SignalCore™
PRESERVING SIGNAL INTEGRITY

SC5511A

100 MHz to 20 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)**
SC5511A	✓	✓	✓	✓	✓	✓

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 SC5511A kit contains the following items:

Quantity	Item
1	SC5511A 20 GHz Signal Source
1	Software Installation USB Flash Drive (may be combined with other products onto a single drive)

Setting Up and Configuring the SC5511A

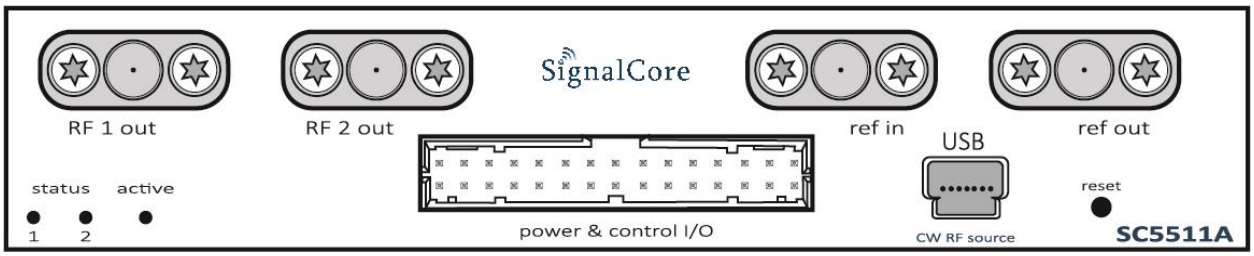

Figure 1 Front view of the SC5511A showing user I/O locations

The SC5511A 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.

Indicator LEDs

The SC5511A provides visual indication of important modes. There are three 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	Orange	Channel powered down or RF Output Disabled
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)

Power and Digital IO Connector

The SC5511A is powered through a rectangular connector from Samtec whose part number is TFM-115-01-L-D-RA. It also serves as the digital connector interface for RS232/SPI, trigger, and other digital signals. The pinout of this connector, viewed from the front of the module, is listed in **Table 2**.

Table 2. Pinout of the SC5511A Power and Control I/O connector.

Pin Number	SPI Function	RS-232 Function
24	MISO	TxD
28	–	CTS
27	MOSI	RxD
26	CS_b	RTS
25	SERIAL READY	–
30	CLK	CLK
16	SPI MODE	BAUD SELECT
14	Device Reset_B	
22	Unused Input	
21	Trigger in	
20	Trigger out	
19	RF1 PLL status	
17	Do not connect	
2,4,6	+Supply (+12V Typ)	
1, 3, 5, 11, 15, 23, 29	GND	
7, 8, 9, 10, 12, 13, 18	Not internally connected	

USB Connector

The SC5511A 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 3**.

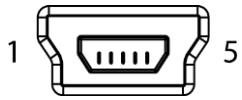

Table 3. Pinout of the SC5511A USB communication connector.

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)

Signal Connections

All signal connections (ports) on the SC5511A are SMA-type. Exercise caution when fastening cables to the signal connections. Over-tightening any connection can cause permanent damage to the device.

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.

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

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

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

THEORY AND OPERATION

Despite its small size, the SC5511A is an instrument-grade, high performance synthesizer with easy to program register-level control. It functions as a standard synthesized CW source with the added capability of a sweep/list mode that makes it ideal for applications ranging from automated test systems to telecommunication equipment to scientific research labs. Being small and modular, it is the ideal solution system integration applications that require a high performance RF source. In addition, it could be used as a general purpose lab source. **Figure 2** shows the block diagram of the device, and the following sub-sections provide details to its operation.

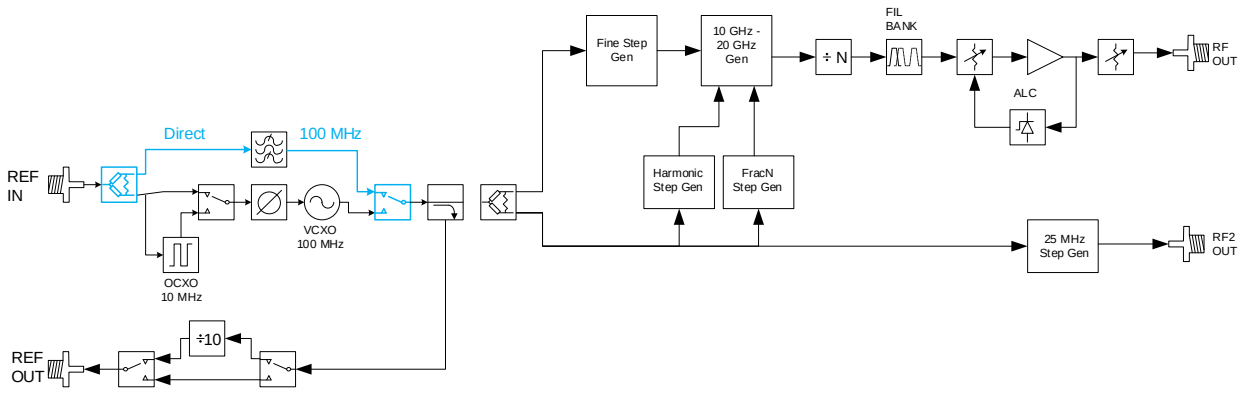

Figure 2. Block diagram of the SC5511A.

RF Generation

The SC5511A is a 100 MHz to 20 GHz low phase noise and low spur synthesizer, using a hybrid architecture comprising of phase lock, harmonic generation, and a DDS functions. Coarse tuning is accomplished by PLL and harmonic generators, while fine tuning is accomplished with the variable modulus DDS, providing exact frequency generation. Isolation between the internal oscillators, their mixed IF products, harmonics, and inter-modulation products is accomplished by internal EMI sealed cavities. A hybrid architecture with well-shielded cavities improves the overall phase noise performance and reduces the spurious signal content of this compact size frequency synthesizer. Signals are synthesized from an internal 10 MHz TCXO reference clock, or an external 10 MHz reference. The reference path indicated in blue is only available in products with hardware revision 16 or later. This path allows for an external 100 MHz, from another SC5511A as an example, to directly clock the synthesizer core, resulting in better phase stability between devices.

The SC5511A has 2 independent channels; channel 1 frequency range is from 100 MHz to 20 GHz with frequency resolution of 1 Hz, while the range on channel 2 is from 25 MHz to 3 GHz with 25 MHz step. Channel 1 has calibrated power level adjustment but this feature is not available on channel 2. Furthermore list and sweep modes are only available for channel 1. A typical use of the 2 channels is to drive LO ports of mixers for a dual stage downconverter where the first RF stage is tunable and the second IF stage is fixed.

Amplitude Control

The output level of the SC5511A is controlled through the automatic leveling control (ALC) circuitry. The ALC can operate in close or open loop. The advantages of the close loop over the open loop operation are that the power levels are more stable and accurate. The disadvantage of the close loop is that it increases the AM noise of the carrier sideband. Although this AM noise is typically lower than the phase noise, it may have impact on some applications. In such applications, it is best to operate the ALC in open loop. Fine amplitude adjusts can be made by changing the ALC DAC value.

Computational Time

The ALC control is accomplished by controlling the ALC DAC and the output step attenuator. The settings of these two components are dynamically calculated based on the level required and a large set of calibration values. Similarly, to change frequency would require four phase lock loops to be programmed and their settings are dynamically calculated based on a set of calibration values. The computational effort to compute these settings is great. Typical computational time and setting up for frequency change is approximately 250 µs, while it is about 350 µs to compute and set up the ALC.

For faster frequency changes, especially for sweeps less than a couple of 100 MHz, it is recommended that the automatic leveling of the power be turned off. This will prevent the SC5511A from having to compute a fresh set of the ALC parameters at each frequency point. Typically the un-calibrated power level does not change by more than a couple of dB over 100 MHz range. See device register 0x14 for details on turning on and off this automatic leveling feature.

Internal EEPROM

The SC5511A contains an EEPROM whose memory space is divided into calibration and operating data spaces. The calibration data space contains SC5511A device information such as serial number, hardware revision, firmware revision, and production date. In addition, this space holds the calibration data for frequency tuning and amplitude control. The operating data space contains the default startup configuration of the device such as the single fixed tone mode frequency and sweep/list mode operation. It also holds the list mode configuration parameters such as sweep behavior (saw or triangular waveform), software or hardware trigger, start/stop/step frequencies, dwell time, sweep/list cycles, etc. Space is allocated for 2048 frequency points that the user may choose to store for list mode operation. The internal EEPROM is not directly accessible for users to store data.

Modes of RF Generation

The SC5511A has both single fixed tone and list mode operation for channel 1. 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 sawtooth 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 to 500 μs. The amplitude computational time alone is close to 350 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 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 a cycle, the frequency may be set to end on the last frequency point or return back the starting point. This 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 pin 21, the hardware trigger pin (TRIGIN). Hardware trigger occurs on a high to low transition state of this pin.

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.

Trigger Out Modes

The device can be set to send out a low to high 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 20 (TRIGOUT).

Communication Interfaces

USB interface is common on the SC5511A, while an alternative interface option is SPI or RS232. As both the SPI and RS232 utilize the same internal communication device, and thus the same connector pins, only one option is possible.

SPI Interface

In addition to the 4-wire SPI ($\overline{CS}$, SDO, SDI, and SCLK) signal lines, there is also an alternative serial ready SRDY line. Upon reception of a register command, the device takes time to execute the command instruction, such as setting a new frequency. While the device is busy, the SRDY line will go low and returns high upon execution completion. Detailed SPI read and write operations are discussed in detail in the **Serial Peripheral Interface (SPI)** section.

USB Interface

The SC5511A has a built-in USB controller configured in client mode. The transfer types supported by the device are control, interrupt, and bulk. More information on the use of the USB interface and its software API are provided in the **USB Interface** section.

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	12 GHz	1.5 GHz
Power	0.00 dBm	Max
RF Output	Enabled	NA
ALC Mode	Closed Loop	NA
Standby	Disabled	Enabled
Auto Level	Enabled	NA
Ref Out Select	10 MHz	
Ext Ref Lock	Disabled	

SETTING THE SC5511A: CONFIGURATION REGISTERS

These are write only registers to configure the device. The registers vary in length to reduce redundant data and improving the communication speed, especially for SPI and RS232 interfaces. Furthermore, for SPI and RS232 interfaces, it is vitally important that the length of data written to a register is exact, because failure to do so will cause the interfaces to misinterpret the incoming data, leaving the device in a stalled state. A summary of the configuration registers are provided in Table 5, and each register is explained in detail in the tables following it.

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 "active" LED
SYNTH_MODE	0X03	[7:0]	Open	Open	Open	Open	Open	Disable SS	Loop gain	Lock mode
RF_MODE	0X04	[7:0]	Open	Open	Open	Open	Open	Open	Open	mode
LIST_MODE_CONFIG	0X05	[7:0]	Trig out mode	Trig out enable	Return to start	Step on trigger	Hw tigger	Saw/Tri wave	Sweep dir	SSS-mode
LIST_START_FREQ	0X06	[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	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_STEP_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_DWELL_TIME	0X09	[7:0]	Time (500us) [7:0]							
		[15:8]	Time (500us) [15:8]							
		[23:16]	Time (500us) [23:16]							
		[31:24]	Time (500us) [31:24]							
LIST_CYCLE_COUNT	0X0A	[7:0]	Count Word [7:0]							
		[15:8]	Count Word [15:8]							

Register Name	Register Address	Serial Range	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
		[23:16]	Count Word [23:16]							
		[31:24]	Count Word [31:24]							
RESERVED	0X0B	[7:0]	Open	Open	Open	Open	Open	Open	Open	mode
LIST_BUFFER_POINTS	0X0C	[7:0]	Points Word [7:0]							
		[15:8]	Points 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		Open	Open	Open	Open	Open	Open	Open	mode
LIST_SOFT_TRIGGER	0X0F		Open	Open	Open	Open	Open	Open	Open	Open
RF_FREQUENCY	0x10	[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]							
RF_LEVEL	0x11	[7:0]	RF Power Word [7:0]							
		[15:8]	Sign Bit	RF Power Word [14:8]						
RF_OUT_ENABLE	0x12	[7:0]	Open	Open	Open	Open	Open	Open	Open	Mode
RF_PHASE	0X13	[7:0]	Phase Word (in tenths of deg)[7:0]							
			Sign bit	Phase Word (in tenths of deg)[14:8]						
AUTO_LEVEL_DISABLE	0x14	[7:0]	Open	Open	Open	Open	Open	Open	Open	Mode
RF_ALC_MODE	0x15	[7:0]	Open	Open	Open	Open	Open	Open	Open	Mode
RF_STANDBY	0x16	[7:0]	Open	Open	Open	Open	Open	Open	Open	Mode
REFERENCE_MODE	0x17	[7:0]	Open	Open	Open	Open	Open	Open	Ref Out Select	Lock Enable
REFERENCE_DAC_SETTING	0x18	[7:0]	DAC word [7:0]							
		[15:8]	Open	Open	DAC word [13:8]					
ALC_DAC_VALUE	0x19	[7:0]	DAC word [7:0]							
		[15:8]	Open	Open	DAC word [13:8]					
RESERVED	0x1A	[7:0]	Open	Open	Open	Open	Open	Open	Open	Open
STORE_DEFAULT_STATE	0x1B	[7:0]	Open	Open	Open	Open	Open	Open	Open	Open
RESERVED	0x1C	[7:0]	Open	Open	Open	Open	Open	Open	Open	Open
RESERVED	0x1D	[7:0]	Open	Open	Open	Open	Open	Open	Open	Open
RF2_STANDBY	0x1E	[7:0]	Open	Open	Open	Open	Open	Open	Open	Open
RF2_FREQENCY	0x1F	[7:0]	Frequency Word (MHz) [7:0]							
		[15:8]	Frequency Word (MHz) [15:8]							

The following tables provide details of each of the registers. These registers are the same for all modes of communication. The USB API functions provided are simply wrappers that properly set up the data bits of these registers to simplify programming.

Table 6. Register 0x01 INTIALIZE (1 Byte)

Bit	Type	Name	Width	Description
[0]	WO	Mode	1	0 = Re-initialize device with current settings 1 = Re-initialize device to power up state
[7:1]	WO	Unused	7	Set all bits to 0

Table 7. Register 0x02 SET_SYS_ACTIVE (1 Byte)

Bit	Type	Name	Width	Description
[0]	WO	Mode	1	0 = turns off access LED 1 = turns on access LED
[7:1]	WO	Unused	7	Set all bits to 0

Table 8. Register 0x03 SYNTH_MODE (1 Byte)

Bit	Type	Name	Width	Description
[0]	WO	Lock Mode	2	0 = harmonic offset mode 1 = fracN PLL offset mode
[1]	WO	Loop Gain	2	0 = Normal loop gain – for better close in phase noise 1 = low loop gain – for better far out phase noise and spur suppression.
[2]	WO	Disable spur suppression	1	Only applies in harmonic offset mode, see bit[0]. 0 = The device automatically switches to fracN offset mode to avoid potentially large spurs due to intermodulation between the carrier and the harmonics of the reference clock. 1 = This disables the function. May speed up tuning speed in some cases.
[7:3]	WO	Unused	5	Set all bits to 0

Table 9. Register 0x04 RF_MODE (1 Byte)

Bit	Type	Name	Width	Description
[0]	WO	RF Mode	1	0 = Single fixed tone mode. This mode must be set to change the frequency value via register 0x1A. 1 = Sweep/list mode. In this mode writing to register 0x10 will be unresponsive. This register must be called first for sweep/list triggering to function.
[7:1]	WO	Unused	7	Set all bits to 0.

Table 10. Register 0x05 LIST_MODE_CONFIG (1 Byte)

Bit	Type	Name	Width	Description
[0]	WO	SSS Mode	1	0 = List mode. Device gets its frequency points from the list buffer uploaded via the LIST_BUFFER_WRITE register (0x0D). 1 = Sweep mode. The device computes the frequency points using the Start, Stop, and Step frequencies.
[1]	WO	Sweep Direction	1	0 = Forward. In the forward direction, the sweeps starts from the lowest start frequency or starts at the beginning of the list buffer. 1 = Reverse. In the reverse direction, the sweep starts with the stop frequency and steps down toward the start frequency or starts at the end and steps toward the beginning of the buffer.
[2]	WO	Triangular Waveform	1	0 = Sawtooth waveform. Frequency returns to the beginning frequency upon reaching the end of a sweep cycle. 1 = Triangular waveform. Frequency reverses direction at the end of the list and steps back towards the beginning to complete a cycle.
[3]	WO	Soft/Hardware Trigger	1	0 = Software trigger. Software trigger can only be used to start and stop a sweep/list cycle. It does not work for step-on-trigger mode. 1 = Hardware trigger. A high-to-low (↓) transition on the TRIGIN pin will trigger the device. It can be used for both start/stop or step-on-trigger functions.
[4]	WO	Step on Trigger	1	0 = Start/Stop behavior. The sweep starts and continues to step through the list for the number of cycles set, dwelling at each step frequency for a period set by the LIST_DWELL_TIME register. The sweep/list will end on a consecutive trigger. 1 = Step-on-trigger. This is only available if hardware triggering is selected. The device will step to the next frequency on a trigger. Upon completion of the number of cycles (set by the LIST_CYCLE_COUNT register), the device will exit from the stepping state and stop. Further triggering will set the device back into the stepping state. To exit the stepping state and stop before reaching the end of a cycle, a software trigger must to be sent or a change in the RF mode to single fixed tone needs to be made.

Bit	Type	Name	Width	Description
[5]	WO	Return to Start	1	0 = Stop at end of sweep/list. The frequency will stop at the last point of the sweep/list. 1 = Return to start. The frequency will return and stop at the beginning point of the sweep or list after a cycle.
[6]	WO	Trigger Output	1	0 = No trigger output. 1 = Puts a trigger pulse on the TRIGOUT pin
[7]	WO	Trigger Out Mode	1	0 = Puts out a trigger pulse at each frequency change, right after all internal devices are configured. 1 = Puts out a trigger pulse at the completion of each sweep/list cycle.

Table 11. Register 0x06 LIST_START_FREQ (5 Byte)

Bit	Type	Name	Width	Description
[39:0]	WO	List Start Frequency	40	Sets the start frequency for a sweep. Start frequency should always be lower than the stop frequency. The Sweep Direction bit [1] of register 0x05 should be used to determine where the sweep should begin.

Table 12. Register 0x07 LIST_STOP_FREQ (5 Bytes)

Bit	Type	Name	Width	Description
[39:0]	WO	List Stop Frequency	40	Sets the stop frequency for a sweep. Stop frequency should always be greater than the start frequency. The Sweep Direction bit [1] of register 0x05 should be used to determine where the sweep should begin.

Table 13. Register 0x08 LIST_STEP_FREQ (5 Bytes)

Bit	Type	Name	Width	Description
[39:0]	WO	List Step Frequency	40	Sets the step frequency for a sweep. Step size should not exceed the difference between the start and stop frequencies.

Table 14. Register 0x09 LIST_DWELL_TIME (4 Bytes)

Bit	Type	Name	Width	Description
[31:0]	WO	List Dwell Time	32	Set the dwell time at each step frequency. The Dwell time is incremented in 500 µs increments. For example, to produce a 10 ms dwell time the value written to this register is 20d.

Table 15. Register 0x0A LIST_CYCLE_COUNT (4 Bytes)

Bit	Type	Name	Width	Description
[31:0]	WO	List Cycle Count	32	0 = Cycle forever. This will set the device to cycle forever. Not 0 will set the number of cycles the device will sweep or step though the list then stop. This applies for both start/stop and step trigger modes.

Table 16. Register 0x0B Reserved

Bit	Type	Name	Width	Description
[7:0]	WO	Reserved	7	

Table 17. Register 0x0C LIST_BUFFER_POINTS (4 Bytes)

Bit	Type	Name	Width	Description
[31:0]	WO	Number of Buffer Points	32	Sets the number of frequency points to step through in the buffer list. The number must be equal to or less than the buffer length. This number will overwrite the count determined from the LIST_BUFFER_WRITE register.

Table 18. Register 0x0D LIST_BUFFER_WRITE (5 Bytes)

Bit	Type	Name	Width	Description
[39:0]	WO	Buffer Frequency	40	Writing this register stores the frequency point into the list buffer held in RAM. Writing 0x0000000000 to this buffer resets the pointer to buffer location [0] and flags the device to store data written to this register. Consecutive non-zero writes to this register will increase the buffer counter up to 2047. Further writes beyond this point are not recognized. Writing 0xFFFFFFFFFF to this register at any time will terminate the write process and stops the pointer increment. The value at which the pointer stops is the new count of list frequency points unless it is overwritten by register LIST_BUFFER_POINTS.

Table 19. Register 0x0E LIST_BUF_MEM_TRNSFER (1 Byte)

Bit	Type	Name	Width	Description
[0]	WO	Transfer Direction	1	0 = Transfers the contents of the list buffer into EEPROM memory. The size of the transfer is set by the list frequency points.

Bit	Type	Name	Width	Description
				1 = Transfers the contents from EEPROM memory to the list buffer (in RAM).
[7:1]	WO	Unused	7	Set all bits to 0.

Table 20. Register 0x0F LIST_SOFT_TRIGGER (1 Byte)

Bit	Type	Name	Width	Description
[7:0]	WO	Soft Trigger	8	Set all bits to 0. Calling this register provides a soft trigger to the device.

Table 21 Register 0x10 RF_FREQUENCY(5 Bytes)

Bit	Type	Name	Width	Description
[39:0]	WO	RF1 frequency word	40	Sets the RF1 frequency in Hz

Table 22 Register 0x11 RF_LEVEL (2 Bytes)

Bit	Type	Name	Width	Description
[14:0]	WO	RF1 Power Level	15	Sets the RF1 Power level in hundreds of dB. To set to 10.25 dB, write 1025 to this register
[15]	WO	Sign bit	1	0 = positive number 1 = negative number

Table 23 Register 0x12 RF_ENABLE (1 Byte)

Bit	Type	Name	Width	Description
[0]	WO	RF1 enable	1	0 = disables the output power 1 = enables the output power
[7:1]	WO	Unused	7	Set all bits to 0

Table 24. Register 0x013 RF_PHASE (2 Byte)

Bit	Type	Name	Width	Description
[14:0]	WO	Phase word	15	Phase word in tenths of degree.
[15]		Sign bit (1 = neg)	1	Set bit to 1 for negative phase word

Table 25 Register 0x14 AUTO_LEVEL_DISABLE (1 Byte)

Bit	Type	Name	Width	Description
[0]	WO	RF1 Auto leveling	1	0 = power is leveled on frequency change 1 = power is not leveled on frequency change with explicitly calling register 0x11 (RF_LEVEL)
[7:1]	WO	Unused	7	Set all bits to 0

Table 26 Register 0x15 RF_ALC_MODE (1 Byte)

Bit	Type	Name	Width	Description
[0]	WO	RF1 ALC mode	1	0 = Amplitude is corrected using ALC closed loop 1 = Amplitude is corrected opened loop.
[7:1]	WO	Unused	7	Set all bits to 0

Table 27 Register 0x16 RF_STANDBY (1 Byte)

Bit	Type	Name	Width	Description
[0]	WO	RF1 standby	1	1 = puts the RF1 channel into standby. Standby powers down all circuitry associated with ch1, thus reducing power consumption.
[7:1]	WO	Unused	7	Set all bits to 0.

Table 28. Register 0x17 REFERENCE_MODE (1 Byte)

Bit	Type	Name	Width	Description
[0]	WO	Lock to external reference source	1	1 = instructs the device to lock to external source. No attempt will be made unless a reference source is detected.
[1]	WO	Reference out select	1	0 = Outputs a 10 MHz signal 1 = Outputs a 100 MHz signal
[2]	WO	Direct Clocking	1	When 100 MHz is selected in [3], it may clock the main synthesis section directly, bypassing the internal reference. *HW Ver >=16
[3]	WO	Reference in select	1	0 = 10 MHz, 1 = 100 MHz. HW Ver >=16
[7:4]	WO	Unused	4	Set all bits to 0.

Table 29 Register 0x18 REFRENCE_DAC_VALUE (2 Bytes)

Bit	Type	Name	Width	Description
[13:0]	WO	DAC Value	14	14 bit word to set/adjust the internal 10 MHz TCXO frequency.
[15:14]	WO	Unused	2	Set all bits to 0.

Table 30 Register 0x18 ALC_DAC_VALUE (2 Bytes)

Bit	Type	Name	Width	Description
[13:0]	WO	DAC Value	14	14 bit word to set/adjust the ALC DAC value. This is useful to make output adjustment. The current ALC DAC value can be read back via register 0x25
[15:14]	WO	Unused	2	Set all bits to 0.

Table 31. Register 0x01A Reserved (1 Byte)

Bit	Type	Name	Width	Description
[7:0]	WO	Reserved	7	

Table 32 Register 0x1B STORE_DEFAULT_STATE (1 Byte)

Bit	Type	Name	Width	Description	
[7:0]	WO	Reserved	8	Set all bits to 0. Calling this register will store the current configuration into memory. On reset or power-up these values are read from memory and set as the default values. Some of the values are:	
				➢ RF1 parameters ➢ List mode configuration ➢ RF mode ➢ List mode parameters	➢ RF2 parameters

Table 33. Register 0x01C Reserved (1 Byte)

Bit	Type	Name	Width	Description
[7:0]	WO	Reserved	7	

Table 34. Register 0x01D Reserved (1 Byte)

Bit	Type	Name	Width	Description
[7:0]	WO	Reserved	7	

Table 35. Register 0x01E RF2_STANDBY(1 Byte)

Bit	Type	Name	Width	Description
[0]	WO	RF2 standby	1	1 = puts the RF2 channel into standby. Standby powers down all circuitry associated with ch2, thus reducing power consumption.
[7:1]	WO	Unused	7	Set all bits to 0.

Table 36. Register 0x01F RF2_FREQUENCY (2 Bytes)

Bit	Type	Name	Width	Description
[14:0]	WO	Frequency (MHz)	14	Frequency word in MHz; 25 to 3000

Table 377. Register 0x47 SYNTH_SELF_CAL (1 Byte)

Bit	Type	Name	Width	Description
[7:0]	WO	Unused	8	Set all bits to 0.

QUERYING THE SC5511A: QUERY REGISTERS

These are request for data registers, in that a request for certain data is made by writing to the specific register first, and then followed by reading back the requested data. Some registers may require instruction data to specify the type of data to return, while others do not need any. For example, the GET_RF_PARAMETERS (0x20) returns sweep dwell time, rf1_frequency, rf2_frequency, etc; this depends on the request instruction byte.

Returned data length is always 5 bytes (40 bits), with the first byte being the most significant (MSB). Not all returned data have 5 valid bytes, and for those data the ending bytes are padded with zeros. For example, for an integer data with first 4 valid bytes, the last byte is 0; that is [MSB][Byte2][Byte1][Byte0][zeros]. It is **important that all 5 bytes** are read in order to clear the interface buffers.

A summary of the query registers are list in Table 38, and their details are provided in the tables that follow.

Table 38. 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_RF_PARAMETERS	0x20	[7:0]	Open	Open	Open	Open	parameter			
GET_TEMPERATURE	0x21	[7:0]	Open	Open	Open	Open	Open	Open	Open	Open
GET_DEVICE_STATUS	0x22	[7:0]	Open	Open	Open	Open	Open	Open	Open	Open
GET_DEVICE_INFO	0x23	[7:0]						info		
GET_LIST_BUFFER	0x24	[7:0]	Buffer Address [7:0]							
		[15:8]	Open	Open	Open	Open	Open	Buffer Address [11:8]		
GET_ALC_DAC_VALUE	0x25	[7:0]	Open	Open	Open	Open	Open	Open	Open	Open
GET_SERIAL_OUT_BUF	0x26	[7:0]	Open	Open	Open	Open	Open	Open	Open	Open

Table 39. Register 0x20 GET_RF_PARAMETERS (1 Byte, 5 Bytes)

Bit	Type	Name	Width	Description
[3:0]	WO	Parameters	4	Data specifies the parameter to retrieve: 0x00 = Current RF1 freq (5 valid bytes return) 0x01 = Sweep Start Freq (5 valid bytes return) 0x02 = Sweep Stop Freq (5 valid bytes return) 0x03 = Sweep Step Freq (5 valid bytes return) 0x04 = Sweep Dwell Time (4 valid bytes return) 0x05 = Sweep Cycle Count (4 valid bytes return) 0x06 = Sweep List Buffer Pts (4 valid bytes return) 0x07 = Current RF1 level 0x08 = Current RF2 Freq(2 valid bytes return) 0x0A = signal phase (firmware > Version 2.5) float phase=*(float*)&read_in_unsigned_int

Bit	Type	Name	Width	Description
[7:4]	WO	Unused	4	Set all bits to 0.
[39:0]	RO	Data	40	Data with varying sizes of unsigned type.

Table 40. Register 0x21 GET_TEMPERATURE (1 byte, 5 Bytes)

Bit	Type	Name	Width	Description
[7:0]	WO	Unused	8	Set all to 0
[39:8]	RO	Valid flatten float type temperature	32	The data is returned in unsigned integer form of flatten float type. To recast the unsigned integer back to float use: float temp = *(float*)&read_in_unsigned_int_var
[7:0]	WO	Invalid data	8	zeros

Table 41. Register 0x22 DEVICE_STATUS (0) (1 Byte, 5 Bytes)

Bit	Type	Name	Width	Description
[0]	WO	0 device status	1	Sets up the read-back buffer with contents of the current device status. Contents are immediately available for USB read. In the case of SPI, contents are transferred to the serial output buffer, so a second query to the SERIAL_OUT_BUFFER register is required to transfer its contents and also to clear the output buffer.
[7:1]	WO		7	Set all to 0
[39]	RO	List Config: Trig Out Mode	1	
[38]	RO	List Config: Trig Out Enable	1	
[37]	RO	List Config: Return to Start	1	
[36]	RO	List Config: Step Trig	1	
[35]	RO	List Config: HW Trig	1	
[34]	RO	List Config: Waveform (Tri/Saw)	1	
[33]	RO	List Config: Sweep Dir	1	
[32]	RO	List Config: SSS mode	1	
[31:30]	RO	Reserved	2	
[29]	RO	Operate: Over Temp	1	Device temperature above threshold of ~ 75 ^{O}C
[28]	RO	Operate: RF1_mode	1	0 = fix, 1 = list/sweep
[27]	RO	Operate: list running	1	1 = device triggered and running list/sweep mode
[26]	RO	Operate: ref_out_select	1	0 = 10 MHz, 1 = 100 MHz
[25]	RO	Operate: ext_ref_detect	1	1 = reference source at ref port
[24]	RO	Operate: ext_ref_lock	1	1 = lock to external sources is enabled
[23]	RO	Operate: RF1_enable	1	1 = RF1 output is enabled

Bit	Type	Name	Width	Description
[22]	RO	Operate: alc_mode	1	0 = closed, 1 = opened
[21]	RO	Operate: auto_level	1	0 = enabled, 1 = disabled
[20]	RO	Operate: rf1_standby	1	0 = standby disabled, 1 = standby enabled
[19]	RO	Operate: rf2_standby	1	0 = standby disabled, 1 = standby enabled
[18]	RO	Operate: device_access	1	1 = device is accessed
[17]	RO	Operate: loop_gain	1	0=normal, 1=low loop gain
[16]	RO	Operate: lock_mode	1	0=harmonic, 1=fracN
[15]	RO	Pll_status:rf2	1	Rf2 pll locked
[14]	RO	Pll_status:ref_TCXO	1	1 = the 10 MHz TCXO is locked
[13]	RO	Pll_status:ref_VCXO	1	1 = the 100 MHz VCXO is locked
[12]	RO	Pll_status:crs_aux	1	1 = the auxiliary crs loop (fracN lock) is locked
[11]	RO	Pll_status:crs_ref	1	1 = the ref source for the crs loop is locked
[10]	RO	Pll_status:fine	1	1 = the DDS based fine loop is locked
[9]	RO	Pll_status:crs	1	1 = the crs (harmonic) loop is locked
[8]	RO	Pll_status:sum	1	1 = the main loop is locked
[7:0]	RO	Invalid data	8	0

Table 42. Register 0x22 DEVICE_STATUS (1) (1 Byte, 5 Bytes)

Bit	Type	Name	Width	Description
[0]	WO	1 reference config	1	Will read back the reference configuration parameters
[7:1]	WO		7	Set all to 0
[39:13]	RO	Invalid data	27	zeros
[12]	RO	Ext_Ref_Select	1	Select ext. freq. 0 = 10 MHz 1 = 100 MHz
[11]	RO	Ext_Direct_Clk	1	Clock synthesizer directly using 100 MHz
[10]	RO	Invalid	1	0
[9]	RO	Out Freq Select	1	Output ref. freq. 0 = 10 MHz 1 = 100 MHz
[8]	RO	Lock External Enable	1	Enable locking to external source
[7:0]	RO	Invalid data	8	0

Table 43. Register 0x23 GET_DEVICE_INFO (1 Byte, 5 Bytes)

Bit	Type	Name	Width	Description
[2:0]	WO	Device Status	3	Writing this register will place the requested contents into the output buffer. Contents are immediately available for USB read. The contents occupy effectively four bytes. In the case of SPI, contents are transferred to the serial output buffer, so a second query to the SERIAL_OUT_BUFFER register is required to transfer its contents and also to clear the output buffer. 0 = Obtain the product serial number 1 = Obtain the hardware revision

Bit	Type	Name	Width	Description
				2 = Obtain the firmware revision 3 = Obtain the manufacture date
[7:3]	WO	Unused	5	
[39:8]	RO	Data	32	Data for the requested parameter: Product Serial Number – 32-bit unsigned Hardware Revision – typecast to 32-bit float Firmware Revision – typecast to 32-bit float Manufacture Date – unsigned 32-bit [31:24] Year (last two digits) [23:16] Month [15:8] Day [7:0] Hour
[7:0]	WO	Invalid data	8	zeros

Table 44. Register 0x24 GET_LIST_BUFFER (2 bytes, 5 bytes)

Bit	Type	Name	Width	Description
[15:0]	WO	Buffer Address	16	The data point (0 – 2047) to read.
[39:0]	RO	Frequency in Hz	40	The data is returned in unsigned 5 bytes. Must be converted to unsigned long long int.

Table 45. Register 0x25 GET_ALC_DAC_VALUE(1 byte, 5Bytes)

Bit	Type	Name	Width	Description
[7:0]	WO	Used	16	zeros
[39:24]	RO	Dac Value	16	The data is returned in unsigned 2 bytes. Must be converted to unsigned short.
[23:0]	RO	Invalid data	24	zeros

Table 46. Register 0x26 SERIAL_OUT_BUFFER (5 Bytes, 5 Bytes)

Bit	Type	Name	Width	Description
[39:0]	WO	Serial Out Buffer	40	Set all bits to 0. Use of this register is only available for the SPI interface.
[39:0]	RO	Request Data	40	The data clocked back are the contents requested by the 0x20 to 0x25 registers.

USB INTERFACE

The SC5511A has a full speed USB interface that works in parallel with the SPI/RS232 interface. Both interfaces are active at the same time if the USB interface is available on the device.

USB Configuration

The SC5511A USB interface is USB 2.0 compliant running at *Full Speed*, capable of 12 Mbits per second transfer rates. The interface supports three transfer or endpoint types:

- Control Transfer
- Interrupt Transfer
- Bulk Transfer

The endpoint addresses are provided in the C-language header file and are listed below:

```
//  Define SignalCore USB Endpoints
#define  SCI_ENDPOINT_IN_INT          0x81
#define  SCI_ENDPOINT_OUT_INT         0x02
#define  SCI_ENDPOINT_IN_BULK         0x83
#define  SCI_ENDPOINT_OUT_BULK        0x04

// Define for Control Endpoints
#define  USB_ENDPOINT_IN              0x80
#define  USB_ENDPOINT_OUT             0x00
#define  USB_TYPE_VENDOR              (0x02 << 5)
#define  USB_RECIP_INTERFACE          0x01
```

The buffer lengths are sixty-four bytes for all endpoint types. The user should not exceed this length or the device may not respond correctly. This information is provided to aid custom driver development on host platforms other than those that are supported by SignalCore.

Writing the Device Registers Directly

Device register for the SC5511A vary between two bytes and six bytes in length. The **most significant byte (MSB)** is the command register address that specifies how the device should handle the subsequent configuration data. The configuration data likewise needs to be ordered MSB first, that is, transmitted first. Input and output buffers of six bytes long are sufficient on the host. To ensure that a register instruction has been fully executed by the device, reading a byte back from the device will confirm that because the device will only return data upon full execution of the instruction, although this is not necessary.

Reading the Device Registers Directly

Valid data is only available to be read back after writing one of the query registers such as 0x20. As soon as one of these registers is written, data is available on the device to be read back through USB. When reading the device, the MSB is returned as the first byte for a total of five bytes. Valid data starts on the first byte, and if the data is less than 5 bytes the tail bytes are padded with zeros.

USB Driver API

The SC5511A USB driver provided by SignalCore is based on libusb-1.0 (www.libusb.org) and its API library is available for the Windows™ and Linux™ operating systems. Source code for both platforms is available upon request by emailing support@signalcore.com. The API functions are nothing more than register wrappers called through the USB bulk transfer function. The C/C++ API library functions are summarized in the list below and each function description is provided in the API description section.

SignalCore's philosophy is to provide our customers with products in which 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 SC5511A API (versions > 2.2.0) dynamic linked library with call type __cdecl (**sc5511a.dll**) are:

- sc5511a_search_devices
- sc5511a_open_device
- sc5511a_close_device
- sc5511a_reg_write
- sc5511a_reg_read
- sc5511a_initialize
- sc5511a_set_freq
- sc5511a_set_signal_phase
- sc5511a_set_synth_mode
- sc5511a_set_rf_mode
- sc5511a_list_mode_config
- sc5511a_list_start_freq
- sc5511a_list_stop_freq
- sc5511a_list_step_freq
- sc5511a_list_dwell_time
- sc5511a_list_cycle_count
- sc5511a_list_buffer_points
- sc5511a_list_buffer_write
- sc5511a_list_buffer_transfer
- sc5511a_list_soft_trigger
- sc5511a_set_power_level
- sc5511a_set_output
- sc5511a_auto_level_disable

- sc5511a_set_alc_mode
- sc5511a_set_standby
- sc5511a_set_clock_reference
- sc5511a_set_reference_dac
- sc5511a_set_alc_dac
- sc5511a_store_default_state
- sc5511a_set_rf2_standby
- sc5511a_set_rf2_freq
- sc5511a_synth_self_cal
- sc5511a_get_rf_parameters
- sc5511a_get_temperature
- sc5511a_get_signal_phase
- sc5511a_get_device_status
- sc5511a_get_device_info
- sc5511a_list_buffer_read
- sc5511a_get_alc_dac
- sc5511a_get_clock_config

Each of these functions is described in more detail on the following pages. For C/C++ development the constants, types, and function prototypes are contained in the C header file, *sc5511a.h*. These constants and types are useful not only as an include for developing applications using the SC5511A API, but also for writing device drivers independent of those provided by SignalCore.

API Description

The prototype functions listed below are found in the **sc5511a.h** header file and the functions are contained in the dynamic linked library (**sc5511a.dll**) for the Windows™ operating system.

Function: **sc5511a_search_devices**

Definition: int sc5511a_search_devices(char **serial_numberList)

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

Description: sc5511a_search_devices searches for SignalCore SC5511A devices connected to the host computer and returns (int) the number of devices found, and it also populates the char array with their serial numbers.

Function: **sc5511a_open_device**

Definition: sc5511a_device_handle_t *sc5511a_open_device (char *dev_serial_num)

Input: char * dev_serial_num (serial number string)

Return: *device_handle (unsigned int number for the sc5511a_device_handle_t)

Description: sc5511a_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: **sc5511a_close_device**

Definition: int sc5511a_close_device (sc5511a_device_handle_t *dev_handle)

Input: sc5511a_device_handle_t *dev_handle (handle to the device to be closed)

Description: sc5511a_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 close the PXIe device:

```
#include "sc5511a.h"
// Declaring
#define MAXDEVICES 50
sc5511a_device_handle_t *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
num_of_devices = sc5511a_search_devices(device_list); //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++;
       }
//** sc5511a_OpenDevice, open device 0
dev_handle = sc5511a_open_device(device_list[0]);
// 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 = sc5511a_close_device(dev_handle);
```

Function: **sc5511a_reg_write**

Definition: int sc5511a_reg_write (sc5511a_device_handle_t *dev_handle,
unsigned char reg_byte,
unsigned long long int instruct_word)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
unsigned char reg_byte (register address)
unsigned long long int instruct_word (the data for the register)

Description: sc5511a_reg_write writes the instruct_word data to the register specified by the reg_byte. See the register maps for more information.

Example: To set the power level to 2.00 dBm:

```
int status = sc5511a_reg_write(dev_handle, RF_POWER, 200);
```

Function: **sc5511a_reg_read**

Definition: int sc5511a_reg_read (sc5511a_device_handle_t *dev_handle,
unsigned char reg_byte,
unsigned long long int instruct_word, unsigned int *receivedWord)

Input: sc5511a_device_handle_t *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: sc5511a_reg_read reads the data requested by the instruct_word data to the register specified by the reg_byte. Data is returned on received_word See the register maps for more information.

Example: To read the status of the device:

```
unsigned long long int frequency;

int status = sc5511a_reg_read(dev_handle, GET_DEVICE_STATUS, 0x00, &frequency);
```

Function: **sc5511a_set_freq**

Definition: int sc5511a_set_freq(sc5511a_device_handle_t *dev_handle,
unsigned long long int freq)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
unsigned long long int frequency (frequency in Hz)

Description: sc5511a_set_frequency sets RF1 frequency.

Function: **sc5511a_set_signal_phase**

Definition: int sc5511a_set_signal_phase (sc5511a_device_handle_t *dev_handle,
float phase)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
float phase (phase)

Description: sc5511a_set_signal_phase set the relative phase of the signal on CH1.

Function: **sc5511a_set_synth_mode**

Definition: sc5511a_set_synth_mode(sc5511a_device_handle_t *dev_handle,
unsigned char disable_spur_suppression,
unsigned char loop_gain,
unsigned char lock_mode)

Input: **sc5511a_device_handle_t** *dev_handle (handle to the opened device)
unsigned char disable_spur_suppression (set the spur suppression behavior)
unsigned char loop_gain (set synth loop gain)
unsigned char lock_mode (set lock mode of RF1)

Description: sc5511a_set_synth_mode sets synthesizer mode of RF1. In harmonic mode, the device switches automatically to fractional-N mode to avoid significant spurs induced through intermodulation between the reference harmonics and the RF signal. Setting disable_spur_suppression to 1 disables this automatic behavior, disabling the spur calculation algorithm, and improves the device switching time. The loop gain adjust the PLL bandwidth to shape the phase noise. The loop gain set to low (1) suppresses the far out phase noise as well as spurs. The default state is normal (0). The lock mode select either the harmonic circuitry or fractional-N circuitry as the offset loop of the hybrid loop architecture.

Function: **sc5511a_set_rf_mode**

Definition: sc5511a_set_rf_mode(sc5511a_device_handle_t *dev_handle,
unsigned char rf_mode)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
unsigned char rf_mode (set RF mode of RF1)

Description: sc5511a_set_rf_mode sets RF1 to fixed tone or sweep.

Function: **sc5511a_list_mode_config**

Definition: int sc5511a_list_mode_config(sc5511a_device_handle_t *dev_handle,
const list_mode_t *list_mode)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
const list_mode_t *list_mode (list mode setup)

Description: sc5511a_ListModeConfig configures the list mode behavior. See the document for more information on the modeConfig structure.

Function: **sc5511a_list_start_freq**

Definition: int sc5511a_list_start_freq(sc5511a_device_handle_t *dev_handle,
unsigned long long int freq)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
unsigned long long int freq (frequency in Hz)

Description: sc5511a_ list_start_freq sets the sweep start frequency.

Function: **sc5511a_list_stop_freq**

Definition: int sc5511a_list_stop_freq(sc5511a_device_handle_t *dev_handle,
unsigned long long int freq)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
unsigned long long int freq (frequency in Hz)

Description: sc5511a_ list_stop_freq sets the sweep stop frequency.

Function: **sc5511a_list_step_freq**

Definition: int sc5511a_list_step_freq(sc5511a_device_handle_t *dev_handle,
unsigned long long int freq)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
unsigned long long int freq (frequency in Hz)

Description: sc5511a_list_step_freq sets the sweep step frequency.

Function: **sc5511a_list_dwell_time**

Definition: int sc5511a_list_dwell_time(sc5511a_device_handle_t *dev_handle,
unsigned int dwell_time)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
unsigned int dwell_time (Time in 500 µs increments)

Description: sc5511a_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: **sc5511a_list_cycle_count**

Definition: int sc5511a_list_cycle_count(sc5511a_device_handle_t *dev_handle,
unsigned int cycle_count)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
unsigned int cycle_count (number of cycles)

Description: sc5511a_list_cycle_count sets the number of sweep cycles to perform before stopping. To repeat the sweep continuously, set the value to 0.

Function: **sc5511a_list_buffer_points**

Definition: int sc5511a_list_buffer_points(sc5511a_device_handle_t *dev_handle,
unsigned int list_points)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
unsigned int list_points (number of points of the list buffer)

Description: sc5511a_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: **sc5511a_list_buffer_write**

Definition: int sc5511a_list_buffer_write(sc5511a_device_handle_t *dev_handle,
unsigned long long int freq)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
unsigned long long int freq (frequency in Hz)

Description: sc5511a_list_buffer_write writes the frequency buffer sequentially. If frequency value = 0, the buffer pointer is reset to position 0 and subsequent writes will increment the pointer. Writing 0xFFFFFFFFF will terminate the sequential write operation and sets the list_buffer_points variable to the last pointer value.

Function: **sc5511a_list_buffer_transfer**

Definition: int sc5511a_list_buffer_transfer(sc5511a_device_handle_t *dev_handle,
unsigned char transfer_mode)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
unsigned char transferMode (transfer to EEPROM or RAM)

Description: sc5511a_list_buffer_transfer transfers the frequency list buffer from RAM to EEPROM or vice versa.

Function: **sc5511a_list_soft_trigger**

Definition: int sc5511a_list_soft_trigger(sc5511a_device_handle_t *dev_handle)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)

Description: sc5511a_list_soft_trigger triggers the device when it is configured for list mode and soft trigger is selected as the trigger source.

Function: **sc5511a_set_level**

Definition: int sc5511a_set_level(sc5511a_device_handle_t *dev_handle,
float power_level)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
float power_level (level in dBm)

Description: sc5511a_set_power_level set the power output level of RF1.

Function: **sc5511a_set_output**

Definition: int sc5511a_set_output(sc5511a_device_handle_t *dev_handle,
unsigned char enable)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
unsigned char enable (enables the output)

Description: sc5511a_set_output enables or disables the output RF1.

Function: **sc5511a_set_auto_level_disable**

Definition: int sc5511a_set_auto_level_disable(sc5511a_device_handle_t *dev_handle,
unsigned char disable)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
unsigned char disable (disable leveling)

Description: sc5511a_set_auto_level_disable disables the leveling compensation after the frequency is changed for channel RF1.

Function: **sc5511a_set_alc_mode**

Definition: int sc5511a_set_alc_mode(sc5511a_device_handle_t *dev_handle,
unsigned char mode)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
unsigned char mode (ALC close/open)

Description: sc5511a_set_alc_mode set the ALC to close (0) or open (1) mode operation for channel RF1.

Function: **sc5511a_set_standby**

Definition: int sc5511a_set_standby(sc5511a_device_handle_t *dev_handle,
unsigned char enable)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
unsigned char enable (enables/disables standby)

Description: sc5511a_set_standby, if enabled, powers down channel RF1.

Function: **sc5511a_set_clock_reference**

Definition: int sc5511a_set_clock_reference(sc5511a_device_handle_t *dev_handle,
unsigned char ext_ref_select,
unsigned char ext_direct_clk
unsigned char select_freq,
unsigned char lock_external)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
unsigned char ext_ref_select (selects input as 10 MHz or 100 MHz)
unsigned char ext_direct_clk (bypass internal reference, clocks synth directly)
unsigned char select_freq (selects 10 MHz or 100 MHz)
unsigned char lock_external (locks to external reference)

Description: sc5511a_set_clock_reference configure the reference clock behavior.

Function: **sc5511a_set_reference_dac**

Definition: int sc5511a_set_reference_dac(sc5511a_device_handle_t *dev_handle,
unsigned short dac_value)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
unsigned short dac_value (DAC value to be written)

Description: sc5511a_set_reference_dac set the DAC value that controls the TCXO frequency.

Function: **sc5511a_set_alc_dac**

Definition: int sc5511a_set_alc_dac(sc5511a_device_handle_t *dev_handle,

unsigned short dac_value)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
unsigned short dac_value (DAC value to be written)

Description: sc5511a_set_alc_dac set the value of the ALC DAC to make amplitude adjustments.

Function: **sc5511a_store_default_state**

Definition: int sc5511a_store_default_state(sc5511a_device_handle_t *dev_handle)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)

Description: sc5511a_store_default_state stores the current configuration into EEPROM memory as the default state upon reset or power-up.

Function: **sc5511a_set_rf2_standby**

Definition: int sc5511a_set_rf2_standby(sc5511a_device_handle_t *dev_handle,
unsigned char enable)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
unsigned char enable (enable the device to go in standby mode)

Description: sc5511a_set_rf2_standby powers down channel RF2.

Function: **sc5511a_set_rf2_freq**

Definition: int sc5511a_set_rf2_freq(sc5511a_device_handle_t *dev_handle,
unsigned short freq)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
unsigned short freq (frequency in MHz)

Description: sc5511a_set_rf2_freq sets the frequency for channel RF2.

Function: **sc5511a_synth_self_cal**

Definition: int sc5511a_synth_self_cal(sc5511a_device_handle_t *dev_handle)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)

Description: sc5511a_synth_self_cal will cause the device to perform a self calibration of the DAC values to properly set the VCO up for phase lock. When the device uses the harmonic - generator as the offset loop, the VCO could potentially lock to a wrong reference harmonic causing the sum PLL to fail. Perform this function if the sum PLL fails when the synthesizer is in harmonic lock mode. Allow 2-3 seconds for the calibration routing to execute, and upon completion the device will reset. The status indicator of RF1 will go off, then red, then amber, and then finally green.

Function: **sc5511a_get_rf_parameters**

Definition: int sc5511a_get_rf_parameters(sc5511a_device_handle_t*dev_handle, rf_params_t *rf_params)

Input: sc5511a_device_handle_t*dev_handle (handle to the opened device)
rf_params_t *rf_params (rf_params)

Description: sc5511a_get_rf_parameters gets the current RF parameters such as RF1 frequency, RF2 frequency, and sweep start frequency, etc.

Function: **sc5511a_get_temperature**

Definition: int sc5511a_get_temperature (sc5511a_device_handle_t *dev_handle, float *temperature)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
float *temperature (temperature)

Description: sc5511a_get_temperature retrieves the device temperature.

Function: **sc5511a_get_signal_phase**

Definition: int sc5511a_get_signal_phase (sc5511a_device_handle_t *dev_handle, float *phase)

Input: sc5511a_device_handle_t *dev_handle (handle to the opened device)
float *phase (phase)

Description: sc5511a_get_signal_phase obtains the current relative phase of the signal on CH1.

Function: **sc5511a_get_device_status**

Definition: int sc5511a_get_device_status(sc5511a_device_handle_t*dev_handle, device_status_t *device_status)

Input: sc5511a_device_handle_t*dev_handle (handle to the opened device)
device_status_t *device_status (current device status)

Description: sc5511a_get_device_status gets the current device status such as the PLL lock status, sweep modes, and other operating conditions.

Example: Code showing how to use function:

```
device_status_t *dev_status;

dev_status = (device_status_t *)malloc(sizeof(device_status_t));

int status = sc5511a_get_device_status(dev_handle, dev_status);

if(dev_status->pll_status.ref_100_pll_ld)
printf("The 100 MHz is phase-locked \n");
else
printf("The 100 MHz is not phase-locked \n");
```

Function: **sc5511a_get_device_info**

Definition: int sc5511a_get_device_info(sc5511a_device_handle_t*dev_handle,
device_info_t *device_info)

Input: sc5511a_device_handle_t*dev_handle (handle to the opened device)

Output: device_info_t *device_info (device information)

Description: sc5511a_get_device_info obtains the device information such as serial number, hardware revision, firmware revision, and manufactured date.

Function: **sc5511a_list_buffer_read**

Definition: int sc5511a_list_buffer_read(sc5511a_device_handle_t*dev_handle,
unsigned int address, unsigned long long int *freq)

Input: sc5511a_device_handle_t*dev_handle (handle to the opened device)
unsigned int address (buffer offset address)

Output: unsigned long long int *freq (frequency)

Description: sc5511a_list_buffer_read reads the frequency at an offset address of the list buffer.

Function: **sc5511a_get_alc_dac**

Definition: int sc5511a_get_alc_adc(sc5511a_device_handle_t*dev_handle,
unsigned short *dac_value)

Input: sc5511a_device_handle_t*dev_handle (handle to the opened device)

Output: unsigned short *dac_value (DAC value)

Description: sc5511a_get_alc_adc retrieves the current value of the ALC DAC which set the power level of channel RF1.

Function: **sc5511a_get_clock_config**

Definition: int sc5511a_get_clock_config(sc5511a_device_handle_t*dev_handle,
clock_config_t *clock_config)

Input: sc5511a_device_handle_t*dev_handle (handle to the opened device)

Output: clock_config_t *clock_config (Clock config struct)

Description: sc5511a_get_clock_config retrieves the current reference clock configuration.

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 16 of the interface 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 16 is pulled high or left unconnected, mode 1 is select.

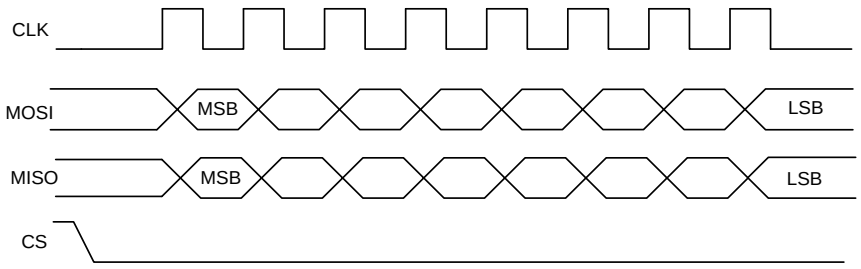

Figure 3. SPI Mode 0 shown.

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 $1\ \mu s$ (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 ($T_C = 0.2\ \mu s$), however if the external SPI signals do not have sufficient integrity due trace issues, the rate should be lowered.

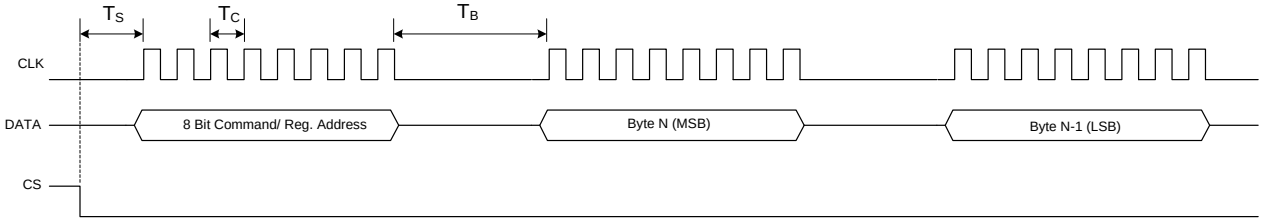

Figure 4 SPI timing.

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 Setting the SC5511A: 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 Setting the SC5511A: Configuration 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”.

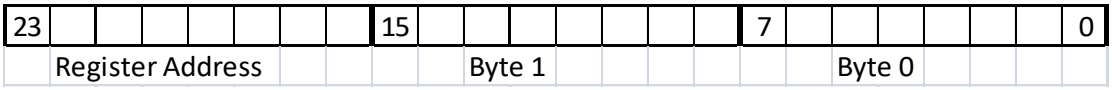

Figure 5. Single 3 byte transfer buffer.

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 (0x26) and 5 empty bytes (MOSI) to clock out the returned data (MISO). An example of reading the RF parameters (list_start_freq) from the device is shown in *Figure 6.*

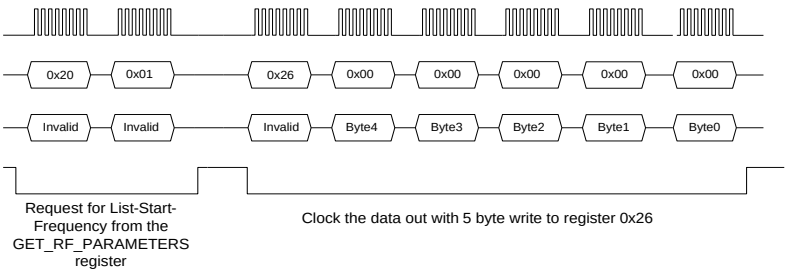

Figure 6. Reading queried data.

In the above example, valid data is present in the last 5 bytes - byte 4 down to byte 0, where byte 4 is the most significant byte. Table 47 shows the valid data bytes associated with each of the querying registers, while Table 48 shows the valid bytes associated with the requested contents of the register.

Table 47. Valid returned data bytes.

Register Name (Address)	Register Code (Hex)	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
GET_RF_PARAMETERS	0x20	valid	valid	valid	valid	valid
GET_TEMPERATURE	0x21	valid	valid	valid	valid	invalid
GET_DEVICE_STATUS	0x22	valid	valid	valid	valid	invalid
GET_DEVICE_INFO	0x23	valid	valid	valid	valid	invalid
GET_LIST_BUFFER	0x24	valid	valid	valid	valid	valid
GET_ALC_DAC_VALUE	0x25	valid	valid	invalid	invalid	invalid

Table 48 Return Valid Data for GET_RF_PARAMETERS Register

Data Return Name	Data Sent	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
RF1 Frequency	0x00	valid	valid	valid	valid	valid
List Start Frequency	0x01	valid	valid	valid	valid	valid
List Stop Frequency	0x02	valid	valid	valid	valid	valid
List Step Frequency	0x03	valid	valid	valid	valid	valid
Sweep Dwell Time	0x04	valid	valid	valid	valid	invalid
Sweep Cycles	0x05	valid	valid	valid	valid	invalid
Sweep Buffer Points	0x06	valid	valid	valid	valid	invalid
RF1 Power Level*	0x07	valid	valid	valid	valid	invalid
RF2 Frequency	0x08	valid	valid	invalid	invalid	invalid
Signal Phase*	0x0A	valid	valid	valid	valid	invalid

While all return data for the GET_RF_PARAMETERS Register (0x20) are unsigned integers, the RF1 power level and phase are returned as a 4-byte flattened floats which need to be re-casted back to a float using *(float*)& in C/C++.Programming the RS-232 Interface

The RS-232 version of the SC5511A 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 49 below.

Table 49. RS-232 communication settings.

Baud rate	Rate of transmission. Pin 16 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. Upon the execution of the register command that was sent, the device will return one (1) byte of data with bit 1 high. This byte must be read by the host to clear its receive buffer so that reading subsequent registers will not contain corrupted data. Furthermore, reading back this byte will ensure that the device is ready for the next register command.

Reading from the Device via RS-232

To query information from the device, the query registers are addressed and data is returned. Returned data vary in length, which are dependent on the register call. Table 38 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 returned data length is also detailed in the Querying the SC5511A: Query Registers section. Returned valid data are detailed in Table 47 and Table 48.

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 sc5511a_rs232.dll library and with the exception of 2 functions, namely sc5511a_search_devices and sc5511a_open_device, all functions are identical to those for USB communication. Please refer to the USB Driver API function descriptions and the sc5511a_rs232.h header file for proper usage. A C/C++ programming example is also provided under the examples subdirectory. For driver support off 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 SC5511A_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 SC5511A is factory calibrated and ships with a certificate of calibration. SignalCore strongly recommends that the SC5511A 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.

Should any customer need to reload calibration data for their SC5511A, 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 SC5511A.

The SC5511A 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 0.1	Development release	Date
Rev 1.0	Initial Release	4-23-15
Rev 1.0.1	Corrected table 43 for register 25	5-13-15
Rev 1.0.2	Corrected API function syntax	
Rev 1.0.3	Corrected figure 6 wording	7-20-15
Rev 1.1.0	Added RS-232 section	7-23-15
Rev 1.2.0	Modified function sc5511a_set_synth_mode. Only for firmware v1.1 or later	
Rev 1.3.0	Added the sc5511a_synth_self_cal function.	
Rev 1.4.0	Modified RS232 write instructions for devices with firmware >2.1	10-21-16
Rev 1.5.0	1) Added ability to change the phase of the signal on CH1 with resolution limitation for lower frequencies. Only for firmware versions 2.5 and higher. Contact SignalCore for firmware updates. 2) Corrected and updated Table 39.	08-03-18
Rev 1.6.0	Function deleted, not applicable.	1-29-19
Rev 1.7.0	Corrected API function syntax	5-22-19
Rev 1.7.1	Corrected description of pin 22 in Table 2	10-12-20
Rev 1.8.0	Added documentation on *get_temperature()* function	2-8-21
Rev 1.8.1	Changed “High to low transition” to “ Low to high transition”	3-3-21
Rev 1.8.2	Updated p. 39 to Mode 0: Data clocked in and out on rising edge; Mode 1: data clocked in and out on falling edge.	6-21-21
Rev 1.8.3	Corrected torque range	12-07-22
Rev 1.9.0	1. Added modifications to register 0x22 to read back the reference input configuration. 2. Added function sc5510a_get_clock_config () to retrieve reference configuration 3. Modified sc5511a_set_clock_reference() function. These changes are valid only for hardware versions 16 or later. API versions 2.2.0 or later are backward compatible with older hardware.	1-11-24
Rev 1.9.1	Revised figure 2 to reflect reference clocking changes in hardware revision 16 or later.	5-14-24