SignalCore™
PRESERVING SIGNAL INTEGRITY

SC800

nanoSynth®

INTEGRATED 25 MHZ TO 6 GHZ SMT SYNTHESIZER

GENERAL DESCRIPTION

The SC800 nanoSynth® is a fully integrated broadband CW signal synthesizer that combines multiple PLL, DDS, and frequency dividers into a rugged and miniature 2"x1" surface mountable package. The output frequency range is 25 MHz to 6 GHz with average output power of +10 dBm. Tuning at 1 Hz resolution, the multiple PLL architecture eliminates close-in phase spurs that are associated with fractional-N PLLs. The SC800 has low phase noise of -118 dBc/Hz at 10 kHz offset from a 1 GHz carrier. The SC800 integrates low noise linear regulators and an output RF amplifier to greatly improve the pushing and pulling performance. To simplify user communication with the device, an onboard microprocessor performs all necessary computations to control and set the output frequency, reducing the complexity and number of instruction registers.

Refer to the theory of operation and register map sections of this document for more information.

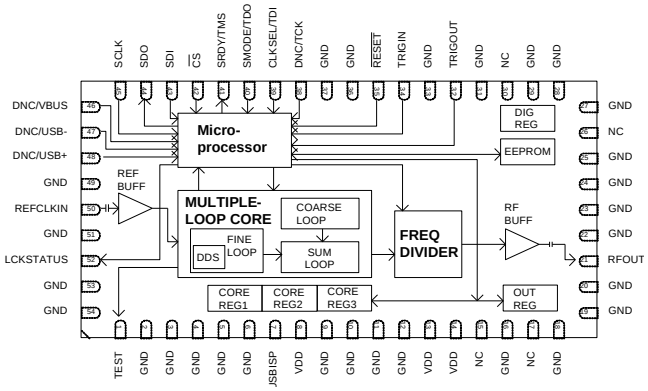
PRODUCT FEATURES

- 25 MHz to 6 GHz output range
- 1 Hz frequency tuning resolution
- Residual phase noise better than -118 dBc/Hz at 10 kHz offset from a 1 GHz carrier
- Rugged and miniature 2"x1" SMT package
- Frequency list mode
- Single supply operation

TYPICAL APPLICATIONS

- Automated device/IC testers
- Test and measurement equipment
- Wireless communication equipment
- Frequency converter local oscillator
- Digital data converter clock source
- Network equipment

FUNCTIONAL BLOCK DIAGRAM



SPECTRAL SPECIFICATIONS

RF output frequency range ..... 25 MHz to 6 GHz

Tuning

Resolution ..... 1 Hz

Speed (settled to .1 ppm) [1] ..... < 500 us

Sideband phase noise [2] (dBc/Hz)

RF Frequency				
Offset	**.1**	**1 GHz**	**3 GHz**	**6 GHz**
100 Hz	-108	-93	-86	-80
1 kHz	-130	-113	-106	-99
10 kHz	-135	-118	-110	-102
100	-136	-118	-110	-103
1 MHz	-150	-140	-132	-125
10 MHz	-152	-152	-149	-147

Sideband phase spurs

< 500 kHz ..... -60 dBc typical

500 kHz to 2 MHz ..... -65 dBc typical

Reference frequency

CLKSEL (state)	0	1
Frequency (MHz)	200	100

AMPLITUDE SPECIFICATIONS

Output power

500 MHz ..... +7 dBm typical

3000 MHz ..... +5 dBm typical

6000 MHz ..... +0 dBm typical

2nd order harmonics ..... < -15 dBc typical

Sub-harmonics [3] ..... < -60 dBc typical

Spurious signals [4] ..... < -65 dBc typical

Reference input power

Minimum ..... +3 dBm

Typical ..... +7 dBm

Maximum ..... +10 dBm

ELECTRICAL SPECIFICATIONS

Parameter	Min	Typ	Max	Units
Voltage supply VDD	3.3	3.6	5.5	V
Supply current IDD	870	890	910	mA
Total power dissipation (3.6V)	3.15	3.20	3.25	W
Low-level input logic voltage	-0.3		0.8	V
High-level input logic voltage	2.0		3.6	V
Low-level output logic voltage			0.4	V
High-level output logic voltage	2.9			V

ABSOLUTE MAXIMUM RATINGS

Parameter	Rating
Supply voltage VDD	5.5 V
Logic voltage	3.6 V
Continuous power dissipation	5 W
Storage temperature	-10 to +125 °C
Operating temperature (measured case temperature)	0 to +85 °C

NOTES

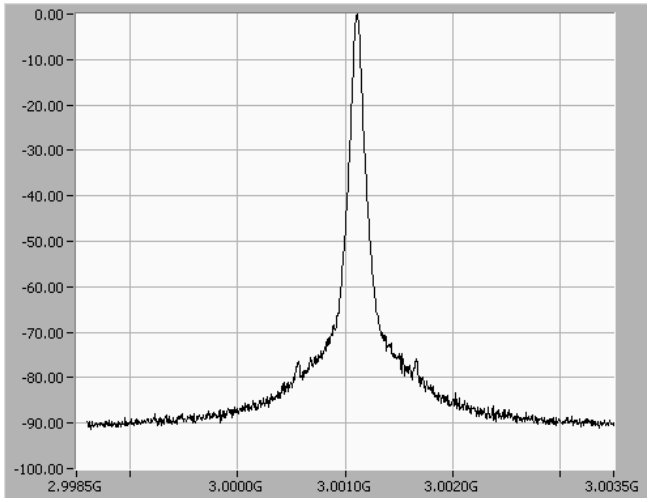
1. For tuning steps less than 10 MHz.
2. The phase noise is largely dependent on the external reference. The specifications are based on a reference signal with the following maximum phase noise levels normalized to 100 MHz:

Offset	100	1k	10k	100k	1M
dBc/Hz	-115	-140	-155	-165	-165

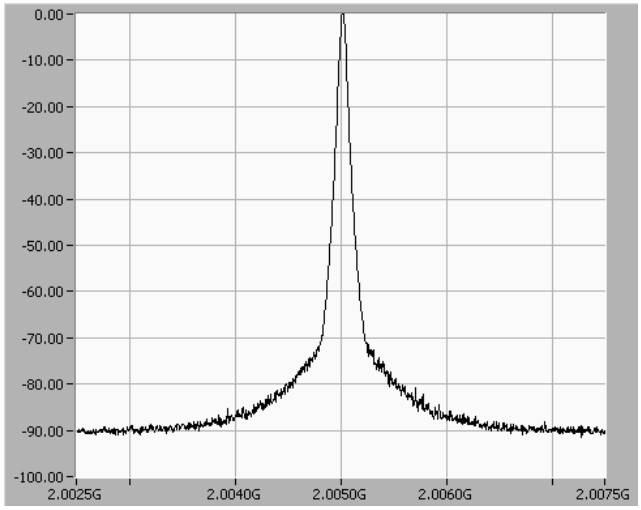
3. The fundamental frequency range of the final oscillator is 3 GHz to 6 GHz. These sub-harmonics are not due to multiplication but are due to leakage of the divided signal.
4. Spurious signals are due to intermodulation of the internal oscillator signals. They generally reside from a few 100 kHz and greater away from the carrier.

PHASE NOISE @ 2.00501 GHZ

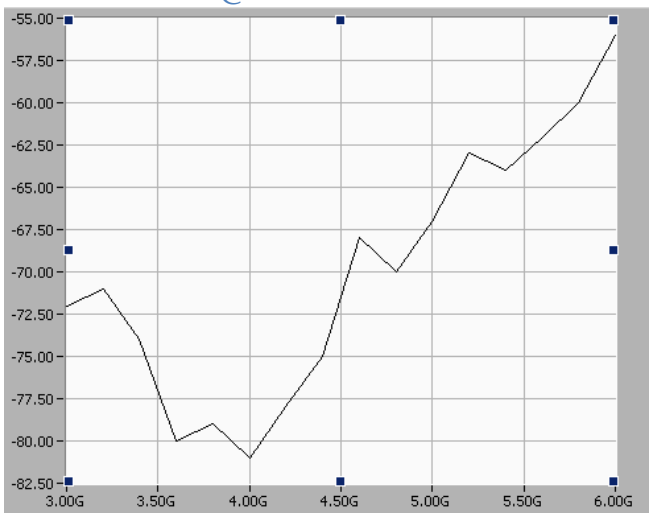
CF = 3.0011 GHZ, SPAN 5 MHZ



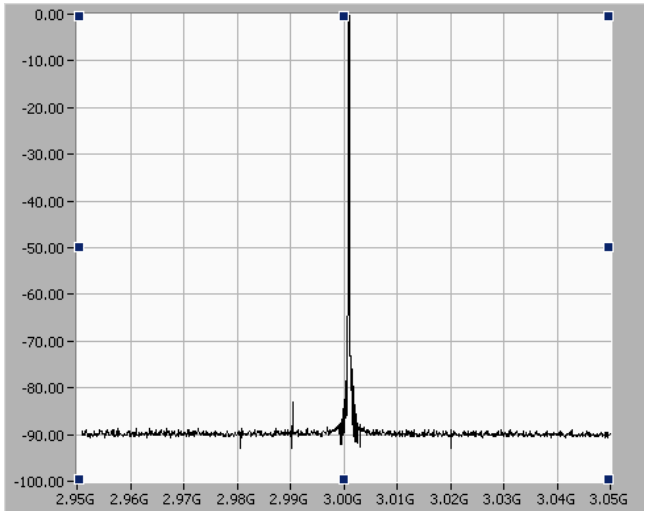
CF = 2.00501 GHZ, SPAN 5 MHZ



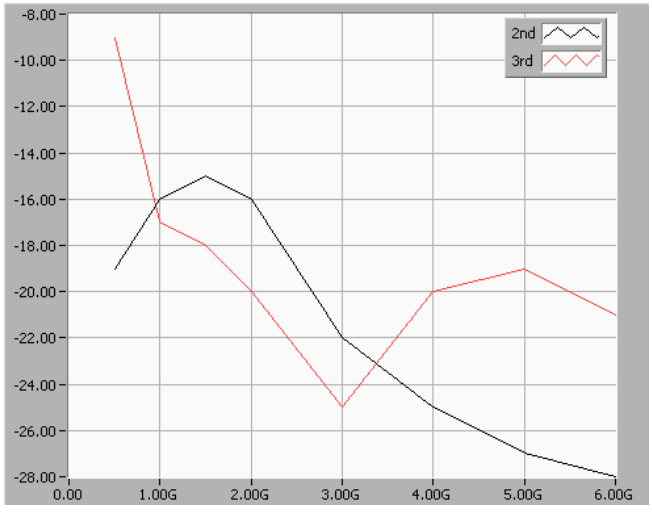
SUB-HARMONICS @ CF= 3-6 GHZ



CF = 3.0011 GHZ, SPAN 100 MHZ



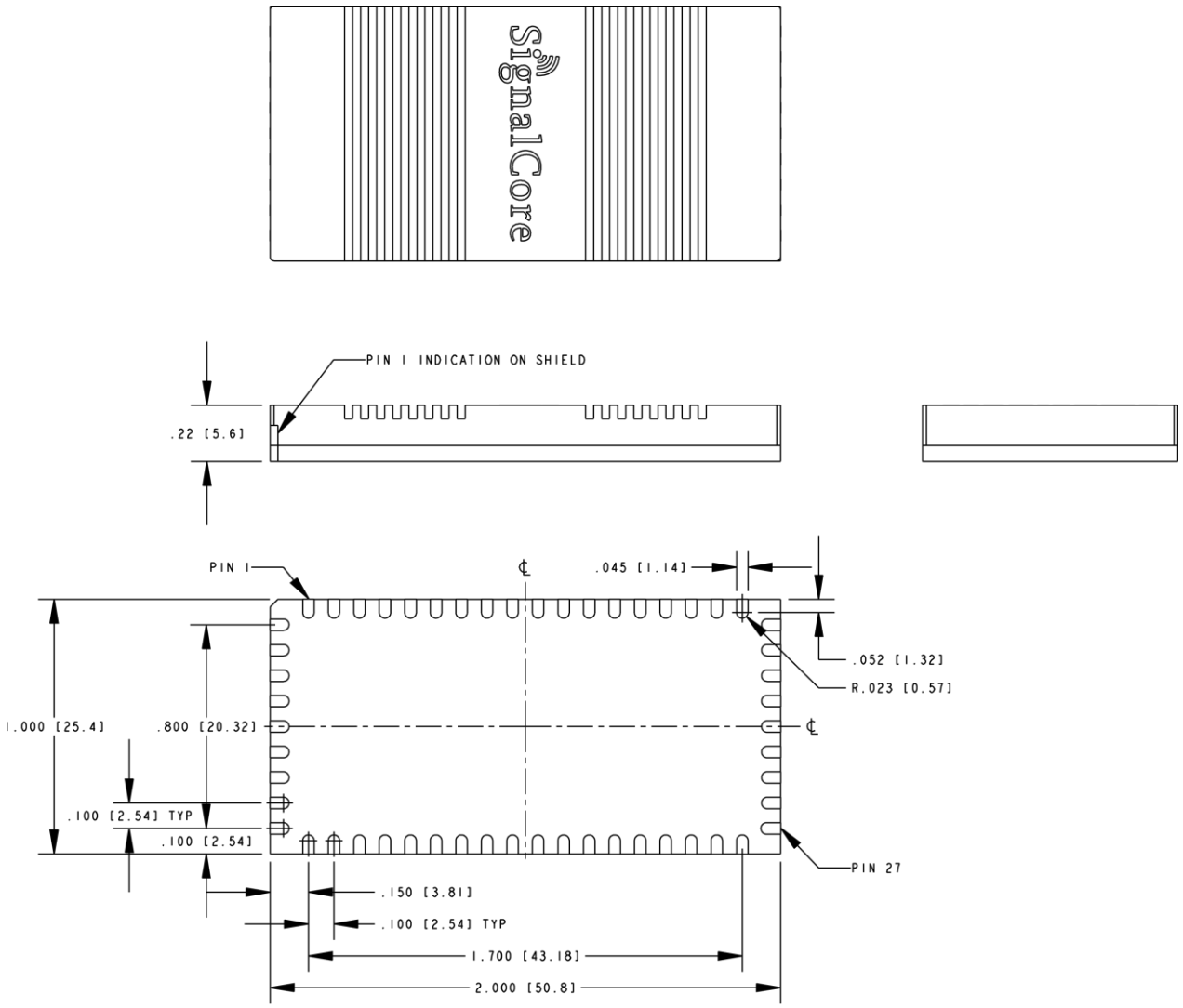
HARMONICS



PIN DESCRIPTION

Pin Number	Function	Description
2, 3, 4, 5, 6, 9, 10, 11, 12, 16, 18, 19, 20, 22, 23, 24, 25, 27, 28, 29, 31, 33, 36, 37, 49, 51, 53, 54	GND	Must be connected to RF or DC ground. It is important to place as many ground vias as possible around or in these ground pads to improve signal performance as well as thermal conduction from the device to the board.
8, 13, 14	VDD	Positive supply voltage inputs.
15, 17, 26, 30	NC	Not physically connected internally.
1	TEST	Factory test pin. Terminate with 1000pF to GND.
7	DNC/USBISP	If held logic low during a reset event or power-up, this pin will put the device into in-system programming (ISP) mode for firmware upgrades. This pin should always be pulled high using a resistor (10k recommended).
21	RFOUT	AC-coupled RF output port.
32	TRIGOUT	May be programmed to put out a trigger pulse for list mode events. See the LIST_MODE_CONFIG register for more information.
34	TRIGIN	Hardware trigger to start/stop or step a sweep/list event. See the LIST_MODE_CONFIG register for more information.
35	$\overline{\text{RESET}}$	Logic low will put the device in reset. On power up, it is recommended to send a reset signal to the device to ensure proper operation.
38, 39, 40, 41	DNC/TCK, CLKSEL/TDI, SMODE/TDO , SRDY/TMS	Along with pin 35, these JTAG programming pins are used by the factory to set the device firmware. The user may wire them for field firmware updates. Under normal operation, these pins should be pulled high with a 10k resistor. Pin 38 is reserved strictly for the JTAG TCK function. Pin 39 (CLK_SEL) selects either a 200/100 MHz reference on 0/1 respectively. Pin 40 (SMODE) pulled low disables the USB port. Pin 41 (SRDY) is the serial ready monitor pin.
42, 43, 44, 45	$\overline{\text{CS}}$, SDI, SDO, SCLK	4-wire SPI bus. Device is always in slave mode. SDI is data from the master/host, SDO is data to the master/host, SCLK is the host clock, and $\overline{\text{CS}}$ is chip select. See section on Serial Programming for more information.
46, 47, 48	VBUS, USB-, USB+	Optional USB interface to the device. They may be left unconnected if the feature is not available or not used on the device.
50	REFCLKIN	Reference signal input. The clock frequency of the reference should match the setting of Pin 39. This pin is AC-coupled.
52	LCKSTATUS	Active high when all phase-locked loops are locked. If one or more of the loops fail to achieve lock, pin will go low.

MECHANICAL DATA



NOTES:

1. NO CUSTOMER ROUTING RESTRICTIONS UNDER COMPONENT. ALL VIAS ON BOTTOM SIDE ARE PROTECTED AGAINST SHORTS TO CUSTOMER'S PCB BY PROTECTIVE SOLDERMASK LAYER. SIGNALCORE RECOMMENDS CUSTOMER'S ALSO PROVIDE PROTECTIVE SOLDERMASK AREA UNDER COMPONENT FOR ADDITIONAL PROTECTION.
2. ALL COMPONENT PADS ARE IMMERSION GOLD OVER ELECTROLESS NICKEL (ENIG) PLATED.
3. COMPONENT IS COMPATIBLE WITH RoHS (LED-FREE) REFLOW PROCESSING PROFILES.

ORDER INFORMATION

7100046-01 ..... SC800, nanoSynth 6 GHz Integrated SMT Synthesizer

THEORY AND OPERATION

Despite its small size, the SC800 is an instrument-grade, high performance surface mount synthesizer with easy to program register-level control. It functions as a standard synthesized source with the added capability of a sweep/list mode that makes it ideal for applications ranging from automated test systems to telecommunication equipment to scientific research labs. Being small and fully integrated, it is the ideal solution for board-level designs that require a high performance RF source(s) without having to invest much engineering effort. Figure 1 shows the block diagram of the device, and the following sub-sections provide details to its operation.

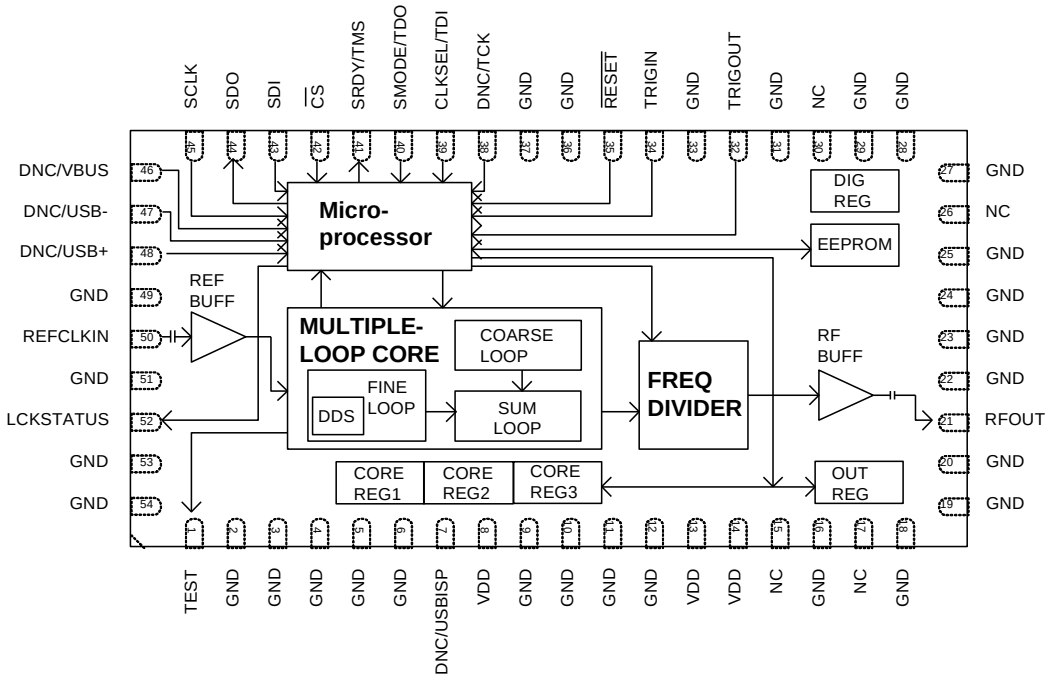


Figure 1. Block diagram of the SC800.

RF GENERATION

The SC800 is a 25 MHz to 6 GHz low phase noise and low spur synthesizer, using a hybrid PLL loop architecture comprised of three phase-locked loops and a DDS function. All PLLs are operated in integer division feedback mode, thus circumventing fractional feedback mode-induced spurs. Fine tuning is accomplished with the variable modulus DDS, providing exact frequency generation. Isolation between the internal oscillators, their mixed IF products, harmonics, and inter-modulation products are accomplished by internal EMI sealed cavities. Using a hybrid PLL loop architecture with well-shielded cavities improves the overall phase noise performance and reduces the spurious signal content of this compact size frequency synthesizer. Signals are synthesized from the external reference clock; either a 100 MHz or 200 MHz clock. A 200 MHz clock reduces the side band spurs. The DDS running at a higher clock rate reduces the amount and levels of DDS-related spurs.

Furthermore there will be less inter-modulation spurs that are products of the reference harmonics and LO signals.

SUPPLY REGULATION

Each section of the synthesizer has an ultra low noise linear regulator, providing not just low supply noise but further signal isolation that could otherwise leak through common supply lines. There are eight such regulators to serve the output RF section, RF feedback section, summing PLL, coarse PLL, fine PLL, mixer IF section, DDS, and the microcontroller and its related circuitry. Most of these regulators can be turn-off to reduce power consumption by invoking the DEVICE_STANDBY register (see the *Device Registers* section).

INTERNAL EEPROM

The SC800 contains an EEPROM whose memory space is divided into calibration and user data spaces. The calibration data space contains SC800 device information such as serial number, hardware revision, firmware revision, and production date. In addition, this space holds the calibration data for the wideband primary 3 GHz – 6 GHz VCO. This VCO calibration data is used to properly program the assisting VCO DAC to assure locking and improve tuning speed. The calibration space of the EEPROM is not user modifiable.

The user data space contains the default startup configuration of the device such as the single fixed tone mode frequency and sweep/list mode operation. It also holds the list mode configuration parameters such as sweep behavior (saw or triangular waveform), software or hardware trigger, start/stop/step frequencies, dwell time, sweep/list cycles, etc. Space is allocated for 2048 frequency points that the user may choose to store for list mode operation.

MODES OF RF GENERATION

The SC800 has both single fixed tone and list mode operation. In single fixed tone mode, it operates as a normal synthesizer where the user writes the frequency (RF_FREQUENCY) register to change the frequency. In list mode, the device is triggered to automatically run through a set of frequency points that are either entered directly by the user or pre-computed by the device based on user parameters. Configuration of the device for list mode operation is accomplished by setting up the LIST_MODE_CONFIG register.

Sweep Function

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

List Function

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

Sweep Direction

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

Sweep Waveform

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

Dwell Time

The dwell time at each frequency, in either sweep or list modes, is determined by writing to the LIST_DWELL_TIME register. The dwell time step increment is 500 µs. However, the recommended minimum dwell time is 1 ms, which allows sufficient time for the signal to settle before a measurement is made. Due to the size limitation of the onboard RAM, it is not possible to have a pre-calculated configuration parameters list that could be used to program the various functions of the device, decreasing the setup time of the device for frequency change. As a result, for each frequency change the configuration parameters are dynamically computed. This overhead computational time to handle the mathematics, triggers, timers, and interrupts may increase the effective settling time close to or slightly exceeding 500 µs.

List Cycles

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

Trigger Sources

The device may be set up for software or hardware triggering. This is defined in bit [4] of the LIST_MODE_CONFIG register. If software trigger is selected, writing the LIST_SOFT_TRIGGER register will trigger the device to perform the sweep/list function defined in the

LIST_MODE_CONFIG register. The device may also be triggered via pin 34, the hardware trigger pin (TRIGIN). Hardware trigger occurs on a high to low transition state of this pin.

Trigger In Modes

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

Trigger Out Modes

The device can be set to send out a high to low transition signal when the configuration of a frequency by the device is completed; that is, it has completed all necessary computations, and has successfully written data to the appropriate components. This trigger pulse can be sent on the completion of every step frequency or on the last frequency of a sweep cycle. This trigger signal is present on pin 32 (TRIGOUT).

COMMUNICATION INTERFACES

The default communication channel to the device is via the SPI bus. An optional USB interface is also available. If the USB port is not used or available for controlling the device, it is nevertheless recommended to wire the USB interface pins to a connector for future firmware upgrade capability.

SPI Interface

In addition to the 4-wire SPI ($\overline{CS}$, SDO, SDI, and SCLK) signal lines, there is also an alternative serial ready SRDY line. Upon reception of a register command, the device takes time to execute the command instruction, such as setting a new frequency. While the device is busy, the SRDY line will go low and returns high upon execution completion. Pulling pin 40 (SMODE) high enables simultaneous operation of SPI and USB, and grounding this pin disables the USB. It is highly recommended that the USB port be disabled if it is not used. Detailed SPI read and write operations are discussed in detail in the *Serial Peripheral Interface (SPI)* section.

USB Interface

The SC800 has an built-in USB controller configured in client mode. The three wires VBUS, USB-, and USB+ can be routed directly to a USB connector or an embedded host port to access the device. The transfer types supported by the device are control, interrupt, and bulk. The USB port can be turned off by grounding or pulling low pin 40. More information on the use of the USB interface and its software API are provided in the *USB Interface* section.

DEVICE REGISTERS

Communication to the SC800 is performed by writing to and reading from its set of control and query registers respectively. The control registers are used to set/configure the device, hence a one-way communication. The query registers on the other hand request the device to perform an operation with the expectancy of returned results, hence a two-way communication. The table below lists the device registers and provides the necessary details for each.

Table 1. Register 0x02 RF_FREQUENCY (5 Bytes)

Bit	Type	Name	Width	Description
[39:0]	WO	Frequency Word	40	Sets the single fixed tone frequency.

Table 2. Register 0x04 RF_MODE (1 Byte)

Bit	Type	Name	Width	Description
[0]	WO	RF Mode	1	1 = Sweep/list mode. In this mode writing to register 0x02 will be unresponsive. This register must be called first for sweep/list triggering to function. 0 = Single fixed tone mode. This mode must be set to change the frequency value via register 0x02.
[7:1]	WO	Reserved	7	Set all bits to 0.

Table 3. Register 0x05 LIST_MODE_CONFIG (2 Bytes)

Bit	Type	Name	Width	Description
[0]	WO	List/Sweep	1	0 = List mode. Device gets its frequency points from the list buffer uploaded via the LIST_BUFFER_WRITE register (0x0D). 1 = Sweep mode. The device computes the frequency points using the start, stop, and step frequencies.
[1]	WO	Sweep Direction	1	0 = Forward. In the forward direction, the sweeps starts from the lowest start frequency or starts at the beginning of the list buffer. 1 = Reverse. In the reverse direction, the sweep starts with the stop frequency and steps down toward the start frequency or starts at the end and steps toward the beginning of the buffer.

[2]	WO	Sweep Waveform	1	0 = Sawtooth waveform. Frequency returns to the beginning frequency upon reaching the end of a sweep cycle. 1 = Triangular waveform. Frequency reverses direction at the end of the list and steps back towards the beginning to complete a cycle.
[3]	WO	Soft/Hardware Trigger	1	0 = Software trigger. Software trigger can only be used to start and stop a sweep/list cycle. It does not work for step-on-trigger mode. 1 = Hardware trigger. A high-to-low transition on the TRIGIN pin will trigger the device. It can be used for both start/stop or step-on-trigger functions.
[4]	WO	Start/Stop or Step	1	0 = Start/Stop behavior. The sweep starts and continues to step through the list for the number of cycles set, dwelling at each step frequency for a period set by the LIST_DWELL_TIME register. The sweep/list will end on a consecutive trigger. 1 = Step-on-trigger. This is only available if hardware triggering is selected. The device will step to the next frequency on a trigger. Upon completion of the number of cycles (set by the LIST_CYCLE_COUNT register), the device will exit from the stepping state and stop. Further triggering will set the device back into the stepping state. To exit the stepping state and stop before reaching the end of a cycle, a software trigger must to be sent or a change in the RF mode to single fixed tone needs to be made.
[5]	WO	Return to Start	1	0 = Stop at end of sweep/list. The frequency will stop at the last point of the sweep/list. 1 = Return to start. The frequency will return and stop at the beginning point of the sweep or list after a cycle.
[6]	WO	Trigger Output	1	0 = No trigger output. 1 = Puts a trigger pulse on the TRIGOUT pin
[7]	WO	Trigger Out Mode	1	0 = Puts out a trigger pulse at each frequency change, right after all internal devices are configured. 1 = Puts out a trigger pulse at the completion of each sweep/list cycle.
[15:8]	WO	Reserved	8	Set all bits to 0.

Table 4. Register 0x06 LIST_SOFT_TRIGGER (1 Byte)

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

Table 5. Register 0x07 LIST_START_FREQ (5 Byte)

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

Table 6. Register 0x08 LIST_STOP_FREQ (5 Bytes)

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

Table 7. Register 0x09 LIST_STEP_FREQ (5 Bytes)

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

Table 8. Register 0x0A LIST_DWELL_TIME (4 Bytes)

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

Table 9. Register 0x0B LIST_CYCLE_COUNT (4 Bytes)

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

Table 10. Register 0x0C LIST_BUFFER_POINTS (4 Bytes)

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

Table 11. Register 0x0D LIST_BUFFER_WRITE (5 Bytes)

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

Table 12. Register 0x0E LIST_BUF_MEM_TRNSFER (1 Byte)

Bit	Type	Name	Width	Description
[0]	WO	Transfer Direction	1	0 = Transfers the contents of the list buffer into EEPROM memory. The size of the transfer is set by the list frequency points. 1 = Transfers the contents from EEPROM memory to the list buffer (in RAM).
[7:1]	WO	Reserved	7	Set all bits to 0.

Table 13. Register 0x0F STORE_DEFAULT_STATE

Bit	Type	Name	Width	Description
[7:0]	WO	Reserved	8	Set all bits to 0. Calling this register will store the current configuration into memory. On reset or power-up these values are read from memory and set as the default values. These values are: ➢ RF frequency ➢ List mode configuration ➢ RF mode ➢ Start/Stop/Frequency ➢ Dwell time ➢ Sweep/List cycles ➢ List buffer from EEPROM

Table 14. Register 0x10 DEVICE_STANDBY

Bit	Type	Name	Width	Description
[0]	WO	Device Standby	1	0 = Takes device out of standby. 1 = Puts device into standby. Most analog circuits are powered down, reducing total power consumption by approximately 70%.
[7:1]	WO	Reserved	7	Set all bits to 0.

Table 15. Register 0x20 DEVICE_STATUS (1 Byte)

Bit	Type	Name	Width	Description
[7:0]	WO	Reserved	8	Set all bits to 0. Sets up the read-back buffer with contents of the device status. Contents are immediately available for USB read. The contents occupy effectively the lower two bytes. In the case of SPI, contents are transferred to the serial output buffer, so a second query to the SERIAL_OUT_BUFFER register is required to transfer its contents and also to clear the output buffer.
[39:16]	RO	Reserved	24	
[15:8]	RO	List Mode Configuration	8	The current list mode configuration parameters.
[7]	RO	Reserved	1	
[6]	RO	RF Mode	1	0 = Single fixed tone mode, 1 = Sweep/list mode
[5]	RO	Device Standby Mode	1	1 = Device in standby.
[4]	RO	Fine PLL Status	1	1 = PLL is phase locked.
[3]	RO	Coarse PLL Status	1	1 = PLL is phase locked
[2]	RO	Sum PLL Status	1	1 = PLL is phase locked

[1]	RO	Sweep/List Triggered	1	0 = The device, although possibly set to sweep/list mode, is not triggered and is in the stopped state. 1 = The device has been triggered and is in the middle of a triggered cycle.
[0]	RO	Reference Clock	1	0 = 200 MHz selected, 1 = 100 MHz selected.

Table 16. Register 0x21 DEVICE_INFO (1 Byte)

Bit	Type	Name	Width	Description
[1:0]	WO	Device Status	2	Writing this register will place the requested contents into the output buffer. Contents are immediately available for USB read. The contents occupy effectively four bytes. In the case of SPI, contents are transferred to the serial output buffer, so a second query to the SERIAL_OUT_BUFFER register is required to transfer its contents and also to clear the output buffer. 0 = Obtain the product serial number 1 = Obtain the hardware revision 2 = Obtain the firmware revision 3 = Obtain the manufacture date
[7:2]	WO	Reserved	6	
[39:32]	RO	Reserved	8	
[31:0]	RO	Data	32	Data for the requested parameter: Product Serial Number – 32-bit unsigned Hardware Revision – typecast to 32-bit float Firmware Revision – typecast to 32-bit float Manufacture Date – unsigned 32-bit [31:24] Year (last two digits) [23:16] Month [15:8] Day [7:0] Hour

Table 17. Register 0x22 LIST_BUFFER_READ (2 bytes)

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

Table 18. Register 0x24 SERIAL_OUT_BUFFER (5 Bytes)

Bit	Type	Name	Width	Description
[39:0]	WO	Serial Out Buffer	40	Set all bits to 0. Use of this register is only available for the SPI interface.
[39:0]	RO	Request Data	40	The data clocked back are the contents requested by the 0x20, 0x21, or 0x22 registers.

Registers 0x20, 0x21, and 0x22 are query registers. With SPI, the write-only portion of the register must be written first followed by reading register 0x24 to read back the requested data. With USB, the data is available to be read into the register following the write of a query register.

Table 19 Register 0x26 GET_SWEEP_PARAM (1 Byte)

Bit	Type	Name	Width	Description
[2:0]	WO	Get Sweep Parameters	2	Writing this register will place the requested contents into the output buffer. Contents are immediately available for USB read. The contents occupy effectively 5 bytes. In the case of SPI, contents are transferred to the serial output buffer, so a second query to the SERIAL_OUT_BUFFER register is required to transfer its contents and also to clear the output buffer. 0 = Current fix tone frequency 1 = Sweep start frequency 2 = Sweep stop frequency 3 = Sweep step frequency 4 = Sweep dwell time 5 = Sweep cycles
[7:3]	WO	Reserved	6	
[39:0]	RO	Data	40	Data for the requested parameter: Fix tone freq – 40-bit unsigned (Hz) Start freq – 40-bit unsigned (Hz) Stop freq – 40-bit unsigned (Hz) Step freq – 40-bit unsigned (Hz) Dwell time – unsigned 32-bit Cycles – unsigned 32-bit

SERIAL PERIPHERAL INTERFACE (SPI)

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

Once a full transfer has been received, the device will proceed to process the command and de-assert low the SRDY pin. The status of this pin may be monitored by the host because when it is de-asserted low, the device will ignore any incoming data. The device SPI is ready when the previous command is fully processed and SRDY pin is re-asserted high. It is important that the host either monitors the SRDY pin or waits for 500 us between register writes.

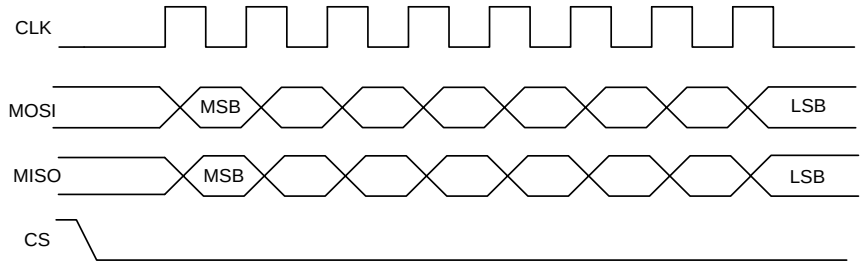


Figure 2. Clock phase.

Register writes are accomplished in a single write operation. Register buffer lengths vary depending on the register; they vary in lengths of 2 to 6 bytes, with the first byte being the register address, followed by the data associated with that register. All data transferred to and from the device are clocked on the falling edge of the clock as shown in *Figure 2*. The ($\overline{CS}$) pin must be asserted low for a minimum period of 1 μs (T_S, see *Figure 3*) before data is clocked in, and must remain low for the entire register write. The clock rate may be as high as 5.0 MHz (T_C = 0.2 μs), however if the external SPI signals do not have sufficient integrity due trace issues then the rate should be lowered.

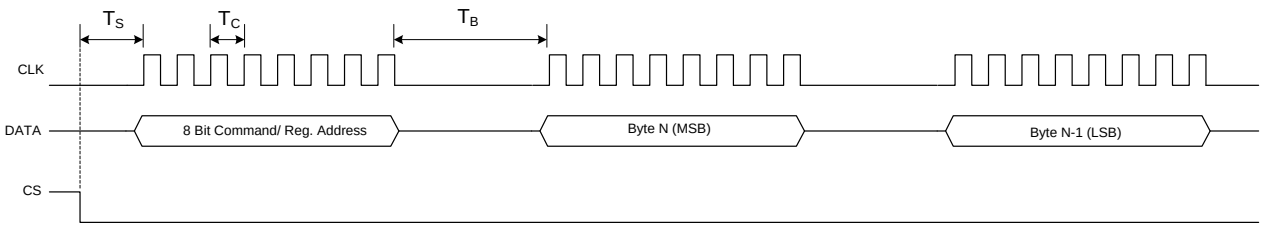


Figure 3 SPI timing.

As mentioned above, the SPI architecture limits the byte rate due to the fact that after every byte transfer the input and output SPI buffers need to be cleared and loaded respectively by the device SPI

engine. Data is transferred between the buffers and the internal registers. The time required to perform this task is indicated by T_B, which is the time interval between the end of one byte transfer and the beginning of another. The recommended minimum time delay for T_B is 5 μs for write only registers, and 7 μs for query registers. The number of bytes transferred depends on the register. It is important that the correct number of bytes is transferred for the associated device register, because once the first byte (MSB) containing the device register is received, the device will wait for the desired number of associated data bytes. The device will hang if an insufficient number of bytes are written to the register. In order to clear the hung condition, the device will need an external hard reset. The time required to process a command is also dependent on the command itself. Measured times for command completions are between 40 μs to 400 μs after reception.

WRITING THE SPI BUS

The SPI transfer size (in bytes) depends on the register being targeted. The MSB byte is the command register address as noted in the *Device Registers* section. The subsequent bytes contain the data associated with the register. As data from the host is being transferred to the device via the SDI (MOSI) line, data present on its SPI output buffer is simultaneously transferred back, MSB first, via the SDO (MISO) line. The data return is invalid for most transfers except for those registers querying for data from the device. See *Reading the SPI Bus* section below for more information on retrieving data from the device. *Figure 4* shows the contents of a single 3 byte SPI command written to the device. The *Device Registers* section provides information on the number of data bytes and their contents for an associated register. There is a minimum of 1 data byte for each register even if the data contents are "zeros".

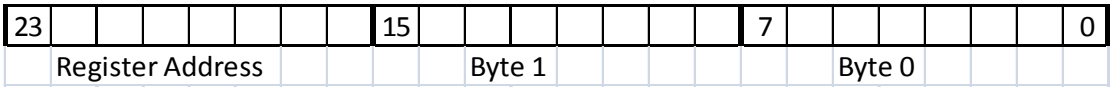


Figure 4. Single 3 byte transfer buffer.

READING THE SPI BUS

Data is simultaneously read back during a SPI transfer cycle. Requested data from a prior command is available on the device SPI output buffers, and these are transferred back to the user host via the SDO pin. To obtain valid requested data would require querying the SERIAL_OUT_BUFFER, which requires 6 bytes of clock cycles; 1 byte for the device register (0x23) and 5 empty bytes (MOSI) to clock out the returned data (MISO). An example of reading the device status from the device is shown in *Figure 5.*

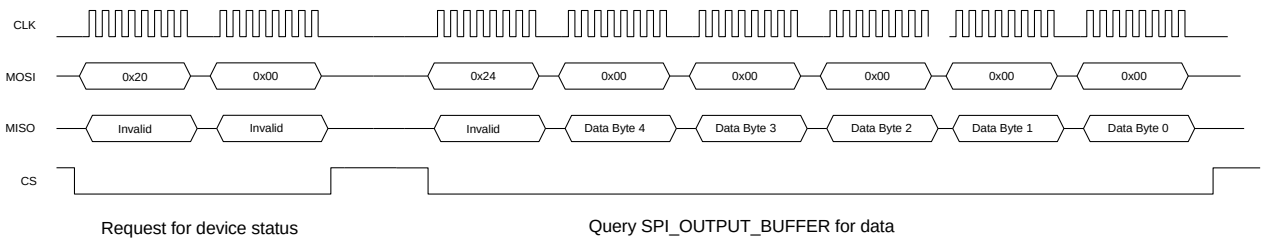


Figure 5. Reading queried data.

USB INTERFACE

The SC800 has a full speed USB interface that works in parallel with the SPI interface. Both interfaces are active at the same time if the USB interface is available on the device and the SMODE pin is pulled high. If the USB interface function is not available or used, it can nevertheless be used for firmware update. It is recommended (as an alternative to JTAG) that pins USB_V, USB+, USB-, and USBISP be routed to a pin header for possible future firmware updates even though the port is unused for device control. The pin USBISP *must be pulled high upon power-up or reset* in order for the device to operate correctly. If the pin is pulled down upon power-up or reset, the device will remain in firmware update mode and will not function.

USB CONFIGURATION

The SC800 USB interface is USB 2.0 compliant running at *Full Speed*, capable of 12 Mbits per second transfer rates. The interface supports three transfer or endpoint types:

- Control Transfer
- Interrupt Transfer
- Bulk Transfer

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

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

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

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

WRITING THE DEVICE REGISTERS DIRECTLY

Device register for the SC800 vary between two bytes and six bytes in length. The most significant byte (MSB) is the command register address that specifies how the device should handle the subsequent configuration data. The configuration data likewise needs to be ordered MSB first, that is,

transmitted first. Input and output buffers of six bytes long are sufficient on the host. To ensure that a register instruction has been fully executed by the device, reading a byte back from the device will confirm that because the device will only return data upon full execution of the instruction, although this is not necessary.

READING THE DEVICE REGISTERS DIRECTLY

Valid data is only available to be read back after writing one of the query registers such as 0x20, 0x21, and 0x22. As soon as one of these registers is written, data is available on the device to be read back. When reading the device, the MSB is returned as the first byte for a total of five bytes. Although not all of the five bytes carry valid data, all five bytes must be read in as valid data beginning at the LSB.

USB DRIVER API

The SC800 USB driver provided by SignalCore is based on libusb-1.0 (www.libusb.org) and its API library is available for the Windows™ and Linux™ operating systems. Source code for both platforms is available upon request by emailing support@signalcore.com. The API functions are nothing more than register wrappers called through the USB bulk transfer function. The C/C++ API library functions are summarized in the table below and each function description is provided in the API description section.

Function	Description
sc800_SearchDevices	Finds all the SC800 Devices connected to the host
sc800_OpenDevice	Opens a USB session for the device
sc800_CloseDevice	Closes a USB session for the device
sc800_SetFrequency	Sets the device frequency for single fixed tone mode
sc800_SetRfMode	Sets the RF output to fixed or sweep/list mode
sc800_ListModeConfig	Configures the sweep/list behavior
sc800_ListSoftTrigger	Software trigger
sc800_ListStartFrequency	Sets the start frequency for sweep
sc800_ListStopFrequency	Sets the stop frequency for sweep
sc800_ListStepFrequency	Sets the step frequency for sweep
sc800_ListDwellTime	Sets the dwell time at each frequency point
sc800_ListCycleCount	Sets the number of cycles to repeat the sweep
sc800_ListBufferPoints	Set the number of list points to step through
sc800_ListBufferWrite	Write the frequency points to the list buffer in RAM
sc800_ListBufferTransfer	Transfers the list points between RAM and EEPROM
sc800_StoreCurrentState	Stores the current configuration as default on reset or power-up
sc800_SetDeviceStandby	Sets the device in standby mode
sc800_GetDeviceStatus	Gets the device status
sc800_GetDeviceInfo	Gets the device information
sc800_ListBufferRead	Reads the list points from list buffer in RAM
sc800_GetSweepParameters	Reads the current sweep/list frequency parameters

API DESCRIPTION

The API functions are contained in the **sc800.dll** for Windows™ operating systems, or libsc800.so.1.0 for Linux™ operating systems. For other operating systems or embedded systems, source code is available for compilation by emailing support@signalcore.com. Information provided below represents the contents of the C/C++ header file, sc800.h, but are expanded here, and listed for convenience.

Function: **sc800_SearchDevices**
Definition: int sc800_SearchDevices(char **serialNumberList)
Output: char **serialNumberList (2-D array pointer list)
Description: sc800_SearchDevices searches for SignalCore SC800 devices connected to the host computer and returns (int) the number of devices found. 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 sc800_OpenDevice function on how to open a device.

Function: **sc800_OpenDevice**
Definition: sc800_deviceHandle_t *sc800_OpenDevice(char *devSerialNum)
Input: char * devSerialNum (serial number string)
Return: sc800_deviceHandle_t* (pointer to the handle)
Description: sc800_OpenDevice opens the device and returns a handle pointer for access.

Function: **sc800_CloseDevice**
Definition: int sc800_CloseDevice(sc800_deviceHandle_t *devHandle)
Input: sc800_deviceHandle_t *devHandle (handle to the device to be closed)
Description: sc800_CloseDevice closes the device associated with the device handle.

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

```
// Declaring
#define MAXDEVICES 50
sc800_deviceHandle_t *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); // 50 serial numbers to search
for (i=0;i<MAXDEVICES; i++) // allocate 8 char for each device
        deviceList[i] = (char*)malloc(sizeof(char)*SCI_SN_LENGTH); // SCI SN has 8 char

numOfDevices = sc800_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 SC800 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++;
            }
// sc800_OpenDevice, open device 0
devHandle = sc800_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
//
status = sc800_CloseDevice(devHandle); // Close the device
```

Function: **sc800_SetFrequency**
Definition: int sc800_SetFrequency(sc800_deviceHandle_t *devHandle,
unsigned long long int frequency)
Input: sc800_deviceHandle_t *devHandle (handle to the opened device)
unsigned long long int frequency (frequency in Hz)
Description: sc800_SetFrequency sets the single fixed tone RF frequency.

Function: **sc800_SetRfMode**
Definition: int sc800_SetRfMode(sc800_deviceHandle_t *devHandle,
unsigned char rfMode)
Input: sc800_deviceHandle_t *devHandle (handle to the opened device)
unsigned char rfMode (See document for bit info)
Description: sc800_SetRfMode sets the mode to single fixed tone generation or sweep/list mode.

Function: **sc800_ListModeConfig**
Definition: int sc800_ListModeConfig(sc800_deviceHandle_t *devHandle,
const listMode_t *modeConfig)
Input: sc800_deviceHandle_t *devHandle (handle to the opened device)
listMode_t *modeConfig (listMode setup)
Description: sc800_ListModeConfig configures the list mode behavior. See the document for more information on the modeConfig structure.

Function: **sc800_ListSoftTrigger**
Definition: int sc800_ListSoftTrigger(sc800_deviceHandle_t *devHandle)
Input: sc800_deviceHandle_t *devHandle (handle to the opened device)
Description: sc800_ListSoftTrigger triggers the device when it is configured for list mode and soft trigger is selected as the trigger source.

Function: **sc800_ListStartFrequency**
Definition: int sc800_ListStartFrequency(sc800_deviceHandle_t *devHandle, unsigned long long int frequency)
Input: sc800_deviceHandle_t *devHandle (handle to the opened device)
unsigned long long int frequency (frequency in Hz)
Description: sc800_ListStartFrequency sets the sweep start frequency.

Function: **sc800_ListStopFrequency**
Definition: int sc800_ListStopFrequency(sc800_deviceHandle_t *devHandle, unsigned long long int frequency)
Input: sc800_deviceHandle_t *devHandle (handle to the opened device)
unsigned long long int frequency (frequency in Hz)
Description: sc800_ListStopFrequency sets the sweep stop frequency.

Function: **sc800_ListStepFrequency**
Definition: int sc800_ListStepFrequency(sc800_deviceHandle_t *devHandle, unsigned long long int frequency)
Input: sc800_deviceHandle_t *devHandle (handle to the opened device)
unsigned long long int frequency (frequency in Hz)
Description: sc800_ListStepFrequency sets the sweep step frequency.

Function: **sc800_ListDwellTime**
Definition: int sc800_ListDwellTime(sc800_deviceHandle_t *devHandle, unsigned int dwellTime)
Input: sc800_deviceHandle_t *devHandle (handle to the opened device)
unsigned int dwellTime (Time in 500 µs increments)
Description: sc800_ListDwellTime stet the sweep/list dwell time at each frequency point. Dwell time is in 500 µs increments (1 = 500 µs, 2 = 1 ms, etc.).

Function: **sc800_ListCycleCount**
Definition: int sc800_ListCycleCount(sc800_deviceHandle_t *devHandle, unsigned int cycleCount)
Input: sc800_deviceHandle_t *devHandle (handle to the opened device)
unsigned int cycleCount (number of cycles)
Description: sc800_ListCycleCount sets the number of sweep cycles to perform before stopping. To repeat the sweep continuously, set the value to 0.

Function: **sc800_ListBufferPoints**
Definition: int sc800_ListPoints(sc800_deviceHandle_t *devHandle, unsigned int listPoints)
Input: sc800_deviceHandle_t *devHandle (handle to the opened device)
unsigned int listPoints (number of points of the list buffer)
Description: sc800_ListPoints sets the number of list points in the list buffer to sweep or step through. The list points must be smaller or equal to the points in the list buffer.

Function: **sc800_ListBufferWrite**
Definition: int sc800_ListBufferWrite(sc800_deviceHandle_t *devHandle,
unsigned long long int frequency)
Input: sc800_deviceHandle_t *devHandle (handle to the opened device)
unsigned long long int frequency (frequency in Hz)
Description: sc800_ListBufferWrite writes the frequency buffer sequentially. If frequency value = 0, the buffer pointer is reset to position 0 and subsequent writes will increment the pointer. Writing 0xFFFFFFFFFF will terminate the sequential write operation and sets the listBufferPoints to the last pointer value.

Function: **sc800_ListBufferTransfer**
Definition: int sc800_ListBufferTransfer(sc800_deviceHandle_t *devHandle,
unsigned char transferMode)
Input: sc800_deviceHandle_t *devHandle (handle to the opened device)
unsigned char transferMode (transfer to EEPROM or RAM)
Description: sc800_ListBufferTransfer transfers the frequency list buffer from RAM to EEPROM or vice versa.

Function: **sc800_StoreCurrentState**
Definition: int sc800_StoreCurrentState(sc800_deviceHandle_t *devHandle)
Input: sc800_deviceHandle_t *devHandle (handle to the opened device)
Description: sc800_StoreCurrentState stores the current configuration into EEPROM memory and is used as the default state upon reset or power-up.

Function: **sc800_SetDeviceStandby**
Definition: int sc800_SetDeviceStandby(sc800_deviceHandle_t *devHandle,
unsigned char standbyEnable)
Input: sc800_deviceHandle_t *devHandle (handle to the opened device)
unsigned char standbyEnable (enable the device to go in standby mode)
Description: sc800_SetDeviceStandby will turn off most analog circuitry, reducing power consumption, stops any current sweeps, and resets the triggers when standbyEnable is set to 1. Setting to 0 will take the device out of standby. Current configurations are not lost when standby is enabled, however a trigger must be applied to restart a sweep.

Function: **sc800_GetDeviceStatus**
Definition: int sc800_GetDeviceStatus(sc800_deviceHandle_t *devHandle, deviceStatus_t *deviceStatus)
Input: sc800_deviceHandle_t *devHandle (handle to the opened device)
deviceStatus_t *deviceStatus (current device status)
Description: sc800_GetDeviceStatus gets the current device status such as the PLL lock status, RF mode, sweep mode, sweep status, as well as the list mode configuration.

Function: **sc800_GetDeviceInfo**
Definition: int sc800_GetDeviceInfo(sc800_deviceHandle_t *devHandle, deviceInfo_t *deviceInfo)
Input: sc800_deviceHandle_t *devHandle (handle to the opened device)
Output: deviceInfo_t *deviceInfo (device information)
Description: sc800_GetDeviceInfo obtains the device information such as serial number, hardware revision, firmware revision, and manufactured date.

Function: **sc800_ListBufferRead**
Definition: int sc800_ListBufferRead(sc800_deviceHandle_t *devHandle, unsigned int address, unsigned long long int *frequency)
Input: sc800_deviceHandle_t *devHandle (handle to the opened device)
unsigned int address (buffer offset address)
Output: unsigned long long int *frequency (frequency)
Description: sc800_ListBufferRead reads the frequency at an offset address of the list buffer.

Function: **sc800_GetSweepParameters**
Definition: int sc800_GetSweepParameters(sc800_deviceHandle_t *devHandle, sweepParams_t *sweepParams)
Input: sc800_deviceHandle_t *devHandle (handle to the opened device)
Output: deviceInfo_t * sweepParams (device information)
Description: sc800_ GetSweepParameters obtains the device sweep parameters such as the fixed tone frequency, sweep start, sweep stop, sweep step frequencies, as well as dwell time and number of the sweep cycles.

EXAMPLE CODE

Code examples in C/C++ demonstrate how the API simplifies programming the device. Code and precompiled 32-bit and 64-bit executables are provided with the software package.

LABVIEW SUPPORT

A LabVIEW USB API is also provided for development on that platform. The API calls the sc800.dll and is simply a wrapper of the C/C++ API. Another LabVIEW API based on NI-VISA is not part of the supplied software package and is available separately by contacting support@signalcore.com for details. The executable SoftFrontPanel.exe was developed in LabVIEW, and its source code is also available from SignalCore.

REVISION NOTES

Rev 1.0 Original document

Rev 1.1
1. Changed spur and phase noise specifications.
2. Added measured data figures.
3. Added theory, device register, and communication buses sections.

Rev 1.2
1. Corrected functions sc800_GetDeviceStatus and sc800_GetDeviceInfo
2. Added function to retrieve current sweep parameters. Nice to read back the startup parameters such as fixed tone frequency.

Rev. 1.2

13401 Pond Springs Road, Suite 100
Austin, TX 78729, USA
Phone: 512 501 6000
Fax: 512 501 6001
Email: info@signalcore.com

SignalCore™
PRESERVING SIGNAL INTEGRITY