

SR2_Analog_810

User's Manual

by



with the collaboration of



November 25 2008

INTRODUCTION	5
1 TECHNICAL DATA	5
1.1 Analog Inputs	5
1.2 Analog Outputs	5
1.3 General-Purpose IOs	6
2 CONNECTOR PINOUTS	6
2.1.1 ADC Connector Pinouts	6
2.1.2 DAC Connector Pinouts	7
2.1.3 Auxiliary +-12 V Power Supplies	8
2.1.4 GPIO Connector Pinout	8
3 INSTALLATION	9
3.1 Manual Installation (LabVIEW users)	9
3.2 Automatic Installation	9
3.3 FPGA Configuration	9
3.4 Mating to <i>Signal_Ranger_mk2</i>	9
4 HARDWARE	10
4.1 Functions and Registers	10
4.1.1 Control Register	10
4.1.2 FIFOs	11
4.1.3 Period Register	11
4.1.4 General-Purpose Inputs and Outputs	11
4.2 Reset and Sequencing	13
4.2.1 Sequencing	13
4.2.2 Start Sequence	13
5 SOFTWARE	14
5.1 <i>SR2_SignalTracker_Analog810</i> Demo Application	14
5.1.1 Time Signal Tab	14
5.1.2 Sxx Tab	15
5.1.3 Acquisition Set-Up Tab	16
5.1.4 GPIOs Tab	17
5.1.5 Board/FPGA Info Tab	18
5.2 AIC810 Driver and Example Code	19
5.2.1 Overview	19

5.2.2	FPGA Logic	19
5.2.3	User-Accessible Structures and Functions	20
5.2.4	Used Resources	21
5.2.5	Restrictions	21
5.2.6	Driver Lock-Up	22
5.3	LabVIEW Library	22

Introduction

SR2_Analog_810 is an analog conversion board for the Signal_Ranger_mk2 DSP board. It is optimized for industrial control and instrumentation applications. It provides two functions:

- 8 analog inputs and 8 analog outputs.
- 22 individually configurable IOs, arranged as one 16-bit port and one 6-bit port.

Its analog I/O channels are capable of operating at up to 150 kHz with a +-10V dynamic range, a very high DC stability, low noise and a very low input-output group-delay. To minimize the group-delay the analog IOs do not have anti-aliasing filters. Therefore this board is not a good choice for vibro-acoustic analysis and general-purpose signal acquisition applications. For these applications we recommend the SR2_Analog_16 or SR2_Analog24Bits boards

Note: Signal_Ranger_mk2's FPGA is used to manage the ADCs and DACs on SR2_Analog_810. To be functional the FPGA must be loaded with a special logic that is different from the factory-default logic implemented in Signal_Ranger_mk2. Any logic implemented in the FPGA of Signal_Ranger_mk2 at power-up that is not the logic designed to manage SR2_Analog_810, including the factory-default logic, has the potential to damage the FPGA and the SR2_Analog_810 board. If SR2_Analog_810 has been purchased separately from Signal_Ranger_mk2 we recommend to either load the SR2_Analog_810 logic, or erase any logic in Flash prior to mating SR2_Analog_810 to Signal_Ranger_mk2. The procedure is described in Signal_Ranger_mk2's user manual.

1 Technical Data

1.1 Analog Inputs

- Number of inputs: 8
- Resolution: 16 bits
- Noise: 1 bit RMS = 150 μ V RMS on +-5V range
1 bit RMS = 300 μ V RMS on +-10V range
- Sampling rate: 11.4 Hz to 150kHz
- Analog input bandwidth: 0 to 10 MHz (includes DC)
- Input type: Single Ended
- Dynamic ranges: +-5V, +-10V
- Input leakage: +-1 μ A max
- Anti-aliasing filter: None
- Group-delay: 2 samples (includes all hardware and software FIFO delay)

1.2 Analog Outputs

- Number of outputs: 8
- Resolution: 16 bits
- Noise: 20MHz bandwidth: up to 55 mV pk-pk on FFFF_H-0000_H alternating code sequence.
20 kHz bandwidth: <25 μ V RMS
- Offset drift with temperature: +-2 ppm FSR / degC
- Gain drift with temperature: +-2 ppm FSR / degC
- Offset drift with time: +-13 ppm FSR / 500 hours
- Sampling rate: 11.4 Hz to 150kHz
- Analog output bandwidth: 0 to >80 kHz (includes DC)
- Output type: Single Ended
- Dynamic range: +-10V
- Source/Sink ability: 4mA

- Anti-aliasing filter: None
- Group-delay: (includes all hardware and software FIFO delay)
 - *Out_0* and *Out_1* 2.5 samples
 - *Out_2* and *Out_3* 2.75 samples
 - *Out_4* and *Out_5* 3 samples
 - *Out_6* and *Out_7* 3.25 samples

1.3 General-Purpose IOs

- Number of IOs: 22 (one 16-bit port and one 6-bit port).
- Configurability: All IOs individually configurable as input or output.
- IO level: 3.3V CMOS (5V-tolerant inputs)

2 Connector Pinouts

2.1.1 ADC Connector Pinouts

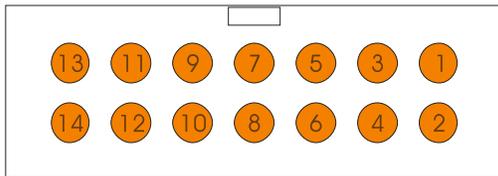


Figure 1 : J1 – J2 connector pinouts

2.1.1.1 J1

No	Function	No	Function
1	-12V	2	+12V
3	Gnd	4	ADC_11 (not managed)
5	Gnd	6	ADC_9 (not managed)
7	Gnd	8	ADC_7
9	Gnd	10	ADC_5
11	Gnd	12	ADC_3
13	Gnd	14	ADC_1

Table 1: Connector J1

2.1.1.2 J2

No	Function	No	Function
1	-12V	2	+12V
3	Gnd	4	ADC_10 (not managed)

5	Gnd	6	ADC_8 (not managed)
7	Gnd	8	ADC_6
9	Gnd	10	ADC_4
11	Gnd	12	ADC_2
13	Gnd	14	ADC_0

Table 2: Connector J2

2.1.2 DAC Connector Pinouts

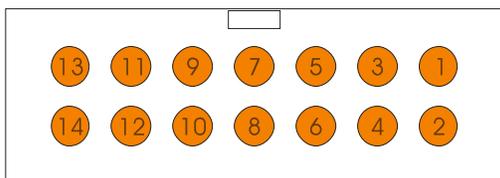


Figure 2: J4 – J5 connector pinouts

2.1.2.1 J4

No	Function	No	Function
1	-12V	2	+12V
3	Gnd	4	Gnd
5	Gnd	6	Gnd
7	Gnd	8	DAC_6
9	Gnd	10	DAC_4
11	Gnd	12	DAC_2
13	Gnd	14	DAC_0

Table 3: Connector J4

2.1.2.2 J5

No	Function	No	Function
1	-12V	2	+12V
3	Gnd	4	Gnd
5	Gnd	6	Gnd
7	Gnd	8	DAC_7

9	Gnd	10	DAC_5
11	Gnd	12	DAC_3
13	Gnd	14	DAC_1

Table 4: Connector J5

2.1.3 Auxiliary +/-12 V Power Supplies

The J1, J2, J3 and J4 connectors provide auxiliary +12 V and -12 V taps that can be used to power external user circuitry. When using these taps the following precautions must be taken:

- The total current drawn from all the +12 V taps must not be greater than 100 mA. The limit is not 100mA per tap, but 100 mA for all taps.
- The total current drawn from all the -12 V taps must not be greater than 100 mA. The limit is not 100mA per tap, but 100 mA for all taps.
- Drawing current from these taps increases the power-supply noise up to 13mV pk-pk for 100 mA.

2.1.4 GPIO Connector Pinout

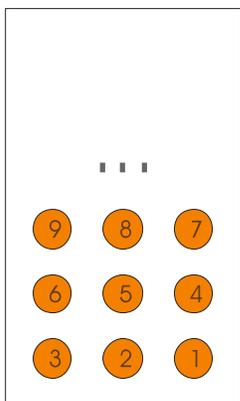


Figure 3: J3 connector pinout

No	Function	No	Function	No	Function
36	GPIO_0(0)	35	Gnd	34	GPIO_0(1)
33	GPIO_0(2)	32	Gnd	31	GPIO_0(3)
30	GPIO_0(4)	29	Gnd	28	GPIO_0(5)
27	GPIO_0(6)	26	Gnd	25	GPIO_0(7)
24	GPIO_0(8)	23	Gnd	22	GPIO_0(9)
21	GPIO_0(10)	20	Gnd	19	GPIO_0(11)
18	GPIO_0(12)	17	Gnd	16	GPIO_0(13)

15	GPIO_0(14)	14	Gnd	13	GPIO_0(15)
12	GPIO_1(0)	11	Gnd	10	GPIO_1(1)
9	GPIO_1(2)	8	Gnd	7	GPIO_1(3)
6	GPIO_1(4)	5	Gnd	4	GPIO_1(5)
3	NC	2	Gnd	1	Presence

Table 5: J3 connector pinout

Note: The Presence pin must be grounded for the GPIO pins to be connected to the FPGA through the bus switches. When experimenting directly on the connector a simple jumper to ground may be used.

3 Installation

Note: LabVIEW developers should not install the executables (run Setup.exe) on a machine that has the LabVIEW development environment installed. The installation of the LabVIEW run-time can sometimes conflict with the development environment and National Instruments discourages this practice. Instead, LabVIEW developers should install the LabVIEW Vis and libraries manually to a directory of their choice and run the applications from the LabVIEW environment. This provides the same functionality, and additionally provides access to the sources of the applications.

3.1 Manual Installation (LabVIEW users)

Unzip the *SR2_Analog_810.zip* file. This installs the following:

3.2 Automatic Installation

Unzip the *SR2_Analog_810.zip* file and run *setup.exe*. This installs the following:

-
-

3.3 FPGA Configuration

Signal_Ranger_mk2's FPGA is used to manage the ADCs and DACs on *SR2_Analog_810*. To be functional the FPGA must be loaded with a special logic that is different from the factory-default logic implemented in *Signal_Ranger_mk2*.

The demo applications that are provided with the board load the FPGA dynamically. They do not need any FPGA configuration to be loaded in Flash.

However any logic present in the FPGA of *Signal_Ranger_mk2*'s Flash will be implemented at power-up. If it is different from the logic designed to manage *SR2_Analog_810* (including the factory-default logic) it has the potential to damage the FPGA and the *SR2_Analog_810* board. If *SR2_Analog_810* has been purchased separately from *Signal_Ranger_mk2* we recommend to either remove any logic from the Flash of *Signal_Ranger_mk2*, or load the *SR2_Analog_810* logic in Flash prior to mating *SR2_Analog_810* to *Signal_Ranger_mk2*. Procedures for doing so are described in *Signal_Ranger_mk2* user's manual. The logic file is called *SR2_Analog_810.rbt*.

3.4 Mating to *Signal_Ranger_mk2*

When *SR2_Analog_810* is purchased with *Signal_Ranger_mk2* the two boards are already mated. If not *SR2_Analog_810* must be mated to connector J4 of *Signal_Ranger_mk2* prior to powering-up.

Before mating the two boards make sure that no FPGA logic other than *SR2_Analog_810* is present in the Flash of *Signal_Ranger_mk2*. Never attempt to mate the two boards while *Signal_Ranger_mk2* is powered.

4 Hardware

4.1 Functions and Registers

All the functions of the FPGA logic are configurable through a set of 7 registers. The registers and their addresses are listed in table 1.

Register	R/W	DSP Byte-Address	DSP Word-Address	Function
Control	R/W	C00002 _H	600001 _H	Main Control Register
FIFO_Data	R/W	C00006 _H	600003 _H	ADC and DAC FIFO access
Period	R/W	C0000A _H	600005 _H	Sampling Period Register
GPIO_0_Data	R/W	C0000E _H	600007 _H	16-bit parallel port 0
GPIO_0_Dir	R/W	C00012 _H	600009 _H	Direction port 0
GPIO_1_Data	R/W	C00016 _H	60000B _H	6-bit parallel port 1
GPIO_1_Dir	R/W	C0001A _H	60000D _H	Direction port 1

Table 2: FPGA registers

4.1.1 Control Register

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name															ADC_Range	Run
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset state	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3: Control register

Bit	Function
Run	Resetting this bit to 0 performs the following: <ul style="list-style-type: none"> • All the FPGA logic is maintained in reset. • All ADC and DAC operations are suspended. • The RSTIN and CLR DAC signals, as well as the RESET ADC signals are activated, maintaining both ADCs and DACs in reset.

	When this bit is set the logic starts operating at the sampling frequency set by <i>Period</i> .
ADC_Range	This bit defines the state of the <i>RANGE</i> signal of the ADC. The range adjustment is as follows: <ul style="list-style-type: none"> • 1 +5V • 0 +10V

Table 4: Control bits

4.1.2 FIFOs

The *FIFO_Data* register accesses both the ADC and DAC FIFOs. To fill the DAC FIFO simply write words in succession to the FIFO's address. To read the ADC FIFO simply read words in succession from the same address. Both FIFOs have a depth of 16 words. This is twice the number of samples required for one sampling period.

After reset the DAC FIFO is empty. This way the first words sent by the DSP as soon as the first INT0 interrupt is triggered are immediately output to the DACs. This minimizes the group delay of the output chain.

After reset the ADC FIFO contains 8 words at 0. These words must be read by the DSP as soon as the INT0 interrupt is triggered. These words represent ADC samples read in the previous sampling period. The first time around there has not been an ADC sampling yet. This is why the ADC FIFO contains 8 samples at zero. At the next INT0, the ADC FIFO contains the 8 words sampled during the first period (see section *Reset and Sequencing*).

4.1.3 Period Register

The *Period* register adjusts the sampling frequency. The sampling frequency is calculated as:

$$F_s = \frac{Clk_In}{N \times 200} \text{ where } N \text{ is the content of } Period.$$

The sampling frequency cannot be higher than 150 kHz, which corresponds to a *Period* value of 5. Any value between 0 and 5 is forced at 5.

The *Period* register should not be changed while the logic is operating. The proper procedure to change the value is:

- Write 0 to the *Run* bit of the *Control* register to stop and reset the FPGA logic.
- Write the new value to the *Period* register.
- Write 1 to the *Run* bit of the *Control* register to start the FPGA logic.

4.1.4 General-Purpose Inputs and Outputs

The board provides 22 general-purpose input-output pins (GPIOs). These pins are accessible on the connector J3. Each pin can be individually configured as input or output via a direction register.

The 22 pins are grouped into one 16-bit port (*GPIO_0*) and one 6-bit port (*GPIO_1*). They are accessed via two 16-bit data registers (*GPIO_Data_0* and *GPIO_Data_1*) and two 16-bit direction registers (*GPIO_Dir_0* and *GPIO_Dir_1*).

4.1.4.1 Bus Switches

The GPIO pins are connected to the corresponding FPGA pins on *Signal_Ranger_mk2* through bus switches.

A *Presence* input is provided on J3 pin 1. This input is used to activate the bus switches that otherwise isolate all the signals of J3 from the *Signal_Ranger_mk2* board.

The bus switches provide two functions:

- They isolate *Signal_Ranger_mk2* from a user board connected on J3 when one board is powered but not the other. This is essential because in the absence of the switches, drivers on one board could be driving un-powered input stages on the other, which could damage them.
- They provide level translation between the user-board, which can drive levels up to 5V and inputs on *Signal_Ranger_mk2*, which are not 5V-tolerant. In short, the switches make the inputs 5V-tolerant.

To provide the first function, the switches should only be activated (placed in low-impedance) when the user daughter board is powered. The *Presence* input serves this purpose. This input is normally pulled-up by a 10 kΩ resistor on *SR2_Analog_810*, which keeps the switches open. The *Presence* pin should be pulled low by an open-drain driver on the user's daughter board, closing the switches, when the daughter board is properly powered. Pulling *Presence* low when the user's daughter board is not properly powered (before it is powered or after it is un-powered) exposes input stages on the user's board to damage inflicted by FPGA pins that may be configured as outputs. Depending on how the FPGA is configured at the time it may not be a problem. For instance if the FPGA is not configured, its IOs are floating and will not drive any current. In this case, *Presence* may be permanently tied to ground. Pulling *Presence* low when the user's daughter board is properly powered but *Signal_Ranger_mk2* is not DOES NOT EXPOSE the FPGA inputs to damage that may be inflicted by drivers on the user's daughter board. This is because the switches are open whenever *Signal_Ranger_mk2* is un-powered.

The level-translation function is provided automatically whenever the switches are activated (whenever *Presence* is low).

4.1.4.2 GPIO Registers

Each GPIO port has two registers, *GPIO_Data_x* and *GPIO_Dir_x*. The direction register (*GPIO_Dir_x*) contains bits that configure each pin as input (0) or output (1).

Reading the data register (*GPIO_Data_x*) returns the state of the corresponding port pins, irrespective of its configuration (input or output).

Writing the data register (*GPIO_Data_x*) sets the state of the port pins that are configured as outputs. It has no impact on the port pins that are configured as inputs.

An open drain function can be emulated by writing 0 to the specified bit position of *GPIO_Data_x* and controlling the specified bit position of *GPIO_Dir_x*.

The register contents are as follows:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset state	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6: GPIO_Data_0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset state	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7: GPIO_Dir_0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R/W	N/A	R/W	R/W	R/W	R/W	R/W	R/W									
Reset state	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8: GPIO_Data_1

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R/W	N/A	R/W	R/W	R/W	R/W	R/W	R/W									
Reset state	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 9: GPIO_Dir_1

Note: N/A always reads 0.

4.2 Reset and Sequencing

Note: The following description assumes the maximum 150 kHz sampling frequency (Period = 5). When the sampling frequency is lower all the timings are lengthened accordingly.

4.2.1 Sequencing

The INT0 interrupts signals the beginning of the sampling period. In response to INT0 the DSP must take the following actions:

1. Write 8 words to the DAC FIFO. To avoid a FIFO under-run the DSP has 3 μ s initially (@ Fs = 150 kHz) to write the first two words, then 1.6 μ s to write each remaining pair of words. This delay leaves enough margin for the FIFO write sequence to be delayed by individual reads and writes via the EMIF, or by a kernel exchange in DARAM of one complete block (256 words).
2. The DSP must then read 8 words from the ADC FIFO. The DSP has up to the next INT0 to read the ADC FIFO to avoid a FIFO overrun. The words read from the ADC FIFO have been sampled in the previous sampling period (between the previous INT0 and the present one).

4.2.2 Start Sequence

The recommended reset and start sequence is as follows:

1. Write to the *Control* register to reset the *Run* bit.
2. Write to the *Period* register to set the desired sampling frequency.
3. If required, prepare the DMAs to read and write the ADC and DAC FIFOs in response to the INT0 interrupt.
4. Write to the *Control* register to set the *Run* bit and start the sequence.

The first INT0 interrupt is triggered 80 ns (@ Fs = 150 kHz) after initiating the last write to the *Control* register, which deactivates the reset. Since the write itself takes more than 80 ns, this means that the first INT0 is triggered immediately after the write.

5 Software

5.1 SR2_SignalTracker_Analog810 Demo Application

The SR2_SignalTracker_Analog810 application has been designed to allow the test and evaluation of the analog input/output channels of the SR2_Analog_810 board. The application allows the user to send test signals to a selected output, and monitor the sampled signal on a selected input. Inputs are displayed both in terms of time signals, as well as instantaneous or averaged energy spectra. Averaged energy spectra are useful to assess the input noise. The front-panel of the application is divided into several tabs, one for each functional group.

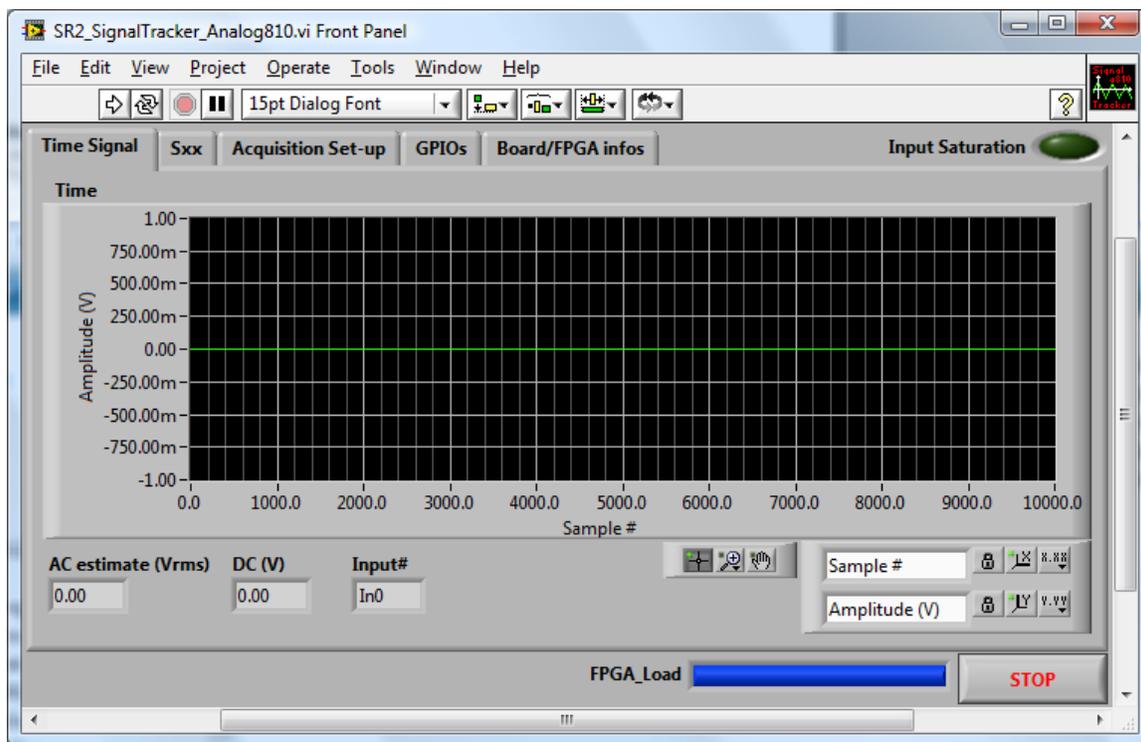


Figure 4: SR2_SignalTracker_Analog810 - Time Signal Tab

To start the application, simply click on the white arrow at the top-left of the window.

The application sends blocks of samples of the specified length and waveform to the selected output, and records blocks of samples of the same length on the selected input. The recorded input samples are synchronous to the output samples, with a fixed and known time relationship between input and output.

5.1.1 Time Signal Tab

Time Indicator

The *Time-Signal* tab presents a time plot of the signal sampled on the selected input. The amplitude scale takes into account the range of the ADC, so that the signal amplitude is represented in Volts at the connector.

AC estimate (Vrms) Indicator

This indicator presents the RMS value of the input signal (any DC offset is removed before the RMS calculation).

DC (V) Indicator

This indicator presents the average DC value of the recorded time signal.

5.1.2 Sxx Tab

Spectrum Indicator

The *Spectrum* indicator presents the instantaneous or averaged power spectrum of the input sampled block.

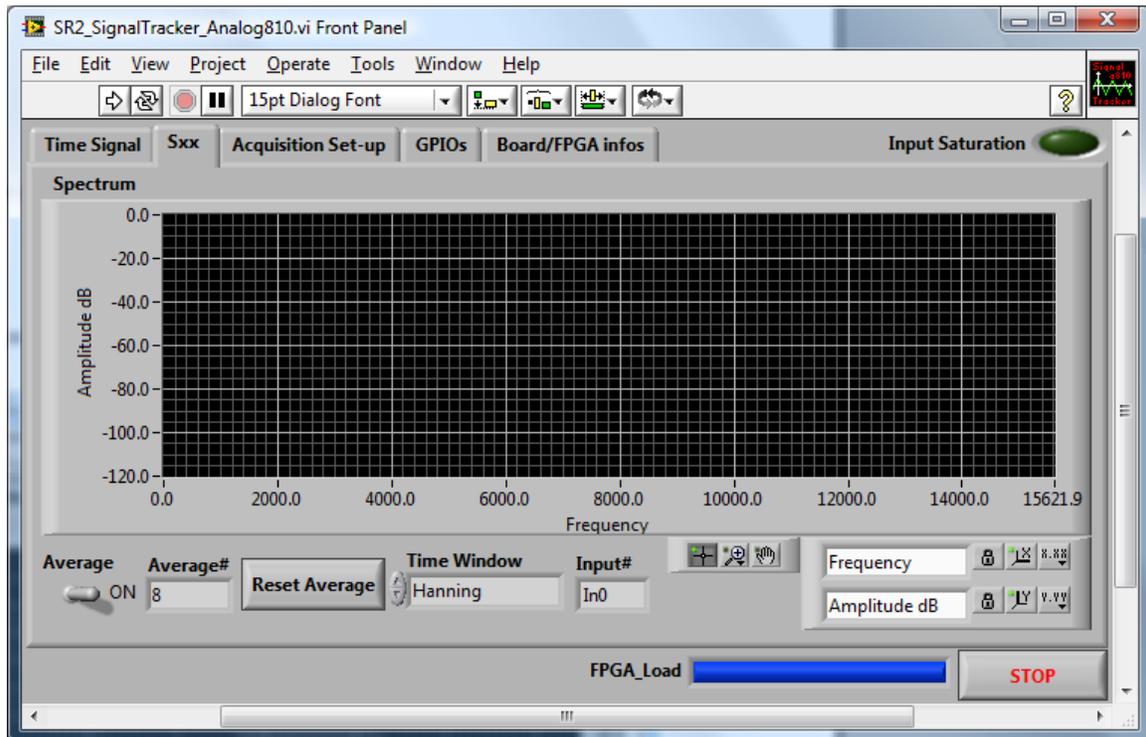


Figure 5: SR2_SignalTracker_Analog810 - Sxx Tab

Average Control

To average the power-spectrum, simply place the *Average* control in the ON position.

Reset Average Button

The *Reset Average* button resets the average.

Time Window selector

An optional weighting window can be chosen from the *Time-Window* list.

Graph and Zoom Controls

Graph controls can be used to change the zoom factor. By default the plot is auto-scaled in X and Y, which is indicated by the closed locks beside each scale name. To disable auto-scale, simply press the lock button.

5.1.3 Acquisition Set-Up Tab

The *Acquisition Set-Up* tab presents the various controls for the acquisition set-up.

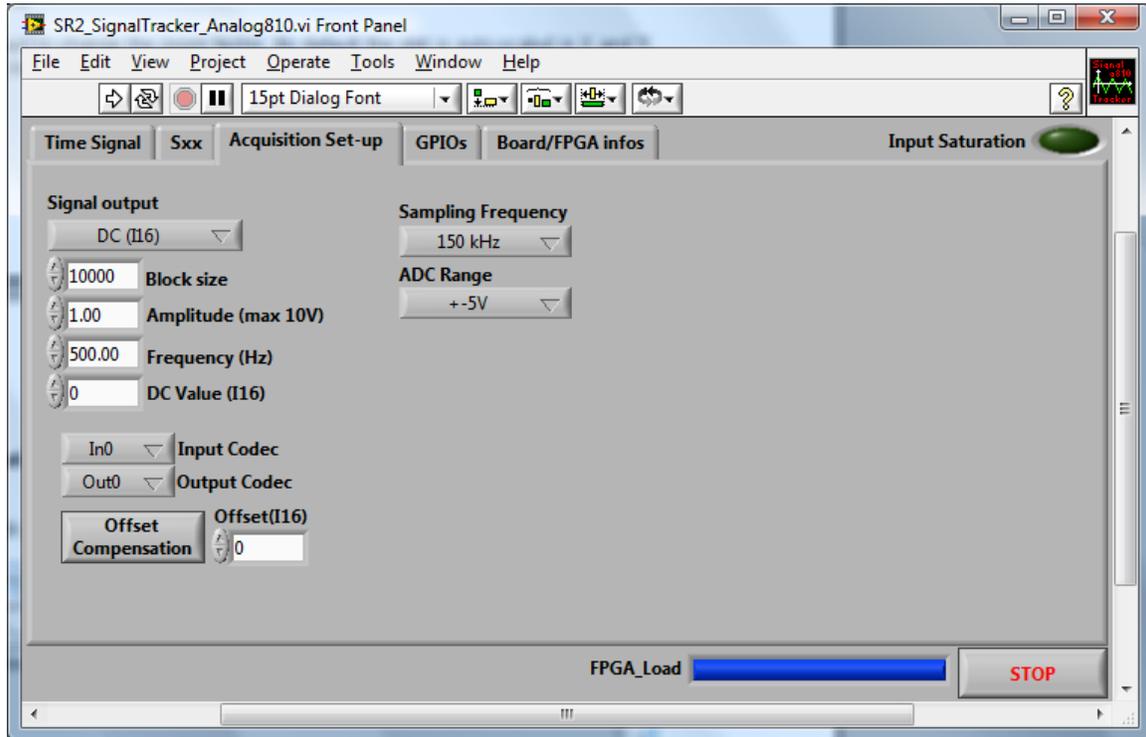


Figure 6: SR2_SignalTracker_Analog810 - Acquisition Tab

Signal Output Control

The *Signal Output* control selects a type of waveform from a list of predefined waveforms. The *No Output* selection sends null samples to the output.

Block Size Control

The *Block Size* control sets the number of samples that are sent to the output, and synchronously recorded from the input.

Amplitude Control

The *Amplitude* control adjusts the amplitude of the output waveform. Note that the output dynamic range is $\pm 10V$.

Frequency Control

The *Frequency* control is only used for periodic waveforms. It adjusts the fundamental frequency of the waveform.

Input Codec Control

The *Input Codec* control selects the input channel between 0 and 7.

Output Codec Control

The *Output Codec* selects the output channel between 0 and 7.

Offset Compensation Control

The *Offset Compensation* button performs an input offset compensation. This procedure reads a block of input samples from the selected input channel while sending zero samples to the selected output channel. The average of the input sample block is then subtracted from any further input samples. Therefore, if any offset is present on the selected input, it is subtracted in all subsequent acquisitions. The average is displayed in the *Offset(I16)* indicator. This indicator is scaled in bits. The input offset compensation is done entirely in software.

Offset(I16) Control

The *Offset(I16)* indicator can also act as a control. Simply changing the content of this field forces a software offset to the recorded input samples.

Sampling Frequency Control

This control allows selecting the sampling frequency of the SR2_Analog_810 board. The list offers a limited number of selections (from 150 kHz to 25 kHz). It means that the N content of the *Period* register is limited between 5 and 30. With a simple software modification of the *SR2_SignalTracker_Analog810*, application the N value can be set higher than 30 (maximum 65536) to select lower sampling frequencies.

ADC Range Control

This is the input range selection. Two selections are possible $\pm 5V$ or $\pm 10V$.

5.1.4 GPIOs Tab

The *GPIOs* tab allows the user to read or to write the digital I/Os. The *Direction* control selects the direction of the I/Os (green for the read direction and red for the write direction). The *State* control acts as an indicator for all I/Os in read direction and as a control for all I/Os in write direction. The levels of the *State* controls are:

-  : I/O at 1 (3.3 V)
-  : I/O at 0 (0 V)

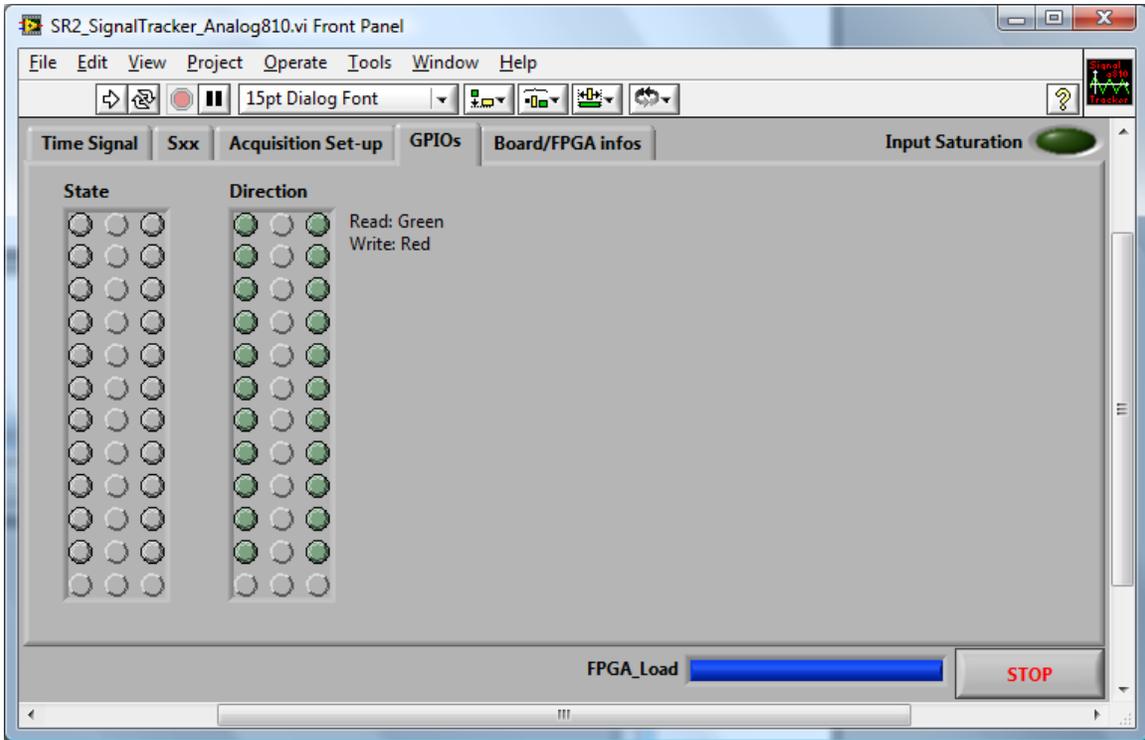


Figure 7: SR2_SignalTracker_Analog810 – GPIOsTab

5.1.5 Board/FPGA Info Tab

This tab presents information about the *Signal_Ranger_mk2* hardware revision, Driver ID number, and FPGA configuration file.

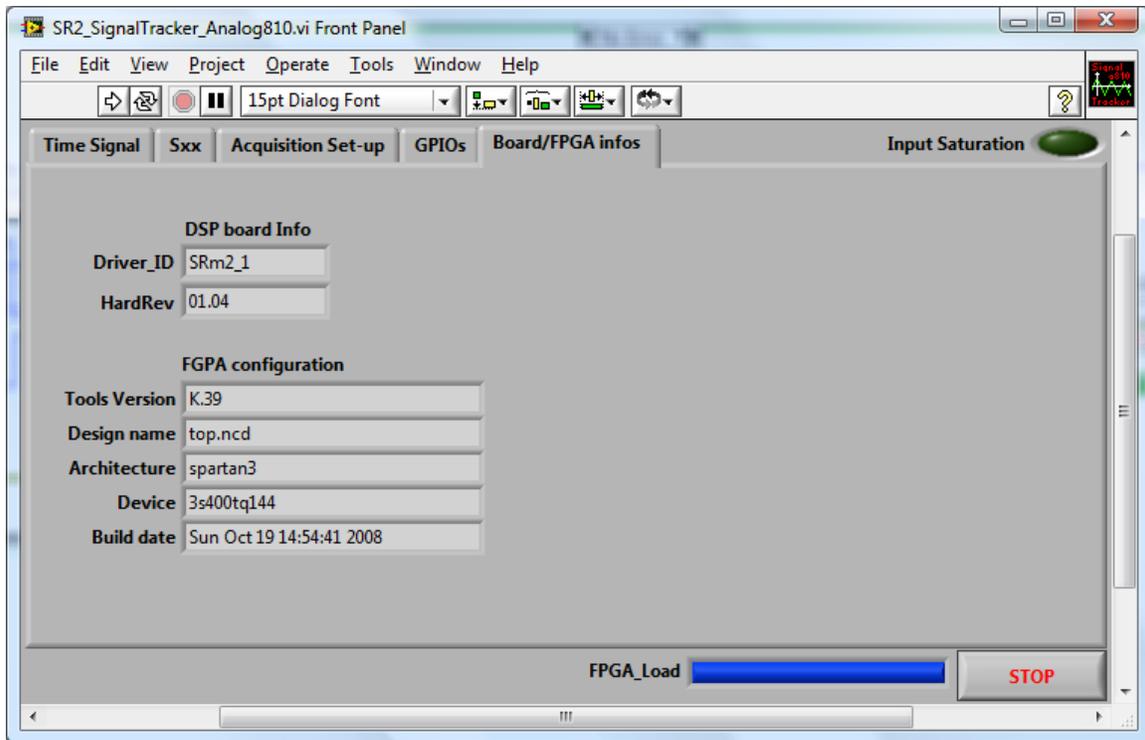


Figure 8: SR2_SignalTracker_Analog810 – Board/FPGA infos Tab

5.2 AIC810 Driver and Example Code

5.2.1 Overview

A driver for the analog I/Os (Analog Interface Circuit) is provided, together with the DSP code of a demo application that uses this driver, as well as an empty “shell” project. Source code for the AIC driver resides in the folder *AIC810Driver*. Source code for the demo application resides in the folder *ST_Analog810_DSP*. This folder contains the DSP code of the *SR2_SignalTracker_Analog810* demo application discussed above. Source code for the shell project resides in the *IO_Shell_Analog810* folder. The shell project constitutes an excellent starting point for developing DSP code that uses the AICs.

The driver has been optimized in assembly language, but can be used in either C, or assembly language. It takes the form of a DSP object libraries *sr2_analog810driver.lib*. The driver contains C-callable functions to configure and use the AICs. A function called *dataprocess* is provided in C, where developers can conveniently place their own analog I/O processing code.

The driver uses the DMA to communicate with the AICs for maximum efficiency.

5.2.2 FPGA Logic

Operation of the driver assumes that the *SR2_Analog_810.rbt* logic is loaded into the FPGA and functional. The entire logic of the module is clocked by a 150 MHz *Clk_In* signal provided by the DSP (*CLKOUT*). *CLKOUT* is provided by the DSP by default at the correct frequency. This signal must not be disabled by DSP user code.

5.2.3 User-Accessible Structures and Functions

The *sr2_analog810driver.lib* library defines and allocates the following user-accessible structures and variables:

FreqDiv

This unsigned short variable (16-bits) determines the sampling frequency selection. This value, between 5 and 65536, is written to the *Period* register of the FPGA logic. The maximum sampling frequency selection is 150 kHz (N=5) and the minimum sampling frequency is 11.44 Hz (N=65535). See the section *Hardware – Period Register* for more details.

ADCRange

This unsigned short variable (16-bits) selects the input range. The value 0 selects the $\pm 10V$ range while the value 1 selects the $\pm 5V$. All others values are not allowed.

iobuf

This structure is designed to contain the input and output samples to/from the AICs.

The user DSP code has one complete sampling period to execute the *dataprocess* function. If the function is not completed within a sampling period input samples are overwritten by the new samples, and the previous output samples are sent to the AICs.

The structure is defined in the *Analog810Driver.h* header file.

The structure is defined as follows:

```
struct iobuf_rec {
int min[8];
int mout[8] ;
};
```

The structure is reserved by the driver library as follows:

```
struct iobuf_rec iobuf;
```

Note: The structure is reserved automatically as a consequence of including the driver library with the user DSP code. There is no need for the user DSP code to reserve this structure.

Note: All symbols must have an “_” prefix when used from assembly code. For instance iobuf becomes iobuf in assembly.

start_Analog810

This function configures the AICs and starts the AIC conversion process. It has no arguments. It uses the configuration values found in the **FreqDiv** and **ADCRange** variables and configures the AICs accordingly through the FPGA configuration registers. These variables must be initialized prior to calling *start_Analog810*. After the execution of this function, the user-defined processing function called *dataprocess* starts being triggered at each sampling period. The LabVIEW AIC interface library includes a VI to generate the configuration variables *FreqDiv* and *ADCRange* automatically as a function of user-selected AIC set-up.

The function is defined in the *Analog810Driver.h* header file.

Note: All symbols must have an “_” prefix when used from assembly code. start_Analog810 must be written _start_Analog810 in assembly.

stop_Analog810

This function stops the AIC conversion process. After the execution of this function, the user-defined *dataprocess* function stops being triggered.

The function is defined in the Analog810Driver.h header file.

Note: All symbols must have an “_” prefix when used from assembly code. stop_Analog810 must be written _stop_Analog810 in assembly.

dataprocess

This function is declared by the AIC driver library, but must be provided by the user. It usually contains the DSP code that reads the input samples from the *iobuf* structure, performs the signal processing between the inputs and outputs and writes the output samples to the *iobuf* structure. Note that the *dataprocess* function is actually a TRAP.

In assembly, this function must protect all registers that it uses, and must be terminated by a RETI instruction. The *dataprocess* symbol must be declared using the `.global` directive.

In C, this function must be defined with the *interrupt* keyword.

Just before the entry into *dataprocess*, the AIC driver enables the global interrupt mask (INTM) and the local masks INT0 and DMAC1. All other interrupts are disabled because of the time-critical nature of this function. The INT0 and DMAC1 interrupts must remain active during the execution of *dataprocess* for the driver to be functional. The local mask registers are saved on the stack and restored after the execution of *dataprocess*. If the user needs to unmask another interrupt within *dataprocess* they must be careful of timing issues. The processing of *dataprocess* must be able to complete within one sampling period. Also all the interrupt masks are restored to their state prior to the execution of *dataprocess* at the function's exit. Therefore any interrupt enabled in *dataprocess* that was not enabled before will be disabled at exit.

Note: All symbols must have an “_” prefix when used from assembly code. dataprocess must be written _dataprocess in assembly.

5.2.4 Used Resources

The AIC driver uses the following hardware resources:

- DMA channels 0 to 1 are used by the driver.
- The DMAC0 and DMAC1 interrupts are used by the driver.
- TRAP #30 is used by the driver to launch the *dataprocess* user function.
- The external INT0 interrupt is used by the FPGA to start the DMA channel 0 and the write operations in the DAC FIFO.
- The driver uses 307 bytes of code and 18 words of data.

5.2.5 Restrictions

When developing C or assembly code using the AIC driver, the following restrictions apply:

- The user-defined I/O processing function *dataprocess* must be present in the user code. If the function is defined in C, it must be defined with the *interrupt* specifier. This way, the compiler does a full context save when entering the function. If it is defined in assembly, all DSP registers used within the function must be protected upon entry, and restored upon return. It must end with the RETI instruction. The *dataprocess* symbol must be defined using the *.global* directive.
- All C-accessible symbols and labels defined in the *sr2_analog810driver.lib* library must have a “_” prefix when used in assembly language.
- The entire code and data sections (including *.bss* section) of the *sr2_analog810driver.lib* library must be linked into the internal DARAM memory.
- The software interrupt #30, the DMAC0, the DMAC1 and the external INTO interrupts are used by the AIC driver. The vectors for these interrupts must be properly declared. See the *ST_Analog810_DSP* or the *IO_Shell_Analog810 DSP* code projects for an example of a correct example of *vectors.asm*.

5.2.6 Driver Lock-Up

Because the driver relies on the DMA for writing and reading in the ADC and DAC FIFOs, there are situations that can lead to a lock-up. The DARAM can only support 2 accesses per DSP cycle, for which the CPU always has priority against DMA and HPI accesses. Some rare sequences of instructions, when executed in a tight loop require those two DARAM accesses per cycle, thereby completely denying access to the DMA. This condition occurs only when the code is executing from the same DARAM block where the DMA buffer resides. Under these conditions the DMA transfers never complete. Therefore the interrupt #30 is never triggered and the *dataprocess* user function is never called. This situation may be accompanied by a USB lock-up if the code executes from the first DARAM block, because the HPI uses the DMA for transfers and is subject to the same priority rules (see *Signal_Ranger_mk2_UserManual.pdf*).

There are several workarounds that can be applied:

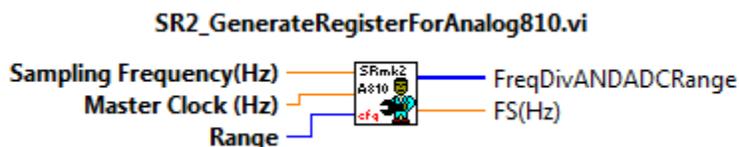
- The DMA section (named *dmabuf* in the AIC driver) may be moved to a DARAM block other from the block containing the offending code loop.
- The offending code loop may be modified to avoid requiring two accesses per cycle. A simple fix is usually to insert NOP instructions in the loop. The situation is very rare so most code reorganizations will usually fix the problem.

5.3 LabVIEW Library

A LabVIEW library is provided to manage the AICs. This library contains only one function, used to determine the contents of the AIC registers.

SR2_GenerateRegistersForAnalog810.vi

This VI determines the values the AIC driver configuration variables. The AIC register contents are determined according to two controls: *Sampling Frequency(Hz)* and *Range*. The *Master Clock* input must be set to 150 MHz at all times.



Controls:

- **Sampling Frequency(Hz):** This control selects the desired sampling frequency in Hz. The VI finds the closest available sampling frequency and returns the sampling frequency found in the *Fs(Hz)* indicator. The maximum sampling frequency is 150 kHz and the minimum is 11.44 Hz.
- **Master Clock (Hz):** This control must be set to 150 MHz at all times.
- **Range:** This control selects the ADC input range. The menu ring control allows the $\pm 5V$ and $\pm 10V$ input range selection.

Indicators:

- **FreqDivANDADCRange:** This vector contains the AIC driver configuration variables *FreqDiv* and *ADCRange*. This vector must be loaded in DSP memory at the symbol address *_FreqDiv* before launching the *_start_Analog810* function of the AIC driver.
- **FS(Hz):** This indicator contains the actual sampling frequency, which is the closest to the requested sampling frequency.