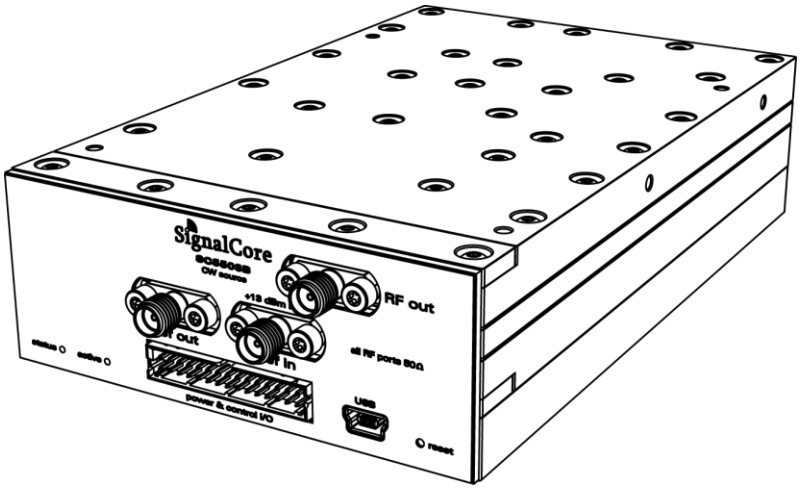
SignalCore™
PRESERVING SIGNAL INTEGRITY




# SC5503B
# 50 MHz to 10 GHz RF Signal Source
# USB, SPI and RS-232 Interfaces

# Operating and Programming Manual


support@signalcore.com

# CONTENTS

# IMPORTANT INFORMATION

## Warranty

*The warranty terms and conditions for all SignalCore products are also provided on our corporate website. Please visit http://www.signalcore.com/warranty for more information.*

This product is warranted against defects in materials and workmanship for a period of one year 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<br>**Model Name** | 鉛<br>**Lead (Pb)** | 汞<br>**Mercury (Hg)** | 镉<br>**Cadmium (Cd)** | 六价铬<br>**Hexavalent Chromium (Cr(VI))** | 多溴联苯<br>**Polybrominated biphenyls (PBB)** | 多溴二苯醚<br>**Polybrominated diphenyl ethers (PBDE)** |
|---|---|---|---|---|---|---|
| SC5503B | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

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

## CE European Union EMC & Safety Compliance Declaration

The European Conformity (CE) marking is affixed to products with input of 50 - 1,000 Vac or 75 - 1,500 Vdc and/or for products which may cause or be affected by electromagnetic disturbance. The CE marking symbolizes conformity of the product with the applicable requirements. CE compliance is a manufacturer's self-declaration allowing products to circulate freely within the European Union (EU). SignalCore products meet the essential requirements of Directives 2004/108/EC (EMC) and 2006/95/EC

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

## Warnings Regarding Use of SignalCore Products

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

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

# GETTING STARTED

## Unpacking

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

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

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

## Verifying the Contents of your Shipment

Verify that your SC5503B kit contains the following items:

| Quantity | Item |
|---|---|
| 1 | SC5503B USB/RS-232 RF CW generator |
| 1 | USB flash drive with installation software |

## Setting Up and Configuring the SC5503B

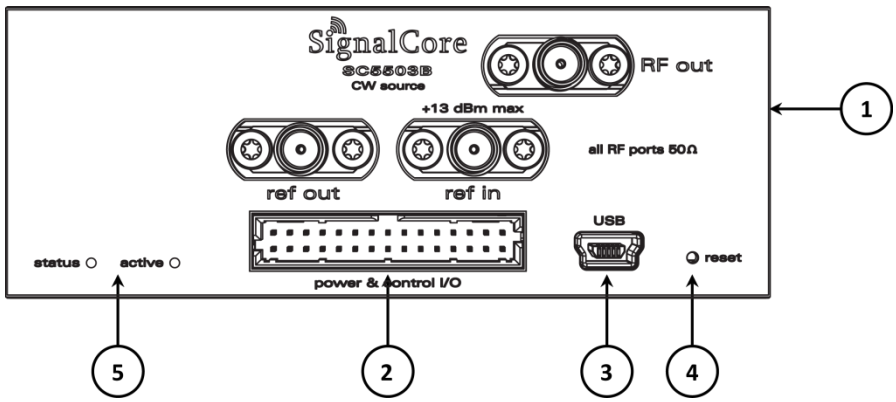



**Figure 1. Front view of the SC5503B showing I/O connection locations.**

The SC5503B is a core module-based RF CW generator with all user I/O located on the front face of the module as shown in Figure 1. Each location is discussed in further detail below.

## 1 Signal Connections

All signal connections (ports) to the SC5503B are SMA-type. Exercise caution when fastening cables to the SMA 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 8.8 in-lb max (100 N-cm max)***

**RF OUT** RF output port with frequency range from 50 MHz to 10 GHz with nominal input impedance is 50 Ω. The port is AC coupled.

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

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

## 2 Communication and Supply Connection

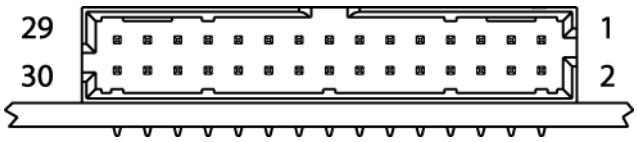


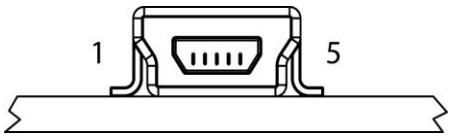

**Figure 2 Power and I/O connector**

Power and communication to the device is provided through a Molex **Milli-Grid**[M] 2.00mm pitch, 30 position, male header connector, whose part number is 87833-3020. A suggested receptacle female connector is the crimp terminal type 51110-3051 or ribbon type 87568-3093 from Molex. The pin definitions of this I/O connector are listed below.

**Communication IO connector pin definitions**

| PIN # | | 21 | Device Active |
|---|---|---|---|
| 1,3 | 12V Supply rail | 22 | Device Status |
| 2,4,10,16, 20,24,28 | GND | 23 | Baudrate/Spi Mode |
| 5,6,7,8,9, 15,18,21 | Reserved, do not connect | 25 | CTS/$\overline{CS}$. This pin is CTS for RS232 or device select for SPI |
| 11 | USB+ | 26 | SPI Clock |
| 13 | USB- | 27 | TX/MISO. This pin is TX for RS232 or MISO for SPI |
| 14 | USB_VBUS | 29 | RX/MOSI. This pin is RX for RS232 or MOSI for SPI |
| 17 | Reserved, pull high to 3.3V or DNC | 30 | RTS/SRDY. This pin is RTS for RS232 or serial ready for SPI |
| 19 | $\overline{Reset}$, logic 0 to reset device | | |

## 3 Mini-USB Connection




| Pin | 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 SC5503B uses a mini-USB Type B connector for USB communication with the device using the standard USB 2.0 protocol (full speed) found on most host computers. The pinout of this connector, viewed from the board edge, is shown in the table above.

## 4 Reset Button (Pin Hole)

Behind this pin hole is the reset button. Using a pin and lightly depressing this momentary-action push button switch will cause a hard reset to the device, putting it back to its default settings. All user settings will be lost. System reset capability can also be accessed through the communication header connector.

## 5 Indicator LEDs

The SC5503B provides visual indication of important modes. There are two redundant pairs of LED indicators on the device, two to the left of the micro-D connector and two on the opposite side of the module. The LED closest to the micro-D connector indicates STATUS. The LED to the left of the STATUS LED indicates ACTIVE. The pair of LEDs on the opposite side of the module are oriented the same way. Their behavior under different operating conditions is shown in the table below.

| LED | Color | Definition |
|---|---|---|
| STATUS | Green | "Power good" and all oscillators phase-locked |
| 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 |
| ACTIVE | Orange | User initiated standby mode |

## Output Amplitude Control

As shown in Figure 3, the SC5503B 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 DSA provides the coarse-step tuning over a wide range while the VCA provides fine tune correction to the DSA. 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, and 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 voltages levels that correspond to calibrated output power levels.

There are advantages and disadvantages with 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 primarily 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 affect the overall output amplitude accuracy. In particular, when the filters within the filter bank are switched from one bank to another the signal experiences abrupt discontinuities in its amplitude which the open loop calibration cannot appropriately account for in its correction algorithm.

Setting of the amplitude control components are performed automatically by the system, although the user may chose to override the ALC DAC value if needed. In Figure 3, the labels in red indicate parameters or devices which the user has direct control over.

## Frequency Synthesizer

The synthesizer section of the SC5503B 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 regions of approximately 1 kHz to 30 kHz. The 100 MHz VCXO provides the reference signal to the main RF signal synthesizer comprised of three phase-locked loops and a direct digital synthesis (DDS) oscillator. The "fine" PLL provides a tuning resolution of few mHz over a narrow frequency range, while the "coarse" PLL tunes in steps of a few MHz over several GHz of range. The "main" or summing PLL combines the signals of the "coarse" and "fine" loops into one broad tuning signal with fine stepping.

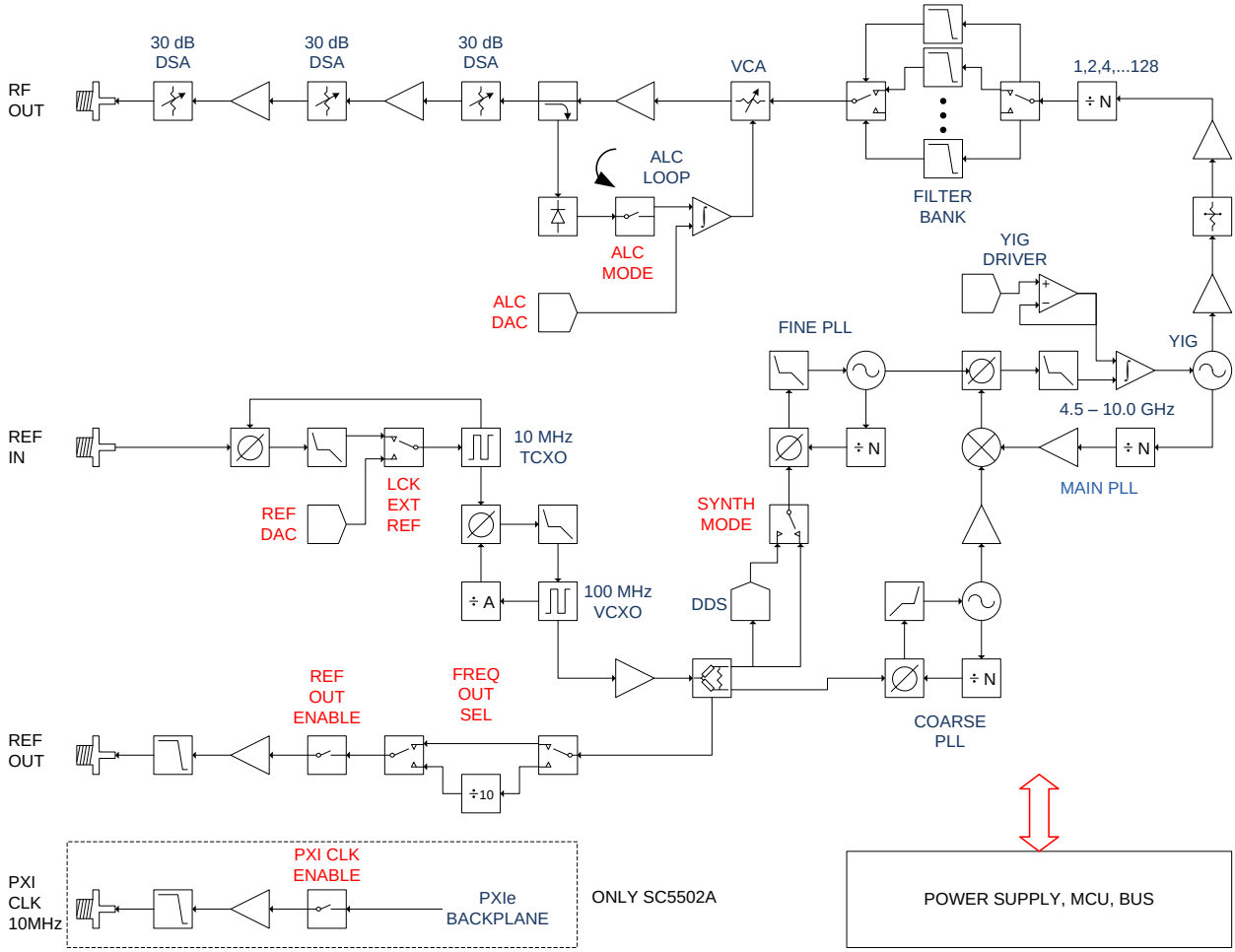



**Figure 3. Simplified block diagram of the SC5503B RF signal source.**

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

VCO because of the YIG's superior far-out phase nose performance in the frequency offset regions of 100 kHz to 10 MHz.

## Frequency Tuning Modes

The synthesizer has two sets of control parameters that can be explored to optimize the device for a particular application. The first set of parameters, TUNE SPEED, sets the tuning and phase locking time as frequency is changed. TUNE SPEED comprises two modes - Fast Tune mode and Normal mode, both of which affect the YIG oscillator configuration. The Fast Tune mode deactivates a noise suppression capacitor across the tuning coil of the YIG oscillator. Doing so increases the rate of current flow through the coil and hence increases the available rate of frequency change. In Normal mode, the capacitor is activated, slowing down the available rate of frequency change. The advantage of activating the capacitor is that its presence in the circuit shunts the noise developed across the coil, decreasing close-in phase noise.

The other set of control parameters, FINE TUNE, sets the tuning resolution of the device. There are three modes to FINE TUNE: 1 MHz, 25 kHz, and 1 Hz steps. In the first two modes, the "fine" PLL uses the VCXO signal directly to synthesize its frequency and step size, while in the third mode it uses a DDS to provide 1 Hz resolution. The PLL-only modes provide the ability to realize exact frequencies with tuning as fine as 25 kHz. These modes offer several advantages including lower phase spurs and less computational demand required to set a new frequency. The 1 MHz mode has the lowest phase spurious signals, below the level of the product specifications. The DDS mode also tunes to exact frequencies. However, it requires many more computing cycles and several more internal device registers to write to in order to set a new frequency. Comparing setup times, the PLL-only modes require up to 175 microseconds to compute and setup the device for a new frequency, whereas the DDS mode can require up to 350 microseconds to perform the same action. At first glance it may seem that these differences could impact the frequency tuning times. However, the tuning times are predominantly set by the physical parameters of the YIG oscillator. Computation and register writes typically account for less than 25% of the total tune time of a 10 MHz step change in frequency.

## Reference Clock Control

As mentioned above, the primary clock reference for the SC5503B 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 either a 10 MHz or 100 MHz reference signal. To adjust the accuracy of the TCXO as needed, for example, to correct for long-term accuracy drift, the user may vary the reference DAC voltage by writing the **REFERENCE_DAC_SETTING** register.

The SC5503B can be programmed to lock to an external reference usually for better stability and accuracy, or for synchronizing the device to the reference. The reference source must be connected to the Ref In port. The device reference can be exported out the Ref Out port, and the user can choose to export a 10 MHz or 100 MHz signal.

## Harmonics and Range Operation

The SC5503B's guaranteed operating frequency range is 50 MHz to 10 GHz. However, typically the device is capable of tuning as low as 35 MHz and as high as 10.2 GHz. At these out-of-range regions, the amplitude may not be within specification but the frequency accuracy will not be affected. At low frequencies of operation (200 MHz and lower), the harmonics of the signal can potentially be observed as high as -10 dBc at 0 dBm output. This is due to the limited space available for additional filtering in these ranges. At lower frequencies, the large physical size of appropriate filters makes it impossible to accommodate them within the compact form of this device. Furthermore, as the low frequencies are synthesized through frequency dividers, their output waveforms become more "square" than sinusoidal, giving rise to higher odd-order harmonics.

The device is specified to a maximum calibrated level of +10 dBm for frequencies up to 9 GHz, and +7 dBm for frequencies greater than that, 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.

## Device Standby and RF Enable

The SC5503B has both standby and RF output enable features. The user may wish to put the device into standby mode to reduce power consumption and thus lower the operating temperature of the device under the same environmental conditions. Taking the device out of standby requires the device to wait for the power rails to settle and all internal components to be reprogrammed, usually about one second.

Disabling the RF output moves the frequency to some very low value so that the step attenuators (DSA) and voltage controlled attenuators (VCA) have the most effective attenuation. This will push the signal level below -100 dBm. Enabling the RF output is nearly instantaneous as all components remain active even when the output is disabled.

## Default Startup Mode

The factory power-up state for the device is detailed in Table 1. The default state can be changed to the current state programmatically, allowing the user to power up the device in the last saved state without having to reprogram.

**Table 1 Factory default power-up state**

| | |
|---|---|
| **Frequency** | 5.0 GHz |
| **Power** | 0.00 dBm |
| **RF Output** | Enabled |
| **ALC Mode** | Closed Loop |
| **Standby** | Disabled |
| **Auto Level** | Enabled |
| **Ref Out** | Disabled |
| **Ext Ref Lock** | Disabled |

# SC5503B PROGRAMMING INTERFACE

## USB Device Drivers

The SC5503B 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 **SC5503B_RegWrite** and **SC5503B_RegRead** functions respectively.

For Microsoft Windows™ operating systems, The SC5503B API is provided as a dynamic linked library, *SC5503B.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 *SC5503B.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 *SC5503B.inf* file under 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 *SC5503B_Readme.txt* file in the same directory as the *SC5503B.inf* file.

For LabVIEW™ support, a full LabVIEW API is provided and is available under 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 wrappers around SC5503B.dll, however code written purely in LabVIEW-G that does not call or depend on external library functions is available on request. Use the National Instruments driver wizard that comes with NI-VISA to create a driver in the operating system of choice if pure G code is used. The Vendor ID is **0x277C** and the PID is **0x0015**.

The Linux driver is in the Linux/ directory. Please read the ReadMe.txt file for installation and compilation instructions.

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 SC5503B. Driver code based on libusb-1.0 is available to our customers by 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 SC5503B 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 **SC5503B_GetDeviceTemperature**, or calls **SC5503B_SetFrequency** to tune the frequency. The software API is covered in detail in the "Software API Library Functions" section.

# SETTING THE SC5503B: WRITING TO CONFIGURATION REGISTERS

## Configuration Registers

The users may write the configuration registers (write only) directly by calling the **SC5503B_RegWrite** function. The syntax for this function is **SC5503B_RegWrite(deviceHandle, registerCommand, instructWord)**. The **instructWord** takes a 64 bit-word. However, it will only send the required number of bytes to the device. **These registers are the same for USB, SPI, and RS232;** see the SPI and RS232 sections for data transfer details. Table 2 summarizes the register addresses (commands) and the effective bytes of command data.

**Table 2. 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 |
| DEVICE_STANDBY | 0x05 | [7:0] | Open | Open | Open | Open | Open | Open | Open | Mode |
| RF_FREQUENCY | 0x10 | [7:0] | Fine Frequency Adjust Word [7:0] | | | | | | | |
| | | [15:8] | Fine Frequency Adjust Word [15:8] | | | | | | | |
| | | [23:16] | Fine Frequency Adjust Word [23:16] | | | | | | | |
| | | [31:24] | Fine Frequency Adjust Word [31:24] | | | | | | | |
| | | [39:32] | Fine Frequency Adjust Word [39:32] | | | | | | | |
| RF_POWER | 0x11 | [7:0] | RF Power Word [7:0] | | | | | | | |
| | | [15:8] | Sign Bit | RF Power Word [14:8] | | | | | | |
| SYNTH_MODE | 0x12 | [7:0] | | | | | | Fast Tune | Fine Tune Modes | |
| RF_ALC_MODE | 0x13 | [7:0] | Open | Open | Open | Open | Open | Open | Open | Mode |
| SET_ALC DAC_VALUE | 0x14 | [7:0] | ALC DAC Word [7:0] | | | | | | | |
| | | [15:8] | ALC DAC Word [13:8] | | | | | | | |
| REFERENCE_MODE | 0x15 | [7:0] | Open | Open | Open | Open | Open | 100 MHz Out Enable | Ref Out Enable | Lock Enable |
| REFERENCE_DAC_SETTING | 0x16 | [7:0] | REFERENCE DAC Word [7:0] | | | | | | | |
| | | [15:8] | Open | Open | REFERENCE DAC Word [13:8] | | | | | |
| USER_EEPROM_WRITE | 0x23 | [7:0] | DATA [7:0] | | | | | | | |
| | | [15:8] | EEPROM Address [15:8] | | | | | | | |
| | | [23:16] | EEPROM Address [23:16] | | | | | | | |
| RF_OUT_ENABLE | 0x26 | [7:0] | Open | Open | Open | Open | Open | Open | Open | Mode |
| STORE_STARTUP_STATE | 0x28 | [7:0] | Open | Open | Open | Open | Open | Open | Open | Open |
| AUTO_POWER_DISABLE | 0x29 | [7:0] | Open | Open | Open | Open | Open | Open | Open | Mode |

To write to the device through USB transfers such as bulk transfer, it is important to send data with the register byte first, followed with the MSB of the data bytes. For example to set the RF level to some power amplitude, the byte stream is [0x11][15:8][7:0].

## Initializing the Device

**INITIALIZE (0x01)** - Writing 0x00 to this register will reset the device to the default power-on state. Writing 0x01 will reset the device but leave it in the current state. The user has the ability to define the default startup state by writing to the **START UP STATE (0x28)** 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 Device Standby Mode

**DEVICE_STANDBY (0x05)** - Writing 0x01 to this register will power-down the analog/RF circuitry. Writing 0x00 to this register will enable the analog/RF circuitry and the device will return to its last programmed state.

## Setting the RF Frequency

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

## Setting the RF Power

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

## Setting the Synthesizer Mode

**SYNTH_MODE (0x12)** - This register has one data byte and provides two tuning modes for the device - Fast Tune and Fine Tune. By default the Fast Tune is disabled (normal mode). Asserting high bit 2 of the data byte will enable Fast Tune. Fast Tune enables the device for faster lock and faster settling times between frequency changes. The Fine Tune mode has 3 options - 1 MHz (PLL), 25 kHz (PLL), and 1 Hz (DDS). Selection of these options requires setting the first 2 bits of the data byte to 0, 1, and 2, respectively. See the "Frequency Tuning Modes" section for more information. As an example, to set the device for Fast Tune and step at 1 Hz resolution, write 0x06.

## Setting the RF Automatic Level Control (ALC) Mode

**RF_ALC_MODE (0x13)** – 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 RF ALC DAC Value

**SET_ALC_DAC_VALUE (0x14)** - 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.

## Setting the Reference Clock

**REFERENCE_MODE (0x15)** - This register sets the behavior of the reference clock section. Bit 3 enables (1) or disables (0) the PXI 10 MHz clock, Bit 2 selects whether the output reference signal is 10 MHz (0) or 100 MHz (1), 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.

## Setting the Reference DAC Value

**REFERENCE_DAC_SETTING (0x16)** - 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.

## Writing to the User EEPROM

**USER_EEPROM_WRITE (0x23)** - 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 RF Output Enable

**RF_OUT_ENABLE (0x26)** - This register enables or disables the RF signal output. Setting bit 0 low (0) disables RF output. Setting bit 0 high (1) enables RF output.

## Storing the Startup State

**STORE_STARTUP_STATE (0x28) –** 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.

## Disabling the Auto Power Feature

**AUTO_POWER_DISABLE (0x29)** - When changing frequency, 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 adjust is enabled. To disable this feature, set bit 0 high (1).

# QUERYING THE SC5503B: WRITING TO REQUEST REGISTERS

The registers to read data back from the device (such as device status) are accessed through the **SC5503B_RegRead** function. The function and parameter format for this command is **SC5503B_RegRead(deviceHandle, registerCommand, instructWord,*dataOut)**. Any instruction in addition to the register call is placed into "instructWord", and data obtained from the device is returned via the pointer value dataOut. The set of request registers are shown in Table 3.

**Table 3. 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_DEVICE_STATUS | 0x17 | [7:0] | Open | Open | Open | Open | Open | Open | Open | Open |
| GET_TEMPERATURE | 0x18 | [7:0] | Open | Open | Open | Open | Open | Open | Open | Open |
| CAL_EEPROM_READ | 0x20 | [7:0] | EEPROM Address [7:0] | | | | | | | |
| | | [15:8] | EEPROM Address [15:8] | | | | | | | |
| USER_EEPROM_READ | 0x22 | [7:0] | EEPROM Address [7:0] | | | | | | | |
| | | [15:8] | EEPROM Address [15:8] | | | | | | | |
| GET_ALC_DAC_VALUE | 0x2A | [7:0] | Open | Open | Open | Open | Open | Open | Open | Open |

To read from the device using native USB transfers instead of the **SC5503B_RegRead** 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 3 bytes. See the register details below.

## Reading the Device Status

**GET_DEVICE_STATUS (0x17)** - This register, summarized in Table 4, returns the device status information such as phase lock status of the PLL, current reference settings, etc. Data is contained in the first three bytes.

**Table 4. Description of the status data bits.**

| Bit | Description |
|---|---|
| [15] | 10 MHz TCXO PLL lock status |
| [14] | 100 MHz VCXO PLL lock status |
| [13] | Main PLL lock status |
| [12] | Reserved |
| [11] | Reserved |
| [10] | Fine PLL lock status |
| [9] | Coarse PLL lock status |

| Bit | Description |
|---|---|
| [8] | Reserved |
| [7] | External reference detected |
| [6] | Reference output enabled |
| [5] | Reference lock enabled |
| [4] | ALC mode |
| [3] | Fast tune state |
| [2] | Device standby state |
| [1] | RF out state |
| [0] | PXI clock state (only used for PXI versions) |

## Reading the Device Temperature

**GET_TEMPERATURE (0x18)** - Data returned by this register needs to be processed to correctly represent data in temperature units of degrees Celsius. Data is returned in the first 14 bits [13:0]. Bit [13] is the polarity bit indicating whether it is positive (0x0) or negative (0x1). The temperature value represented in the raw data is contained in the next 13 bits [12:0]. To obtain the temperature ADC code, the raw data should be masked (bitwise AND'ed) with 0x1FFF, and the polarity should be masked with 0x2000. The conversion from 12 bit ADC code to an actual temperature reading in degrees Celsius is shown below:

Positive Temperature (bit 13 is 0) = ADC code / 32

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

It is not recommended to read the temperature too frequently, especially once the SC5503B 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 SC5503B 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 Calibration EEPROM

**CAL_EEPROM_READ (0x20)** - Reading a single byte from an address in the device EEPROM is performed by writing this register with the address for the instructWord. The data is returned as a byte. The CAL EEPROM maximum address is 0x7FFF. Reading above this address will cause the device to retrieve data from the lower addresses. For example, addressing 0x8000 will return data stored in address location 0x0000. The calibration EEPROM map is shown in **Error! Reference source not found.**

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

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

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

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

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

## Reading the User EEPROM

**USER_EEPROM_READ (0x22)** - Once data has been written to the user EEPROM, it can be retrieved by calling this register and using the process outlined above for reading calibration data. The maximum address for this EEPROM is also 0x7FFF.

## Reading the RF ALC DAC Value

**GET_ALC_DAC_VALUE (0x2A)** - The user may be interested to obtain the current value of the ALC DAC for the purpose of making minor adjustments to the RF output power level. Data is returned in the first 14 bits.

# CALIBRATION EEPROM MAP

**Table 5. Calibration EEPROM map.**

| EEPROM ADDRESS (HEX) | # DATA POINTS | TYPE | DESCRIPTION |
|---|---|---|---|
| 0 | 1 | U32 | Manufacturing Information |
| 4 | 1 | U32 | Product serial number |
| 8 | 1 | U32 | RF module number |
| C | 1 | U32 | Product manufacture date |
| 10 | 1 | U32 | Last calibration date |
| 14 | 4 | U32 | Reserved |
| 2C | 1 | F32 | Firmware revision |
| 30 | 1 | F32 | Synthesizer CCA hardware revision |
| 34 | 3 | F32 | Signal conditioning CCA hardware revision |
| 40 | 2 | U32 | Startup RF frequency |
| 48 | 1 | F32 | Startup RF power |
| 4C | 1 | U32 | Other startup states |
| 50 | 1 | F32 | Calibration temperature |
| 54 | 1 | U32 | TCXO DAC calibration value |
| 58 | 112 | U8 | Reserved |
| C8 | 40 | F32 | YIG calibration frequency |
| 168 | 40 | U32 | YIG calibration DAC values |
| 208 | 40 | F32 | YIG spline interpolants |
| 2A8 | 88 | U8 | Reserved |
| 300 | 5 | F32 | ALC enabled temperature coefficient |
| 314 | 5 | F32 | ALC disabled temperature coefficient |
| 328 | 1 | U8 | Reference attenuation value |
| 329 | 245 | U16 | ALC calibration frequencies (MHz) |
| 513 | 125 | U16 | Attenuator calibration frequencies (MHz) |
| 60D | 245 | F32 | ALC close reference RF power |
| 9E1 | 245 | U16 | ALC close reference DAC values |
| BCB | 735 | F32 | ALC close coefficient |
| 1747 | 245 | F32 | ALC open reference RF power |
| 1B1B | 245 | U16 | ALC open reference DAC value |
| 1D05 | 735 | F32 | ALC open coefficient |
| 2881 | 5520 | F32 | Attenuator calibration values |

# 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 SC5503B API dynamic linked library or LabVIEW library are:

- **SC5503B_ SearchDevices**
- **SC5503B_OpenDevice**
- **SC5503B_CloseDevice**
- **SC5503B_RegWrite**
- **SC5503B_RegRead**
- **SC5503B_InitDevice**
- **SC5503B_SetStandby**
- **SC5503B_SetFrequency**
- **SC5503B_SetPowerLevel**
- **SC5503B_SetRfOut**
- **SC5503B_SetAlcMode**
- **SC5503B_SetAlcDac**
- **SC5503B_SetSynthesizerMode**
- **SC5503B_SetReferenceClock**
- **SC5503B_SetReferenceDac**
- **SC5503B_WriteUserEeprom**
- **SC5503B_StoreCurrentState**
- **SC5503B_DisableAutoLevel**
- **SC5503B_GetDeviceInfo**
- **SC5503B_GetDeviceStatus**
- **SC5503B_GetTemperature**
- **SC5503B_GetAlcDac**
- **SC5503B_ReadCalEeprom**
- **SC5503B_ReadUserEeprom**

Each of these functions is described in more detail on the following pages. Example code in C/C++ in the *Win**Driver**src* directory is available to show how these functions are called and used. First, for C/C++ we define the constants and types which are contained in the C header file, *SC5503B.h*. These constants and types are useful not only as an include for developing user applications using the SC5503B API, but also for writing device drivers independent of those provided by SignalCore.

## Constants Definitions

```
// Parameters for storing data in the onboard EEPROMs
#define CALEEPROMSIZE            32768    // size in bytes
#define USEREEPROMSIZE           32768    // size in bytes

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

// Define error codes
#define SUCCESS                      0
#define DEVICEERROR                 -1
#define DATAERROR                   -2
#define INPUTNULL                   -3
#define COMMERROR                   -4
#define INPUTNOTALLOC               -5
#define EEPROMOUTBOUNDS             -6
#define INVALIDARGUMENT             -7
#define INPUTOUTRANGE               -8
#define NOREFWHENLOCK               -9
#define NORESOURCEFOUND            -10
#define INVALIDCOMMAND             -11

// Define device registers
#define INITIALIZE                0x01    // initialize the device
#define SET_SYSTEM_ACTIVE         0x02    // set the device “active” LED
#define DEVICE_STANDBY            0x05    // place the device into standby mode
#define RF_FREQUENCY              0x10    // set the frequency
#define RF_POWER                  0x11    // set the power level of the RF output
#define SYNTH_MODE                0x12    // synth mode control
#define RF_ALC_MODE               0x13    // closed or open loop ALC mode
#define SET_ALC_DAC_VALUE         0x14    // set the RF ALC DAC value
#define REFERENCE_MODE            0x15    // reference clock settings
#define REFERENCE_DAC_SETTING     0x16    // set reference clock DAC
#define GET_DEVICE_STATUS         0x17    // read the device status
#define GET_TEMPERATURE           0x18    // get the internal temperature of he device
#define CAL_EEPROM_READ           0x20    // read a byte from the calibration EEPROM
#define CAL_EEPROM_WRITE          0x21    // cal EEPROM write (not accessible)
#define USER_EEPROM_READ          0x22    // read a byte from the user EEPROM
#define USER_EEPROM_WRITE         0x23    // write a byte to the user EEPROM
#define RF_OUT_ENABLE             0x26    // enable RF output
#define STORE_STARTUP_STATE       0x28    // store new default state
#define AUTO_POWER_DISABLE        0x29    // disable auto power leveling
```

**Type Definitions**

```
typedef    struct deviceInfo_t
{
           unsigned int productSerialNumber;
           unsigned int rfModuleSerialNumber;
           float firmwareRevision;
           float synthHardwareRevision;
           float signalHardwareRevision;
           unsigned int calDate; // year, month, day, hour:&(0xFF000000,0x00FF0000,
                                 0x0000FF00,0x000000FF)
           unsigned int manDate; // year, month, day, hour:&(0xFF000000,0x00FF0000,
                                 0x0000FF00,0x000000FF)
}          deviceInfo_t;
```

```
typedef    struct deviceStatus_t
{
           bool tcxoPllLock;//Master 10 MHz TCXO
           bool vcxoPllLock;//100 MHz VCXO
           bool finePllLock;//Fine Tuning PLL
           bool coarsePllLock; //Coarse tuning PLL
           bool sumPllLock;    //Main tuning PLL
           bool extRefDetected; //Indicates whether an external source is detected
           bool refClkOutEnable; //Indicates if the reference output port is enabled
           bool extRefLockEnable; //Indicates if the TCXO lock is enabled
           bool alcOpen; // Indicates that the ALC is in open loop
           bool fastTuneEnable; // Indicates if Tuning mode
           bool standbyEnable; // Power down of analog circuit
           bool rfEnable; // RF port is enabled
           bool pxiClkEnable; // Only used in PXIe platform
}          deviceStatus_t;
```

## Function Definitions and Usage

The functions listed below are found in the **SC5503B.dll** dynamic linked library for the Windows™ operating system. These functions are also provided in the LabView library, **SC5503B.llb**. The LabView functions contain context help (Cntrl-H) to help with the input and output parameters.

*Function:* **SC5503B_SearchDevices**

*Definition:* **int SC5503B_SearchDevices(char **serialNumberList)**

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

*Description:* **SC5503B_SearchDevices** searches for SignalCore SC5503B 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. The user can use this information to open specific device(s) based on their unique serial numbers. See **SC5503B_OpenDevice** function on how to open a device.

*Function:* **SC5503B_OpenDevice**

*Definition:* **deviceHandle *SC5503B_OpenDevice(char *devSerialNum)**

*Input:* **char** * devSerialNum (serial number string)

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

*Description:* **SC5503B_OpenDevice** opens the device and turns the front panel “active” LED on if it is successful. It returns a handle to the device for other function calls.

*Function:* **SC5503B_CloseDevice**

*Definition:* **int SC5503B_CloseDevice(deviceHandle *devHandle)**

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

*Description:* **SC5503B_CloseDevice** closes the device associated with the device handle and turns off the “active” LED on the front panel if it is successful.

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

```
// Declaring
devicehandle *devHandle; //device handle
int numOfDevices; // the number of device types found
char **deviceList;  // 2D to hold serial numbers of the devices found
int status; // status reporting of functions

deviceList = (char**)malloc(sizeof(char*)*MAXDEVICES); // MAXDEVICES serial numbers to
search
for (i=0;i<MAXDEVICES; i++)
       deviceList[i] = (char*)malloc(sizeof(char)*SCI_SN_LENGTH); // SCI SN has 8 char

numOfDevices = SC5503B_SearchDevices(deviceList); //searches for SCI for device type
if (numOfDevices == 0)
{
       printf("No signal core devices found or cannot not obtain serial numbers\n");
       for(i = 0; i<MAXDEVICES;i++) free(deviceList[i]);
       free(deviceList);
       return 1;
}
printf("\n There are %d SignalCore %s USB devices found. \n \n", numOfDevices,
SCI_PRODUCT_NAME);
       i = 0;
       while ( i < numOfDevices)
       {
              printf("      Device %d has Serial Number: %s \n", i+1, deviceList[i]);
              i++;
       }
//** SC5503B_OpenDevice, open device 0
devHandle = SC5503B_OpenDevice(deviceList[0]);
// Free memory
       for(i = 0; i<MAXDEVICES;i++) free(deviceList[i]);
       free(deviceList); // Done with the deviceList
 //
// Do something with the device
// Close the device
status = SC5503B_CloseDevice(devHandle);
```

*Function:* **SC5503B_RegWrite**

*Definition:* **int SC5503B_RegWrite(deviceHandle *devHandle, unsigned char commandByte, unsigned long long int instructWord)**

*Input:* **deviceHandle** *devHandle (handle to the opened device)
**unsigned char** commandByte (register address)
**unsigned long long int** instructWord (the data for the register)

*Description:* **SC5503B_RegWrite** writes the instructWord data to the register specified by the commandByte. See the register map on **Error! Reference source not found.** for more nformation.

*Example:* To set the power level to 2.00 dBm:

```
int status = SC5503B_RegWrite(devHandle, RF_POWER, 200);
```

*Function:* **SC5503B_RegRead**

*Definition:* **int SC5503B_RegRead(deviceHandle *devHandle, unsigned char commandByte, unsigned long long int instructWord, unsigned int *receivedWord)**

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

*Description:* **SC5503B_RegRead** reads the data requested by the instructWord data to the register specified by the commandByte. See the register map on Table 3 for more information.

*Example:* To read the status of the device:

```
unsigned int deviceStatus;

int status = SC5503B_RegRead(devHandle,
GET_DEVICE_STATUS,0x00,&deviceStatus);
```

*Function:* **SC5503B_InitDevice**

*Definition:* **int SC5503B_InitDevice(deviceHandle *devHandle, bool mode)**

*Input:* **deviceHandle** *devHandle (handle to the opened device)
**bool** mode (set the mode of initialization)

*Description:* **SC5503B_InitDevice** initializes (resets) the device. Mode = 0 resets the device to the default power up state. Mode = 1 resets the device but leaves it in its current state.

*Function:* **SC5503B_SetStandby**

*Definition:* **int SC5503B_SetStandby(deviceHandle *devHandle, bool standbyStatus)**

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

*Description:* **SC5503B_SetStandby** puts a channel in standby mode where the power to the analog circuits for that channel are disabled, conserving power.

*Function:* **SC5503B_SetFrequency**

*Definition:* **int SC5503B_SetFrequency(deviceHandle *devHandle,**
**unsigned long long int frequency)**

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

*Description:* **SC5503B_SetFrequency** sets the channel RF frequency.

*Function:* **SC5503B_SetPowerLevel**

*Definition:* **int SC5503B_SetPowerLevel(deviceHandle *devHandle, float powerLevel)**

*Input:* **deviceHandle** *devHandle (handle to the opened device)
**float** powerLevel (set power in dBm)

*Description:* **SC5503B_SetPowerLevel** sets the value of the desired output power level for the channel.

*Function:* **SC5503B_SetRfOut**

*Definition:* **int SC5503B_SetRfOut(deviceHandle *devHandle, bool mode)**

*Input:* **deviceHandle** *devHandle (handle to the opened device)
**bool** mode (disable/enable RF output)

*Description:* **SC5503B_SetRfOut** enables or disables the RF output on a channel.

*Function:* **SC5503B_SetAlcMode**

*Definition:* **int SC5503B_SetAlcMode(deviceHandle *devHandle, bool mode)**

*Input:* **deviceHandle** *devHandle (handle to the opened device)
**bool** mode (closed or open loop operation of the ALC circuit)

*Description:* **SC5503B_SetAlcMode** sets the ALC loop to closed or open loop operation for a channel.

*Definition:* **int SC5503B_SetAlcDac(deviceHandle *devHandle, unsigned int dacValue)**

*Input:* **deviceHandle** *devHandle (handle to the opened device)
**unsigned int** dacValue (14 bit value for adjusting the ALC DAC)

*Description:* **SC5503B_SetAlcDac** writes a value to the ALC DAC to control the RF output level for a channel.

*Function:* **SC5503B_SetSynthesizerMode**

*Definition:* **int SC5503B_SetSynthesizerMode(deviceHandle *devHandle,**
**bool fastTuneEnable, unsigned int fineTuneMode)**

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

*Description:* **SC5503B_SetSynthesizerMode** enables/disables fast tuning and sets the step resolution of the device.

*Function:* **SC5503B_SetReferenceClock**

*Definition:* **int SC5503B_SetReferenceClock(deviceHandle *devHandle, bool lockExtEnable, bool refOutEnable, bool clk100Enable)**

*Input:* **deviceHandle** *devHandle (handle to the opened device)
**bool** lockExtEnable (enables phase locking to an external source)
**bool** refOutEnable (enables the internal clock to be exported on the "ref out" port)
**bool** clk100Enable (choose 100 MHz to export instead of 10 MHz)

*Description:* **SC5503B_SetReferenceClock** configures the reference clock behavior of the device**.**

*Function:* **SC5503B_SetReferenceDac**

*Definition:* **int SC5503B_SetReferenceDac(deviceHandle *devHandle, unsigned int dacValue)**

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

*Description:* SC5503B_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:* **SC5503B_WriteUserEeprom**

*Definition:* **int SC5503B_WriteUserEeprom(deviceHandle *devHandle, unsigned int memAdd, unsigned char byteData)**

*Input:* **deviceHandle** *devHandle (handle to the opened device)
**unsigned int** memAdd (memory address to write to)
**unsigned char** byteData (byte to be written to the address)

*Description:* **SC5503B_WriteUserEeprom** writes one byte of data to the memory address specified.

*Function:* **SC5503B_StoreCurrentState**

*Definition:* **int SC5503B_StoreCurrentState(deviceHandle *devHandle)**

*Input:* **deviceHandle** *devHandle (handle to the opened device)

*Description:* **SC5503B_StoreCurrentState** stores the current state of the devices as the default power-up state.

*Function:* **SC5503B_DisableAutoLevel**

*Definition:* **int SC5503B_DisableAutoLevel(deviceHandle *devHandle, bool mode)**

*Input:* **deviceHandle** *devHandle (handle to the opened device)
**bool** mode (defines when to disable auto-level)

*Description:* **SC5503B_DisableAutoLevel** disables the device from auto adjusting the power level to the current state when frequency is changed. Disabling the auto adjust feature allows

the device to return faster after calling **SC5503B_SetPowerLevel**. If the power remains the same for the next tuned frequency, this should not be disabled.

*Function:* **SC5503B_GetDeviceInfo**

*Definition:* **int SC5503B_GetDeviceInfo(deviceHandle *devHandle, deviceInfo_t *devInfo)**

*Input:* **deviceHandle** *devHandle (handle to the opened device)

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

*Description:* **SC5503B_GetDeviceInfo** retrieves device information such as serial number, calibration date, revisions, etc.

*Function:* **SC5503B_GetDeviceStatus**

*Definition:* **int SC5503B_GetDeviceStatus(deviceHandle *devHandle, deviceStatus_t *deviceStatus)**

*Input:* **deviceHandle** *devHandle (handle to the opened device)

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

*Description:* **SC5503B_GetDeviceStatus** retrieves the status of the device such as phase lock status and current device settings.

*Example:* Code showing how to use this function:

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

int status = SC5503B_GetDeviceStatus(devHandle, devStatus);

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

free(deviceStatus);
```

*Function:* **SC5503B_GetTemperature**

*Definition:* **int SC5503B_GetTemperature(deviceHandle *devHandle, float *temperature)**

*Input:* **deviceHandle** *devHandle (handle to the opened device)

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

*Description:* **SC5503B_GetTemperature** retrieves the internal temperature of the device.

*Function:* **SC5503B_GetAlcDac**

*Definition:* **int SC5503B_GetAlcDac(deviceHandle *devHandle, unsigned int *dacValue)**

*Input:* **deviceHandle** *devHandle (handle to the opened device)

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

*Description:* **SC5503B_GetAlcDac** reads back the current ALC DAC value.

*Function:* **SC5503B_ReadCalEeprom**

*Definition:* **int SC5503B_ReadCalEeprom(deviceHandle *devHandle, unsigned int memAdd, unsigned char *byteData)**

*Input:* **deviceHandle** *devHandle (handle to the opened device)
**unsigned int** memAdd (EEPROM memory address)

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

*Description:* **SC5503B_ReadCalEeprom** reads back a byte from a specific memory address of the calibration EEPROM.

*Function:* **SC5503B_ReadUserEeprom**

*Definition:* **int SC5503B_ReadUserEeprom(deviceHandle *devHandle, unsigned int memAdd, unsigned char *byteData)**

*Input:* **deviceHandle** *devHandle (handle to the opened device)
**unsigned int** memAdd (EEPROM memory address)

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

*Description:* **SC5503B_ReadUserEeprom** reads back a byte from a specific memory address of the user EEPROM.

# PROGRAMMING THE RS232 INTERFACE

## Function Definitions and Usage of the RS232 API

The driver functions to access the device through the RS232 port are written upon NI-VISA, so NI-VISA must be installed to use them. There is no separate installer (.inf file) requirement as NI-VISA contains the RS232 protocol. The driver functions are provided as a DLL which can be called from C/C++, BASIC, LabVIEW, etc. The LabVIEW functions provided are simply wrappers of this DLL.

The functions are near identical to the USB set with the exception that SC5503B_SearchDevices is not provided, and the SC5503B_OpenDevice returns the device handle differently; see the rs232 header file for more information. Also, refer to the USB function descriptions for details on function usage. The device handle is defined as an *unsigned integer*, and is passed into the function as a pointer in the RS232 implementation, which differs from the USB version where the device handle is a return type of *deviceHandle*. Furthermore, the baud rate must be passed into the function at this point. The device supports 2 baud rates: 57600 and 115200. The baud rate for the device must be set at power up or on hardware reset of the device through pin 23 of the communication I/O connector. This pin is defaulted to logic high, which set the baud rate at 57600. Pulling this pin to logic low will set the rate at 115200.

| | | |
|---|---|---|
| *Function:* | **SC5503B_OpenDevice** | |
| *Definition:* | **int *SC5503B_OpenDevice(char *visaResource, unsigned int baudrate, unsigned int *deviceHandle)** | |
| *Return:* | int | (status) |
| *Output:* | char *visaResource | (resource name such as “COM1”) |
| | unsigned int baudrate | (sets the host to the baudrate of the device) |
| *Output:* | unsigned int* deviceHandle | (deviceHandle) |
| *Description:* | **SC5503B_OpenDevice** opens the device and turns on the front panel “active” LED if successful. This function returns a handle to the device for other function calls. | |

## Addressing the RS232 Registers Directly

When the user uses the supplied API, data sent and returned is appropriately handled and the user gets the proper end result. However, if the user chooses to read and write to the RS232 registers directly, the following information and the register descriptions in the USB section are needed.

The device with the RS232 option has a standard interface buffered by an RS232 transceiver, so that it may interface directly with many host devices such as a desktop computer The interface is provide through the communication I/O connector; refer to **Figure 1** and item 3 for position and pin-out information. The device communication control is provided in Table 6 below.

**Table 6.RS232 communication settings**

| | |
|---|---|
| Baud rate | Rate of transmission. Pin 12 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 (jumper to pin 11) or grounding it, the rate is set to 115200. |
| 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 or none |

## Writing to the device via RS232

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 4 bytes (address plus data) then all 4 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 data bytes associated with it before acting on the register instruction. Failure to complete the register transmission will cause the device to behave erratically or hang. Information for the configuration or write-to registers is given in Table 2.

When the device receives all the information for a register and finishes performing its instruction, it will **return a byte back** to the host. Querying this return byte ensures that the prior configuration command has been successfully executed and the device is ready for the next register command. **It is important to clear the incoming RX buffer on the host by querying it** or force flushing it to avoid in-coming data corruption of querying registers. The return byte value is 1 on success and 0 on an unsuccessful configuration.

## Reading from the device via RS232

To query information from the device, the query registers are addressed and data is returned. The returned RS232 data length for querying registers is always 2 bytes, however valid returned data depends on the queried register;  Table 3 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. Data is returned MSB first. The returned valid data length is also detailed below in Table 7. Note that temperature is returned as a digital code so please refer to the Reading Temperature Data section for information to convert it to floating number in degree Celsius

**Table 7 Valid returned data**

| REGISTER | BYTE 1 | BYTE 0 |
|---|---|---|
| GET_DEVICE_STATUS (0x17) | valid | valid |
| GET_TEMPERATURE (0x18) | valid | valid |
| READ_CAL_EEPROM (0x20) | non-valid | valid |
| READ_USER_EEPROM (0x22) | non-valid | valid |
| GET_ALC_DAC_VALUE (0x2A) | valid | valid |

# CALIBRATION & MAINTENANCE

The SC5503B is factory calibrated and ships with a certificate of calibration. SignalCore strongly recommends that the SC5503B 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.

SC5503B 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 SC5503B, 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 SC5503B.

The SC5503B 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 NOTE

| | |
|---|---|
| Rev 1.0 | Original Document |

SignalCore, Inc.
13401 Pond Springs Rd.
Suite 100
Austin, TX 78729, USA
Phone: 512-501-6000
Fax: 512-501-6001