implies a negative magnitude. For example, to write 10.05 dBm to the register, the data is simply 1005 (0x03ED). For -10.05 dBm, the data is 33773 (0x83ED).

Setting RF Output Enable

RF_ENABLE (0x12) - This register enables or disables the RF signal output for each channel. Setting bit 0 low (0) disables RF output. Setting bit 0 high (1) enables RF output.

Disabling the Auto Power Feature

AUTO_PWR_DISABLE (0x13) – When changing frequency on either channel, the device will calculate new settings for the amplitude control components such that the amplitude remains the same as the last setting. If the amplitude is also changed at the new frequency setting, the user has the option to turn off this auto power adjustment. By default, auto power adjustment is enabled. To disable auto power adjustment, set bit 0 of this register high (1).

Setting the RF Automatic Level Control (ALC) Mode

RF_ALC_MODE (0x14) – For each channel independently, writing 0x00 to this register puts the ALC in a closed loop operation. Writing 0x01 will run the ALC in an open loop. See the "Output Amplitude Control" section to understand the differences between the modes.

Setting the Device Standby Mode

POWER_STANDBY (0x15) – For each channel, writing 0x01 to this register will power-down the analog/RF circuitry. Writing 0x00 to this register will enable the analog/RF circuitry and the channel will return to its last programmed state.

Setting the Reference Clock

REFERENCE_SETTING (0x16) - This register sets the behavior of the reference clock section. Bit 2 enables (1) or disables (0) the PXI 10 MHz clock, Bit 1 enables (1) or disables (0) the output reference signal, and Bit 0 enables (1) or disables (0) the device to phase-lock to an external source. It is important that if the device is not intended to lock externally, the external source connection should be removed from the "ref in" connector. Even with external locking disabled, the presence of a large signal from the external source on the reference input terminal could potentially modulate the internal references, causing a spur offset in the RF signal.

Writing to the User EEPROM

USER_EEPROM_WRITE (0x1B) - There is an onboard 32 kilobyte EEPROM for the user to store data. User data is sent one byte at a time and is contained in the last (least significant) of the three bytes of data written to the register. The other two bytes contain the write address in the EEPROM. For example, to write user data 0x22 into address 0x1F00 requires writing 0x1F0022 to this register.

Setting the Reference DAC Value

REFERENCE_DAC (0x1D) - The frequency precision of the device's 10 MHz TCXO is set by the device internally and the factory calibrated unsigned 14 bit value is written to the reference DAC on power-up from the EEPROM. The user may choose to write a different value to the reference DAC by accessing this register, for example, to correct for long-term accuracy drift.

Storing the Startup State

START_UP_STATE (0x23) – Writing to this register will save the current device state as the new default power on (startup) state. All data written to this register will be ignored as only the write command is needed to initiate the save.

Setting the RF ALC DAC Value

ALC_DAC_VALUE_SET (0x24) - Writing a 14 bit control word to the ALC DAC register adjusts output amplitude. This is useful when the user wants to make small adjustments to the power level.

QUERYING THE SC5505A: WRITING TO REQUEST REGISTERS

The registers to read data back from the device (such as device status) are accessed through the **sc5505a_RegRead** function. The function and parameter format for this command is **sc5505a_RegRead(deviceHandle, registerCommand, instructWord,*dataOut)**. Any instructions in addition to the register call are placed into "instructWord", and data obtained from the device is returned via the pointer value dataOut. The set of request registers are shown in Table 4.

Table 4. Query registers.

Register Name	Register Address	Serial Range	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
GET_TEMPERATURE	0x17	[7:0]	Open	Open	Open	Open	Open	Open	Open	Open
GET_DEVICE_STATUS	0x18	[7:0]	Open	Open	Open	Open	Open	Open	Open	Open
USER EEPROM_READ	0x1A	[7:0]	EEPROM Address [7:0]							
		[15:8]	EEPROM Address [15:8]							
CAL EEPROM_READ	0x21	[7:0]	EEPROM Address [7:0]							
		[15:8]	EEPROM Address [15:8]							
ALC_DAC_VALUE_GET	0x39	[7:0]	Open	Open	Open	Open	Open	Open	Open	Channel

Reading the Device Temperature

GET_TEMPERATURE (0x17) - Data returned by this register needs to be processed to correctly represent 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 positive (0x0) or negative (0x1). 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 (bitwise AND'ed) 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 (bit 13 is 0) = ADC code / 32

Negative Temperature (bit 13 is 1) = (ADC code – 8192) / 32

It is not recommended to read the temperature too frequently, especially once the SC5505A has stabilized in temperature. 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 could potentially modulate the internal oscillators. Furthermore, once the SC5505A 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 less than -0.01 dB/°C, gain changes between readings will be negligible.

Reading the Device Status

GET_DEVICE_STATUS (0x18) - This register, summarized in Table 5, returns the device status information such as phase lock status of the PLL, current reference settings, etc. Data is contained in the first three bytes.

Table 5. Description of the status data bits.

Bit	Description
[21]	Channel 1 ALC open
[20]	Channel 2 ALC open
[19]	Channel 1 RF output enable
[18]	Channel 2 RF output enable
[17]	Channel 1 standby enable
[16]	Channel 2 standby enable
[15]	Channel 1 sum PLL status
[14]	Channel 1 coarse PLL status
[13]	Channel 1 fine PLL status
[12]	Reserved
[11]	Channel 2 sum PLL status
[10]	Channel 2 coarse PLL status
[9]	Channel 2 fine PLL status
[8]	Reserved
[7]	VCXO PLL status
[6]	TCXO PLL status
[5]	External reference detected
[4]	Lock to external reference enable
[3]	Reference clock output enable
[2]	Device accessed
[1]	PXI clock out enable

Reading the User EEPROM

USER_EEPROM_READ (0x1A) - Once data has been written to the user EEPROM, it can be retrieved by calling this register and using the process outlined next for reading calibration data. The maximum address for this EEPROM is 0x7FFF.

Reading the Calibration EEPROM

CAL_EEPROM_READ (0x21) - Reading a single byte from an address in the device EEPROM is performed by writing this register with the address for the instructWord. The data is returned as a byte. The CAL EEPROM maximum address is also 0x7FFF. Reading above this address will cause the device to retrieve data from the lower addresses. For example, addressing 0x8000 will return data stored in address location 0x0000. The calibration EEPROM map is shown in Table 6.

EEPROM ADDRESS (HEX)	NUMBER OF DATA POINTS	TYPE	DESCRIPTION

Table					
	0	1	U32	Manufacturing Information	6All
	4	1	U32	Product serial number	
	8	1	U32	RF module number	
	C	1	U32	Product manufacture date	
	10	1	U32	Last calibration date	
	14	4	NA	Reserved	
	24	1	F32	Firmware revision	
	28	1	F32	Hardware revision	
	2C	40	F32	Reserved	
	CF	1	U8	Startup state (reference)	
	D0	32	U8	Startup state (synth)	
	F0	1	U32	TCXO DAC calibration value	
	F4	1	F32	Calibration temperature	
	F8	1	F32	ALC enabled temperature coefficient	
	FC	1	F32	ALC disabled temperature coefficient	
	100	128	U8	Ch0 VCO value	
	180	128	U8	Ch0 VCO tune value	
	200	128	U8	Ch1 VCO value	
	280	128	U8	Ch1 VCO tune value	
	300	1	U8	Reference attenuation value	
	301	125	U16	Cal frequencies (MHz)	
	3FB	3875	F32	Ch0 attenuator values	
	4087	125	F32	Ch0 ALC closed ref RF power	
	427B	125	U16	Ch0 ALC closed ref DAC value	
	4375	375	F32	Ch0 ALC closed coefficients (2^{nd} order)	
	4951	125	F32	Ch0 ALC opened ref RF power	
	4B45	125	U16	Ch0 ALC opened ref DAC value	
	4C3F	375	F32	Ch0 ALC opened coefficient	
	521B	3875	F32	Ch1 attenuator values	
	8EA7	125	F32	Ch1 ALC closed ref RF power	
	909B	125	U16	Ch1 ALC closed ref DAC value	
	9195	375	F32	Ch1 ALC closed coefficients (2^{nd} order)	
	9771	125	F32	Ch1 ALC opened ref RF power	
	9965	125	U16	Ch1 ALC opened ref DAC value	
	9A5F	375	F32	Ch1 ALC opened coefficient	

calibration data, whether floats, unsigned 8-bit, unsigned 16-bit or unsigned 32-bit integers, are stored as flattened unsigned byte representation. A float is flattened to 4 unsigned bytes, so once it is read back it needs to be un-flattened back to its original type. Unsigned values containing more than a single byte are converted (un-flattened) simply by concatenation of the bytes through bit-shifting. Converting to floating point representation is slightly more involved. First, convert the 4 bytes into an unsigned 32-bit integer value, and then (in C/C++) 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:

```
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 RF ALC DAC Value

ALC_DAC_VALUE_GET (0x39) - The user may be interested to obtain the current value of the ALC DAC for the purpose of making minor adjustments to the RF output power level. Data is returned in the first 14 bits.

CALIBRATION EEPROM MAP

Table 6. Calibration EEPROM map.

EEPROM ADDRESS (HEX)	NUMBER OF DATA POINTS	TYPE	DESCRIPTION
0	1	U32	Manufacturing Information
4	1	U32	Product serial number
8	1	U32	RF module number
C	1	U32	Product manufacture date
10	1	U32	Last calibration date
14	4	NA	Reserved
24	1	F32	Firmware revision
28	1	F32	Hardware revision
2C	40	F32	Reserved
CF	1	U8	Startup state (reference)
D0	32	U8	Startup state (synth)
F0	1	U32	TCXO DAC calibration value
F4	1	F32	Calibration temperature
F8	1	F32	ALC enabled temperature coefficient
FC	1	F32	ALC disabled temperature coefficient
100	128	U8	Ch0 VCO value
180	128	U8	Ch0 VCO tune value
200	128	U8	Ch1 VCO value
280	128	U8	Ch1 VCO tune value
300	1	U8	Reference attenuation value
301	125	U16	Cal frequencies (MHz)
3FB	3875	F32	Ch0 attenuator values
4087	125	F32	Ch0 ALC closed ref RF power
427B	125	U16	Ch0 ALC closed ref DAC value
4375	375	F32	Ch0 ALC closed coefficients (2^{nd} order)
4951	125	F32	Ch0 ALC opened ref RF power
4B45	125	U16	Ch0 ALC opened ref DAC value
4C3F	375	F32	Ch0 ALC opened coefficient
521B	3875	F32	Ch1 attenuator values
8EA7	125	F32	Ch1 ALC closed ref RF power
909B	125	U16	Ch1 ALC closed ref DAC value
9195	375	F32	Ch1 ALC closed coefficients (2^{nd} order)
9771	125	F32	Ch1 ALC opened ref RF power
9965	125	U16	Ch1 ALC opened ref DAC value
9A5F	375	F32	Ch1 ALC opened coefficient

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 provided in the SC5505A API dynamic linked library or LabVIEW library are:

- **sc5505a_ListResources**
- **sc5505a_OpenDevice**
- **sc5505a_CloseDevice**
- **sc5505a_RegWrite**
- **sc5505a_RegRead**
- **sc5505a_InitDevice**
- **sc5505a_SetDeviceStandby**
- **sc5505a_SetFrequency**
- **sc5505a_SetPowerLevel**
- **sc5505a_SetRfOutput**
- **sc5505a_SetAlcMode**
- **sc5505a_DisableAutoLevel**
- **sc5505a_SetClockReference**
- **sc5505a_SetReferenceDac**
- **sc5505a_SetAlcDac**
- **sc5505a_WriteUserEeprom**
- **sc5505a_StoreCurrentState**
- **sc5505a_GetDeviceInfo**
- **sc5505a_GetDeviceStatus**
- **sc5505a_GetTemperature**
- **sc5505a_ReadCalEeprom**
- **sc5505a_GetAlcDac**
- **sc5505a_ReadUserEeprom**

Each of these functions is described in more detail on the following pages. Example code in C/C++ is available to show how these functions are called and used. First, for C/C++ we define the constants and types which are contained in the C header file, *sc5505a.h*. These constants and types are useful not only as an include for developing user applications using the SC5505A API, but also for writing device drivers independent of those provided by SignalCore.

Constants Definitions

```
// Parameters for storing data in the onboard EEPROM
#define MAXDEVICES                  50    // bytes
#define MAXDESCRIPTORSIZE           25    // bytes
#define CALEEPROMSIZE            65536    // bytes
#define USEREEPROMSIZE           32768    // bytes

// Assign channel names
#define CH1                          0
#define CH2                          1

// Synthesizer fine tune modes
#define DISABLEDFINEMODE             0    // 1 MHz tuning steps, PLL implementation
#define PLLFINEMONDE                 1    // 25 KHz tuning steps, PLL implementation
#define DDSFINEMODE                  2    // 1 Hz tuning steps, DDS implementation

// Define error codes
#define RESERVEDERROR               -1
#define NOTCORRECTDEVICE            -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

// Define device registers
#define INITIALIZE                0x01    // initialize the device
#define SET_SYSTEM_ACTIVE         0x02    // set the device “active” LED
#define RF_FREQUENCY              0x10    // set the frequency
#define RF_POWER                  0x11    // set power of LO1
#define RF_ENABLE                 0x12    // enable RF output
#define AUTO_PWR_DISABLE          0x13    // disable auto power leveling
#define RF_ALC_MODE               0x14    // select closed (0) or open (1) loop ALC modes
#define POWER_STANDBY             0x15    // place the selected channel in standby mode
#define REFERENCE_SETTING         0x16    // reference settings
#define GET_TEMPERATURE           0x17    // get the internal temperature of the device
#define GET_DEVICE_STATUS         0x18    // read the device status
#define USER_EEPROM_READ          0x1A    // read a byte from the user EEPROM
#define USER_EEPROM_WRITE         0x1B    // write a byte to the user EEPROM
#define REFERENCE_DAC             0x1D    // set the reference DAC value
#define CAL_EEPROM_READ           0x21    // read a byte from the calibration EEPROM
#define START_UP_STATE            0x23    // store the new default state
#define ALC_DAC_VALUE_SET         0x24    // set the RF ALC DAC value
#define ALC_DAC_VALUE_GET         0x39    // read back the RF ALC DAC value
```

Type Definitions

```
typedef struct deviceInfo_t
{
       unsigned int productSerialNumber;
       unsigned int rfModuleSerialNumber;
```

```
        float firmwareRevision;
        float hardwareRevision;
        unsigned int calDate; // year, month, day, hour:&(0xFF000000,0xFF0000,0xFF00,0xFF)
        unsigned int manDate; // year, month, day, hour:&(0xFF000000,0xFF0000,0xFF00,0xFF)
}       deviceInfo_t;

typedef struct // deviceStatus_t
{
        bool ch1AlcOpen;
        bool ch2AlcOpen;
        bool ch1OutEnable;
        bool ch2OutEnable;
        bool ch1StandbyEnable;
        bool ch2StandbyEnable;
        bool ch1SumPllStatus;
        bool ch1CrsPllStatus;
        bool ch1FinePllStatus;
        bool ch2SumPllStatus;
        bool ch2CrsPllStatus;
        bool ch2FinePllStatus;
        bool vcxoPllStatus;
        bool tcxoPllStatus;
        bool extRefDetected;
        bool extRefLockEnable;
        bool refClkOutEnable;
        bool deviceAccess;
        bool pxiClk10Enable;
} deviceStatus_t;
```

Function Definitions and Usage

The functions listed below are found in the *sc5505a.dll* dynamic linked library for the Windows™ operating system. These functions are also provided in the LabVIEW library, *sc5505a.llb*. The LabVIEW functions contain context help (Cntrl-H) to help with the input and output parameters.

Function: **sc5505a_ListResources**

Definition: **int sc5505a_ListResources(char **visaResource, unsigned int *size)**

Output: **char** **visaResource (pointer list to visaResource)
unsigned int *size (the numbers of devices found)

Description: **sc5505a_ListResources** searches for SignalCore SC5505A devices connected to the host computer and returns an array containing their VISA resource names. The user can use this information to open the device(s) with the resource name of choice. See **sc5505a_OpenDevice** function on how to open a device.

Function: **sc5505a_OpenDevice**

Definition: **int sc5505a_OpenDevice(char *visaResource, unsigned int *deviceHandle)**

Input: **char** *visaResource (pointer list to visaResource)

Output: **unsigned int** *deviceHandle (unsigned int number for the deviceHandle)

Description: **sc5505a_OpenDevice** opens the device and turns the front panel "active" LED on if it is successful. It returns a handle to the device for other function calls.

Function: **sc5505a_CloseDevice**

Definition: **int sc5505a_CloseDevice(unsigned int deviceHandle)**

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

Description: **sc5505a_CloseDevice** closes the device associated with the device handle and turns off the "active" LED on the front panel if it is successful.

Example: Code to exercise the functions that open and close the PXIe device:

```
// Declaring
char **visaResource = (char**)malloc(sizeof(char*)*MAXDEVICES);
unsigned int devHandle;
int devicesFound;
int i,status;
// Allocate memory
for(i = 0; i< 100; i++)
       visaResource[i] = (char*)malloc(sizeof(char)*MAXDESCRIPTORSIZE);

status = sc5505a_ListResources(visaResource, &devicesFound);
 printf("There are %d SignalCore SC5505A devices found. \n", devicesFound);

if(devicesFound == 0)
//If no device are found deallocate memory and end the program
{
       for(i = 0; i<MAXDEVICES;i++)
              free(visaResource[i]);
       free(visaResource);
       printf("No sc5505a devices detected. Press enter to continue.\n");
       return 1;
}

//** sc5505a_OpenDevice, open device 0
status = sc5505a_OpenDevice(visaResource[0], &devHandle);

// Free memory
for(i = 0; i<MAXDEVICES;i++)
       free(visaResource[i]);
free(visaResource);
 //
// Do something with the device
//
//Close the device
int status = sc5505a_CloseDevice(devHandle);
```

Function: **sc5505a_RegWrite**

Definition: **int sc5505a_RegWrite(unsigned int deviceHandle, unsigned char commandByte, unsigned long long int instructWord)**

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

unsigned char commandByte (register address)

unsigned long long int instructWord (the data for the register)

Description: **sc5505a_RegWrite** writes the instructWord data to the register specified by the commandByte. See the register map on Table 3 for more information.

Example: To set the power level to 2.00 dBm:

```
int status = sc5505a_RegWrite(devHandle, RF_POWER, 200);
```

Function: **sc5505a_RegRead**

Definition: **int sc5505a_RegRead(unsigned int deviceHandle, unsigned char commandByte, unsigned long long int instructWord, unsigned int *receivedWord)**

Input:
unsigned int deviceHandle (handle to the opened device)
unsigned char commandByte (The address byte of the register to write to)
unsigned long long int instructWord (the data for the register)
unsigned int *receivedWord (data to be received)

Description: **sc5505a_RegRead** reads the data requested by the instructWord data to the register specified by the commandByte. See the register map on Table 4 for more information.

Example: To read the status of the device:

```
unsigned int deviceStatus;

int status = sc5505a_RegRead(devHandle,GET_DEVICE_STATUS,0x00,&deviceStatus);
```

Function: **sc5505a_InitDevice**

Definition: **int sc5505a_InitDevice(unsigned int deviceHandle, bool Mode)**

Input:
unsigned int deviceHandle (handle to the opened device)
bool mode (Set the mode of initialization)

Description: **sc5505A_InitDevice** initializes (resets) the device. Mode = 0 resets the device to the default power up state. Mode = 1 resets the device but leaves it in its current state.

Function: **sc5505a_SetDeviceStandby**

Definition: **int sc5505a_SetDeviceStandby(unsigned int deviceHandle, unsigned char channel, bool standbyStatus)**

Input:
unsigned int deviceHandle (handle to the opened device)
unsigned char channel (channel name to put into standby)
bool standbyStatus Set to true (1) to set device in standby mode

Description: **sc5505a_SetStandby** puts a channel in standby mode where the power to the analog circuits for that channel are disabled, conserving power.

Function: **sc5505a_SetFrequency**

Definition: **int sc5505a_SetFrequency(unsigned int deviceHandle, unsigned char channel, unsigned long long int frequency)**

Input: **unsigned int** deviceHandle (handle to the opened device)
unsigned char channel (channel name of target frequency change)
unsigned long long int frequency (frequency in Hz)

Description: **sc5505a_SetFrequency** sets the channel RF frequency.

Function: **sc5505a_SetPowerLevel**

Definition: **int sc5505a_SetPowerLevel(unsigned int deviceHandle, unsigned char channel, float powerLevel)**

Input: **unsigned int** deviceHandle (handle to the opened device)
unsigned char channel (channel name of target power level change)
float powerLevel (set power in dBm)

Description: **sc5505a_SetPowerLevel** sets the value of the desired output power level for the channel.

Function: **sc5505a_SetRfOutput**

Definition: **int sc5505a_SetRfOutput(unsigned int deviceHandle, unsigned char channel, bool mode)**

Input: **unsigned int** deviceHandle (handle to the opened device)
unsigned char channel (channel name of target RF output toggle)
bool mode (disable/enable RF output)

Description: **sc5505a_SetRfOutput** enables or disables the RF output on a channel.

Function: **sc5505a_SetAlcMode**

Definition: **int sc5505a_SetAlcMode(unsigned int deviceHandle, unsigned char channel, bool mode)**

Input: **unsigned int** deviceHandle (handle to the opened device)
unsigned char channel (channel name for desired ALC mode change)
bool mode (closed or open loop operation of the ALC circuit)

Description: **sc5505a_SetAlcMode** sets the ALC loop to closed or open loop operation for a channel.

Function: **sc5505a_DisableAutoLevel**

Definition: **int sc5505a_DisableAutoLevel(unsigned int deviceHandle, unsigned char channel, unsigned char mode)**

Input: **unsigned int** deviceHandle (handle to the opened device)
unsigned char channel (channel name to disable auto-level)
unsigned char mode (defines when to disable auto-level)

Description: **sc5505a_DisableAutoLevel** disables the device from auto adjusting the power level to the current state when frequency is changed. Disabling the auto adjust feature allows

the device to return faster after calling **sc5505a_SetPowerLevel**. If the power remains the same for the next tuned frequency, this should not be disabled.

Function: **sc5505a_SetClockReference**

Definition: **int sc5505a_SetClockReference(unsigned int deviceHandle, bool lockExtEnable, bool refOutEnable, bool pxiClk)**

Input:
unsigned int deviceHandle (handle to the opened device)
bool lockExtEnable (enables phase locking to an external source)
bool refOutEnable (enables the internal clock to be exported on the REF OUT port)
bool pxiClk (enables the PXI 10 MHz clock)

Description: **sc5505a_SetReferenceClock** configures the reference clock behavior of the device**.**

Function: **sc5505a_SetReferenceDac**

Definition: **int sc5505a_SetReferenceDac(unsigned int deviceHandle, unsigned int dacValue)**

Input:
unsigned int deviceHandle (handle to the opened device)
unsigned int dacValue (14bit value for the reference DAC)

Description: **sc5505a_SetReferenceDac** sets the value of the DAC that tunes the internal reference TXCO. The user may choose to override the value stored in memory for example, to correct for long-term accuracy drift.

Function: **sc5505a_SetAlcDac**

Definition: **int sc5505a_SetAlcDac(unsigned int deviceHandle, unsigned char channel, unsigned int dacValue)**

Input:
unsigned int deviceHandle (handle to the opened device)
unsigned char channel (channel name of target ALC DAC)
unsigned int dacValue (14 bit value for adjusting the ALC DAC)

Description: **sc5505a_SetAlcDac** writes a value to the ALC DAC to control the RF output level for a channel.

Function: **sc5505a_WriteUserEeprom**

Definition: **int sc5505a_WriteUserEeprom(unsigned int deviceHandle, unsigned int memAdd, unsigned char byteData)**

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: **sc5505a_WriteUserEeprom** writes one byte of data to the memory address specified.

Function: **sc5505a_StoreCurrentState**

Definition: **int sc5505a_StoreCurrentState(unsigned int deviceHandle)**

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

Description: **sc5505a_StoreCurrentState** stores the current state of the device as the default power-up state.

Function: **sc5505a_GetDeviceInfo**

Definition: **int sc5505a_GetDeviceInfo(unsigned int deviceHandle, deviceInfo_t *devInfo)**

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

Output: **deviceInfo_t** *devInfo (device info struct)

Description: **sc5505a_GetDeviceInfo** retrieves device information such as serial number, calibration date, revisions, etc.

Function: **sc5505a_GetDeviceStatus**

Definition: **int sc5505a_GetDeviceStatus (unsigned int deviceHandle, deviceStatus_t *deviceStatus)**

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

Output: **deviceStatus_t** *deviceStatus (deviceStatus struct)

Description: **sc5505a_GetDeviceStatus** retrieves the status of the device such as 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 = sc5505a_GetDeviceStatus(devHandle, 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: **sc5505a_GetTemperature**

Definition: **int sc5505a_GetTemperature (unsigned int deviceHandle, float *temperature)**

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

Output: **float** *temperature (temperature in degrees Celsius)

Description: **sc5505a_GetTemperature** retrieves the internal temperature of the device.

Function: **sc5505a_ReadCalEeprom**

Definition: **int sc5505a_ReadCalEeprom(unsigned int deviceHandle, unsigned int memAdd, unsigned char *byteData)**

Input: **unsigned int** deviceHandle (handle to the opened device)
unsigned int memAdd (EEPROM memory address)
Output: **unsigned char** *byteData (the read back byte data)
Description: **sc5505a_ReadUserEeprom** reads back a byte from a specific memory address of the calibration EEPROM.

Function: **sc5505a_GetAlcDac**

Definition: **int sc5505a_GetAlcDac(unsigned int deviceHandle, unsigned char channel, unsigned int *dacValue)**

Input: **unsigned int** deviceHandle (handle to the opened device)
unsigned char channel (channel name of target ALC DAC)
Output: **unsigned char** *dacValue (the read back byte data)
Description: **sc5505a_GetAlcDac** reads back the current ALC DAC value.

Function: **sc5505a_ReadUserEeprom**

Definition: **int sc5505a_ReadUserEeprom(unsigned int deviceHandle, unsigned int memAdd, unsigned char *byteData)**

Input: **unsigned int** deviceHandle (handle to the opened device)
unsigned int memAdd (EEPROM memory address)
Output: **unsigned char** *byteData (the read back byte data)
Description: **sc5505a_ReadUserEeprom** reads back a byte from a specific memory address of the user EEPROM.

CALIBRATION & MAINTENANCE

The SC5505A is factory calibrated and ships with a certificate of calibration. SignalCore strongly recommends that the SC5505A 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.

SC5505A 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 SC5505A, 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 SC5505A.

The SC5505A 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 user.

REVISION NOTES

Rev 1.0	Initial Release	2013
Rev 1.0.1	Added programming instructions	2013
Rev 1.1	Removed outdated address	7-10-2020