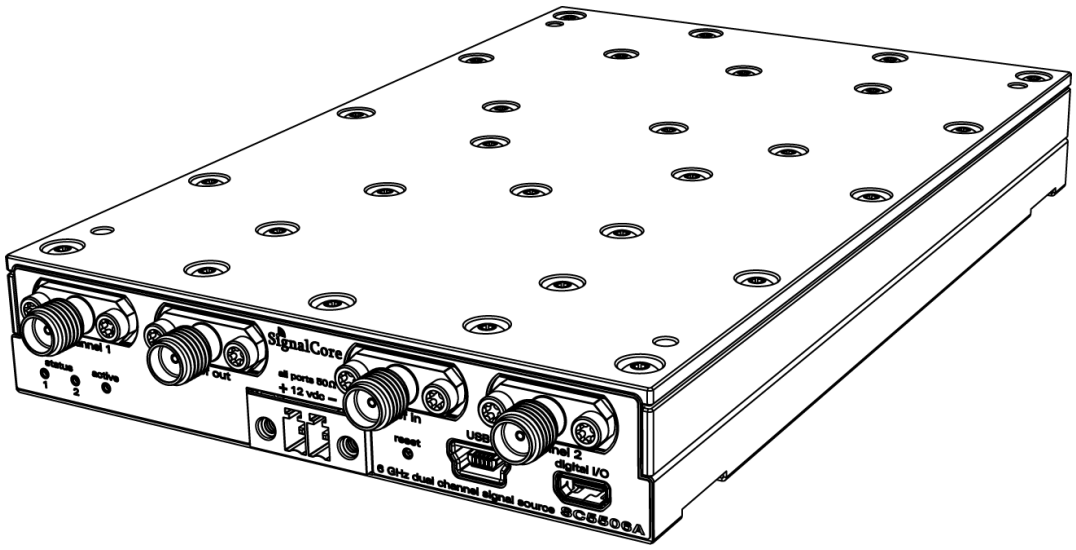
SignalCore™
PRESERVING SIGNAL INTEGRITY



SC5506B

25 MHz to 6 GHz RF Signal Source

USB, SPI and RS-232 Interfaces

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 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)**
SC5506B	✓	✓	✓	✓	✓	✓

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 SC5506B kit contains the following items:

Quantity	Item
1	SC5506B Dual Channel RF Signal Source
1	Software Installation USB Flash Drive (may be combined with other products onto a single drive)

Setting Up and Configuring the SC5506B

The SC5506B is a core module-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.

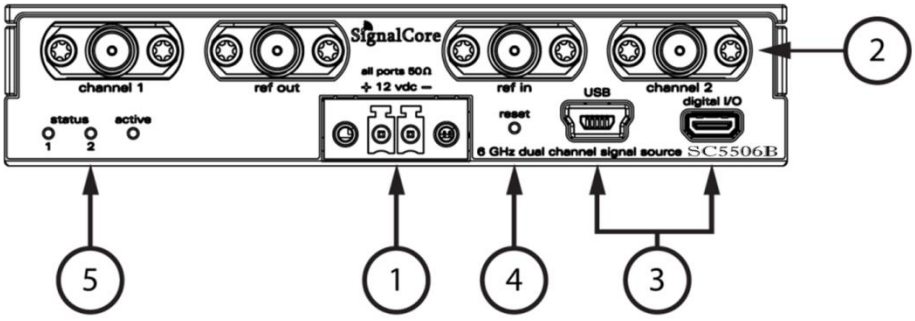


Figure 1. Front view of the SC5506B showing user I/O locations.

1 Power Connection

Power is provided to the device through a two-position screw terminal block connection as shown in Figure 1. Proper operation of the device requires +12 VDC source and ground return wires capable of delivering a minimum current of 1.5 Amps. The polarity of the connector is shown on the front panel of the RF module, just above the screw terminal block.

2 Signal Connections

All signal connections (ports) on the SC5506B 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 take several days to fully evaporate and may degrade measurement performance until fully evaporated.

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.

3 Communication Connections

The SC5506B uses a mini-USB Type B connector (for USB communication) and a micro-HDMI (for SPI or RS-232 communication, depending on the version ordered) to communicate with the device. The USB port uses the standard USB 2.0 protocol found on most host computers. The pinout of this connector, viewed from the front of the module, is listed in Table 1.

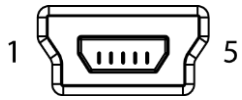


Table 1. Pinout of the SC5506B USB communication connector.

Pin Number	USB Function	Description
1	VBUS	Vcc (+5 Volts)
2	D –	Serial data
3	D +	Serial data
4	ID	Not used
5	GND	Device ground (also tied to connector shell)

The user can also communicate with the device through the micro-HDMI type-D port. Depending on the version ordered, this connector provides either the SPI or RS-232 communication path, as well as hardware trigger. The pinout of this connector, viewed from the front of the module, is listed in Table 2.

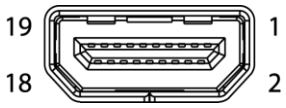


Table 2. Pinout of the SC5506B micro-HDMI connector for either SPI or RS-232 communication.

Pin Number	SPI Function	RS-232 Function
3	MISO	TxD
5	–	RTS
8	reset_b	
9	MOSI	RxD
11	CS	CTS
15	SRLDY	–
17	CLK	CLK
18	SPI MODE	BAUD SELECT
6	Trigger In	
1	Trigger Out	
4, 7, 10, 13, 16	GND	GND
2, 8, 12, 14, 19	NC	NC

4

Reset Button

Depressing this momentary-action push button switch will reset the device to its default state. The SC5506B has the ability to store the current configuration at any point as the default setting. If the factory setting has been overwritten with a saved user configuration, resetting the device will reinitialize the device to the saved user configuration. Otherwise, resetting the device will restore the factory configuration. Refer to the table in the "Default Startup Mode" section of this manual for default settings.

5 Indicator LEDs

The SC5506B provides visual indication of important modes. There are three LED indicators on the unit. Their behavior under different operating conditions is shown in Table 3.

Table 3. LED indicator states.

LED	Color	Definition
STATUS	Green	"Power good" and all oscillators phase-locked
STATUS	Orange	Channel powered down
STATUS	Red	One or more oscillators off lock
STATUS	Off	Power fault
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 SC5506B 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 user 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 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 are performed automatically by the system, although the user may chose to override the ALC DAC value if needed. In Figure 2, the labels in red indicate parameters or devices which the user has direct control over.

Frequency Synthesizer

The synthesizer section of the SC5506B 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 few millihertz over a narrow frequency range, while the “coarse” PLL tunes in steps of a few megahertz over several gigahertz 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.

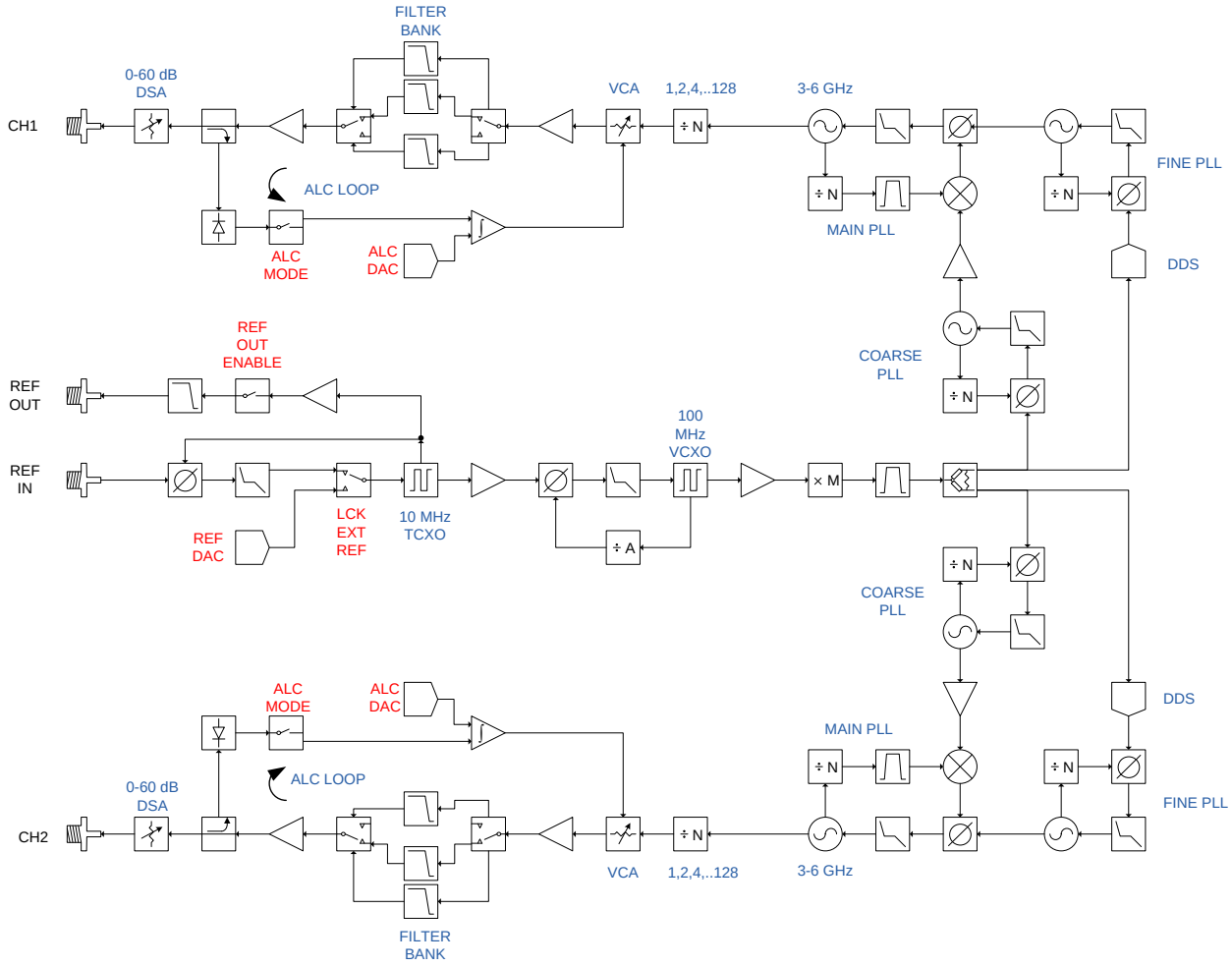


Figure 2. Simplified block diagram of the SC5506B dual channel RF signal source.

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 in certain parts of the band - especially at frequency regions close the integer boundaries. A multiple loop architecture allows fine tuning with extremely low phase spurs.

Reference Clock Control

As mentioned above, the primary clock reference for the SC5506B 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_VALUE** register.

Harmonics and Range of Operation

The SC5506B's guaranteed operating frequency range is 25 MHz to 6.0 GHz. At lower 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 SC5506B 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.

Modes of RF Generation

The SC5506B has both single fixed tone and list mode operation for both channels, however only one channel can be selected for sweep/list mode at time. In single fixed tone mode, it operates as a normal

synthesizer where the user writes the frequency (RF_FREQUENCY) register to change the frequency. In list mode, the device is triggered to automatically run through a set of frequency points that are either entered directly by the user or pre-computed by the device based on user parameters. Configuration of the device for list mode operation is accomplished by setting up the LIST_MODE_CONFIG register.

Sweep Function

When frequency points are generated based on the start/stop/step set of frequencies, this is (in the context of this product) known as putting the device into *sweep*. When the sweep function is enabled, the frequency points are incrementally stepped with a constant step size either in a linearly increasing or linearly decreasing fashion.

List Function

The list function requires that the frequency points are read in from a list provided by the user. The user will need to load the frequency points into the list buffer via the LIST_BUFFER_WRITE register, or have the device read the frequency points from the EEPROM into it.

Sweep Direction

The sweep can be chosen to start at the beginning of a list and incrementally step to the end of the list or vice versa.

Sweep Waveform

The list of frequency points may be swept in a saw-tooth manner or triangular manner. If sawtooth is selected, upon reaching the last frequency point the device returns back to the starting point. Plotting frequency versus time reveals a sawtooth pattern. If triangular is selected, the device will sweep linearly from the starting point, then reverse its direction after the last (highest or lowest) frequency and sweep backwards toward the start point, mapping out a triangular waveform on a frequency versus time graph.

Dwell Time

The dwell time at each frequency, in either sweep or list modes, is determined by writing to the LIST_DWELL_TIME register. The dwell time step increment is 500 μs. However, the recommended minimum dwell time is 1 ms, which allows sufficient time for the signal to settle before a measurement is made. Due to the size limitation of the onboard RAM, it is not possible to have a pre-calculated configuration parameters list that could be used to program the various functions of the device, decreasing the setup time of the device for frequency change. As a result, for each frequency change the configuration parameters are dynamically computed. This overhead computational time to handle the mathematics, triggers, timers, and interrupts may increase the effective frequency settling time close 500 μs. The amplitude computational time alone is close to 300 μs. If the sweep is over a narrow range, it is best to disable the automatic power leveling feature, allowing faster frequency sweeps. By default whenever the frequency is changes, the device re-computes a set of new parameters to set the ALC. Over short range

frequencies, the parameters are similar so the amplitude variation may be acceptable. If automatic power leveling is turned on, allow for a minimal dwell time of 2-5 ms.

List Cycles

The number of repeat cycles for a sweep or list is set by writing the LIST_CYCLE_COUNT register. Writing the value 0 to the register will cause the device to repeat the sweep/list forever until a trigger is sent or the RF mode is changed to single fixed tone mode via the RF_MODE register. Upon completion of the a cycle, the frequency may be set to end on the last frequency point or return back the starting point. This is cycle ending behavior is configured with bit [5] of the LIST_MODE_CONFIG register.

Trigger Sources

The device may be set up for software or hardware triggering. This is defined in bit [4] of the LIST_MODE_CONFIG register. If software trigger is selected, writing the LIST_SOFT_TRIGGER register will trigger the device to perform the sweep/list function defined in the LIST_MODE_CONFIG register. The device may also be triggered via the PXI trigger_0 line on the back plane. Hardware trigger occurs on a high to low transition state of this line.

Hardware Trigger Modes

The device may be triggered to start a sweep or list then uses the next trigger to stop it. In triggered start/stop mode, alternating triggers will start and stop the sweep/list. In this mode, start triggering will always return the frequency point to the beginning of the sweep/list. It does not continue from where it had left off from a stop trigger. The device may also be triggered to step to the next frequency with each start trigger. This is known as the triggered step mode. Software triggering cannot perform the step trigger function. This can only be done through hardware triggering. When hardware step triggering has started, performing a software trigger or changing the RF mode to single fixed tone will take the device out of step trigger state before a cycle is completed. The trigger input is on pin #6 of the micro-HDMI type-D digital connector.

Trigger Out Modes

The device can be set to send out a high to low transition signal when the configuration of a frequency by the device is completed; that is, it has completed all necessary computations, and has successfully written data to the appropriate components. This trigger pulse can be sent on the completion of every step frequency or on the last frequency of a sweep cycle. This trigger signal is present on pin #1 (Trigger Out) of the micro HDMI type-D digital I/O connector.

Default Startup Mode

The factory power-up state for the device is detailed in Table 4. 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 4. 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
RF Mode	Fix	Fix
Auto Level	Enabled	Enabled
Ref Out	Disabled	
Ext Ref Lock	Disabled	

SC5506B PROGRAMMING INTERFACE

Device Drivers

The SC5506B is programmed 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 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 **sc5506b_RegWrite** and **sc5506b_RegRead** functions respectively.

For Microsoft Windows™ operating systems, The SC5506B API is provided as a dynamic linked library, *sc5506b.dll*, which is available for 32bit and 64bit operating systems. This API is based on the libusb-1.0 library and therefore it is required to be installed on the system prior to development. The *libusb-1.0.dll* will install along with the *sc5506b.dll*, and along with the header files for development. However for possible newer versions of libusb-1.0, visit http://libusbx.org to check for version updates and downloads. To install the necessary drivers, right click on the *sc5506b.inf* file in the *Win*\ directory and chose install. When the device is connected to a USB port, the host computer should identify the device and load the appropriate driver. For more information, see the *SC5506B_Readme.txt* file in the *Win*\ directory.

For LabVIEW™ support, a full LabVIEW API is provided and is available in the *Win**API**LabVIEW*\ directory. To use the library, copy the "SignalCore" folder in that directory to *%LabVIEW path%**instr.lib* location of your LabVIEW installation directory. The LabVIEW functions are simply VI wrappers around *sc5506b.dll*. Code written purely in G that does not call or depend on external library functions is available to our customers on request. If pure G code SC5506B API functions are to be used, the National Instruments driver wizard packaged with NI-VISA should be used to create a driver for the intended operating system. SignalCore's vendor ID is **0x277C** and the SC5506B product ID (PID) is **0x0022**.

For other operating systems, users will need to write and compile their own drivers. The device register map provides the necessary information to successfully implement a driver for the SC5506B. Driver code based on libusb-1.0 is available to our customers on request. 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 SC5506B 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. Using the API, commands to control the device are greatly simplified. For example, to obtain the device temperature, the user simply calls the function **sc5506b_get_device_temperature**, or calls **sc5506b_set_frequency** to tune the frequency. The software API is covered in detail in the "USB Software API Library Functions" section.

SPI and RS-232 Programming

Please see the SPI and RS-232 sections of this manual for interfacing to the device registers using either of these communication methods.

SETTING THE SC5506B: WRITING TO CONFIGURATION REGISTERS

Configuration Registers

The users may write the configuration registers (write only) directly by calling the **sc5506b_reg_write** function. The syntax for this function is **sc5506b_reg_write(dev_handle, reg_byte, instruct_word)**. The **instruct_word** takes a 64 bit-word. However, the function will only send the required number of bytes to the device. These registers are the same for USB, SPI, and RS-232. Table 5 summarizes the register addresses (commands) and the effective bytes of command data.

Table 5. 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 LED
RF_MODE	0x05	[7:0]	Open	Open	Open	Open	Open	Open	Channel	Mode
LIST_MODE_CONFIG	0x06	[7:0]	Trig out mode	Trig out enable	Return to start	Step on trig	HW/SW trig	Trig /saw	Dir	Sweep /list
LIST_START_FREQ	0x07	[7:0]	Frequency Word (Hz) [7:0]							
		[15:8]	Frequency Word (Hz) [15:8]							
		[23:16]	Frequency Word (Hz) [23:16]							
		[31:24]	Frequency Word (Hz) [31:24]							
		[39:32]	Frequency Word (Hz) [39:32]							
LIST_STOP_FREQ	0x08	[7:0]	Frequency Word (Hz) [7:0]							
		[15:8]	Frequency Word (Hz) [15:8]							
		[23:16]	Frequency Word (Hz) [23:16]							
		[31:24]	Frequency Word (Hz) [31:24]							
		[39:32]	Frequency Word (Hz) [39:32]							
LIST_STEP_FREQ	0x09	[7:0]	Frequency Word (Hz) [7:0]							
		[15:8]	Frequency Word (Hz) [15:8]							
		[23:16]	Frequency Word (Hz) [23:16]							
		[31:24]	Frequency Word (Hz) [31:24]							
		[39:32]	Frequency Word (Hz) [39:32]							
LIST_DWELL_TIME	0x0A	[7:0]	Word [7:0]							
		[15:8]	Word [15:8]							
		[23:16]	Word [23:16]							
		[31:24]	Word [31:24]							
LIST_CYCLE_COUNT	0x0B	[7:0]	Word [7:0]							
		[15:8]	Word [15:8]							
		[23:16]	Word [23:16]							
		[31:24]	Word [31:24]							
LIST_BUFFER_POINTS	0x0C	[7:0]	Word [7:0]							
		[15:8]	Word [15:8]							

LIST_BUFFER_WRITE	0x0D	[7:0]	Frequency Word (Hz) [7:0]							
		[15:8]	Frequency Word (Hz) [15:8]							
		[23:16]	Frequency Word (Hz) [23:16]							
		[31:24]	Frequency Word (Hz) [31:24]							
		[39:32]	Frequency Word (Hz) [39:32]							
LIST_BUF_MEM_XFER	0x0E	[7:0]	Open	Open	Open	Open	Open	Open	Open	Dir
LIST_SOFT_TRIGGER	0x0F	[7:0]	Open	Open	Open	Open	Open	Open	Open	Open
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_LEVEL	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_LEVEL_DISABLE	0x13	[7:0]	Open	Open	Open	Open	Open	Open	Channel	Mode
RF_ALC_DISABLE	0x14	[7:0]	Open	Open	Open	Open	Open	Open	Channel	Mode
CHANNEL_STANDBY	0x15	[7:0]	Open	Open	Open	Open	Open	Open	Channel	Mode
REFERENCE_SETTING	0x16	[7:0]	Open	Open	Open	Open	Open	Open	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_VALUE	0x1D	[7:0]	DAC word [7:0]							
		[15:8]	Open	Open	DAC word [13:8]					
STORE_DEFAULT_STATE	0x23	[7:0]	Open	Open	Open	Open	Open	Open	Open	Open
SET_ALC_DAC_VALUE	0x24	[7:0]	ALC DAC Word [7:0] (1/100 dBm)							
		[15:8]	Sign (+ / -)	ALC DAC Word[14:8] (1/100 dBm)						
		[23:16]	Open	Open	Open	Open	Open	Open	Open	Channel

To write to the device through USB transfers such as bulk transfer, it is important to send data with the register byte first, followed by the most significant bit (MSB) of the data bytes. For example, to set the RF power level of channel 2 to a particular amplitude, the byte stream would be [0x11][0x01][15:8][7:0].

Initializing the Device

INITIALIZE (0x01) - Writing 0x01 to this register will reset the device to the default power-on state. Writing 0x00 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 **STORE STARTUP STATE (0x23)** register, described later in this section.

Setting the System Active LED

SET_SYSTEM_ACTIVE (0x02) - This register simply 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 Mode (Fix/List)

RF_MODE (0x05) - This register puts the selected channel into fix frequency mode or list/sweep mode. Bit 1 selects the channel 1 or channel 2 with 0 or 1 respectively. Bit 0 chooses the fix or list mode with 0 or 1 respectively.

Setting the List Behavior

LIST_MODE_CONFIG (0x06) - This register configures the list behavior. Each bit is detailed below.

- Bit 7: **Trigger-Out-Mode** selects whether the output trigger pin (Pin #1 of the digital I/O connector) is triggered on every frequency step or at the end of a cycle.
- Bit 6: **Trigger-Out-Enable** enables output triggering.
- Bit 5: **Return-To-Start** returns the frequency to the starting point after a cycle count is completed.
- Bit 4: **Hardware-Trigger-Stepping** will step to the next frequency point on the list if enable. Otherwise it behaviors as Start-Stop triggers
- Bit 3: **Hardware-Trigger** if selected the device expects triggering from the Trigger Input pin of the device (Pin #6 of the digital I/O connector), otherwise it expects a software trigger to start the sweep/list.
- Bit 2: **Triangular-Waveform** if selected will traverse the list from the beginning point to the ending point and then reverses direction back to the beginning point as a complete cycle. If unselected
- Bit 1: **High-To-Low** if selected will start from the end of the list to the beginning or from the stop frequency to the start frequency, others it will start at the beginning of the list.
- Bit 0: **Start-Stop-Step (Sweep-Mode)** will use frequency list points computed using the stop, start, and step frequency points, otherwise it will use the list stored in the list buffer.

Setting the Sweep Frequencies

A list of frequency points used for sweep can be generated dynamically using the start, stop and step frequencies that at enter into the device through the following registers:

LIST_START_FREQ (0x07) - This register sets the list start frequency. Data is sent as a 40 bit word with the LSB in Hz.

LIST_STOP_FREQ (0x08) - This register sets the list stop frequency. Data is sent as a 40 bit word with the LSB in Hz.

LIST_STEP_FREQ (0x09) - This register sets the list step frequency. Data is sent as a 40 bit word with the LSB in Hz.

Setting the Sweep/List Dwell Time

LIST_DWELL_TIME (0x0A) - This register sets the dwell time at each frequency when the device is set to sweep through a list of frequencies. This parameter is ignored when the device is set to step using the external hardware trigger.

Setting the List Cycle Count

LIST_CYCLE_COUNT (0x0B) - This register set the number of cycles to sweep through when the device is set up to start a sweep on trigger.

Setting the List Buffer Size

LIST_BUFFER_POINTS (0x0C) - This register sets the size of the list buffer to sweep/step through.

Populating the List Buffer

LIST_BUFFER_WRITE (0x0D) - This register enters the frequencies into the buffer that is in memory. The maximum number of frequency points is 2048. To set the device into write mode, 0x0000000000 must be written first. This will indicate to the device that to save successive writes into the buffer incrementally beginning at index 0. When all the frequencies are entered, writing 0xFFFFFFFFFF to the buffer will terminate the write process and the list buffer size automatically assigned as the last index. The size can be override using register 0x0C.

Transferring the List Buffer to and from EEPROM

LIST_BUF_MEM_XFER (0x0E) - This register transfers the buffer in RAM memory to the EEPROM or vise versa. Writing a “0” will transfer the list from RAM to EEPROM, and “1” transfers it from EEPROM to RAM.

Software Trigger

LIST_SOFT_TRIGGER (0x0F) - This register trigger the start or stop of a sweep when the device is configured via register 0x06 for software triggering.

Setting the RF Frequency

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

Setting the RF Power

RF_LEVEL (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; bits 14 down to 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 Level Feature

AUTO_LEVEL_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_DISABLE (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

CHANNEL_STANDBY (0x15) – Writing 0x01 to this register will power-down the analog/RF circuitry of the channel. 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 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) byte 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_VALUE (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

STORE_DEFAULT_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

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

QUERYING THE SC5506B: WRITING TO REQUEST REGISTERS

The registers to read data back from the device (such as device status) are accessed through the **sc5506b_reg_read** function. The function and parameter format for this command is **sc5506b_reg_read(dev_hand, reg_byte, instruct_word,*data_out)**. Any instructions in addition to the register call is placed into "instructWord", and data obtained from the device is returned via the pointer value data_out. The set of request registers are shown in Table 6.

Table 6. 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]							
GET_RF_PARAMETERS	0x25	[7:0]					RF parameter to get[0-9]			
GET_DEVICE_INFO	0x26	[7:0]					Device info to get [0-5]			
GET_LIST_BUFFER	0x27	[7:0]	EEPROM Address [7:0]							
		[15:8]	EEPROM Address [15:8]							
GET_ALC_DAC_VALUE	0x39	[7:0]	Open	Open	Open	Open	Open	Open	Open	Open

To read from the device using native USB transfers instead of the **sc5506b_reg_read** function requires two operations. First, a write transfer is made to the device **ENPOINT_OUT** to tell the device what data needs to be read back. Then, a read transfer is made from **ENDPOINT_IN** to obtain the data. The number of valid bytes returned varies from 1 to 5 bytes. See the register details below.

Reading the Device Temperature

GET_TEMPERATURE (0x17) - Data returned by this register is 4 bytes long that needs to be type-casted correctly to floating point representation to obtain the temperature in degrees Celsius. It is not recommended to read the temperature too frequently, especially once the SC5506B 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 SC5506B stabilizes in temperature, repeated readings will likely differ by as little as 0.25 ^{O}C over extended periods of time. Given that the gain-to-temperature coefficient is on the order of less than -0.01 dB/^{O}C, gain changes between readings will be negligible.

Reading the Device Status

GET_DEVICE_STATUS (0x18) - This register, summarized in Table 7, returns the device status information such as phase lock status of the PLL, current reference settings, etc. Data is contained in the first four bytes.

Table 7. Description of the status data bits.

Bit	Description
[31]	10 MHz TCXO Reference PLL Status
[30]	100 MHz VCXO Reference PLL Status
[29]	Channel 2 fine PLL status
[28]	Channel 2 coarse PLL status
[27]	Channel 2 sum PLL status
[26]	Channel 1 fine PLL status
[25]	Channel 1 coarse PLL status
[24]	Channel 1 sum PLL status
[23]	Current list mode channel selected
[22]	Ch2 auto level disabled
[21]	Ch2 auto level disabled
[20]	Ch2 ALC disabled
[19]	Ch1 ALC disabled
[18]	Ch2 output enabled
[17]	Ch1 output enabled
[16]	Ch2 RF mode
[15]	Ch1 RF mode
[14]	Ch2 Standby status
[13]	Ch1 Standby status
[12]	List mode running status (cycle started)
[11]	Device accessed
[10]	External reference detected
[9]	Reference out enable
[8]	Reference lock enable
[7]	Out on Step/ Out on cycle
[6]	Trig Out Enable
[5]	Return to Start/Stop at End
[4]	Trig to Step/Trig to Start-Stop
[3]	Hard Trig/Soft Trig
[2]	Triangular/Saw tooth
[1]	High-To-Low or Low to High
[0]	Start-Stop-Step Sweep Mode or List

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 32768. A single byte (first byte) is returned.

Reading the Current RF Parameters

GET_RF_PARAMETERS (0x25) – There are 9 different parameters to return. Data is returned in 5 bytes, however not every parameter has 5 bytes of valid data. The following details the data returned and its valid bytes.

Table 8 RF parameters

Parameter	Description	Number of Valid bytes
0	Ch1 RF frequency	First 5 Bytes, convert to ULONG64 in Hz
1	Ch2 RF frequency	First 5 Bytes, convert to ULONG64 in Hz
2	List start frequency	First 5 Bytes, convert to ULONG64 in Hz
3	List stop frequency	First 5 Bytes, convert to ULONG64 in Hz
4	List step frequency	First 5 Bytes, convert to ULONG64 in Hz
5	List dwell time	First 4 Bytes, convert to U32 in 0.5ms
6	List cycles	First 4 Bytes, convert to U32
7	List buffer point	First 4 Bytes, convert to U32
8	Ch1 RF level	First 4 Bytes, typecast to float
9	Ch2 RF level	First 4 Bytes, typecast to float

Reading the Device Information

GET_DEVICE_INFO (0x26) Device information such as the product serial number, firmware version, and hardware version can be obtained from querying the register with the required information parameter. Table 9 details the returned contents of each information parameter.

Table 9 Device Information

Parameter	Description	Number of Valid bytes
0	Product Serial Number	First 4 Bytes, convert to U32
1	Hardware revision	First 4 Bytes, typecast to float
2	Firmware revision	First 4 Bytes, typecast to float
3	Manufacturing date	First 4 Bytes : First Byte – Year (eg 15) : Second – Month :Third – Day :Fourth – hour

Reading the LIST BUFFER Value

GET_LIST_BUFFER (0x27) The register reads the frequency value at the requested address. Data is returned in the first 5 bytes and needs to be converted to U64 word in Hz.

Reading the RF ALC DAC Value

GET_ALC_DAC_VALUE (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 2 bytes and only the first 14 bits contain valid data.

CALIBRATION EEPROM MAP

Table 10. Calibration EEPROM map.

EEPROM ADDRESS (HEX)	NUMBER OF DATA POINTS	TYPE	DESCRIPTION
0	1	U32	Factory 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

USB SOFTWARE API LIBRARY FUNCTIONS

SignalCore's philosophy is to provide products to our customers whose lower hardware functions are easily accessible. For experienced users who wish to use direct, low-level control of frequency and gain settings, having the ability to access the registers directly is a necessity. However, others may wish for simpler product integration using higher level function libraries and not having to program registers directly. The functions provided in the USB SC5506B API dynamic linked library (sc5506b.dll) or LabVIEW library are:

- **sc5506b_search_devices**
- **sc5506b_open_device**
- **sc5506b_close_device**
- **sc5506b_reg_write**
- **sc5506b_reg_read**
- **sc5506b_init_device**
- **sc5506b_SetStandby**
- **sc5506b_set_frequency**
- **sc5506b_set_rf_mode**
- **sc5506b_list_mode_config**
- **sc5506b_list_start_frequency**
- **sc5506b_list_stop_frequency**
- **sc5506b_list_step_frequency**
- **sc5506b_list_dwell_time**
- **sc5506b_list_cycle_count**
- **sc5506b_list_buffer_points**
- **sc5506b_list_buffer_write**
- **sc5506b_list_buffer_transfer**
- **sc5506b_list_soft_trigger**
- **sc5506b_set_power_level**
- **sc5506b_set_rf_output**
- **sc5506b_set_alc_mode**
- **sc5506b_set_channel_standby**
- **sc5506b_set_out_level_disable**
- **sc5506b_set_clock_reference**
- **sc5506b_set_reference_dac**
- **sc5506b_wrtie_user_memory**
- **sc5506b_store_default_state**
- **sc5506b_set_alc_dac**
- **sc5506b_read_user_memory**
- **sc5506b_get_rf_parameters**
- **sc5506b_get_temperature**

- **sc5506b_get_device_status**
- **sc5506b_get_device-info**
- **sc5506b_list_buffer_read**
- **sc5506b_get_alc_dac**

Each of these functions is described in more detail on the following pages. Example code written in C/C++ is located in the *Win**Driver**src* directory 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, *sc5506b.h*. These constants and types are useful not only as an include for developing applications using the USB SC5506B API, but also for writing device drivers independent of those provided by SignalCore.

Constants Definitions

```
// Parameters for storing data in the onboard EEPROM
#define USEREEPROMSIZE           32768    // bytes

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

// Attenuator assignments
#define CH1ATTEN0                    0
#define CH1ATTEN1                    1
#define CH2ATTEN0                    2
#define CH2ATTEN1                    3


// Define error codes
#define SUCCESS                      0
#define USBDEVICEERROR              -1
#define USBTRANSFERERROR            -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 INITIALIZE                 0x01   // init device
#define SET_SYS_ACTIVE             0x02   // set the active led indicator on/off
#define RF_MODE                    0x05   // sets the RF Mode: Single, Sweep/list
#define LIST_MODE_CONFIG           0x06   // Configure the sweep/list behavior
#define LIST_START_FREQ            0x07   // Set the list start frequency
#define LIST_STOP_FREQ             0x08   // Set the list stop frequency
#define LIST_STEP_FREQ             0x09   // Set the step frequency
#define LIST_DWELL_TIME            0x0A   // The step time interval in 500 microseconds
#define LIST_CYCLE_COUNT           0x0B   // number of cycles to run the sweep or list
#define LIST_BUFFER_POINTS         0x0C   // Points to step though in the list buffer
#define LIST_BUFFER_WRITE          0x0D   // write the frequency to the list buffer in RAM
#define LIST_BUF_MEM_XFER          0x0E   // transfer the list between RAM and EEPROM
#define LIST_SOFT_TRIGGER          0x0F   // Soft trigger.

#define RF_FREQUENCY               0x10   // set the frequency
#define RF_LEVEL                   0x11   // Set Power of LO1
#define RF_ENABLE                  0x12   // Enable RF output
#define AUTO_LEVEL_DISABLE         0x13   // Adjust the signal phase
#define RF_ALC_DISABLE             0x14   // Disable the ALC close loop mode
#define CHANNEL_STANDBY            0x15   // puts the channel into standby mode
#define REFERENCE_SETTING          0x16   // Reference Settings
#define GET_TEMPERATURE            0x17   // Device temperature
#define GET_DEVICE_STATUS          0x18   // load the board status
#define SERIAL_OUT_BUFFER          0x19   // Read the data from the SPI output buffer
#define USER_EEPROM_READ           0x1A   // byte read from user EEPROM space
#define USER_EEPROM_WRITE          0x1B   // user EEPROM write
#define REFERENCE_DAC_VALUE        0x1D   // set reference DAC

#define STORE_DEFAULT_STATE        0x23   // store the new default state
#define SET_ALC_DAC_VALUE          0x24   // write the ALC DAC value
#define GET_RF_PARAMETERS          0x25   // Get the current Frequency
```

```
#define GET_DEVICE_INFO           0x26   // load the device info
#define GET_LIST_BUFFER           0x27   // read the list buffer
#define GET_ALC_DAC_VALUE         0x39   // read the current alc DAC value
```

Type Definitions

```
typedef struct device_info_t
{
	unsigned int product_serial_number;
	float hardware_revision;
	float firmware_revision;
	struct date
	{
		unsigned char year;
		unsigned char month;
		unsigned char day;
		unsigned char hour;
	} man_date;
}	device_info_t;


typedef struct list_mode_t
{
	unsigned char sss_mode;
	unsigned char sweep_dir;
	unsigned char tri_waveform;
	unsigned char hw_trigger;
	unsigned char step_on_hw_trig;
	unsigned char return_to_start;
	unsigned char trig_out_enable;
	unsigned char trig_out_on_cycle
} list_mode_t;

typedef struct
{
	unsigned char ch1_sum_pll_ld;
	unsigned char ch1_crs_pll_ld;
	unsigned char ch1_fine_pll_ld;
	unsigned char ch2_sum_pll_ld;
	unsigned char ch2_crs_pll_ld;
	unsigned char ch2_fine_pll_ld;
	unsigned char ref_100_pll_ld;
	unsigned char ref_10_pll_ld;
 } pll_status_t;

typedef struct
{
	unsigned char ref_lock_enable;
	unsigned char ref_out_enable;
	unsigned char ext_ref_detected;
	unsigned char device_accessed;
	unsigned char list_mode_running;
	unsigned char ch1_standby_enable;
	unsigned char ch2_standby_enable;
	unsigned char ch1_rf_mode;
	unsigned char ch2_rf_mode;
	unsigned char ch1_out_enable;
	unsigned char ch2_out_enable;
	unsigned char ch1_alc_disable;
```

```
	unsigned char ch2_alc_disable;
	unsigned char ch1_auto_level_disable;
	unsigned char ch2_auto_level_disable;
} operate_status_t;

typedef struct
{
	list_mode_t   list_mode;
	operate_status_t operate_status;
	pll_status_t pll_status;
} device_status_t;

typedef struct device_rf_params_t
{
	unsigned long long int ch1_rf_freq;
	unsigned long long int ch2_rf_freq;
	float ch1_rf_level;
	float ch2_rf_level;
	unsigned long long int list_start_freq;
	unsigned long long int list_stop_freq;
	unsigned long long int list_step_freq;
	unsigned int list_dwell_time;
	unsigned int list_cycles;
	unsigned int list_buffer_points;
} device_rf_params_t;
```

Function Definitions and Usage

The functions listed below are found in the **sc5506b.dll** dynamic linked library for the Windows™ operating system. These functions are also provided in the LabView library, **sc5506b.llb**. The LabView functions contain context help (Cntrl-H) to help with the input and output parameters.

Function: **sc5506b_search_devices**

Definition: int sc5506b_search_devices (char **serial_number_list, int *count)

Output: char **serial_number_list (2-D array pointer list)

Description: **sc5506b_search_devices** searches for SignalCore SC5506B devices connected to the host computer and outputs the number of devices found, and it also populates the char array with their serial numbers. The user can use this information to open specific device(s) based on their unique serial numbers. See **sc5506b_open_device** function on how to open a device.

Function: **sc5506b_open_device**

Definition: int sc5506b_ open_device (char *dev_serial_num, HANDLE* dev_handle)

Input: char * dev_serial_num (serial number string)

Return: * dev_handle (device handle pointer)

Description: **sc5506b_open_device** 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: **sc5506b_close_device**

Definition: int sc5506b_close_device(HANDLE dev_handle)

Input: HANDLE dev_handle(handle to the device to be closed)

Description: **sc5506b_close_device** 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 and close the device:

```
// Declaring
HANDLE dev_handle; //device handle
int num_of_devices; // the number of device types found
char **device_list;  // 2D to hold serial numbers of the devices found
int status; // status reporting of functions

device_list = (char**)malloc(sizeof(char*)*MAXDEVICES); // MAXDEVICES serial numbers
to search
for (i=0;i<MAXDEVICES; i++)
	device_list [i] = (char*)malloc(sizeof(char)*SCI_SN_LENGTH); // SCI SN has 8
char
status = sc5506b_search_devices(device_list, num_of_devices); //searches for SCI for
device type
if (num_of_devices == 0)
{
	printf("No signal core devices found or cannot not obtain serial numbers\n");
	for(i = 0; i<MAXDEVICES;i++) free(device_list[i]);
	free(device_list);
	return 1;
}
printf("\n There are %d SignalCore %s USB devices found. \n \n", num_of_devices,
SCI_PRODUCT_NAME);
	i = 0;
	while ( i < num_of_devices)
	{
		printf("	Device %d has Serial Number: %s \n", i+1, device_list[i]);
		i++;
	}
//** sc5506b_open_device, open device 0
status = sc5506b_open_device(device_list[0], &dev_handle);
// Free memory
	for(i = 0; i<MAXDEVICES;i++) free(device_list[i]);
	free(device_list); // Done with the device_list
 //
// Do something with the device
// Close the device
status = sc5506b_close_device(dev_handle);
```

Function: **sc5506b_reg_write**

Definition: int sc5506b_reg_write(HANDLE dev_handle, unsigned char reg_byte,
unsigned long long int instruct_word)

Input: HANDLE dev_handle (handle to the opened device)
unsigned char reg_byte (register address)
unsigned long long int instructWord (the data for the register)

Description: **sc5506b_reg_write** writes the instruct_word data to the register specified by the reg_byte.

Example: To set the power level to 2.00 dBm:

```
int status = sc5506b_RegWrite(devHandle, RF_POWER, 200);
```

Function: **sc5506b_reg_read**

Definition: int sc5506b_reg_read(HANDLE dev_handle,
unsigned char reg_byte,
unsigned long long int instruct_word,
unsigned long long int *received_word)

Input: HANDLE dev_handle (handle to the opened device)
unsigned char reg_byte (The address byte of the register to write to)
unsigned long long int instruct_word (the data for the register)
unsigned long long int *received_word (data to be received)

Description: **sc5506b_reg_read** reads the data requested by the instruct_word to the register specified by the reg_byte. See the register map on Table 6 for more information.

Example: To read the status of the device:

```
unsigned int device_status;
unsigned long long int tmp_u64;

int status = sc5506b_reg_read(dev_handle, GET_DEVICE_STATUS,0x00,&tmp_u64);
device_status = (unsigned int)tmp_u64;
```

Function: **sc5506b_init_device**

Definition: int sc5506b_init_device(HANDLE dev_handle, unsigned char mode)

Input: HANDLE dev_handle (handle to the opened device)
Unsigned char mode (Set the mode of initialization)

Description: sc5506b_init_device 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: **sc5506b_SetFrequency**

Definition: int sc5506b_SetFrequency(HANDLE dev_handle,
unsigned char channel,
unsigned long long int frequency)

Input: HANDLE dev_handle (handle to the opened device)
unsigned char channel (channel name of target frequency change)
unsigned long long int frequency (frequency in Hz)
Description: **sc5506b_SetFrequency** sets the channel RF frequency.

Function: **sc5506b_set_rf_mode**

Definition: sc5506b_set_rf_mode(HANDLE dev_handle,
Unsigned char channel,
unsigned char rf_mode)

Input: HANDLE dev_handle (handle to the opened device)
unsigned char channel (channel to apply mode)
unsigned char rf_mode (set RF mode of the channels)

Description: sc5506b_set_rf_mode sets the channels to fixed tone or sweep.

Function: **sc5506b_list_mode_config**

Definition: int sc5506b_list_mode_config(HANDLE dev_handle,
const list_mode_t *list_mode)

Input: HANDLE dev_handle (handle to the opened device)
const list_mode_t *list_mode (list mode setup)

Description: sc5506b_ListModeConfig configures the list mode behavior. See the document for more information on the modeConfig structure.

Function: **sc5506b_list_start_freq**

Definition: int sc5506b_list_start_freq(HANDLE dev_handle,
unsigned long long int freq)

Input: HANDLE dev_handle (handle to the opened device)
unsigned long long int freq (frequency in Hz)

Description: sc5506b_ list_start_freq sets the sweep start frequency.

Function: **sc5506b_list_stop_freq**

Definition: int sc5506b_list_stop_freq(HANDLE dev_handle,
unsigned long long int freq)

Input: HANDLE dev_handle (handle to the opened device)
unsigned long long int freq (frequency in Hz)

Description: sc5506b_ list_stop_freq sets the sweep stop frequency.

Function: **sc5506b_list_step_freq**

Definition: int sc5506b_list_step_freq(HANDLE dev_handle,
unsigned long long int freq)

Input: HANDLE dev_handle (handle to the opened device)

unsigned long long int freq (frequency in Hz)

Description: sc5506b_list_step_freq sets the sweep step frequency.

Function: **sc5506b_list_dwell_time**

Definition: int sc5506b_list_dwell_time(HANDLE dev_handle,
unsigned int dwell_time)

Input: HANDLE dev_handle (handle to the opened device)
unsigned int dwell_time (Time in 500 µs increments)

Description: sc5506b_list_dwell_time stet the sweep/list dwell time at each frequency point. Dwell time is in 500 µs increments (1 = 500 µs, 2 = 1 ms, etc.).

Function: **sc5506b_list_cycle_count**

Definition: int sc5506b_list_cycle_count(HANDLE dev_handle,
unsigned int cycle_count)

Input: HANDLE dev_handle (handle to the opened device)
unsigned int cycle_count (number of cycles)

Description: sc5506b_list_cycle_count sets the number of sweep cycles to perform before stopping. To repeat the sweep continuously, set the value to 0.

Function: **sc5506b_list_buffer_points**

Definition: int sc5506b_list_buffer_points(HANDLE dev_handle,
unsigned int list_points)

Input: HANDLE dev_handle (handle to the opened device)
unsigned int list_points (number of points of the list buffer)

Description: sc5506b_list_buffer_points sets the number of list points in the list buffer to sweep or step through. The list points must be smaller or equal to the points in the list buffer.

Function: **sc5506b_list_buffer_write**

Definition: int sc5506b_list_buffer_write(HANDLE dev_handle,
unsigned long long int freq)

Input: HANDLE dev_handle (handle to the opened device)
unsigned long long int freq (frequency in Hz)

Description: sc5506b_list_buffer_write writes the frequency buffer sequentially. If frequency value = 0, the buffer counter is reset to position 0 and subsequent writes will increment the counter. Writing 0xFFFFFFFFF will terminate the sequential write operation and sets the list_buffer_points variable to the last counter value.

Function: **sc5506b_list_buffer_transfer**

Definition: int sc5506b_list_buffer_transfer(HANDLE dev_handle,
unsigned char transfer_mode)

Input: HANDLE dev_handle (handle to the opened device)

unsigned char transferMode (transfer to EEPROM or RAM)

Description: sc5506b_list_buffer_transfer transfers the frequency list buffer from RAM to EEPROM or vice versa.

Function: **sc5506b_list_soft_trigger**

Definition: int sc5506b_list_soft_trigger(HANDLE dev_handle)

Input: HANDLE dev_handle (handle to the opened device)

Description: sc5506b_list_soft_trigger triggers the device when it is configured for list mode and soft trigger is selected as the trigger source.

Function: **sc5506b_set_power_level**

Definition: int sc5506b_set_power_level(HANDLE dev_handle,
unsigned char channel,
float powerLevel)

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

Description: **sc5506b_set_power_level** sets the value of the desired output power level for the channel.

Function: **sc5506b_set_rf_output**

Definition: int sc5506b_set_rf_output(HANDLE dev_handle,
unsigned char channel,
unsigned char mode)

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

Description: **sc5506b_SetRfOut** enables or disables the RF output on a channel.

Function: **sc5506b_set_alc_mode**

Definition: int sc5506b_set_alc_mode(HANDLE dev_handle,
unsigned char channel,
unsigned char mode)

Input: HANDLE dev_handle (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: **sc5506b_set_alc_mode** sets the ALC loop to closed or open loop operation for a channel.

Function: **sc5506b_set_channel_standby**

Definition: int sc5506b_set_channel_standby(HANDLE dev_handle,
unsigned char channel,
unsigned char enable)

Input: HANDLE dev_handle (handle to the opened device)
unsigned char channel (channel name to put into standby)
bool enable Set to true (1) to set device in standby mode

Description: **sc5506b_set_channel_standby** puts a channel in standby mode where the power to the analog circuits for that channel are disabled, conserving power.

Function: **sc5506b_set_auto_level_disable**

Definition: int sc5506b_set_auto_level_disable(HANDLE dev_handle,
unsigned char channel,
unsigned char mode)

Input: HANDLE dev_handle (handle to the opened device)
unsigned int channel (channel name to disable auto-level)
bool mode (defines when to disable auto-level)

Description: **sc5506b_ set_auto_level_disable** 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 **sc5506b_set_power_level**. If the power remains the same for the next tuned frequency, this should not be disabled.

Function: **sc5506b_set_clock_reference**

Definition: int sc5506b_set_clock_reference(HANDLE dev_handle,
bool ext_lock_enable,
bool ref_out_enable)

Input: HANDLE dev_handle (handle to the opened device)
bool ext_lock_enable (enables phase locking to an external source)
bool ref_out_enable (enables the internal clock to be exported on the “ref out” port)

Description: **sc5506b_set_clock_reference** configures the reference clock behavior of the device**.**

Function: **sc5506b_set_reference_dac**

Definition: int sc5506b_set_reference_dac(HANDLE dev_handle,
unsigned int dac_value)

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

Description: sc5506b_SetReferenceDac set 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: **sc5506b_write_user_memory**

Definition: int sc5506b_write_user_memory(HANDLE dev_handle,
unsigned short mem_add,
unsigned char data)

Input: HANDLE dev_handle (handle to the opened device)
unsigned int mem_add (memory address to write to)
unsigned char data (byte to be written to the address)

Description: **sc5506b_ write_user_memory** writes one byte of data to the memory address specified.

Function: **sc5506b_store_default_state**

Definition: int sc5506b_ store_default_state (HANDLE dev_handle, unsigned char channel)

Input: HANDLE dev_handle (handle to the opened device)
Unsigned char channel (channel)

Description: **sc5506b_ store_default_state** stores the current state of the channel of the device as the default power-up state.

Function: **sc5506b_set_alc_dac**

Definition: int sc5506b_set_alc_dac(HANDLE dev_handle,
unsigned char channel,
unsigned short dac_value)

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

Description: **sc5506b_ set_alc_dac** writes a value to the ALC DAC to control the RF output level for a channel.

Function: **sc5506b_read_user_memory**

Definition: int sc5506b_read_user_memory(HANDLE dev_handle,
unsigned short mem_add,
unsigned char *data)

Input: HANDLE dev_handle (handle to the opened device)
unsigned short mem_add (EEPROM memory address)

Output: unsigned char *data (the read back byte data)

Description: **sc5506b_ReadUserEeprom** reads back a byte from a specific memory address of the user EEPROM.

Function: **sc5506b_get_device_status**

Definition: int sc5506b_ get_device_status(HANDLE dev_handle,
deviceStatus_t *deviceStatus)

Input: HANDLE dev_handle (handle to the opened device)

Output: deviceStatus_t *deviceStatus (deviceStatus struct)

Description: **sc5506b_ get_device_status** retrieves the status of the device such as phase lock status and current device settings.

Example: Code showing how to use function:

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

int status = sc5506b_get_device_status(dev_handle, dev_status);

if(dev_status->pll_status.ref_10_pll_ld)
printf("The 10 MHz is phase-locked \n");
else
printf("The 10 MHz is not phase-locked \n");

free(dev_status);
```

Function: **sc5506b_get_temperature**

Definition: int sc5506b_ get_temperature (HANDLE dev_handle,
float *temp)

Input: HANDLE dev_handle (handle to the opened device)

Output: float *temp (temperature in degrees Celsius)

Description: **sc5506b_ get_temperature** retrieves the internal temperature of the device.

Function: **sc5506b_get_rf_parameters**

Definition: int sc5506b_get_rf_parameters(sc5506b_device_handle_t*dev_handle,
device_rf_params_t *device_rf_params)

Input: sc5506b_device_handle_t*dev_handle (handle to the opened device)
device_rf_params_t *device_rf_params (rf parameters)
Description: sc5506b_get_rf_parameters gets the current RF parameters such as RF1 frequency, RF2 frequency, and sweep start frequency, etc.

Function: **sc5506b_get_device_info**
Definition: int sc5506b_get_device_info(sc5506b_device_handle_t*dev_handle,
device_info_t *device_info)
Input: sc5506b_device_handle_t*dev_handle (handle to the opened device)
Output: device_info_t *device_info (device information)
Description: sc5506b_get_device_info obtains the device information such as serial number, hardware revision, firmware revision, and manufactured date.

Function: **sc5506b_list_buffer_read**
Definition: int sc5506b_list_buffer_read(sc5506b_device_handle_t*dev_handle,
unsigned int point, unsigned long long int *freq)
Input: sc5506b_device_handle_t*dev_handle (handle to the opened device)
unsigned int point (buffer point)
Output: unsigned long long int *freq (frequency)
Description: sc5506b_list_buffer_read reads the frequency at a point offset.

Function: **sc5506b_get_alc_dac**
Definition: int sc5506b_get_alc_dac(HANDLE dev_handle,
unsigned char channel,
unsigned int *dac_value)
Input: HANDLE dev_handle (handle to the opened device)
unsigned char channel (channel name of target ALC DAC)
Output: unsigned char *dacValue (the read back byte data)
Description: **sc5506b_get_alc_dac** reads back the current ALC DAC value.

SERIAL PERIPHERAL INTERFACE (SPI)

The SPI interface is implemented using 8-bit length physical buffers for both the input and output, hence they need to be read and cleared before consecutive bytes can be transferred to and from them. The process of clearing the SPI buffer and decisively moving it into the appropriate register takes CPU time, so a time delay is required between consecutive bytes written to or read from the device by the host. The chip-select pin ($\overline{CS}$) must be asserted low before data is clocked in or out of the product. Pin $\overline{CS}$ must be asserted for the entire duration of a register transfer.

Once a full transfer has been received, the device will proceed to process the command and de-assert low the SRDY pin. The status of this pin may be monitored by the host because when it is de-asserted low, the device will ignore any incoming data. The device SPI is ready when the previous command is fully processed and SRDY pin is re-asserted high. It is important that the host either monitors the SRDY pin or waits for 750 us between register writes.

There are 2 SPI modes; 0 and 1. The default mode is 1, where data is clocked in and out of the device on the falling edge of the clock signal. In mode 0, data is clocked in and out on the rising edge. To select mode 0, pin 18 of the digital I/O connector must be pulled low to ground as the device is powered on or as the reset line (pin 14) is toggle low-high. If pin 18 is pulled high or left unconnected, mode 1 (default) is select.

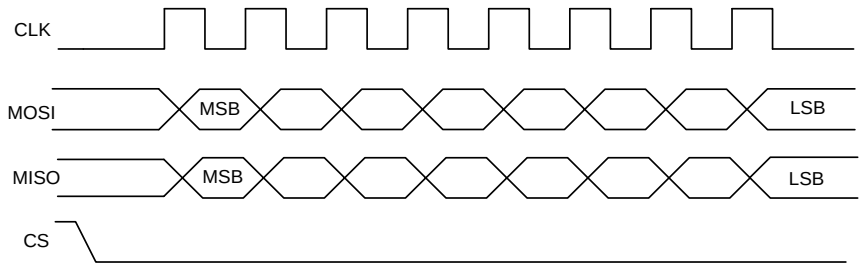


Figure 3. SPI Mode 1 shown.

Register writes are accomplished in a single write operation. Register buffer lengths vary depending on the register; they vary in lengths of 2 to 6 bytes, with the first byte being the register address, followed by the data associated with that register. The ($\overline{CS}$) pin must be asserted low for a minimum period of 1 μs (T_S, see *Figure 4*) before data is clocked in, and must remain low for the entire register write. The clock rate may be as high as 5.0 MHz (T_C = 0.2 μs), however if the external SPI signals do not have sufficient integrity due trace issues, the rate should be lowered.

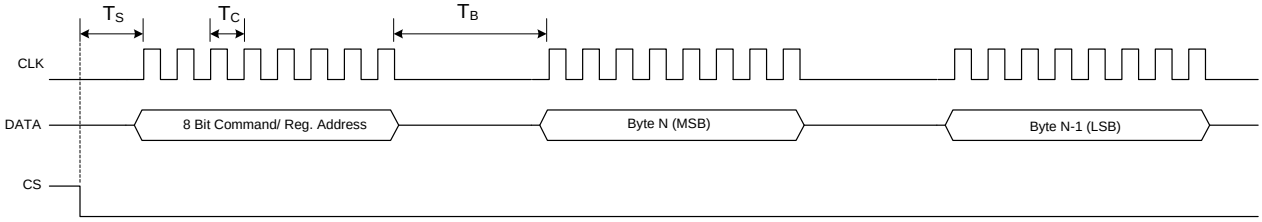


Figure 4 SPI timing.

As mentioned above, the SPI architecture limits the byte rate due to the fact that after every byte transfer the input and output SPI buffers need to be cleared and loaded respectively by the device SPI engine. Data is transferred between the buffers and the internal registers. The time required to perform this task is indicated by T_B, which is the time interval between the end of one byte transfer and the beginning of another. The recommended minimum time delay for T_B is $1\ \mu s$. The number of bytes transferred depends on the register. It is important that the correct number of bytes is transferred for the associated device register, because once the first byte (MSB) containing the device register address is received, the device will wait for the desired number of associated data bytes. The device will hang if an insufficient number of bytes are written to the register. In order to clear the hung condition, the device will need an external hard reset. The time required to process a command is also dependent on the command itself. Measured times for command completions are between $50\ \mu s$ to $600\ \mu s$ after reception. To change the frequency with auto leveling turned on requires the most computational time. The computational time to change frequencies is approximately $250\ \mu s$ and to computational time to change power level is approximately $350\ \mu s$.

Writing the SPI Bus

The SPI transfer size (in bytes) depends on the register being targeted. The MSB byte is the command register address as noted in the *Configuration Registers* section. The subsequent bytes contain the data associated with the register. As data from the host is being transferred to the device via the SDI (MOSI) line, data present on its SPI output buffer is simultaneously transferred back, MSB first, via the SDO (MISO) line. The data return is invalid for most transfers except for those registers querying for data from the device. See Reading the SPI Bus section below for more information on retrieving data from the device. *Figure 5* shows the contents of a single 3 byte SPI command written to the device. The *Querying the SC5506B: Writing to Request Registers* section provides information on the number of data bytes and their contents for an associated register. There is a minimum of 1 data byte for each register even if the data contents are “zeros”.

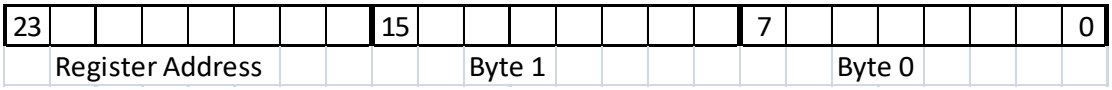


Figure 5. Single 3 byte transfer buffer.

Reading the SPI Bus

Data is simultaneously read back during a SPI transfer cycle. Requested data from a prior command is available on the device SPI output buffers, and these are transferred back to the user host via the SDO pin. To obtain valid requested data would require querying the SERIAL_OUT_BUFFER, which requires 6 bytes of clock cycles; 1 byte for the device register (0x19) and 5 empty bytes (MOSI) to clock out the returned data (MISO). An example of reading the device status from the device is shown in *Figure 6.*

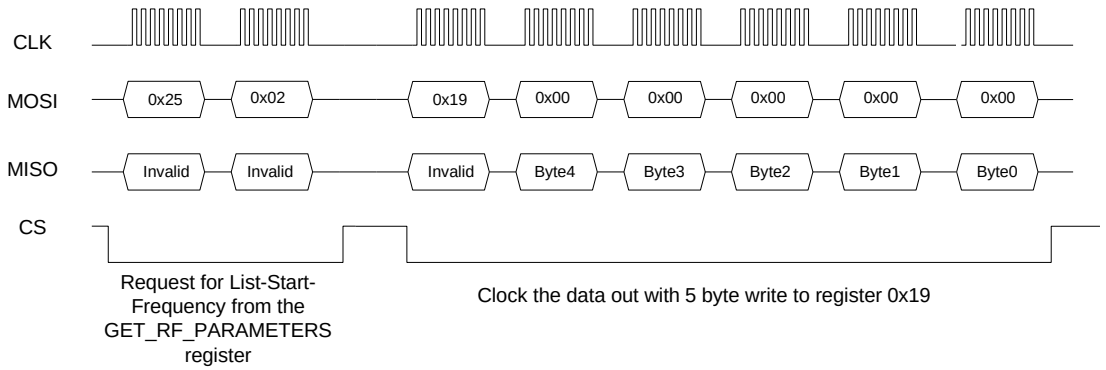


Figure 6. Reading queried data.

In the above example, valid data is present in the first 5 bytes - byte 4 down to byte 0, where byte 4 is the most significant byte. Table 11 shows the valid data bytes associated with each of the querying registers, while Table 12 shows the valid bytes associated with the requested contents of the register 0x25.

Table 11. Valid returned data bytes.

Register Name (Address)	Register Code (Hex)	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
GET_TEMPERATURE	0x17	valid	valid	valid	valid	invalid
GET_DEVICE_STATUS	0x18	valid	valid	valid	valid	invalid
GET_RF_PARAMETERS	0x25	valid	valid	valid	valid	valid
GET_DEVICE_INFO	0x26	valid	valid	valid	valid	invalid
GET_LIST_BUFFER	0x27	valid	valid	valid	valid	valid
GET_ALC_DAC_VALUE	0x39	valid	valid	invalid	invalid	invalid

Table 12 Return Valid Data for GET_RF_PARAMETERS Register

Data Return Name	Data Sent	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
Ch1 RF Frequency	0x00	valid	valid	valid	valid	valid
Ch2 RF Frequency	0x01	valid	valid	valid	valid	valid
List Start Frequency	0x02	valid	valid	valid	valid	valid
List Stop Frequency	0x03	valid	valid	valid	valid	valid
List Step Frequency	0x04	valid	valid	valid	valid	valid
Sweep Dwell Time	0x05	valid	valid	valid	valid	invalid
Sweep Cycles	0x06	valid	valid	valid	valid	invalid
Sweep Buffer Points	0x07	valid	valid	valid	valid	invalid
Ch1 Power Level*	0x07	valid	valid	valid	valid	invalid
Ch1 Power Level*	0x08	valid	valid	valid	valid	invalid

While all return data for the GET_RF_PARAMETERS Register (0x25) are unsigned integers, the RF1 power level is returned as a 4-byte flattened float which needs to be re-casted back to a float using *(float*)&.

PROGRAMMING THE RS-232 INTERFACE

The RS-232 version of the SC5506B has a standard interface buffered by an RS-232 transceiver so that it may interface directly with many host devices, such as a desktop computer. The interface connector for RS-232 communication is labeled “Digital I/O” on the front of the panel. Refer to Table 2 for position and pin-out information. The device communication control set is provided in Table 13 below.

Table 13. RS-232 communication settings.

Baud rate	Rate of transmission. Pin 18 of the Digital IO connector selects the rate. By default if the pin is pulled high or open, the rate is set 56700 at power up or upon HW reset. When the pin is pulled low or grounding it, the rate is set to 115200 upon reset or power up.
Data bits	The number of bits in the data is fixed at 8.
Parity	Parity is 0 (zero).
Stop bits	1 stop bit.
Flow control	0 (zero) or none.

Writing to the Device via RS-232

It is important that all necessary bytes associated with any one register are fully sent. In other words, if a register requires a total of six bytes (address plus data) then all six bytes must be sent even though the last byte may be a null. The device, upon receiving the first register addressing byte, will wait for all the associated data bytes before acting on the register instruction. Failure to complete the register transmission will cause the device to behave erratically or hang. Information for writing to the configuration registers is provided in Table 5.

Reading from the Device via RS-232

To query information from the device, the query registers are addressed and data is returned. Valid returned data vary in length, which are dependent on the register call. However total returned data length (bytes 4 to 0) is the same as for SPI, that is 5 bytes long. Table 6 contains the query register information. As with the configuration registers, it is important that the data byte(s) associated with the query registers are sent even if they are nulls. The valid returned data length is also detailed in the “Querying the SC5506B: Writing to Request Registers” section. Returned valid data are detailed in Table 11 and Table 12.

RS-232 Windows™ API

The API for RS-232 control is provided only for the Windows operating system under the api\rs232\c directory of the installation path. All API functions are provided in the sc5506b_rs232.dll library and with the exception of 2 functions, namely sc5506b_search_devices and sc5506b_open_device, all functions are identical to those for USB communication. Please refer to the *Using the Application Programming Interface (API)* function descriptions and the sc5506b_rs232.h header file for proper usage. A C/C++ programming example is also provided under the examples subdirectory. For driver support of other operating systems, please contact SignalCore support.

RS-232 LabVIEW functions

LabVIEW function wrappers of the C/C++ API DLL are also provided for programming in the LabVIEW environment. To use these functions and have them appear on the *function palette*, copy the SignalCore/ and its sub-directories (containing the SC5506B_RS232 dir) into the instr.lib directory of the LabVIEW installation path. A RS-232 version of the soft-front-panel software is available in LabVIEW and may be compiled into an executable.

CALIBRATION & MAINTENANCE

The SC5506B is factory calibrated and ships with a certificate of calibration. SignalCore strongly recommends that the SC5506B 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.

SC5506B 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 SC5506B, 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 SC5506B.

The SC5506B 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 1.0.0	Initial release.
Rev 1.1.0	Change the open_device API function in its device handle return type
Rev 1.2.0	Added pin 8 to Table 2 as reset_b
Rev 1.2.1	Address deleted
Rev 1.2.2	Changed initialize 0x00 to reset the device but leave in current state, and 0x01 to reset the device to the default power-on state