



SC5413A

400 MHz to 6 GHz IQ Modulator

USB, SPI and RS-232 Interface

Operating & Programming Manual

CONTENTS

Important Information	1
Warranty	1
Copyright & Trademarks	1
International Materials Declarations	2
CE European Union EMC & Safety Compliance Declaration	2
Warnings Regarding Use of SignalCore Products	3
Getting Started	4
Unpacking	4
Verifying the Contents of your Shipment	4
Setting Up and Configuring the SC5413A	4
SC5413A Theory of Operation	8
Overview	8
IF Input Section	8
RF Output Section	9
LO Input Section	11
SC5413A Programming Interface	12
Device Drivers	12
Using the Application Programming Interface (API)	12
Setting the SC5413A: Writing to Configuration Registers	13
Configuration Registers	13
Initializing the Device	14
Setting the System Active LED	14
Setting the RF Frequency	14
Setting RF Input RF Amplifiers	14
Setting the RF Attenuation	14
Setting the RF Path	14
Selecting the RF Filter	14
Selecting the LO Filter	15

Enabling LO Output	15
Removing DC Offset in Differential Amplifiers	15
Setting the Output Linearity of the IQ Modulator	15
Storing the Startup State	15
Writing to the User EEPROM	15
Querying the SC5413A: Writing to Request Registers	16
Reading the Device Temperature	16
Reading the Device Status	17
Reading the User EEPROM	17
Reading the Calibration EEPROM	17
Calibration EEPROM Map	18
Software API Library Functions	19
Constants Definitions	20
Type Definitions	21
Function Definitions and Usage	21
Programming the Serial Peripheral Interface	28
The SPI Architecture:	28
Additional SPI registers	29
Writing the SPI Bus	29
Reading the SPI Bus	30
Programming the RS-232 Interface	30
Writing to the Device via RS-232	31
Reading from the Device via RS-232	31
RS232 Windows™ API	32
Using the LabVIEW Functions and NI-VISA	32
Calibration & Maintenance	33
Revision Notes	1

IMPORTANT INFORMATION

Warranty

This product is warranted against defects in materials and workmanship for a period of three years from the date of shipment. SignalCore will, at its option, repair or replace equipment that proves to be defective during the warranty period. This warranty includes parts and labor.

Before any equipment will be accepted for warranty repair or replacement, a Return Material Authorization (RMA) number must be obtained from a SignalCore customer service representative and clearly marked on the outside of the return package. SignalCore will pay all shipping costs relating to warranty repair or replacement.

SignalCore strives to make the information in this document as accurate as possible. The document has been carefully reviewed for technical and typographic accuracy. In the event that technical or typographical errors exist, SignalCore reserves the right to make changes to subsequent editions of this document without prior notice to possessors of this edition. Please contact SignalCore if errors are suspected. In no event shall SignalCore be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, SIGNALCORE, INCORPORATED MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF SIGNALCORE, INCORPORATED SHALL BE LIMITED TO THE AMOUNT THEREFORE 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

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of SignalCore, Incorporated.

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

“SignalCore”, “signalcore.com”, and the phrase “preserving signal integrity” are registered trademarks of SignalCore, Incorporated. Other product and company names mentioned herein are trademarks or trade names of their respective companies.

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)	鎘 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)
SC5413A	✓	✓	✓	✓	✓	✓

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 ✕ 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 SC5413A kit contains the following items:

Quantity	Item
1	SC5413A IQ Modulator
1	Installation Software
1	Getting Started Guide

Setting Up and Configuring the SC5413A

The SC5413A is a core module-based IQ modulator 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.

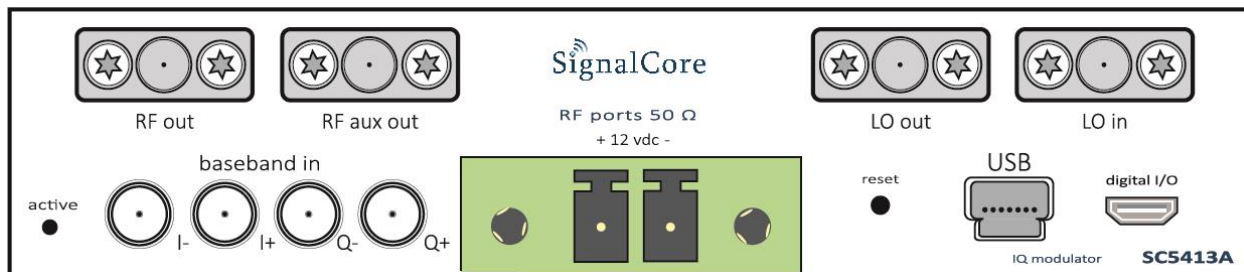


Figure 1. Front view of the SC5413A showing user I/O locations.

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.

RF Signal Connections

All RF signal connections (ports) on the SC5413A 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.



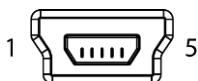
Tighten all SMA connections to 5 in-lb max (56 N-cm max)

- | | |
|-------------------|--|
| 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 OUT | This is the main 400MHz to 6 GHz RF signal output port. The connector is SMA female. The nominal output impedance is 50 Ω . |
| RF AUX OUT | This is the 400 MHz to 6 GHz RF auxiliary output port. This port can be used as an alternate path for system-level calibration. The connector is SMA female. The nominal output impedance is 50 Ω . |
| LO IN | This port accepts a tunable LO signal from an external source to drive the modulator. The connector is SMA female. This port is AC-coupled with a nominal input impedance of 50 Ω . Maximum input power is +10 dBm. |

Baseband Connections

The SC5413A has four baseband input ports, comprised of differential in-phase (I+ and I-) and differential quadrature (Q+ and Q-) inputs. Nominal differential input impedance is 100 Ω . The modulator can also be configured for single-ended or differential IF input. 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.

Communication Connections



The SC5413A 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 SC5413A 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.

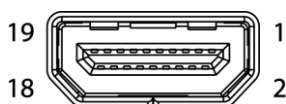


Table 2. Pinout of the SC5413A micro-HDMI connector for either SPI or RS-232 communication.

Pin Number	SPI Function	RS-232 Function
3	MISO	TxD
5	—	—
9	MOSI	RxD
11	CS	—
1	SRDY	—
17	CLK	—
18	SPI_MODE	BAUD SELECT
4, 7, 10, 13, 16	GND	GND
2, 8, 12, 14, 15, 19	DNC	DNC

Reset Button

Depressing this momentary-action push button switch will reset the device to its default state. The SC5413A 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.

Indicator LED

The SC5413A provides visual indication of important modes. There one LED indicator on the unit. Its behavior under different operating conditions is shown in Table 2.

Table 2. 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.

SC5413A THEORY OF OPERATION

Overview

The SC5413A is a single-stage, direct conversion Inphase-Quadrature (IQ) modulator. The modulator operates in the 400 MHz to 6 GHz RF range with a typical 3 dB IQ IF bandwidth of 320 MHz. IF signals are conditioned by input differential drivers prior to the IQ mixers. These differential drivers adjust the DC level and minimize the DC offset effects on the modulated RF signal. The RF output stage has adjustable gain to allow the user to adjust the signal levels, and also to optimize for signal dynamic range. The SC5413A has the necessary RF amplifiers, attenuators and IF amplifiers to give the user optimal control of the device over the entire frequency range. Figure 3 shows a simplified block diagram of the SC5413A, showing only the signal conditioning components critical for the following discussion. The following sections 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.

IF Input Section

The IF inputs are typically driven differentially, however they may also be driven single-ended. The IF input is DC coupled with a differential impedance of 100 Ω . Singled-ended operation is recommended for AC-coupled operation due to the fact that DC-coupled single-ended operation may cause large DC offsets at the input driver amplifier that the DC offset compensation may not be about to overcome. A large DC offset will lead to high LO leakage, so unless the DC points of the differential inputs are bias the same, AC coupling is recommended for single-end operation. For differential DC-coupled input it is recommended to drive the common mode input voltage between 1.5 V to 2.5 V for best linearity performance. See Figure 2 for details of the input section.

From Figure 3, the IF signals are first filtered by a 160 MHz filter, and then passed to the input differential drivers before being driven into the IQ mixers. The DC characteristics of the output of each differential driver are controlled by two 14 bit DACs. One DAC controls the driver common output voltage, while a separate DAC controls the DC offset of the differential output pair. The common output voltage has a range from 0V to 2.5V (DAC code 0 to 16383), and a differential voltage range of -50 mV to +50 mV (DAC code 0 to 16383, mid code is 0V). The common output voltage controls the bias to the mixer inputs, which affects the mixer linearity. The user may need to adjust this voltage to optimize linearity (IP3 and harmonics), which is dependent on IF input power level and the operating frequency. A typical bias voltage may range between 1 V and 3 V.

DC offset to the differential modulating mixer is controlled using the DC offset DAC. This DAC provides approximately -50 mV to +50 mV of differential voltage adjustment. Adjusting the DAC offset on both the I and Q channels can minimize the LO leakage at the RF output. This is commonly known as “nulling” out the LO. The process typically involves a few adjustment steps alternating between the I-Channel DAC and the Q-Channel DAC until an optimal level of LO suppression is achieved. However, nulling of the LO is typically only effective over a small amplitude, frequency, and temperature range. Once the operating condition changes significantly, nulling may need to be performed again. For example, if the LO is nulled

at 25 °C, then it will most likely need to be nulled again at 45 °C although the frequency and input amplitude is held relatively constant. Digital data must be corrected to compensate for the modulator IQ amplitude and phase imbalances in addition to nulling out the LO leakage for best application results.

The IQ core, comprising two mixers driven at LO quadrature signals, should not be driven too hard with IF signals because the resulting IMD3 of the RF signal may be too high. It is recommended to use an IF level of 0.5 V peak-to-peak differential or less to lower the IMD3. However if the IF is too low, the RF signal-to-noise ratio may suffer. Subsequently, the user must look at the RF result to determine the optimal IF input level.

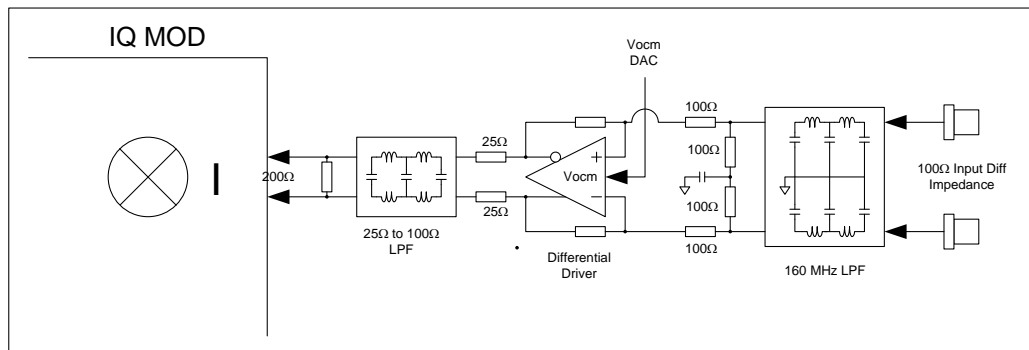


Figure 2. IF input section.

RF Output Section

The output section comprises adjustable digital step attenuators (DSA), RF amplifiers (one is selectable), and a bank of low pass filters to suppress RF harmonics. These are used to condition the RF signal generated in the IQ core component. The RF signal has its best performance exiting the IQ core; therefore it is important to select a suitable RF gain to minimize further degradation of the signal. A general rule is to apply more attenuation earlier in the RF path (close to the IQ core) to improve linearity, and more gain to improve signal-to-noise performance. Generally, for RF frequencies less than 3 GHz, to vary signal levels greater than -20dBm the attenuators RF ATTEN#1, and RF ATTEN#2 should be used as controls. To lower signal levels below -20 dBm, ATTEN#3 and ATTEN#4 should then be used. Although all attenuators in the path have 30 dB of adjustment with 1 dB steps, it is not always recommended to maximize the full range of an attenuator before moving to the next one in the path. As an example, attenuation of ATTEN#1 should rarely be more than 10 dB because any further attenuation will degrade the signal-to-noise ratio at a faster rate than the improvement of IMD3. If any further IMD3 improvement is needed, ATTEN#2 or ATTEN#3 should be used to create the desired attenuation. ATTEN#4 should be fully utilized first to generate signals lower than -30 dBm.

The selectable RF amplifier is used to improve the overall gain of the device for frequencies in the 5 GHz to 6 GHz range, as the RF loss in the RF path become more significant at these higher frequencies. The gain flatness roll-off over the entire operating band is typically 15-16 dB. The RF amplifier has 14 dB of gain. If the overall gain is too excessive at the desired operating frequency, ATTEN#2 should be used to readjust the gain to the required level.

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 the cutoff point may experience a slightly faster roll-off of their IF bandwidth. A typical 1 dB IQ IF bandwidth is about 160 MHz. The user may want to choose a higher frequency filter if this becomes a problem. See the section “Setting the SC5412A: Writing to Configuration Registers” 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 SC5413A 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 modulator. 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 modulator, for example. Driving multiple modulators (or demodulators when working with SignalCore’s SC5313A) 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.

SC5413A PROGRAMMING INTERFACE

Device Drivers

The SC5413A 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 **sc5413a_RegWrite** and **sc5413a_RegRead** functions respectively.

For Microsoft Windows™ operating systems, The SC5413A API is provided as a dynamic linked library, *sc5413a.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 *SC5413A.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://libusb.org>. To install the necessary drivers, right click on the *sc5413a.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 *SC5413A_Readme.txt* file also located under the *\Win* directory.

A LabVIEW API is provided and it consists of function wrappers that call the *sc5413a.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 SC5413A, the Vendor ID is **0x277C** and the PID is **0x0018**.

For Linux systems, the shared library *libsc5413a.so* is provided. To install the shared files and links, type “*make -f makefile.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 this 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 SC5413A. 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 SC5413A 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 **sc5413a_GetDeviceTemperature**, or calls **sc5413a_SetFrequency** to tune the frequency. The software API is covered in detail in the “Software API Library Functions” section.

SETTING THE SC5413A: WRITING TO CONFIGURATION REGISTERS

Configuration Registers

The users may write the configuration registers (write only) directly by calling the **sc5413a_RegWrite** function. The syntax for this function is **sc5413a_RegWrite(deviceHandle, registerCommand, instructWord)**. The **instructWord** takes a 64 bit-word. However, it will only send the required number of bytes to the device. Table 3 summarizes the register addresses (commands) and the effective bytes of command data.

Table 3. Configuration registers.

Register Name	Register Address	Serial Range	Bit 7 (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
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]					
		[23:16]								Channel
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 ATTN#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 amplifier. Setting bit 0 low (0) disables RF amplifier. Setting bit 0 high (1) enables RF amplifier.

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, 0 = ATTN#1, and 3 = ATTN#4. Byte0 and bit [4:0] specifies the attenuation value of the corresponding attenuator set by byte1.

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 to the “LO out” connector. Note that there is always some LO leakage from 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.

Removing DC Offset in Differential Amplifiers

DC_OFFSET_DAC (0x1A) – The DC offsets at the input of the modulator causes LO leakage, adjustment to both the I & Q channel differential DC offsets can minimize the LO leakage. Varying the DAC value from 0 to 16383 can correct up to approximately $\pm 50mV$ of DC offset error. This correction resolution is less than 0.010 mV per LSB. An approximation of the DAC value to offset voltage is given below.

$$DAC\ Value = 16383 \left(\frac{V_{offset} + 0.05V}{0.1V} \right)$$

Setting the Output Linearity of the IQ Modulator

LINEARITY_DAC (0x19) – This DAC controls the bias point of the IQ modulator. A typical value for the bias point is 1.2V, however adjustments are needed to improve linearity at different operating frequency and input IF power levels. Typically, the DAC is set around 1.2 V using the following equation:

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

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 SC5413A: WRITING TO REQUEST REGISTERS

The registers to read data back from the device (such as device status) are accessed through the **sc5413a_RegRead** function. The function and parameter format for this command is **sc5413a_RegRead(deviceHandle, registerCommand, instructWord,*dataOut)**. Any instructions in addition to the register call are placed into “instructWord”, and data obtained from the device is returned via the pointer value **dataOut**. The set of request registers are shown in Table 4.

Table 4. Query registers.

Register Name	Register Address	Serial Range	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
GET_TEMPERATURE	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 **sc5413a_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:

$$\text{Positive Temperature (bit 13 is 0)} = \text{ADC code} / 32$$

$$\text{Negative Temperature (bit 13 is 1)} = (\text{ADC code} - 8192) / 32$$

It is not recommended to read the temperature too frequently, especially once the temperature of the SC5413A 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 returns the device status information such as phase lock status of the PLL, current reference settings, etc. Data is contained in the first three bytes.

Table 5. Description of the status data bits.

Bit	Description
[3]	RF AMP 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 next 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 6.

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 = uint32_value | (byte_value[count] <<
(count*8));
    count++;
}

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

CALIBRATION EEPROM MAP

Table 6. Calibration EEPROM map.

EEPROM ADDRESS (HEX)	NUMBER OF DATA POINTS	TYPE	DESCRIPTION
0	1	U32	Manufacturing information
4	1	U32	Product serial number
8	1	U32	RF module number
C	1	U32	Product manufacture date
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 SC5413A API dynamic linked library or LabVIEW library are:

- **sc5413a_SearchDevices**
- **sc5413a_OpenDevice**
- **sc5413a_CloseDevice**
- **sc5413a_RegWrite**
- **sc5413a_RegRead**
- **sc5413a_InitDevice**
- **sc5413a_SetDeviceStandby**
- **sc5413a_SetFrequency**
- **sc5413a_SetLinearityDac**
- **sc5413a_SetLoFilter**
- **sc5413a_SetLoOut**
- **sc5413a_SetRfAmplifier**
- **sc5413a_SetRfAttenuators**
- **sc5413a_WriteUserEeprom**
- **sc5413a_StoreStartupState**
- **sc5413a_SetRfFilter**
- **sc5413a_GetDeviceInfo**
- **sc5413a_GetDeviceStatus**
- **sc5413a_GetTemperature**
- **sc5413a_ReadCalEeprom**
- **sc5413a_ReadUserEeprom**
- **sc5413a_SetDcOffsetDac**
- **sc5413a_SetRfGain**
- **sc5413a_SetRfPath**

Each of these functions is described in more detail on the following pages. Example code written in C/C++ is located in the `\Win\Driver\src` directory to show how these functions are called and used. First, for C/C++, we define the constants and types which are contained in the C header file, *sc5413a.h*. These constants and types are useful not only as an include for developing user applications using the SC5413A 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 0xFFFF)
#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 *sc5413a.dll* dynamic linked library for the Windows™ operating system. These functions are also provided in the LabVIEW library, *sc5413a.llb*. The LabVIEW functions contain context-sensitive help (Ctrl-H) to assist with understanding the input and output parameters.

Function: **sc5413a_SearchDevices**

Definition: **int** sc5413a_SearchDevices(**char** **serialNumberList)

Output: **char** **serialNumberList (pointer list to serialNumberList)

Description: **sc5413a_SearchDevices** searches for SignalCore SC5506A 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 **sc5413a_OpenDevice** function on how to open a device.

Function: **sc5413a_OpenDevice**

Definition: **deviceHandle** *sc5413a_OpenDevice(**char** *devSerialNum)

Input: **char** *devSerialNum (pointer serial number list)

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

Description: **sc5413a_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: `sc5413a_CloseDevice`

Definition: `int sc5413a_CloseDevice (deviceHandle *deviceHandle)`

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

Description: `sc5413a_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
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 = sc5413a_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++;
}
/** sc5413a_OpenDevice, open device 0
devHandle = sc5413a_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 = sc5413a_CloseDevice(devHandle);
```

Function: `sc5413a_RegWrite`

Definition: `int sc5413a_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: `sc5413a_RegWrite` writes the instructWord data to the register specified by the commandByte. See the register map on Table 3 for more information.

Example: To set the frequency to 2 GHz:

```
int status = sc5413a_RegWrite(devHandle, RF_FREQUENCY, 2000000000); // set frequency to 2 GHz
```

Function: **sc5413a_RegRead**

Definition: **int** sc5413a_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: **sc5413a_RegRead** reads the data requested by the instructWord data to the register specified by the commandByte. See

Table 4 (register map) for more information.

Example: To read the status of the device:

```
unsigned int deviceStatus;  
  
int status = sc5413a_RegRead(devHandle,  
GET_DEVICE_STATUS, 0x00, &deviceStatus);
```

Function: **sc5413a_InitDevice**

Definition: **int** sc5413a_InitDevice (**deviceHandle** *deviceHandle, **bool** mode)

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

Description: **sc5413a_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: **sc5413a_SetFrequency**

Definition: **int** sc5413a_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: **sc5413a_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: **sc5413a_SetRfAmplifier**

Definition: **int** sc5413a_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: **sc5413a_SetRfAmplifier** enables or disables the RF amplifiers.

Function: **sc5413a_SetRfPath**

Definition: **int sc5413a_SetRfPath (deviceHandle *deviceHandle, bool mode)**

Input: **deviceHandle** *deviceHandle (handle to the opened device)
bool mode (0=main path, 1=aux path)

Description: **sc5413a_SetRfPath** select the RF input port.

Function: **sc5413a_SetLoOut**

Definition: **int sc5413a_SetLoOut(deviceHandle *deviceHandle, bool mode)**

Input: **deviceHandle** *deviceHandle (handle to the opened device)
bool mode (0=disable, 1= enable)

Description: **sc5413a_SetLoOut** enables the LO output port. The LO input signal is replicated and piped out through these LO output port.

Function: **sc5413a_SetRfAttenuation**

Definition: **int sc5413a_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: **sc5413a_SetRfAttenuation** Sets the attenuation of the RF attenuators.

Function: **sc5413a_SetRfFilter**

Definition: **int sc5413a_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: **sc5413a_SetRfFilter** selects the active filter in the RF filter bank.

Function: **sc5413a_SetLoFilter**

Definition: **int sc5413a_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: **sc5413a_SetLoFilter** selects the active filter in the LO filter bank.

Function: **sc5413a_SetDcOffsetDac**

Definition: **int** sc5413a_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: **sc5412a_SetDcOffsetDac** sets the DC offset voltage at the IQ modulator core. Voltage adjust is approximately +/- 0.05 V. The default factory setting is 8038.

Function: **sc5413a_SetLinearityDac**

Definition: **int** sc5413a_SetLinearityDac (**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: **sc5412a_SetLinearityDac** sets the bias point of the IQ modulator, which affects the linearity of the device. A DAC value of 3935 is recommended and is also the default factory setting.

Function: **sc5413a_WriteUserEeprom**

Definition: **int** sc5413a_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: **sc5413a_WriteUserEeprom** writes one byte of data to the memory address specified.

Function: **sc5413a_StoreCurrentState**

Definition: **int** sc5413a_StoreCurrentState(**deviceHandle** *deviceHandle)

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

Description: **sc5413a_StoreCurrentState** stores the current state of the devices as the default power-up state.

Function: **sc5413a_GetDeviceInfo**

Definition: **int** sc5413a_GetDeviceInfo(**deviceHandle** *deviceHandle, **deviceInfo_t** *devInfo)

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

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

Description: **sc5413a_GetDeviceInfo** retrieves device information such as serial number, calibration date, revisions, etc.

Function: **sc5413a_GetDeviceStatus**

Definition: **int** sc5413a_GetDeviceStatus (**deviceHandle** *deviceHandle, **deviceStatus_t** *deviceStatus)

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

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

Description: **sc5413a_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 = SC5413A_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: **sc5413a_GetTemperature**

Definition: **int** sc5413a_GetTemperature (**deviceHandle** *deviceHandle, **float** *temperature)

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

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

Description: **sc5413a_GetTemperature** retrieves the internal temperature of the device.

Function: **sc5413a_ReadCalEeprom**

Definition: **int** sc5413a_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: **sc5413a_ReadCalEeprom** reads back a byte from a specific memory address of the calibration EEPROM.

Function: **sc5413a_ReadUserEeprom**

Definition: **int** sc5413a_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: **sc5413a_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, which is 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 controller monitors the SERIAL_READY bit and performs transfers only when it is asserted high to avoid miscommunication.

All data transferred to and from the device are clocked on the falling edge (MODE 1) of the clock as shown in Figure 4; data is clocked in on the rising edge when pin #18 is pulled to ground (MODE 0). 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 μs 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 μs to 150 μs after reception. The user may choose to wait a minimum of 150 μs or query the SERIAL_READY bit before sending in another command; the latter is recommended for robustness.

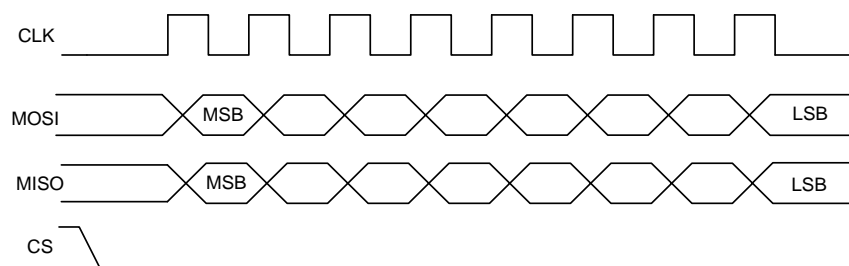


Figure 4. SPI Mode - Data clocked in on falling clock edge.

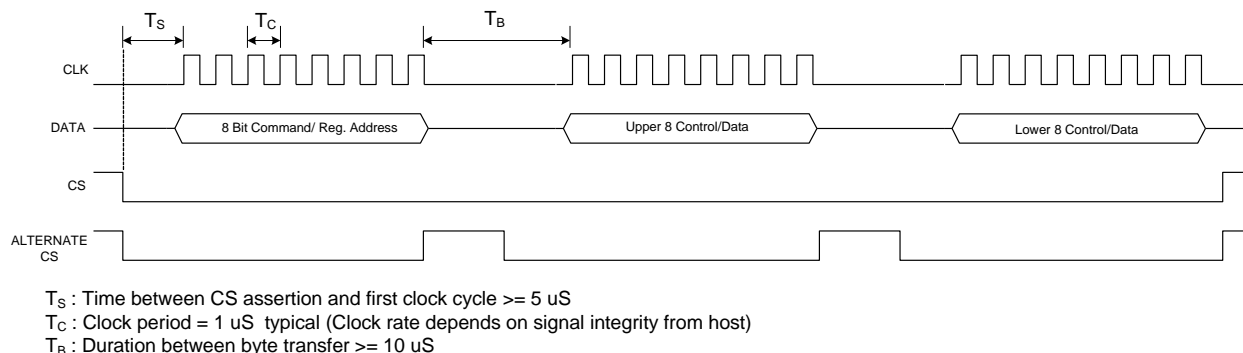


Figure 5. SPI timing.

Additional SPI registers

There are 2 additional registers available for SPI communication as shown in Table 7. 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 7 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)
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 note in the above set of registers in Table 3. 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 the 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 3 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”.

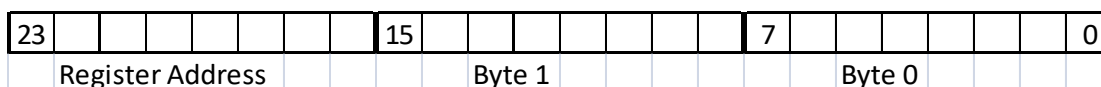


Figure 6. Single 3 byte transfer buffer.

Reading the SPI Bus

Data is simultaneously read back during a SPI transfer cycle. Requested data from a prior command (see Table 8) 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.

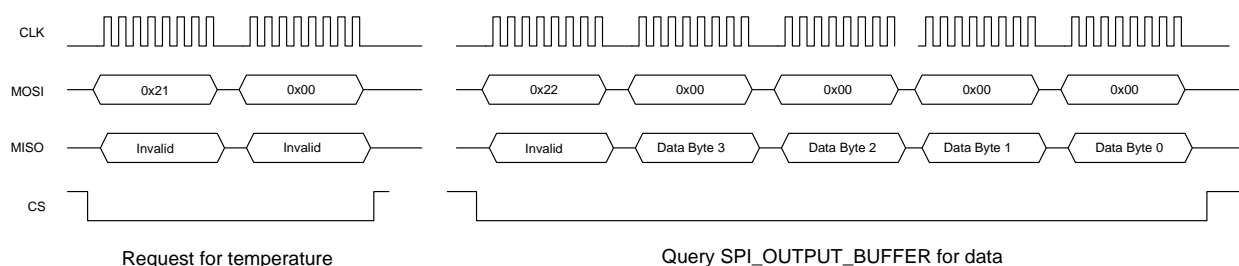


Figure 7. Reading queried data.

In the above example, valid data is present in the last 2 bytes; byte 1 and byte 0. Table 8 shows the valid data bytes associated with the querying register.

Table 8. 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

PROGRAMMING THE RS-232 INTERFACE

The RS-232 version of the SC5413A has a standard interface buffered by an RS-232 transceiver so that it may interface directly with many host devices, such as a desktop computer. The interface connector for RS-232 communication is labeled “Digital I/O” on the front of the panel. Refer to **Error! Reference source not found.** and **Error! Reference source not found.** for position and pin-out information. The device communication control set is provided in Table 9 below.

Table 9. RS-232 communication settings.

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

Writing to the Device via RS-232

It is important that all necessary bytes associated with any one register are fully sent. In other words, if a register requires a total of four bytes (address plus data) then all four bytes must be sent even though the last byte may be a null. The device, upon receiving the first register addressing byte, will wait for all the associated data bytes before acting on the register instruction. Failure to complete the register transmission will cause the device to behave erratically or hang. Information for writing to the configuration registers is provided in **Error! Reference source not found.**

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 that the device is ready for the next register command. It is important to **clear the incoming RX buffer** on the host by querying or force flushing it to avoid incoming data corruption of querying registers. The return byte value is 1 for a successful configuration and 0 for an unsuccessful configuration.

Reading from the Device via RS-232

To query information from the device, the query registers are addressed and data is returned. Returned data vary in length, which are dependent on the register call. Table 4 contains the query register information. As with the configuration registers, it is important that the data byte(s) associated with the query registers are sent even if they are nulls. The returned data length is also detailed in the “Querying the SC5413A: Writing to Request Registers” section. Table 11 summarizes the number of returned bytes for RS232; note that this is not the same as the SPI return byte lengths.

Table 10 RS232 Returned data bytes for query registers

Register Name	Register Address	Returned Bytes
GET_TEMPERATURE	0x20	2
GET_DEVICE_STATUS	0x21	1
USER_EEPROM_READ	0x23	1
CAL_EEPROM_READ	0x24	1

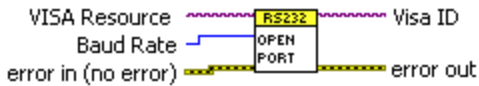
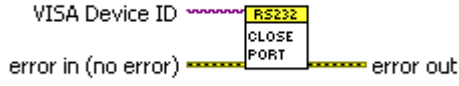

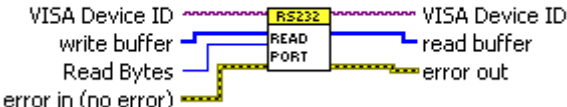
RS232 Windows™ API

An API for the windows platform is provided in the sc5413a_rs232.dll dynamic library, which is located in the installation directory or the USB driver under the Win\API\RS232 directory. An example code in C/C++ is provided to demonstrate how this DLL is called. Source code is available upon request.

Using the LabVIEW Functions and NI-VISA

Functions for RS-232 control are provided in LabVIEW and use NI-VISA. The low level port control functions are unique to RS-232 and are contained in the subdirectory **UsartPort**. These functions are listed in Table 11 below for convenience. All provided LabVIEW VI functions are not protected, so the user may open them to understand how the register calls are made.

Table 11. LabVIEW RS-232 port access functions.

Function	Description
<p>Rs232OpenPort.vi</p> 	Opens the VISA session to the serial port associated with the device. Option to select between 2 Baud rates.
<p>Rs232ClosePort.vi</p> 	Closes the VISA session associated with the serial port.
<p>Rs232WritePort.vi</p> 	Writes the data bytes in the buffer to the opened port. The data buffer is a byte array. Most significant data is sent first.
<p>Rs232ReadPort.vi</p> 	Writes the query register data and reads back the data associated with the register. The buffers are byte arrays. Most significant data is sent and received first. The number of bytes to be read back from the queried register needs to be specified.

CALIBRATION & MAINTENANCE

The SC5413A does not receive a factory calibration. The SC5413A 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 SC5413A. Should users require SignalCore to perform any calibration, please contact SignalCore support directly.

REVISION NOTES

Rev 1.1.0	Added RS232 programming information
Rev 1.2.0	<ul style="list-style-type: none">- Corrected pin outs for the digital connector- Removed legacy information for SPI communication
Rev 1.3.0	Address Removed