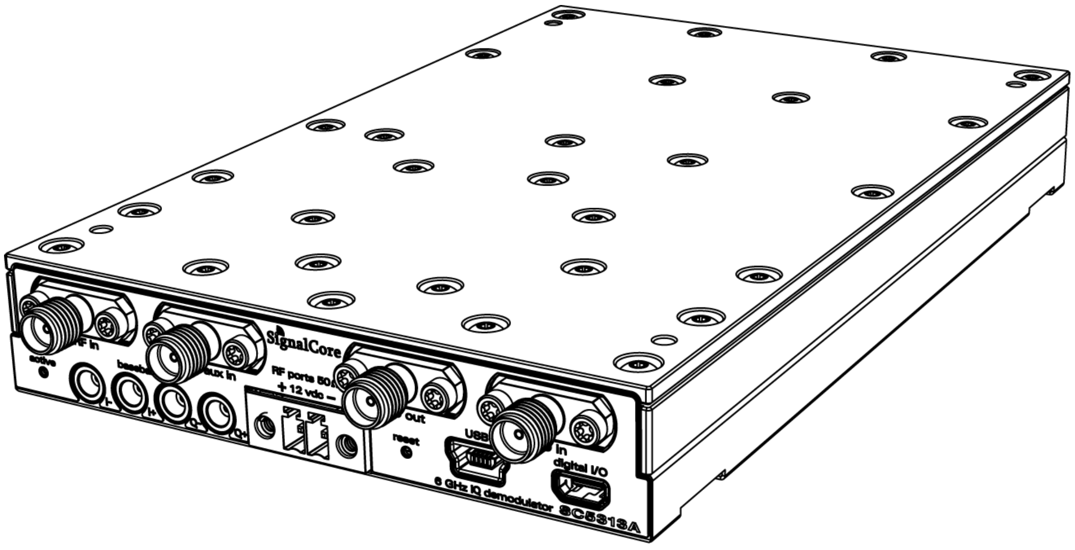
SignalCore™
PRESERVING SIGNAL INTEGRITY



SC5313A
400 MHz to 6 GHz IQ Demodulator
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 **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)**
SC5313A	✓	✓	✓	✓	✓	✓

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 SC5313A kit contains the following items:

Quantity	Item
1	SC5313A IQ Demodulator
1	USB Flash Drive Installation Software (may be combined with other products onto a single drive)
1	Getting Started Guide

Setting Up and Configuring the SC5313A

The SC5313A is a core module-based IQ demodulator with all I/O connections and indicators located on the front face of the module as shown in Figure 1. Each location is discussed in further detail below.

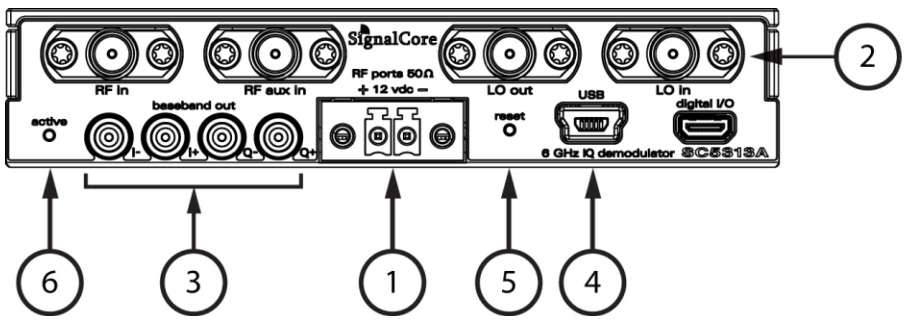


Figure 1. Front view of the SC5313A showing user I/O locations.

1 Power Connection

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

2 RF Signal Connections

All RF signal connections (ports) on the SC5313A are SMA-type. Exercise caution when fastening cables to the signal connections. Over-tightening any connection can cause permanent damage to the device.

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

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

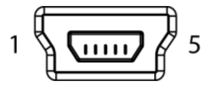
Tighten all SMA connections to 5 in-lb max (56 N-cm max)

LO OUT	This port outputs the tunable LO signal allowing phase-coherent daisy-chaining of multiple IQ demodulator modules. The connector is SMA female. The nominal output impedance is 50 Ω.
RF IN	This port accepts an RF signal ranging from 400 MHz to 6 GHz. The connector is SMA female. The nominal input impedance is 50 Ω. Maximum input power is +23 dBm with ATTEN #1 set to at least 10 dB attenuation.
RF AUX IN	This port accepts an RF signal ranging from 400 MHz to 6 GHz. This port can be used as an alternate path for system-level calibration. The connector is SMA female. The nominal input impedance is 50 Ω.
LO IN	This port accepts a tunable LO signal from an external source to drive the demodulator. The connector is SMA female. This port is AC-coupled with a nominal input impedance of 50 Ω. Maximum input power is +10 dBm.

3 Baseband Connections

The SC5313A has four baseband output ports, comprised of differential in-phase (I+ and I-) and differential quadrature (Q+ and Q-) outputs. Nominal differential output impedance is 100 Ω. The demodulator can also be configured for single-ended or differential IF output. When configured for single-ended operation, it is recommended to terminate the other half of the differential pair using a 50 Ω terminator. All baseband connectors are MCX female.

4 Communication Connection



The SC5313A uses a mini-USB Type B connector for primary communication with the device using the standard USB 2.0 protocol found on most host computers. The pinout of this connector, viewed from the front of the module, is listed in Table 1.

Table 1. Pinout of the SC5313A USB communication connector.

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

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

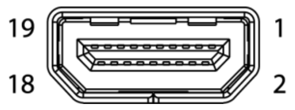


Table 2. Pinout of the SC5506A micro-HDMI connector for either SPI or RS-232 communication.

Pin Number	SPI Function	RS-232 Function
3	MISO	TxD
5	–	RTS
9	MOSI	RxD
11	CS	CTS
15	SRDY	–
17	CLK	CLK
4, 7, 10, 13, 16	GND	GND
2, 8, 12, 14, 18, 19	NC	NC

5 Reset Button

Depressing this momentary-action push button switch will reset the device to its default state. The SC5313A has the ability to store the current configuration at any point as the default setting. If the factory setting has been overwritten with a saved user configuration, resetting the device will reinitialize the device to the saved user configuration.

6 Indicator LED

The SC5313A provides visual indication of important modes. There one LED indicator on the unit. Its behavior under different operating conditions is shown in Table 3.

Table 3. LED indicator states.

LED	Color	Definition
ACTIVE	Orange	Device is powered on and working properly.
ACTIVE	Green	Device is open (communication has been established). This indicator is also user programmable. See register map.
ACTIVE	Off	Power fault. Contact SignalCore.

SC5313A Theory of Operation

Overview

The SC5313A is a single-stage, direct coversion Inphase-Quadrature (IQ) demodulating mixer, or simply an IQ demodulator. The SC5313A can operate as a single-stage downconverter or as an IQ demodulator. The SC5313A demodulator operates in the 400 MHz to 6 GHz RF range with a typical 3 dB IF bandwidth of 160 MHz in single-stage converter mode, or 320 MHz in IQ mode. The RF input stage has adjustable gain to allow the user to adjust the incoming RF signal prior to the demodulation process for the purpose of optimizing RF dynamic range. The IF stage has adjustable gain to ensure that linearity and noise of the IF output are optimized. The SC5313A has the necessary RF amplifiers, attenuators, IF amplifiers, and IF control via DACs to allow the user to optimally operate the device over the entire frequency range as well as for both small and large RF input levels. Figure 2 shows a simplified block diagram of the SC5313A, showing only the signal conditioning components critical for the following discussion. The following sections below provide more in-depth discussion on how to optimize the converter for linearity and signal-noise dynamic range. Power supply generation and regulation, and digital control functions are not covered. Should the user require more information than what is provided in this manual, please contact SignalCore.

RF Input Section

In the design of the RF input section, care was taken to ensure that the dynamic range of the IQ demodulator is preserved as seen at the input port of the device. This requires that the demodulator is not driven too hard (high signal amplitude) nor too soft (low signal amplitude). When the device is driven hard, nonlinear effects dominate the system. When driven too softly, signal-to-noise dynamic range suffers. A general rule is to apply more attenuation earlier in the RF signal path to improve linearity, and more gain to improve signal-to-noise performance. As an example, for a given input signal level and while maintaining a relatively constant output IF level, the user would switch in RF AMP#1 and apply attenuation on ATTEN#3 to improve signal-to-noise dynamic range. The factory default state sets all the RF amplifiers off, all attenuators set to 0 dB attenuation, and the IF gain set to 8 dB (DAC code of 32). In this default state, the device is optimized for a -10 dBm RF signal in the 1.0 GHz to 2.4 GHz range. The IF output is typically 0.5 V – 1.0 V peak-to-peak differential at these settings.

The RF amplifiers are used to improve the gain of the device if the input signal is too low or when the losses at higher frequencies are large. RF AMP#1 is usually selected when the RF signals are lower than -25 dBm at the input port. With RF AMP#1 enabled, the device sensitivity is improved and the detection of low level signals is better resolved. RF AMP#2 should be selected and switched into the signal path at RF frequencies greater than 5 GHz, where the signal power loss through the front end prior to the demodulator can be as high as 15 dB due to filter and switch insertion losses. At these high RF frequencies, if the IF gain is at its maximum of 15.75 dB (DAC code = 63) and the IF output level falls below -10 dBm or outside the digitizers optimal levels, RF AMP#2 should be enabled.

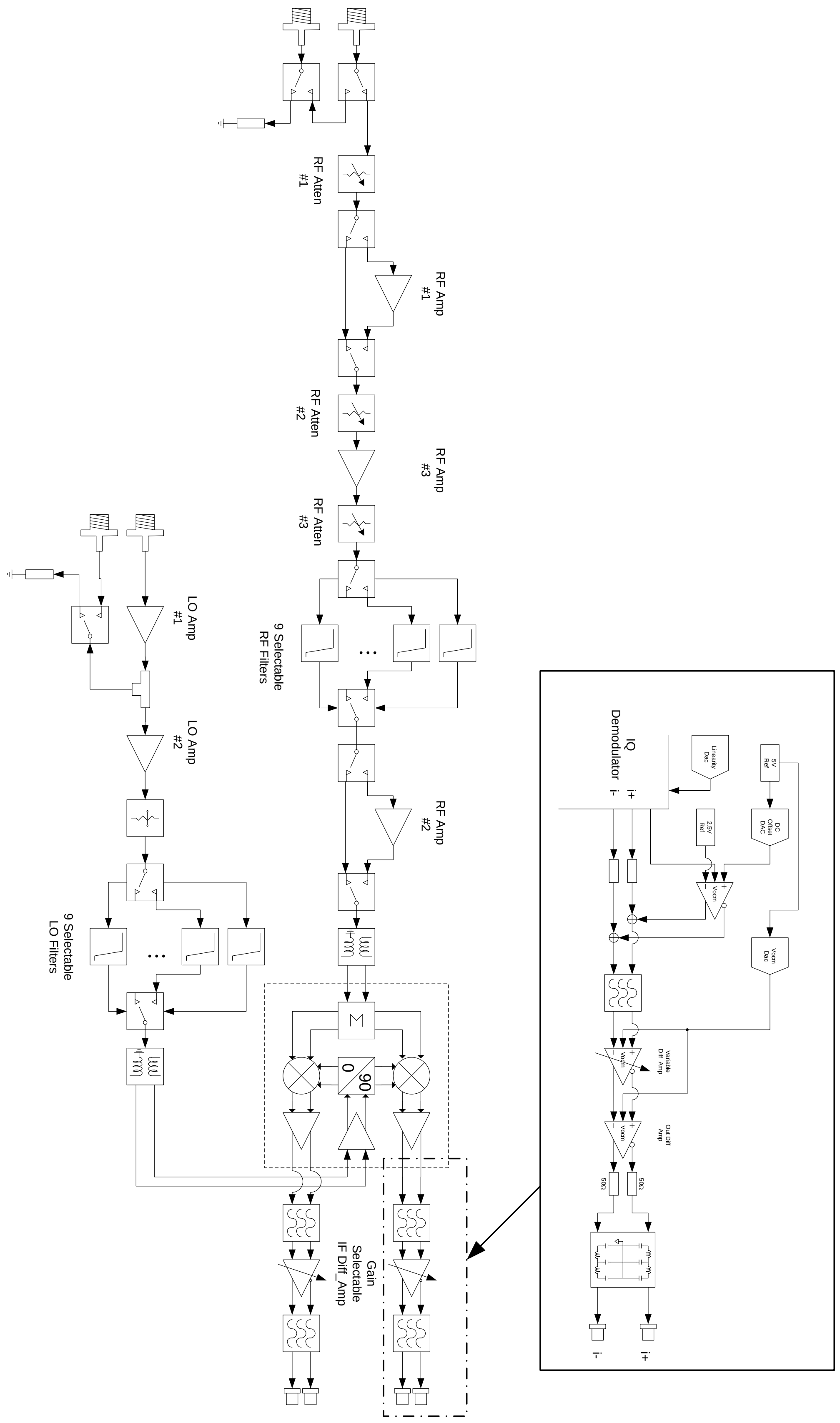


Figure 2. Simplified block diagram of the SC5313A.

The RF attenuators provide attenuation when required. RF ATTEN#1 attenuation should be stepped up as the signal power at the RF port increases above -10 dBm. Nonlinear components of the signal such as IMD3 and second order harmonics will increase in magnitude as the input signal increases, therefore the user should exercise good judgment to determine when to use RF ATTEN#1. Do not over attenuate because this will hurt the signal to noise ratio.

RF ATTEN#2 is used when if the input signal needs further suppression to improve linearity. It should also be used if RF AMP#1 is enabled to improve sensitivity, but as a result the level at the input of RF AMP#3 (always in the path) may be too high. Step up the attenuation of RF ATTEN#2 to ensure the system (resulting from RF AMP#3) is not driven too hard. Finally RF ATTEN#3 is used to control the level to the IQ demodulator when RF AMP#2 is enabled (switched into the signal path).

There is also an auxiliary RF input to the device. This input is almost identical to the main RF input with the exception of having an extra switch path. The intended use of this port is to allow the user a calibration path without having to detach the device under test (DUT) that is already attached to the main RF input port. The user must perform in-situ equalization to remove IQ errors such as phase imbalance and quadrature gain offsets that are inherent to the device.

There are nine low pass filters in the RF filter bank. These filters are automatically selected when the user enters the operating frequency. These filters can also be selected manually should the user choose to do so. As with all filters there is generally an amplitude roll-off as the frequency nears its 3 dB cutoff point so it is important to understand that frequencies near to the cutoff point may experience a slightly faster roll-off of its IF bandwidth. A typical 1 dB IF bandwidth (IQ) is about 160 MHz. The user may want to choose a higher frequency filter if this becomes a problem. See the programming section in this manual for more details. The filters in both the RF and LO filter banks are identical and are listed below.

Filter Number	1 dB Cutoff Frequency
0	400 MHz
1	500 MHz
2	650 MHz
3	1000 MHz
4	1400 MHz
5	2000 MHz
6	2825 MHz
7	3800 MHz
8	6000 MHz

LO Input Section

The SC5313A requires an external RF signal as its “Local Oscillator” (LO) for the frequency conversion process. The external RF signal must be connected to the “LO in” port. The typical required input level is -3 dBm to 3 dBm. These levels are required to sufficiently drive the IQ demodulator for good linearity performance and conversion loss. The LO signal is conditioned through a bank of low-pass filters to reduce the signal harmonics. Reducing the harmonics produces a “purer” signal tone, improving the duty cycle of the LO as it drives the mixers of the demodulator. Additionally, the LO signal can be passed out of the device via the “LO out” port. This output can be used as the input LO source for another demodulator, for example. Driving multiple demodulators (or modulators when working with SignalCore’s SC5413A) with the same derived LO signal optimizes phase coherency between them. When this port is not in use, it is highly recommended to terminate it into a 50 Ω load.

IF Output Section

The IF outputs are differentially driven. Each of the in-phase and quadrature components of the demodulator is conditioned prior to leaving the IF ports. The user can programmatically adjust the parameters of the differential signal such as the common output voltage, DC offset between the (-) and (+) terminals, and its amplitude. The differential output impedance of each component is 100 Ω. However, all ports can be operated as single-ended 50 Ω ports. All unused ports should be terminated into 50 Ω loads.

There are voltage DACs within the device to control the signal parameters of each of the IQ components. For each component, the Vcom (common voltage) DAC controls the common output voltage of the differential outputs. The Vcom DAC values range from 0 to 16383 (14 bits) and change the voltage between 1 V to 3.5 V. For a wider output voltage swing range, this voltage should be set to around 2.4 V to 2.5 V. Having a wider swing range improves the output compression point of the device. This is not a hard requirement and the user will need to adjust the voltage levels to suit their specific requirements. As an example, setting to some other voltage may be required to optimize the dynamic range of the receiving digitizer and as a result better optimize the entire system.

DC offsets may limit the dynamic range of the receiving digitizer, and where it is critical the user can “tune out” to minimize these offsets using the DC Offset DAC. This 14 bit DAC can correct offsets up to +/-0.050 V with about 0.025 mV resolution.

The IF amplifiers have an adjustable gain range of 15.75 dB, with a tuning resolution of 0.25 dB. The gain is controlled by programming a 6-bit DAC whose codes range from 0 to 63. Writing 63 to the DAC provides the highest gain. Increasing the IF gain instead of the RF gain to achieve a required IF level will improve the linearity of the system, but with the chance of a slight increase in output noise. For a common output voltage of 2.4 V, the output compression/saturation point of the amplifier is around 10 dBm. It is recommended to operate the output at least 6 dBm below this value to avoid running into saturation from signals with high crest factors. When deciding the operating point of the digitizer, it is recommended that the user not operate the output voltage too close to the saturation point of the digitizer input.

Digitized data may require digital correction to compensate for the IQ imbalances of the Demodulator. The process by which digital correction is applied to compensate for amplitude and phase imbalances of the demodulator is usually called *digital equalization*. In-situ digital equalization should be performed to achieve the best application results with the SC5313A.

SC5313A Programming Interface

Device Drivers

The SC5313A is programmed by writing to its set of configuration registers, and its data is read back through its set of query registers. The user may 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 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 **sc5313a_RegWrite** and **sc5313a_RegRead** functions respectively.

For Microsoft Windows^{TM} operating systems, The SC5313A API is provided as a dynamic linked library, *sc5313a.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 *SC5313A.dll* as well as the header files needed for development. Files required for development are in the *\win\Driver* directory. SignalCore makes every effort to bundle the latest third-party tools in our software installer. Occasionally however, third-party updates may not be identified and bundled in time for a given product shipment. Therefore, for the latest version of libusb-1.0, please visit http://libusbx.org. To install the necessary drivers, right click on the *sc5313a.inf* file under the *\win* directory and select “Install.” After installation is completed, when the device is plugged into a USB port and powered on, the host computer should identify the device and load the appropriate driver. For more information, please see the *SC5313A_Readme.txt* file also located under the *\Win* directory.

A LabVIEW API is provided and it consists of function wrappers that call the sc5313a.dll. A LabVIEW API written in G that uses NI-VISA is also available from the SignalCore website. The National Instruments driver wizard that is part of NI-VISA can be used to create a driver for most operating systems. For the SC5313A, the Vendor ID is **0x277C** and the PID is **0x0018**.

For Linux systems, the shared library *libsc5313a.so* is provided. To install the shared files and links, type “*make install*” in the */Linux/* directory. The user may need to acquire root privileges to perform the install operation. Ensure the *libsub-1.0.X.dev* package for the Linux distribution is installed on the host computer. 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 SC5313A. 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 SC5313A 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 **sc5313a_GetDeviceTemperature**, or calls **sc5313a_SetFrequency** to tune the frequency. The software API is covered in detail in the “Software API Library Functions” section.

SETTING THE SC5313A: WRITING TO CONFIGURATION REGISTERS

Configuration Registers

The users may write the configuration registers (write only) directly by calling the **sc5313a_RegWrite** function. The syntax for this function is **sc5313a_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 common USB, SPI, and RS232 interfaces; see the SPI and RS232 sections for data transfer details. Table 4 summarizes the register addresses (commands) and the effective bytes of command data.

Table 4. Configuration registers.

Register Name	Register Address	Serial Range	Bit 7 (MSB)	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 (LSB)
INITIALIZE	0x01	[7:0]								Mode
SET_SYSTEM_ACTIVE	0x02	[7:0]								Enable “active” LED
RF_FREQUENCY	0x10	[7:0]	MHz Frequency Word [7:0]							
		[15:8]	MHz Frequency Word [15:8]							
		[23:16]	MHz Frequency Word [23:16]							
		[31:24]	MHz Frequency Word [31:24]							
		[39:32]	MHz Frequency Word [39:32]							
RF_AMPLIFIER	0x12	[7:0]							Amplifier	Mode
RF_ATTENUATION	0x13	[7:0]			Attenuation					
		[15:8]							Atten #	
RF_PATH	0x14	[7:0]								Path
RF_FILTER_SELECT	0x15	[7:0]					Filter [3:0]			
LO_FILTER_SELECT	0x16	[7:0]					Filter [3:0]			
LO_OUT_ENABLE	0x17	[7:0]								Enable “LO out” port
IF_GAIN_DAC	0x18	[7:0]			DAC value [5:0]					
		[15:8]								Channel
VCOM_OUT_DAC	0x19	[7:0]	DAC value [7:0]							
		[15:8]			DAC value [13:8]					
		[23:16]								Channel
DC_OFFSET_DAC	0x1A	[7:0]	DAC value [7:0]							
		[15:8]			DAC value [13:8]					
		[23:16]								Channel
LINEARITY_DAC	0x1B	[7:0]	DAC value [7:0]							
		[15:8]			DAC value [13:8]					
STORE_STARTUP_STATE	0x1D	[7:0]								
USER_EEPROM_WRITE	0x1F	[7:0]	Data [7:0]							
		[15:8]	Address [7:0]							
		[23:16]	Address [15:8]							

To write to the device through USB transfers such as bulk transfer, it is important to send the data with the register byte first, followed by the most significant bit (MSB) of the data bytes. For example, to set the attenuation value of ATTEN#2, the byte stream would be [0x13][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 **STORE_STARTUP_STATE (0x1D)** register, described later in this section.

Setting the System Active LED

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

Setting the RF Frequency

RF_FREQUENCY (0x10) - This register provides the device frequency information to set up the filters appropriately. Data is sent as a 40 bit word with the LSB in Hz.

Setting RF Input RF Amplifiers

RF_AMPLIFIER (0x12) - This register enables or disables the RF amplifiers. Setting bit 0 low (0) disables RF amplifier. Setting bit 0 high (1) enables RF amplifier. Bit 1 selects the amplifier; 0 for RF AMP#1, 1 for RF AMP#2.

Setting the RF Attenuation

RF_ATTENUATION (0x13) – Each of the attenuators is a 5 bit digital step attenuator with 1 dB per LSB. Data is sent in 2 bytes; byte1 and bits [1:0] specifies the attenuator to program, and byte0 and bit [4:0] specifies the attenuation value.

Setting the RF Path

RF_PATH (0x14) – Setting bit 0 low selects the main RF input path, while high will select the RF auxiliary path.

Selecting the RF Filter

RF_FILTER_SELECT (0x15) – There are 9 RF filters to select from to improve RF input second harmonic suppression. Bits [3:0] are used.

Selecting the LO Filter

LO_FILTER_SELECT (0x16) – There are 9 RF filters to select from to improve LO input second harmonic suppression. Bits [3:0] are used.

Enabling LO Output

LO_OUT_ENABLE (0x17) – Setting bit 0 high enables the LO signal to be ported out the LO output connector. Note there is always a leakage out of this port and the levels could be as high as -30 dBm. It is recommended to terminate this port into a 50 Ω load if it is not used.

Setting the IF Gain

IF_GAIN_DAC (0x18) – Each of the channels has an adjustable IF amplifier with a step resolution of 0.25 dB per LSB. Writing the associated 6-bit DAC provides a gain range of 0 dB to 15.75 dB. Byte 1 selects the channel, while byte 0 determines the DAC value. A maximum DAC value of 63 will provide maximum gain.

Setting the Common Output Voltage

VCOM_OUT_DAC (0x19) – The common output voltage of each channel of differential amplifiers can be adjusted by writing to this DAC. The output voltage is linear in the region of 1.0 V to 3.5 V and follows the equation:

$$DAC\ Value = 16383\left({}^{V_{com}}/_{5V}\right)$$

Removing DC Offset in Differential Amplifiers

DC_OFFSET_DAC (0x1A) – The DC offset between the (+) and (-) terminals of the differential amplifier output can be minimized by writing this DAC. Varying the DAC value from 0 to 16383 can correct up to approximately $\mp 50mV$ of DC offset error. This correction resolution is less than 0.01 mV per LSB. An approximation of the DAC value to offset voltage is given below.

$$DAC\ Value = 16383\left({}^{V_{Offset} + 0.05V}/_{0.1V}\right)$$

Setting the Output Linearity of the IQ Demodulator

LINEARITY_DAC (0x19) – This DAC controls the current draw of the IQ modulator. As rule of thumb, the more current, the better the linearity. However, the user may find that the linearity can be improved with slight adjustments to the current consumption. The linearity is additionally dependent on the operating frequency and input RF power levels. Typically, the DAC is set around 4.5 V using the following equation:

$$DAC\ Value = 16383\left({}^{V_{com}}/_{5V}\right)$$

Storing the Startup State

STORE_STARTUP_STATE (0x1D) – 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.

Writing to the User EEPROM

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

Querying the SC5313A: Writing to Request Registers

The registers to read data back from the device (such as device status) are accessed through the **sc5313a_RegRead** function. The function and parameter format for this command is **sc5313a_RegRead(deviceHandle, registerCommand, instructWord,*dataOut)**. Any instructions in addition to the register call are placed into "instructWord", and data obtained from the device is returned via the pointer value **dataOut**. The set of request registers are shown in Table 5.

Table 5. Query registers.

Register Name	Register Address	Serial Range	Bit 7 (MSB)	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 (LSB)
GET_TEMPERATURE	0x20	[7:0]	Open	Open	Open	Open	Open	Open	Open	Open
GET_DEVICE_STATUS	0x21	[7:0]	Open	Open	Open	Open	Open	Open	Open	Open
USER_EEPROM_READ	0x23	[7:0]	EEPROM Address [7:0]							
		[15:8]	EEPROM Address [15:8]							
CAL_EEPROM_READ	0x24	[7:0]	EEPROM Address [7:0]							
		[15:8]	EEPROM Address [15:8]							

To read from the device using native USB transfers instead of the **sc5313a_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 Temperature

GET_TEMPERATURE (0x17) - Data returned by this register needs to be processed to correctly represent data in temperature units of degrees Celsius. Data is returned in the first 14 bits [13:0]. Bit [13] is the polarity bit indicating whether it is positive (0x0) or negative (0x1). For an ENDPOINT_IN transfer, data is returned in 2 bytes with the MSB first. 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 temperature of the SC5313A has stabilized. The temperature sensor is a serial device located inside the RF module. Therefore, like any other serial device, reading the temperature sensor requires a 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 externally-supplied LO signal within the device.

Reading the Device Status

GET_DEVICE_STATUS (0x21) - This register, summarized in Table 6, 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 6. Description of the status data bits.

Bit	Description
[4]	RF AMP#1 Enable
[3]	RF AMP#2 Enable
[2]	RF Path Selection
[1]	LO Output Enable
[0]	Device accessed

Reading the User EEPROM

USER_EEPROM_READ (0x23) - Once data has been written to the user EEPROM, it can be retrieved by calling this register and using the process outlined below for reading calibration data. The maximum address for this EEPROM is 0x7FFF. A single byte is returned.

Reading the Calibration EEPROM

CAL_EEPROM_READ (0x24) - Reading a single byte from an address in the device EEPROM is performed by writing this register with the address for the instructWord. The data is returned as a byte. The CAL EEPROM maximum address is also 0x7FFF. Reading above this address will cause the device to retrieve data from the lower addresses. For example, addressing 0x8000 will return data stored in address location 0x0000. The calibration EEPROM map is shown in Table 7.

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;
```

Calibration EEPROM Map

Table 7. Calibration EEPROM map.

EEPROM ADDRESS (HEX)	NUMBER OF DATA POINTS	TYPE	DESCRIPTION
0	1	U32	Manufacturing information
4	1	U32	Product serial number
8	1	U32	RF module number
C	1	U32	Product manufacture date
24	1	F32	Firmware revision
28	1	F32	Hardware revision
2C	40	F32	Reserved
CF	33	U8	Startup state
F4	1	F32	Calibration temperature

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 SC5313A API dynamic linked library or LabVIEW library are:

- **sc5313a_SearchDevices**
- **sc5313a_OpenDevice**
- **sc5313a_CloseDevice**
- **sc5313a_RegWrite**
- **sc5313a_RegRead**
- **sc5313a_InitDevice**
- **sc5313a_SetDeviceStandby**
- **sc5313a_SetFrequency**
- **sc5313a_SetIfGainDac**
- **sc5313a_SetLinearityDac**
- **sc5313a_SetLoFilter**
- **sc5313a_SetLoOut**
- **sc5313a_SetRfAmplifier**
- **sc5313a_SetRfAttenuators**
- **sc5313a_WriteUserEeprom**
- **sc5313a_StoreStartupState**
- **sc5313a_SetRfFilter**
- **sc5313a_GetDeviceInfo**
- **sc5313a_GetDeviceStatus**
- **sc5313a_GetTemperature**
- **sc5313a_ReadCalEeprom**
- **sc5313a_ReadUserEeprom**
- **sc5313a_AutoDcOffsetComp**
- **sc5313a_SetDcOffsetDac**
- **sc5313a_SetRfGain**
- **sc5313a_SetRfPath**
- **sc5313a_SetVcomDac**

Each of these functions is described in more detail on the following pages. Example code written in C/C++ is located in the *CD:\Win\Driver\src* directory to show how these functions are called and used. First, for C/C++, we define the constants and types which are contained in the C header file, *sc5313a.h.* These constants and types are useful not only as an include for developing user applications using the SC5313A API, but also for writing device drivers independent of those provided by SignalCore.

Constants Definitions

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

// Define labels
#define CH_I                      0x00
#define CH_Q                      0x01
#define RF_ATTEN1                 0x00
#define RF_ATTEN2                 0x01
#define RF_ATTEN3                 0x02
#define RF_AMP1                   0x00
#define RF_AMP2                   0x01

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

// Define device registers
#define INITIALIZE                0x01   // initialize the devices
#define SET_SYSTEM_ACTIVE         0x02   // set the device “active” LED
#define RF_FREQUENCY              0x10   // set the frequency
#define RF_AMPLIFIER              0x12   // enable amplifiers
#define RF_ATTENUATION            0x13   // set attenuation for digital step attenuators
#define RF_PATH                   0x14   // select the RF path
#define RF_FILTER_SELECT          0x15   // manually select the RF filter
#define LO_FILTER_SELECT          0x16   // manually select the LO filter
#define LO_OUT_ENABLE             0x17   // enable LO output
#define IF_GAIN_DAC               0x18   // set the I and Q chain IF gain
#define VCOM_OUT_DAC              0x19   // sets common output voltage
#define DC_OFFSET_DAC             0x1A   // sets the DC offset
#define LINEARITY_DAC             0x1B   // sets the Linearity DAC (0 to 0xFFF)
#define STORE_STARTUP_STATE       0x1D   // store the current state as default
#define USER_EEPROM_WRITE         0x1F   // write a byte to the user EEPROM
#define GET_DEVICE_STATUS         0x20   // read the device status
#define GET_TEMPERATURE           0x21   // get the internal temperature of the device
#define USER_EEPROM_READ          0x23   // read a byte from the user EEPROM
#define CAL_EEPROM_READ           0x24   // read a byte from the calibration EEPROM
```

Type Definitions

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

typedef struct
{
	bool rfAmp1Enable;
	bool rfAmp2Enable;
	bool rfPath;
	bool LoEnable;
	bool deviceAccess;
}	deviceStatus_t;
```

Function Definitions and Usage

The functions listed below are found in the *sc5313a.dll* dynamic linked library for the Windows^{TM} operating system. These functions are also provided in the LabVIEW library, *sc5313a.llb.* The LabVIEW functions contain context-sensitive help (Ctrl-H) to assist with understanding the input and output parameters.

Function:	**sc5313a_SearchDevices**	
Definition:	**int sc5313a_SearchDevices(char **serialNumberList)**	
Output:	char **serialNumberList	(pointer list to serialNumberList)
Description:	**sc5313a_SearchDevices** searches for SignalCore SC5313A devices connected to the host computer and returns an array containing their unique serial numbers. This information can be used to open the device(s) using individual device serial numbers as unique identifiers in the system. See **sc5313a_OpenDevice** function on how to open a device.	

Function:	**sc5313a_OpenDevice**	
Definition:	**deviceHandle *sc5313a_OpenDevice(char *devSerialNum)**	
Input:	char *devSerialNum	(pointer serial number list)
Output:	deviceHandle *deviceHandle	(unsigned integer number for the deviceHandle)
Description:	**sc5313a_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: **sc5313a_CloseDevice**

Definition: **int sc5313a_CloseDevice (deviceHandle *deviceHandle)**

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

Description: **sc5313a_CloseDevice** closes the device associated with the device handle and turns off the "active" LED on the front panel if it is successful.

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

```
// Declaring
#define MAXDEVICES 50
libusb_device_handle *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 = sc5313a_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++;
       }
//** sc5313a_OpenDevice, open device 0
devHandle = sc5313a_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 = sc5313a_CloseDevice(devHandle);
```

Function: **sc5313a_RegWrite**

Definition: **int sc5313a_RegWrite (deviceHandle *deviceHandle, unsigned char commandByte, unsigned long long int instructWord)**

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

Description: **sc5313a_RegWrite** writes the instructWord data to the register specified by the commandByte. See the register map on Table 4 for more information.

Example: To set the frequency to 2 GHz:

```
int status = sc5313a_RegWrite(devHandle, RF_FREQUENCY, 2000000000); // set frequency to 2 GHz
```

Function:	**sc5313a_RegRead**
Definition:	**int sc5313a_RegRead (deviceHandle *deviceHandle, unsigned char commandByte, unsigned long long int instructWord, unsigned int *receivedWord)**
Input:	**deviceHandle** *deviceHandle (handle to the opened device)
	unsigned char commandByte (address byte of the register to write to)
	unsigned long long int instructWord (data for the register)
	unsigned int *receivedWord (data to be received)
Description:	**sc5313a_RegRead** reads the data requested by the instructWord data to the register specified by the commandByte. See

Table 5 (register map) for more information.

Example: To read the status of the device:

```
unsigned int deviceStatus;

int status = sc5313a_RegRead(devHandle,
GET_DEVICE_STATUS,0x00,&deviceStatus);
```

Function:	**sc5313a_InitDevice**
Definition:	**int sc5313a_InitDevice (deviceHandle *deviceHandle, bool mode)**
Input:	**deviceHandle** *deviceHandle (handle to the opened device)
	bool mode (set the mode of initialization)
Description:	**sc5313a_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:	**sc5313a_SetFrequency**
Definition:	**int sc5313a_SetFrequency (deviceHandle *deviceHandle, unsigned long long int frequency)**
Input:	**deviceHandle** *deviceHandle (handle to the opened device)
	unsigned long long int frequency (frequency in Hz)
Description:	**sc5313a_SetFrequency** sets the RF frequency so the device can automatically use the information to set the optimal filters in the LO and RF filter banks.

Function:	**sc5313a_SetRfAmplifier**
Definition:	**int sc5313a_SetRfAmplifier(deviceHandle *devHandle, bool amplifier, bool mode)**
Input:	**deviceHandle** *deviceHandle (handle to the opened device)
	bool amplifier (0=AMP#1, 1=AMP#2)
	bool mode (disable/enable)
Description:	**sc5313a_SetRfAmplifier** enables or disables the RF amplifiers.

Function:	**sc5313a_SetRfPath**	
Definition:	**int sc5313a_SetRfPath (deviceHandle *deviceHandle, bool mode)**	
Input:	**deviceHandle** *deviceHandle	(handle to the opened device)
	bool mode	(0=main path, 1=aux path)
Description:	**sc5313a_SetRfPath** select the RF input port.	

Function:	**sc5313a_SetLoOut**	
Definition:	**int sc5313a_SetLoOut(deviceHandle *deviceHandle, bool mode)**	
Input:	**deviceHandle** *deviceHandle	(handle to the opened device)
	bool mode	(0=disable, 1= enable)
Description:	**sc5313a_SetLoOut** enables the LO output port. The LO input signal is replicated and piped out through these LO output port.	

Function:	**sc5313a_SetRfAttenuation**	
Definition:	**int sc5313a_SetRfAttenuation (deviceHandle *deviceHandle, unsigned char attenuator, unsigned char atten)**	
Input:	**deviceHandle** *deviceHandle	(handle to the opened device)
	unsigned char attenuator	(selects the attenuator to program)
	unsigned char atten	(attenuation value (0-31 dB))
Description:	**sc5313a_SetRfAttenuation** Sets the attenuation of the RF attenuators**.**	

Function:	**sc5313a_SetRfFilter**	
Definition:	**int sc5313a_SetRfFilter (deviceHandle *deviceHandle, unsigned char filter)**	
Input:	**deviceHandle** *deviceHandle	(handle to the opened device)
	unsigned char filter	(select the appropriate filter number 0-8)
Description:	**sc5313a_SetRfFilter** selects the active filter in the RF filter bank.	

Function:	**sc5313a_SetLoFilter**	
Definition:	**int sc5313a_SetLoFilter (deviceHandle *deviceHandle, unsigned char filter)**	
Input:	**deviceHandle** *deviceHandle	(handle to the opened device)
	unsigned char filter	(select the appropriate filter number 0-8)
Description:	**sc5313a_SetLoFilter** selects the active filter in the LO filter bank.	

Function:	**sc5313a_SetIfGainDac**
Definition:	int **sc5313a_SetIfGainDac (**deviceHandle ***deviceHandle,** unsigned char **channel,** unsigned char **dacValue)**
Input:	**deviceHandle** *deviceHandle (handle to the opened device) **unsigned char** channel (select the I or Q channel) **unsigned char** dacValue (DAC value 0 - 63)
Description:	**sc5313a_SetIfGainDac** sets the gain of the IF amplifier.

Function:	**sc5313a_SetVcomDac**
Definition:	int **sc5313a_SetVcomDac (**deviceHandle ***deviceHandle,** unsigned char **channel,** unsigned short **dacValue)**
Input:	**deviceHandle** *deviceHandle (handle to the opened device) **unsigned char** channel (select the I or Q channel) **unsigned short** dacValue (DAC value 0 - 16383)
Description:	**sc5313a_SetVcomDac** sets the common output voltage of the differential amplifiers. The default factory setting is 2008.

Function:	**sc5313a_SetDcOffsetDac**
Definition:	int **sc5313a_SetDcOffsetDac (**deviceHandle ***deviceHandle,** unsigned char **channel,** unsigned short **dacValue)**
Input:	**deviceHandle** *deviceHandle (handle to the opened device) **unsigned char** channel (select the I or Q channel) **unsigned short** dacValue (DAC value 0 - 16383)
Description:	**sc5313a_SetDcOffsetDac** sets the DC offset voltage of the differential amplifiers. Voltage adjust is approximately +/- 0.05 V. The default factory setting is 2048.

Function:	**sc5313a_SetLinearityDac**
Definition:	int **sc5313a_SetLinearityDac (**deviceHandle ***deviceHandle,** unsigned short **dacValue)**
Input:	**deviceHandle** *deviceHandle (handle to the opened device) **unsigned short** dacValue (DAC value 0 - 16383)
Description:	**sc5313a_SetLinearityDac** sets the current consumption of the IQ demodulator, which affects the linearity of the device. A DAC value of 3685 is recommended and is also the default factory setting.

Function:	**sc5313a_WriteUserEeprom**
Definition:	**int sc5313a_WriteUserEeprom(deviceHandle *deviceHandle, unsigned int memAdd, unsigned char byteData)**
Input:	deviceHandle *deviceHandle (handle to the opened device)
	unsigned int memAdd (memory address to write to)
	unsigned char byteData (byte to be written to the address)
Description:	**sc5313a_WriteUserEeprom** writes one byte of data to the memory address specified.

Function:	**sc5313a_StoreCurrentState**
Definition:	**int sc5313a_StoreCurrentState(deviceHandle *deviceHandle)**
Input:	deviceHandle *deviceHandle (handle to the opened device)
Description:	**sc5313a_StoreCurrentState** stores the current state of the devices as the default power-up state.

Function:	**sc5313a_GetDeviceInfo**
Definition:	**int sc5313a_GetDeviceInfo(deviceHandle *deviceHandle, deviceInfo_t *devInfo)**
Input:	deviceHandle *deviceHandle (handle to the opened device)
Output:	deviceInfo_t *devInfo (device info struct)
Description:	**sc5313a_GetDeviceInfo** retrieves device information such as serial number, calibration date, revisions, etc.

Function:	**sc5313a_GetDeviceStatus**
Definition:	**int sc5313a_GetDeviceStatus (deviceHandle *deviceHandle, deviceStatus_t *deviceStatus)**
Input:	deviceHandle *deviceHandle (handle to the opened device)
Output:	deviceStatus_t *deviceStatus (deviceStatus struct)
Description:	**sc5313a_GetDeviceStatus** retrieves the status of the device such as phase lock status and current device settings.
Example:	Code showing how to use function:

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

int status = SC5313A_GetDeviceStatus(devHandle, devStatus);

if(devStatus->loEnable)
printf("The LO Output Port is Enabled \n");
else
printf("The LO Output Port is disabled \n");

free(deviceStatus);
```

Function:	**sc5313a_GetTemperature**	
Definition:	**int sc5313a_GetTemperature (deviceHandle *deviceHandle, float *temperature)**	
Input:	**unsigned int** deviceHandle	(handle to the opened device)
Output:	**float** *temperature	(temperature in degrees Celsius)
Description:	**sc5313a_GetTemperature** retrieves the internal temperature of the device.	

Function:	**sc5313a_ReadCalEeprom**	
Definition:	**int sc5313a_ReadCalEeprom(deviceHandle *deviceHandle, unsigned int memAdd, unsigned char *byteData)**	
Input:	**unsigned int** deviceHandle	(handle to the opened device)
	unsigned int memAdd	(EEPROM memory address)
Output:	**unsigned char** *byteData	(the read back byte data)
Description:	**sc5313a_ReadCalEeprom** reads back a byte from a specific memory address of the calibration EEPROM.	

Function:	**sc5313a_ReadUserEeprom**	
Definition:	**int sc5313a_ReadUserEeprom(deviceHandle *deviceHandle, unsigned int memAdd, unsigned char *byteData)**	
Input:	**unsigned int** deviceHandle	(handle to the opened device)
	unsigned int memAdd	(EEPROM memory address)
Output:	**unsigned char** *byteData	(the read back byte data)
Description:	**sc5313a_ReadUserEeprom** reads back a byte from a specific memory address of the user EEPROM.	

PROGRAMMING THE SERIAL PERIPHERAL INTERFACE

The SPI Architecture

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. In other words, 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. $\overline{CS}$ must be asserted for the entire duration of the transfer.

Once a full transfer has been received, the device will proceed to process the command and de-assert low the SERIAL_READY bit. The status of this bit can be read by the host by invoking the SERIAL_READY register (0x04). The SERIAL_READY bit can also be monitored on pin 15 of the SPI interface (digital I/O) connector. While SERIAL_READY is de-asserted low, the device will ignore any incoming commands. It is only ready when the previous command is fully processed and SERIAL_READY is re-asserted high. It is important that the host monitors the SERIAL_READY bit and performs transfers only when it is asserted high to avoid miscommunication. Figure 3 shows the command transfer for obtaining the SERIAL_READY bit, which is returned in bit 0 of the device MISO line.

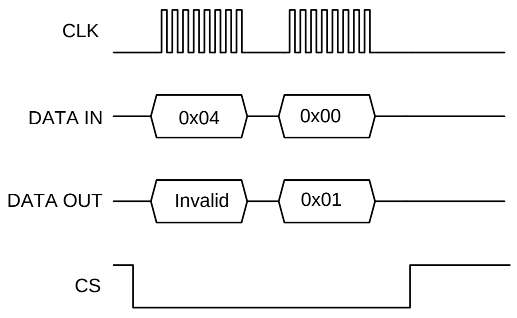


Figure 3. Querying the SerialReady bit.

All data transferred to and from the device are clocked on the falling edge of the clock as shown in Figure 4. Figure 5 shows a 3 byte SPI transfer initiated by the host; the device is always in slave mode. The CS pin must be asserted low for a minimum period of 5 μs before data is clocked in. The clock rate may be as high as 1.0 MHz, however if the external SPI signals do not have sufficient integrity due to cabling problems the rate should be lowered. It is recommended that the clock rate not exceed 1.0 MHz to ensure proper serial operation. As mentioned above, the SPI architecture limits the byte rate because after every byte transfer, the input and output SPI buffers need to be cleared and loaded respectively by the device SPI engine. The time required to perform this task is indicated in Figure 5 by T_B, which is the time interval between the end of one byte transfer and the beginning of another. The recommended time delay for T_B is $10\mu s$ or greater. The number of bytes transferred depends on the command. 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 is received, the device will wait for the desired number of bytes associated with it. The device will hang if insufficient number of bytes is written and the device will need to be reset externally. The time required to process a command is also dependent on the command itself; measured times for command completions are typically between $40\,\mu s$ to $150\,\mu s$ after reception. The user may choose to wait a minimum of $150\,\mu s$ or query the SERIAL_READY bit before sending in another command; the latter is recommended for robustness. The minimum wait time between successive polls of the SERIAL_READY register is $10\,\mu s$.

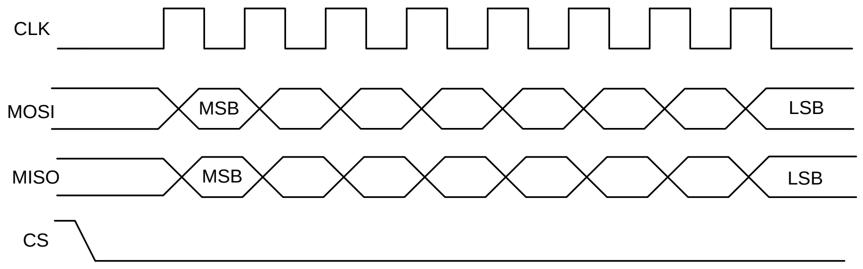


Figure 4: SPI Mode - Data clocked in on falling clock edge.

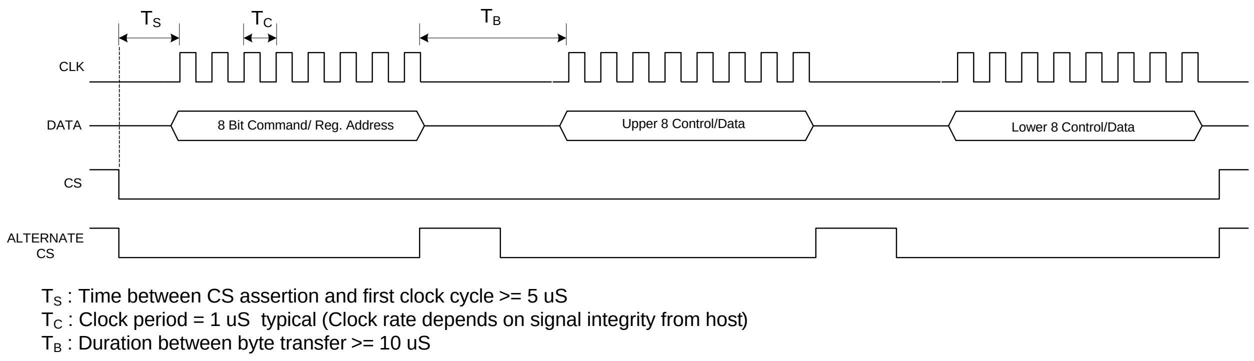

T_S
T_C
T_B
CLK
DATA
8 Bit Command/ Reg. Address
Upper 8 Control/Data
Lower 8 Control/Data
CS
ALTERNATE CS
T_S : Time between CS assertion and first clock cycle >= 5 uS
T_C : Clock period = 1 uS typical (Clock rate depends on signal integrity from host)
T_B : Duration between byte transfer >= 10 uS

Figure 5. SPI timing.

Additional SPI Registers

There are two additional registers available for SPI communication as shown in Table 8. Data byte(s) associated with registers can be “zeros” or “one”; it doesn’t matter which value since the device ignores them. They are only required for clocking out the returned data from the device.

Table 8. Additional SPI registers.

Register Name	Register Address	Serial Range	Bit 7 (MSB)	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 (LSB)
SERIAL_READY	0x04	[7:0]								
SPI_OUT_BUFFER	0x22	[7:0]	Open							
		[15:8]	Open							
		[23:16]	Open							
		[31:24]	Open							

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 Table 8. The subsequent bytes contain the data associated with the register. As data from the host is being transferred to the device, data present on its SPI output buffer is simultaneously transferred back, MSB first, via the master-in-slave-out (MISO) line. The data return is invalid for most transfers except for those register commands querying for data from the device. See "Reading the SPI Bus" section for more information on retrieving data from the device. Figure 6 shows the contents of a single 3 byte SPI command written to the device. Table 4 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".

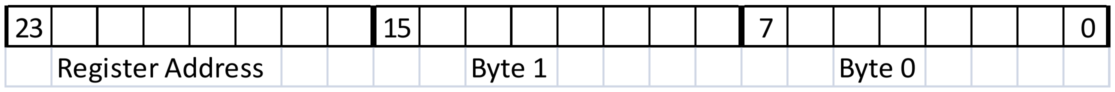


Figure 6. Single 3 byte transfer buffer.

Reading the SPI Bus

Data is simultaneously read back during a SPI transfer cycle. Requested data from a prior command (see Table 9) is available on the device SPI output buffers, and these are transferred back to the user host via the MISO line. To obtain valid requested data would require querying the SPI_OUT_BUFFER, which requires 5 bytes of clock cycles; 1 byte for the device register (0x22) and 4 empty bytes (MOSI) to clock out the returned data (MISO). An example of reading the temperature from the device is shown in Figure 7.

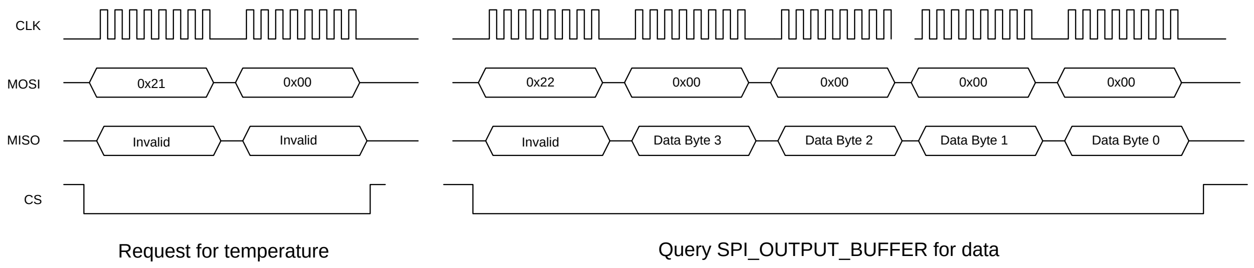


Figure 7. Reading queried data.

In the above example, valid data is present in the last 2 bytes; byte 1 and byte 0. Table 9 shows the valid data bytes associated with the querying register.

Table 9. Valid returned data bytes.

Register Name	Register Address	Byte 3	Byte 2	Byte 1	Byte 0
GET_TEMPERATURE	0x20			valid	valid
GET_DEVICE_STATUS	0x21				valid
USER_EEPROM_READ	0x23				valid
CAL_EEPROM_READ	0x24				valid

Calibration & Maintenance

The SC5313A does not receive a factory calibration. The SC5313A is sold as a component and users will need to perform amplitude and IQ correction as part of their system, which may minimally include a digitizer, LO source, and the SC5513A. Should users require SignalCore to perform any calibration, please contact SignalCore support directly.

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