# Application Programming Interface (API) Reference Manual

797 North Grove Rd, Suite 101
Richardson, TX 75081
Phone: 972-671-9570
www.redrapids.com

Red Rapids reserves the right to alter product specifications or discontinue any product without notice. All products are sold subject to the terms and conditions of sale supplied at the time of order. This product is not designed, authorized, or warranted for use in a life-support system or other critical application.

All product names, logos, and brands are property of their respective owners. All company, product and service names used in this document are for identification purposes only. Use of these names, logos, and brands does not imply endorsement.

# Table of Contents

# 1.0 Introduction

Red Rapids offers a common application programming interface (API) that is shared across multiple products. The API consists of C functions that are used to load hardware configuration variables, query status, and manage data transfers over the host bus. These functions essentially form a bridge between the application code and the register settings that orchestrate data flow through the hardware.

Many of the API functions are accompanied by data structures that help group variables by hardware function.

The API is divided into seven separate libraries:

- Global functions to manage channel independent features of the hardware.
- Channel functions to configure and monitor individual datapaths.
- DMA functions to manage the movement of data to/from the host.
- Product functions to initialize each type of device.
- Chip functions to communicate with various semiconductor peripherals.
- Utility functions that provide performance monitoring and debug capabilities.
- DSP functions to configure optional signal conditioning elements in the datapath.

## 1.1 Conventions

This manual uses the following conventions:

- Hexadecimal numbers are prefixed by “0x” (e.g. 0x00058C).
- *Italic* font is used for names of registers.
- Blue font is used for names of directories, files, and OS commands.
- Green font is used to designate source code.

| | |
|---|---|
| ☞ | Text in this format highlights useful or important information. |

| | |
|---|---|
| **!** | Text shown in this format is a warning. It describes a situation that could potentially damage your equipment. Please read each warning carefully. |

The following are some of the acronyms and abbreviations used in this manual.

- **ADC** Analog to Digital Converter
- **AGC** Automatic Gain Control
- **API** Application Program Interface
- **BAR** Base Address Register
- **DAC** Digital to Analog Converter
- **DMA** Direct Memory Access
- **FIFO** First-in / First-out
- **GPIO** General Purpose IO
- **I2C** Inter-Integrated Circuit
- **IF** Intermediate Frequency

- **ISR** Interrupt Service Routine
- **LED** Light Emitting Diode
- **PCI** Peripheral Component Interconnect
- **QDR** Quad Data Rate
- **RSSI** Received Signal Strength Indicator
- **RX** Receiver
- **Rxx** Any Revision Number
- **SPI** Serial Peripheral Interface
- **SRAM** Static Random Access Memory
- **TOD** Time of Day
- **TX** Transmitter
- **USER** User defined IO
- **VITA** VME International Trade Association

## 1.2 Revision History

| Version | Date | Description |
|---|---|---|
| R05 | 04/09/2020 | Added new functions to support channelizer operation. |
| R04 | 03/11/2019 | Correction to DMASave() and DataSave() descriptions. |
| R03 | 03/07/2019 | Major update to the entire API software release. |
| R02 | 12/01/2017 | Add new devices and filter section. |
| R01 | 03/13/2015 | Correction to LoadTODSettings(). |
| R00 | 11/14/2014 | Initial release. |

# 2.0 Global Functions (global_functions.h)

The global functions are used to perform device operations that are independent of any specific data channel. A typical Red Rapids product may consist of several ADC or DAC channels that can be uniquely configured. However, there are also supporting functions such as clock generation or environmental status reporting that are not associated with any specific channel.

## 2.1 ChannelCount()

Report the number of receiver or transmitter channels supported by the device. Each ADC is considered a receiver channel and each DAC is considered a transmitter channel. This function allows a single code base to adapt to the unique configuration of any product variant.

| Name | Type | Description |
|---|---|---|
| ChannelCount() | unsigned | Number of channels available matching the type requested by the RxTxSelect flag. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| RxTxSelect | unsigned | Flag to indicate whether to return the number of receiver channels (0) or transmitter channels (1). |

## 2.2 ChannelizerCount()

Report the number of parallel channelizer datapaths assigned to a specific IF channel. An IF channel is typically associated with an individual ADC, individual DAC, or an ADC/DAC pair. The channelizer feature is not available on every product.

| Name | Type | Description |
|---|---|---|
| ChannelizerCount() | unsigned | Number of channelizer datapaths available. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the IF channel number to target with the function. |

## 2.3 ClockFrequency()

Report the measured frequency of the transmitter clock to calculate DAC register settings.

| Name | Type | Description |
|---|---|---|
| ClockFrequency() | unsigned | Measured clock frequency in Hertz. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |

## 2.4 ClockInit()

Command the device to synchronize all on-board clocks for products that include the Analog Devices AD9512 clock distribution chip. The specific initialization sequence is unique to each product, but the process generally involves resetting any phase locked

loops and performing dynamic phase alignment on high speed chip-to-chip interfaces. This function should be called following the hardware initialization sequence or anytime a clock setting is changed.

| Name | Type | Description |
|---|---|---|
| ClockInit() | int | Zero indicates successful completion.<br>Non-zero indicates an error condition. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |

## 2.5 ClockInitLMK01000()

Command the device to synchronize all on-board clocks for products that include the Texas Instruments LMK01000 clock distribution chip. The specific initialization sequence is unique to each product, but the process generally involves resetting any phase locked loops and performing dynamic phase alignment on high speed chip-to-chip interfaces. This function should be called following the hardware initialization sequence or anytime a clock setting is changed.

| Name | Type | Description |
|---|---|---|
| ClockInitLMK01000() | int | Zero indicates successful completion.<br>Non-zero indicates an error condition. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| CLKout0 | unsigned | Output control register setting for CLKout0 pin. |
| CLKout4 | unsigned | Output control register setting for CLKout4 pin. |
| CLKout5 | unsigned | Output control register setting for CLKout5 pin. |

## 2.6 ClockStatus()

Report the status of all clocks operating on the device. This function can be called anytime the clock status is needed by the software application. It is particularly useful to verify that an externally supplied sample clock or reference clock has been detected by the device.

| Name | Type | Description |
|---|---|---|
| ClockStatus() | int | Numeric status code indicating clock state:<br>0 External sample clock active, digital clocks are locked.<br>1 External sample clock active, digital clocks are not locked.<br>2 Sample clock locked to internal reference, digital clocks are locked.<br>3 Sample clock locked to internal reference, digital clocks are not locked.<br>4 Sample clock locked to external reference, digital clocks are locked.<br>5 Sample clock locked to external reference, digital clocks are not locked.<br>>5 Sample clock is not locked. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |

## 2.7 ErrorMask()

Set the error interrupt mask bits to enable (1) or disable (0) interrupt generation for each type of error monitored by the device. The device will continue to record faults even if the interrupt is masked, but the software application will have to query for status since there will be no independent notification.

The BAR mask bits are assigned to the following four types of errors:

Bar[0]: Write to an illegal BAR0 address detected.

Bar[1]: Read from an illegal BAR0 address detected.

Bar[2]: Write to an illegal BAR2 address detected.

Bar[3]: Read from an illegal BAR2 address detected.

The type of errors assigned to the clock mask bits are unique to a specific product. Most products do not use all eight available bits.

Clock[0]: Digital calibration clock out of lock.

Clock[1]: Flash programmer clock out of lock.

Clock[2]: QDR-A SRAM interface calibration failed.

Clock[3]: QDR-B SRAM interface calibration failed.

Clock[4+]: ADC interface calibration failed.

| Name | Type | Description |
|---|---|---|
| ErrorMask() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ErrorIntMask | s_ErrorStatus | Variable assigned to the s_ErrorStatus structure consisting of the following members:<br>unsigned Bar[4]<br>Interrupt mask assigned to the four BAR error status flags. Zero disables interrupt generation when the corresponding error is detected, and one enables interrupts.<br>unsigned Clock[8]<br>Interrupt mask assigned to the eight available clock error status flags. Zero disables interrupt generation when the corresponding error is detected, and one enables interrupts.<br>unsigned Alert<br>Interrupt mask assigned to the optional alert input from the port expander (FXL6408) or temperature monitor (LTC2991) chips. Zero disables interrupt generation when an alert is detected, and one enables interrupts.<br>unsigned Fan<br>Interrupt mask assigned to the fan monitor on products equipped with an active heat sink. Zero disables interrupt generation when a fan failure is detected, and one enables interrupts. |

## 2.8 ErrorStatus()

Report the state of individual fault status flags. This function returns the total number of flags that are currently active. The contents of the s_FaultStatus structure are modified by the function to provide details of which specific flags are active. This function will automatically clear the error status register.

The BAR mask bits are assigned to the following four types of errors:

Bar[0]: Write to an illegal BAR0 address detected.

Bar[1]: Read from an illegal BAR0 address detected.

Bar[2]: Write to an illegal BAR2 address detected.

Bar[3]: Read from an illegal BAR2 address detected.

The type of errors assigned to the clock mask bits are unique to a specific product. Most products do not use all eight available bits.

Clock[0]: Digital calibration clock out of lock.

Clock[1]: Flash programmer clock out of lock.

Clock[2]: QDR-A SRAM interface calibration failed.

Clock[3]: QDR-B SRAM interface calibration failed.

Clock[4+]: ADC interface calibration failed.

| Name | Type | Description |
|---|---|---|
| ErrorStatus() | unsigned long | Total number of active error status flags recorded. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| *p_Error | s_ErrorStatus | Pointer to the s_ErrorStatus structure consisting of the following members:<br>unsigned Bar[4]<br>Conveys the current state of four BAR error status flags. Zero indicates that no error has been detected.<br>unsigned Clock[8]<br>Conveys the current state of clock error status flags. Zero indicates that no error has been detected.<br>unsigned Alert<br>Conveys event status reported by the port expander (FXL6408) or temperature monitor (LTC2991) chips.<br>unsigned Fan<br>Reports fan failures on products equipped with an active heat sink. |

## 2.9 FirmwareRevision()

Report the firmware revision date (MMDDYYYY) of the device.

| Name | Type | Description |
|---|---|---|
| FirmwareRevision() | unsigned | Firmware revision date in hex format, but it is not a hex value (e.g. 04/27/2019 returns 0x04272019). |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |

## 2.10 GlobalStatus()

Report the state of the global channel and error status flags. There are up to eight channel status flags and one error flag that can be monitored by software. This function returns the total number of flags that are currently active. The contents of the s_GlobalStatus structure are modified by the function to provide details of which specific flags are active.

Each channel is assigned a global status flag to inform the software application that some type of service is required. Refer to the ChannelStatus() function for a detailed description about the type of information reported.

A single global error flag is assigned to the device. Refer to the ErrorStatus() function for a detailed description about the type of errors reported.

| Name | Type | Description |
|---|---|---|
| GlobalStatus() | int | Total number of status flags that require service. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| *p_Global | s_GlobalStatus | Pointer to the s_GlobalStatus structure consisting of the following members:<br>unsigned Channel[8]<br>Conveys the current state of each channel status flag (8 max). Zero indicates that the channel does not currently require service.<br>unsigned Error<br>Conveys the current state of the device error status flag. Zero indicates that no errors have been detected. |

## 2.11 GlobalSync()

Issue a global synchronization strobe that is synchronous to all channels. This action is equivalent to the synchronization strobe that can be applied just prior to starting a channel but is independent of any scheduled channel activity. This is a convenient mechanism to synchronize the tuner internal to each digital down converter (DDC) and allow the filters to flush prior to scheduling channel activity.

| Name | Type | Description |
|---|---|---|
| GlobalSync() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |

## 2.12 GPIOConnect()

Electrically connect the general purpose I/O (GPIO) port by enabling all pins.

| Name | Type | Description |
|---|---|---|
| GPIOConnect() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |

## 2.13 GPIODisconnect()

Electrically isolate the general purpose I/O (GPIO) port by tristating all pins.

| Name | Type | Description |
|---|---|---|
| GPIODisconnect() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |

## 2.14 GPIOReadWrite()

The general purpose I/O (GPIO) port provides a means to monitor or control external equipment using the API. Each GPIO pin can be software configured as an input or output, allowing this function to either read or write an external logic value.

| Name | Type | Description |
|---|---|---|
| GPIOReadWrite() | unsigned | Value read from the GPIO inputs as a six-bit field (5:0) corresponding to GPIO #6 to GPIO #1. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| GPIO | unsigned | Value written to the GPIO outputs as a six-bit field (5:0) corresponding to GPIO #6 to GPIO #1. |

## 2.15 GPIOSettings()

Configure each general purpose I/O (GPIO) pin as an input or output. This function is also used to electrically isolate the GPIO port.

| Name | Type | Description |
|---|---|---|
| GPIOSettings() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| GPIO | s_GPIOSettings | Variable assigned to the s_GPIOSettings structure consisting of the following members:<br>unsigned Disconnect<br>One electrically disconnects the entire GPIO port by tri-stating all pins and zero connects the port.<br>unsigned Direction[8]<br>Each member of the array corresponds to a GPIO pin starting at index zero for GPIO #1. Each pin can be configured as an input (0), an output (1), or an input with a digital debounce function (2). |

## 2.16 InterruptMask()

Set the global interrupt mask to enable (1) or disable (0) hardware interrupts from the device. Mask bits are also available to disable interrupts originating from specific errors using the ErrorMask() function and specific channel events using the ChannelMask() function.

| Name | Type | Description |
|---|---|---|
| InterruptMask() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| GlobalIntMask | unsigned | Zero disables hardware interrupts from the device and one enables hardware interrupts. |

## 2.17 LEDSettings()

Select the type of status to be reported by illuminating the yellow or green light emitting diode (LED).

The yellow LED conveys four types of status:

Yellow[0]: Flash when a software trigger is detected. (0 => disable, 1 => enable)

Yellow[1]: Flash when a time of day (TOD) start/stop trigger is detected.
(0 => disable, 1 => start trigger, 2 => stop trigger, 3 => start or stop trigger)

Yellow[2]: Flash when a transition on the coax trigger is detected.
(0 => disable, 1 => rising edge, 2 => falling edge, 3 => rising or falling edge)

Yellow[3]: Flash when a transition on the GPIO trigger is detected.
(0 => disable, 1 => rising edge, 2 => falling edge, 3 => rising or falling edge)

The green LED can be illuminated when an individual channel is active.

| Name | Type | Description |
|---|---|---|
| LEDSettings() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| LED | s_LEDSettings | Variable assigned to the s_LEDSettings structure consisting of the following members:<br>unsigned Yellow[8]<br>Zero masks the designated activity from the indicator and a non-zero value causes the LED to flash if a specific activity is detected.<br>unsigned Green[8]<br>Each member of the array corresponds to an individual channel starting at index zero for Channel #1. Zero masks channel activity from the indicator and one causes the LED to illuminate when the corresponding channel is active. |

## 2.18 ModelNumber()

Report the three-digit model number of the device.

| Name | Type | Description |
|---|---|---|
| ModelNumber() | unsigned | Device model number in hex format, but it is not a hex value (e.g. Model 266 returns 0x266). |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |

## 2.19 PinDirection()

Set the direction of signals on external connector pins. The general purpose I/O (GPIO) and USER ports allow the device to communicate with external hardware. Each pin can be software configured as an input or output through this function.

| Name | Type | Description |
|---|---|---|
| PinDirection() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| PinDirection | s_Pin | Variable assigned to the s_PinDirection structure consisting of the following members:<br>unsigned GPIODirection[6]<br>Sets the direction of pins on the GPIO connector to input (0) or output (1).<br>unsigned USERDirection[62]<br>Sets the direction of pins on the USER connector to input (0) or output (1). |

## 2.20 PowerStatus()

Reports the voltage, current, and power dissipation of up to three primary sources supplying the device. The voltage and current measurements are obtained directly from a system monitor chip, the power dissipation is calculated.

The number of supplies reporting, and the voltage associated with each supply will be unique to a specific product. Unused supplies may report a voltage, but zero current.

| Name | Type | Description |
|---|---|---|
| PowerStatus() | double | Total power dissipation across all supplies. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| *p_Power | s_Power | Pointer to the s_Power structure consisting of the following members:<br>double Supply1[3]<br>Conveys the bus voltage (0), current (1), and power dissipation (2) of supply #1.<br>double Supply2[3]<br>Conveys the bus voltage (0), current (1), and power dissipation (2) of supply #2.<br>double Supply3[3]<br>Conveys the bus voltage (0), current (1), and power dissipation (2) of supply #3. |

## 2.21 SerDesInit()

Command the device to synchronize all deserializers for products that include chips with a high-speed serial data interface. This function should be called following the hardware initialization sequence or anytime a clock setting is changed.

| Name | Type | Description |
|---|---|---|
| SerDesInit() | int | Zero indicates successful completion.<br>Non-zero indicates an error condition. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |

## 2.22 Serial()

Program a chip on the device through a serial interface port (SPI, I2C, etc.). Many chips, such as an ADC or DAC, include a serial port to program internal configuration and status registers. This function allows the software application to access the SPI port of any chip available on the device.

A typical software application will not call this function directly, it is primarily used by other functions.

| Name | Type | Description |
|---|---|---|
| Serial() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| BusSelect | unsigned | Code to select a specific type of serial bus. |
| MulitFunction | unsigned | Code that selects either the active edge of serial clock or the I2C transaction type depending on context. |
| Instruction | unsigned | Instruction in the format supported by a particular serial bus, usually including an address. |
| InstructionSize | unsigned | Size in bits of the instruction segment of a payload. |
| Data | unsigned | Variable containing value to be written for write transactions, not used for read transactions. |
| DataSize | unsigned | Size in bits of the data segment of a payload. |
| *p_ReadValue | unsigned | Pointer to a variable that will contain the value retrieved from a read transaction. |

## 2.23 SoftwareReset()

Issue a software reset that is equivalent to a power-on reset.

| Name | Type | Description |
|---|---|---|
| SoftwareReset() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |

## 2.24 SoftwareTrigger()

Issue a software trigger that is synchronous to all channels. The arrival time of the software trigger internal to the device is non-deterministic, but this function does ensure that all active channels are triggered simultaneously.

| Name | Type | Description |
|---|---|---|
| SoftwareTrigger() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |

## 2.25 TemperatureStatus()

Report the temperature measured by five sensors on the device. All five sensors are connected to a single monitor chip that also reports the internal supply voltage.

The location of sensors on the device will be unique to a specific product.

| Name | Type | Description |
|---|---|---|
| TemperatureStatus() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| *p_Temperature | s_Temperature | Pointer to the s_Temperature structure consisting of the following members:<br>double Sensor[5]<br>Conveys the temperature recorded by five sensors.<br>double Vcc<br>Conveys the voltage measurement of the power supply to the temperature sensor. |

## 2.26 TODSeconds()

Read the current time of day (TOD) seconds value from the device. The fractional seconds value is not available.

| Name | Type | Description |
|---|---|---|
| TODSeconds() | unsigned | The current value of the time of day (TOD) seconds counter. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |

## 2.27 TODSettings()

Set the time of day (TOD) clock internal to the device. The TOD clock consists of two counters; seconds and fractional seconds.

The fractional seconds counter increments with each rising edge of the ADC/DAC sample clock, which establishes the resolution of the TOD.

The seconds counter increments on the rising edge of an externally supplied 1 PPS trigger, or an internal seconds timer based on the fractional seconds count. The TOD clock will assume that an external 1 PPS trigger is supplied if the ClockFrequency value is set to zero. If there is no external 1 PPS trigger, the ClockFrequency value should be set to the frequency of the clock incrementing the fractional seconds counter, which is usually the ADC/DAC sample clock.

| Name | Type | Description |
|---|---|---|
| TODSettings() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| TOD | s_TODSettings | Variable assigned to the s_TODSettings structure consisting of the following members:<br>unsigned long Seconds<br>Value used to set the device clock current time of day (TOD) in seconds (max = (2^32)-1).<br>unsigned long ClockFrequency<br>Clock frequency used to increment the time of day fractional seconds, usually the ADC/DAC sample clock frequency, if there is no 1 PPS external source (max = (2^32)-1).<br>unsigned long TSFCode<br>TSF code defined by the VITA 49 specification.<br>unsigned long TSICode<br>TSI code defined by the VITA 49 specification. |

## 2.28 TriggerSettings()

Enable the optional trigger lock function on external trigger inputs. Trigger lock is a feature that centers the rising or falling edge of a trigger within a sample clock period to eliminate potential ambiguity that can occur if an active edge coincides with the clock transition.

The trigger lock function is available on two triggers:

Source[0]: Coax connector trigger input.

Source[1]: GPIO connector trigger input.

| Name | Type | Description |
|---|---|---|
| TriggerSettings() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| Trigger | s_TriggerSettings | Variable assigned to the s_TriggerSettings structure consisting of the following members:<br>unsigned Source[4]<br>Zero disables the clock lock function on the corresponding trigger. A value of one enables a lock to the rising edge of trigger and two locks to the falling edge. |

## 2.29 USERConnect()

Electrically connect the USER port by enabling all pins.

| Name | Type | Description |
|---|---|---|
| USERConnect() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |

## 2.30 USERDisconnect()

Electrically isolate the USER port by tristating all pins.

| Name | Type | Description |
|---|---|---|
| USERDisconnect() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |

# 3.0 Channel Functions (channel_functions.h)

The channel functions are used to perform device operations that are targeted to individual channels. A typical Red Rapids product may consist of several ADC or DAC channels that can be uniquely configured. The exact number of channels depends on the configuration of a specific product.

## 3.1 ChannelMask()

Set the channel interrupt mask bits to enable (1) or disable (0) interrupt generation for each type of channel event monitored by the device. The device will continue to record events even if the interrupt is masked, but the software application will have to query for status since there will be no independent notification.

The DMAMarker status flag is used to inform the application software that the DMA buffer assigned to the designated channel requires service. In the case of a receiver channel, this indicates that new data is available in the buffer. In the case of a transmitter channel, this indicates that new data is needed by the buffer.

The error mask bits are assigned to the following types of errors:

Error[0] Converter (ADC/DAC) error detected..

Error[1] FIFO overflow/underflow error detected.

| Name | Type | Description |
|---|---|---|
| ChannelMask() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |
| ChannelIntMask | s_ChannelIntMask | Variable assigned to the s_ChannelIntMask structure consisting of the following members:<br>unsigned DMAMarker<br>Interrupt mask assigned to the channel DMA marker status flags. Zero disables interrupt generation when the DMA buffer requires service and one enables interrupts.<br>unsigned Error[2]<br>Interrupt mask assigned to the two channel error status flags. Zero disables interrupt generation when the corresponding error is detected and one enables interrupts. |

## 3.2 ChannelReset()

Issue a reset to restore the channel configuration registers to their power-on state.

| Name | Type | Description |
|---|---|---|
| ChannelReset() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |

## 3.3 ChannelSettings()

Load all configuration settings to a specific channel. This function calls a sequence of finer grain configuration functions that address individual channel features.

| Name | Type | Description |
|---|---|---|
| ChannelSettings() | int | Zero indicates successful completion.<br>Non-zero indicates an error condition. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |
| Channel | s_Channel | The s_Channel structure carries all configuration variables assigned to a channel. |

## 3.4 ChannelStatus()

Report the state of individual channel status flags. This function returns the total number of flags that are currently active. The contents of the s_ChannelStatus structure are modified by the function to provide details of which specific flags are active. This function will automatically clear the channel status register.

The DMAMarker status flag is used to inform the application software that the DMA buffer assigned to the designated channel requires service. In the case of a receiver channel, this indicates that new data is available in the buffer. In the case of a transmitter channel, this indicates that new data is needed by the buffer.

The error mask bits are assigned to the following types of errors:

Error[0] Converter (ADC/DAC) error detected.

Error[1] FIFO overflow/underflow error detected.

| Name | Type | Description |
|---|---|---|
| ChannelStatus() | int | Total number of active error status flags recorded. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |
| *p_Channel | s_ChannelStatus | Pointer to the s_ChannelStatus structure consisting of the following members:<br>unsigned DMAMarker<br>Zero indicates the DMA buffer does not require service and one indicates service is needed.<br>unsigned Error[2]<br>Zero indicates no error has been detected and one conveys a past error condition. |

## 3.5 ChannelStatusClear()

Clear the channel status register to erase previously recorded error conditions.

| Name | Type | Description |
|---|---|---|
| ChannelStatusClear() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |

## 3.6 DatapathControlSettings()

Load the datapath configuration settings to a specific channel. The datapath controls select which processing and synchronization features of the datapath will be enabled. It also manages the disposition of data residue that may remain in the datapath when it is not active.

| Name | Type | Description |
|---|---|---|
| DatapathControlSettings() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |
| DatapathControl | s_DatapathControl | The s_DatapathControl structure carries the datapath control configuration variables assigned to a channel. |

## 3.7 EventDurationSettings()

Load the event duration configuration settings to a specific channel. Dataflow through a channel is controlled by selected events. Some of these events are timed by counters that track the number of samples or periodic cycles to process.

| Name | Type | Description |
|---|---|---|
| EventDurationSettings() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |
| EventDuration | s_EventDuration | The s_EventDuration structure carries the event duration configuration variables assigned to a channel. |

## 3.8 FlowControl()

Start (OnOff = 1) or stop (OnOff = 0) processing on the designated channel.

This function is required to initiate any channel processing, even if triggers are used to begin signal acquisition or generation. The start command arms a channel to react when the requested start event is detected (software trigger, hardware trigger, etc.). Processing begins immediately if there is no start event specified in the channel configuration.

The stop command is not necessary to terminate channel processing since other stop events can be selected (software trigger, hardware trigger, etc.). However, the stop command can be used to immediately shut off a channel even if it is waiting for a start or stop event.

| Name | Type | Description |
|---|---|---|
| FlowControl() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |
| OnOff | unsigned | Zero stops channel processing and one enables channel processing. |

## 3.9 FlowStatus()

Report whether the channel is currently active (1) or inactive (0). This function can be used to monitor the state of a channel that has been programmed to start or stop in response to a trigger.

| Name | Type | Description |
|---|---|---|
| FlowStatus() | int | Zero indicates the channel is off and one indicates the channel is on. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |

### 3.10 FrameFilename()

Create a name for the frame descriptor file used to demonstrate the scheduler. The character string includes the full directory path to the descriptor file as follows:

Windows: ..\files\frame.txt

Linux: ../files/frame.txt

| Name | Type | Description |
|---|---|---|
| FrameFilename() | void | Nothing returned. |
| *p_file | char | Full path name to the frame descriptor text file. |

### 3.11 IFdpktFormatSettings()

Load the IF data packet configuration settings to a specific channel. The format of the IF data packet is defined by the VITA Radio Transport Standard (VITA 49.0).

| Name | Type | Description |
|---|---|---|
| IFdpktFormatSettings() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |
| IFdpktFormat | s_IFdpktFormat | The s_IFdpktFormat structure carries the IF data packet configuration variables assigned to a channel. |

### 3.12 PayloadFormatSettings()

Load the IF data payload configuration settings to a specific channel. The format of the IF data payload is defined by the VITA Radio Transport Standard (VITA 49.0).

| Name | Type | Description |
|---|---|---|
| PayloadFormatSettings() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |
| PayloadFormat | s_PayloadFormat | The s_PayloadFormat structure carries the payload format configuration variables assigned to a channel. |

### 3.13 ReadyStatus()

Report whether a specific channel is waiting for a start event. The FlowControl() function is used to arm a channel, but it takes time to prepare the channel to start processing data. This function can be used after the channel is armed to determine when it is ready to

receive the start event. If the start event occurs before the channel is ready, it will be missed.

| Name | Type | Description |
|---|---|---|
| ReadyStatus() | int | One indicates that the channel is waiting for a start event and zero indicates that the channel is not currently waiting for a start event. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |

## 3.14 SchedulerFrameSettings()

Load the frame descriptor settings to a specific channel when the scheduler is selected as the cycle event. The descriptors are stored in a text file organized as multiple pairs of slot start time followed by slot duration in sample count.

| Name | Type | Description |
|---|---|---|
| SchedulerFrameSettings() | int | Non-zero indicates an error encountered parsing the frame descriptor file. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |
| *p_file | char | Full path name to the frame descriptor text file. |

## 3.15 SwitchSelectSettings()

Load the switch select configuration settings to a specific channel. Most channels can obtain input data from multiple sources. The switch select setting determines which specific source is connected to the datapath.

| Name | Type | Description |
|---|---|---|
| SwitchSelectSettings() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |
| SwitchSelect | s_SwitchSelect | The s_SwitchSelect structure carries the switch select configuration variables assigned to a channel. |

## 3.16 SynchronizerSettings()

Load the synchronizer configuration settings to a specific channel. The synchronizer manages all dataflow through the channel based on the settings loaded by this function.

The number of unique settings is determined by the features available in a specific product.

| Name | Type | Description |
|---|---|---|
| SynchronizerSettings() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |
| Synchronizer | s_Synchronizer | The s_Synchronizer structure carries the synchronizer configuration variables assigned to a channel. |

## 3.17 TODTriggerSettings()

Load the time of day (TOD) trigger configuration settings to a specific channel. Dataflow through a channel is controlled by selected events. Time of day can be selected as a start or stop event. This function sets the TOD seconds and fractional seconds count that will activate a start or stop trigger.

| Name | Type | Description |
|---|---|---|
| TODTriggerSettings() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |
| TODTriggers | s_TODTriggers | The s_TODTriggers structure carries the TOD trigger configuration variables assigned to a channel. |

# 4.0 DMA Functions (dma_functions.h)

The DMA functions are used to manage DMA transactions between the device and the host computer. Refer to the DMA on Demand Operating Guide for detailed information about the techniques employed by Red Rapids products.

## 4.1 DMAAssign()

Assign DMA buffer space in host memory to the requested buffer index. The buffer index does not have to be associated with a hardware channel number, but that is frequently the case.

| Name | Type | Description |
|---|---|---|
| DMAAssign() | void (Linux)<br>int (Windows) | Non-zero indicates a memory allocation error in Windows. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| BufferIndex | unsigned | Sequential index number assigned to DMA buffers starting from zero. |

## 4.2 DMAConfigCheck()

Verify that each of the assigned DMA settings fall within the acceptable range.

| Name | Type | Description |
|---|---|---|
| DMAConfigCheck() | int | Non-zero indicates a configuration error was detected. |
| DMA | s_DMABuf | Variable assigned to the s_DMABuf structure consisting of the following members:<br>unsigned PageNumber<br>Number of the active DMA page starting from zero.<br>unsigned PageCount<br>Number of pages that form the circular buffer.<br>unsigned PagesPerMark<br>Number of pages between DMA markers.<br>unsigned BurstSizeB<br>Size of a single DMA burst transaction in bytes.<br>unsigned BurstCount<br>Number of bursts within a page.<br>unsigned long long **Address<br>Virtual address assigned to each page. |
| *ConfigError | unsigned | A non-zero value in each array index indicates a different type of configuration error:<br>0. Burst Size<br>1. Burst Count<br>2. Page Count<br>3. Pages per Marker |

## 4.3 DMAInit()

Initialize all variables in the s_DMABuf structure to default values.

| Name | Type | Description |
|---|---|---|
| DMAInit() | void | Nothing returned. |
| DMA | s_DMABuf | Set all s_DMABuf structure values to zero and pointers to NULL. |

## 4.4 DMALoad()

Copy data samples from a text file to the designated DMA buffer. This function is typically used to populate a DAC DMA buffer to demonstrate transmitter capability.

| Name | Type | Description |
|---|---|---|
| DMALoad() | int | Non-zero indicates a failure opening the sample data text file to read. |
| *p_file | char | Full path name to the sample data text file. |
| *DMA | s_DMABuf | Pointer to the s_DMABuf structure consisting of the following members:<br>unsigned PageNumber<br>Number of the active DMA page starting from zero.<br>unsigned PageCount<br>Number of pages that form the circular buffer.<br>unsigned PagesPerMark<br>Number of pages between DMA markers.<br>unsigned BurstSizeB<br>Size of a single DMA burst transaction in bytes.<br>unsigned BurstCount<br>Number of bursts within a page.<br>unsigned long long **Address<br>Virtual address assigned to each page. |
| DataItemSize | unsigned | Sample data size in bits. |

## 4.5 DMARelease()

Free memory allocated to DMA buffers by the Windows driver.

| Name | Type | Description |
|---|---|---|
| DMARelease() | int | Non-zero indicates an error condition. |
| *DMA | s_DMABuf | Pointer to the s_DMABuf structure consisting of the following members:<br>unsigned PageNumber<br>Number of the active DMA page starting from zero.<br>unsigned PageCount<br>Number of pages that form the circular buffer.<br>unsigned PagesPerMark<br>Number of pages between DMA markers.<br>unsigned BurstSizeB<br>Size of a single DMA burst transaction in bytes.<br>unsigned BurstCount<br>Number of bursts within a page.<br>unsigned long long **Address<br>Virtual address assigned to each page. |
| DevNum | unsigned | Device number assigned to the buffers that will be |

## 4.6 DMASave()

Writes the DMA buffer contents to a text file. The save operation starts at the first page in the buffer and sequences through the requested number of pages. There are two different implementations of the function depending on product features.

The following description applies to the streamlined DMASave() function that does not include packet processing features.

| Name | Type | Description |
|---|---|---|
| DMASave() | int | Non-zero indicates a failure opening the output data text file to write. |
| *p_file | char | Name of the output text file. |
| *DMA | s_DMABuf | Pointer to the s_DMABuf structure consisting of the following members:<br>unsigned PageNumber<br>Number of the active DMA page starting from zero.<br>unsigned PageCount<br>Number of pages that form the circular buffer.<br>unsigned PagesPerMark<br>Number of pages between DMA markers.<br>unsigned BurstSizeB<br>Size of a single DMA burst transaction in bytes.<br>unsigned BurstCount<br>Number of bursts within a page.<br>unsigned long long **Address<br>Virtual address assigned to each page. |
| NumPages | unsigned | Number of DMA pages to write to the file. |
| DataItemSize | unsigned | Sample data size in bits. |

The following description applies to the full featured DMASave() function that can strip packet information included with the raw data.

| Name | Type | Description |
|---|---|---|
| DMASave() | int | Non-zero indicates a failure opening the output data text file to write. |
| *p_file | char | Name of the output text file. |
| *DMA | s_DMABuf | Pointer to the s_DMABuf structure consisting of the following members:<br>unsigned PageNumber<br>Number of the active DMA page starting from zero.<br>unsigned PageCount<br>Number of pages that form the circular buffer.<br>unsigned PagesPerMark<br>Number of pages between DMA markers.<br>unsigned BurstSizeB<br>Size of a single DMA burst transaction in bytes.<br>unsigned BurstCount<br>Number of bursts within a page.<br>unsigned long long **Address<br>Virtual address assigned to each page. |
| RXChannel | s_Channel | The number and function of s_Channel structure members will be unique to each Red Rapids product. |
| TODClock | s_TODSettings | Variable assigned to the s_TODSettings structure consisting of the following members:<br>unsigned long TODSeconds<br>Value used to set the device clock current time of day (TOD) in seconds (max = (2^32)-1).<br>unsigned long ClockFrequency<br>Clock frequency used to increment the time of day fractional seconds, usually the ADC/DAC sample clock frequency, if there is no 1 PPS external source (max = (2^32)-1).<br>unsigned long TSFCode<br>TSF code defined by the VITA 49 specification.<br>unsigned long TSICode<br>TSI code defined by the VITA 49 specification. |
| NumPages | unsigned | Number of DMA pages to write to the file. |
| PacketInfo | unsigned | A non-zero value causes packet information to be printed to the destination file along with sample data when packets are enabled. |

## 4.7 DMAStatus()

Report the current status of the DMA engine that is assigned to the selected channel number. This register can be polled by software to track the progress of DMA transfers through individual pages and bursts.

The DMA engine status conveys the following information:

Engine[0]: The DMA engine is enabled (1) or disabled (0).

Engine[1]: A DMA request is pending (1) or not pending(0).

Engine[2]: The DMA engine is busy processing a transaction (1) or idle (0).

| Name | Type | Description |
|---|---|---|
| DMAStatus() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |
| *p_DMA | s_DMAStatus | Pointer to the s_DMAStatus structure consisting of the following members:<br>unsigned CurrentPage<br>Page number that is currently active. Page numbering begins with zero.<br>unsigned CurrentBurst<br>Burst number that is currently active. Burst numbering begins with zero.<br>unsigned Engine[2]<br>Status of the DMA Engine controls. |

# 5.0 Product Functions (product_functions.h)

The product functions are unique to each Red Rapids model number. They typically perform a series of hardware initialization tasks that are closely tied to the specific chips used on the product.

## 5.1 HWIntialize()

Select an initialization sequence based on the specific model number of the device.

| Name | Type | Description |
|---|---|---|
| HWInitialize() | int | Non-zero indicates an error condition. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| Model | unsigned | Model number of the device to initialize. |

## 5.2 MxxxIntialize()

Initialize all hardware features unique to a specific model number (Model xxx).

| Name | Type | Description |
|---|---|---|
| MxxxInitialize() | int | Non-zero indicates an error condition. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| MxxxDevices | s_MxxxDevices | Variable assigned to the s_MxxxDevices structure that contains a member for every chip on the product. |

# 6.0 Chip Functions (chip_functions.h)

The chip functions are used to access the configuration and status registers of individual semiconductor devices embedded in the product. Most ADC, DAC, and clock generation chips include a serial port that supports read and write transactions to internal registers. These functions access those registers through a serial bus controller on the device.

Consult the datasheet of each specific chip for a description of the available registers

## 6.1 ADA4927Write()

Set the enable bit on the Analog Devices ADA4927-2 dual differential amplifier chip.

| Name | Type | Description |
|---|---|---|
| ADA4927Write() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| PortNumber | unsigned | Port number assigned to each amplifier by Red Rapids. |
| Data | unsigned | Must be either zero (disable) or one (enable). |

## 6.2 AD5628Write()

Write internal registers through the serial control port of the Analog Devices AD5628 digital-to-analog converter chip.

| Name | Type | Description |
|---|---|---|
| AD5628Write() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| Command | unsigned | Internal instruction to write to the chip. |
| Address | unsigned | Internal address of the register to write to the chip. |
| Data | unsigned | Data value written to the selected register address. |

## 6.3 AD8000Write()

Set the enable bit on the Analog Devices AD8000 differential amplifier chip.

| Name | Type | Description |
|---|---|---|
| AD8000Write() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| PortNumber | unsigned | Port number assigned to each amplifier by Red Rapids. |
| Data | unsigned | Must be either zero (disable) or one (enable). |

## 6.4 AD9142ARead()

Read internal registers through the serial control port of the Analog Devices AD9142A digital-to-analog converter chip.

| Name | Type | Description |
|---|---|---|
| AD9142ARead() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| Address | unsigned | Internal address of the register to read from the chip. |

## 6.5 AD9142AWrite()

Write internal registers through the serial control port of the Analog Devices AD9142A digital-to-analog converter chip. It is important to note that the frequency tuning word and NCO phase offsets only change when the frequency tuning word update bit is set.

| Name | Type | Description |
|---|---|---|
| AD9512Write() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| Address | unsigned | Internal address of the register to write to the chip. |
| Data | unsigned | Data value written to the selected register address. |

## 6.6 AD9512Read()

Read internal registers through the serial control port of the Analog Devices AD9512 clock distribution chip.

| Name | Type | Description |
|---|---|---|
| AD9512Read() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| Address | unsigned | Internal address of the register to read from the chip. |

## 6.7 AD9512Write()

Write internal registers through the serial control port of the Analog Devices AD9512 clock distribution chip. It is important to note that the AD9512 assigns a buffer register to each control register. Write operations modify the buffer register, but not the control register. A register update command must be issued to transfer the contents of the buffer registers to the actual control registers.

| Name | Type | Description |
|---|---|---|
| AD9512Write() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| Address | unsigned | Internal address of the register to write to the chip. |
| Data | unsigned | Data value written to the selected register address. |

## 6.8 AD9652Cal()

Initiate a calibration cycle through the serial interface of the Analog Devices AD9652 analog-to-digital converter. This function includes a 1.3 second wait for the fast start-up calibration to complete.

| Name | Type | Description |
|---|---|---|
| AD9652Cal() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |

## 6.9 AD9652Read()

Read internal registers through the serial interface of the Analog Devices AD9652 analog-to-digital converter.

| Name | Type | Description |
|---|---|---|
| AD9652Read() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| Address | unsigned | Internal address of the register to read from the chip. |

## 6.10 AD9652Write()

Write internal registers through the serial interface of the Analog Devices AD9652 analog-to-digital converter.

| Name | Type | Description |
|---|---|---|
| AD9652Write() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| Address | unsigned | Internal address of the register to write to the chip. |
| Data | unsigned | Data value written to the selected register address. |

## 6.11 AD9653Read()

Read internal registers through the serial interface of the Analog Devices AD9653 analog-to-digital converter.

| Name | Type | Description |
|---|---|---|
| AD9653Read() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| Address | unsigned | Internal address of the register to read from the chip. |

## 6.12 AD9653Write()

Write internal registers through the serial interface of the Analog Devices AD9653 analog-to-digital converter.

| Name | Type | Description |
|---|---|---|
| AD9653Write() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| Address | unsigned | Internal address of the register to write to the chip. |
| Data | unsigned | Data value written to the selected register address. |

## 6.13 ADC12D1600Cal()

Initiate a calibration cycle through the serial interface of the Texas Instruments ADC12D1600 analog-to-digital converter. This function performs a read-modify-write operation to the configuration register that commands a calibration cycle.

| Name | Type | Description |
|---|---|---|
| ADC12D1600Cal() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |

## 6.14 ADC12D1600Read()

Read internal registers through the serial interface of the Texas Instruments ADC12D1600 analog-to-digital converter.

| Name | Type | Description |
|---|---|---|
| ADC12D1600Read() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| Address | unsigned | Internal address of the register to read from the chip. |

## 6.15 ADC12D1600Write()

Write internal registers through the serial interface of the Texas Instruments ADC12D1600 analog-to-digital converter.

| Name | Type | Description |
|---|---|---|
| ADC12D1600Write() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| Address | unsigned | Internal address of the register to write to the chip. |
| Data | unsigned | Data value written to the selected register address. |

## 6.16 ADL5566Write()

Set the enable bit on the Analog Devices ADL5566 dual differential amplifier chip.

| Name | Type | Description |
|---|---|---|
| ADL5566Write() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| PortNumber | unsigned | Port number assigned to each amplifier by Red Rapids. |
| Data | unsigned | Must be either zero (disable) or one (enable). |

## 6.17 ADS42LB69Read()

Read internal registers through the serial interface of the Texas Instruments ADS42LB69 analog-to-digital converter.

| Name | Type | Description |
|---|---|---|
| ADS42LB69Read() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| Address | unsigned | Internal address of the register to read from the chip. |

## 6.18 ADS42LB69Write()

Write internal registers through the serial interface of the Texas Instruments ADS42LB69 analog-to-digital converter.

| Name | Type | Description |
|---|---|---|
| ADS42LB69Write() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| Address | unsigned | Internal address of the register to write to the chip. |
| Data | unsigned | Data value written to the selected register address. |

## 6.19 FXL6408Read()

Read internal registers through the I2C bus of the Fairchild FXL6408 GPIO port expander. A typical software application will not call this function directly, it is primarily used by other functions.

| Name | Type | Description |
|---|---|---|
| FXL6408Read() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| Address | unsigned | Internal address of the register to read from the chip. |

## 6.20 FXL6408Write()

Write internal registers through the I2C bus of the Fairchild FXL6408 GPIO port expander. A typical software application will not call this function directly, it is primarily used by other functions.

| Name | Type | Description |
|---|---|---|
| FXL6408Write() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| Address | unsigned | Internal address of the register to write to the chip. |
| Data | unsigned | Data value written to the selected register address. |

## 6.21 INA3221Read()

Read internal registers through the I2C bus of the Texas Instruments INA3221 voltage monitor chip. A typical software application will not call this function directly, it is primarily used by other functions.

| Name | Type | Description |
|---|---|---|
| INA3221Read() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| Address | unsigned | Internal address of the register to read from the chip. |

## 6.22 INA3221Write()

Write internal registers through the I2C bus of the Texas Instruments INA3221 voltage monitor chip. A typical software application will not call this function directly, it is primarily used by other functions.

| Name | Type | Description |
|---|---|---|
| INA3221Write() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| Address | unsigned | Internal address of the register to write to the chip. |
| Data | unsigned | Data value written to the selected register address. |

## 6.23 LMK01000Sync()

Generate an external synchronization pulse on the Texas Instruments LMK01000 clock distribution chip SYNC pin.

| Name | Type | Description |
|---|---|---|
| LMK01000Sync() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |

## 6.24 LMK01000Write()

Write internal registers through the serial interface of the Texas Instruments LMK01000 clock distribution chip.

| Name | Type | Description |
|---|---|---|
| LMK01000Write() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| Address | unsigned | Internal address of the register to write to the chip. |
| Data | unsigned | Data value written to the selected register address. |

## 6.25 LTC1661Write()

Write internal registers through the serial interface of the Linear Technology LTC1661 digital-to-analog converter.

| Name | Type | Description |
|---|---|---|
| LTC1661Write() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| Address | unsigned | Internal address of the register to write to the chip. |
| Data | unsigned | Data value written to the selected register address. |

## 6.26 LTC2991Read()

Read internal registers through the I2C bus of the Linear Technology LTC2991 temperature monitor chip. A typical software application will not call this function directly, it is primarily used by other functions.

| Name | Type | Description |
|---|---|---|
| LTC2991Read() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| Address | unsigned | Internal address of the register to read from the chip. |

## 6.27 LTC2991Write()

Write internal registers through the I2C bus of the Linear Technology LTC2991 temperature monitor chip. A typical software application will not call this function directly, it is primarily used by other functions.

| Name | Type | Description |
|---|---|---|
| LTC2991Write() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChipSelect | unsigned | Code to select the desired chip on the serial bus. |
| Address | unsigned | Internal address of the register to write to the chip. |
| Data | unsigned | Data value written to the selected register address. |

# 7.0 Utility Functions (utility_functions.h)

The utility functions are used by the product demonstration software but may not apply to another application.

## 7.1 ChannelState()

Report the current state of the synchronizer for the selected channel.

| Name | Type | Description |
|---|---|---|
| ChannelState() | unsigned | 0: Idle<br>1: Initialize/Prime<br>2: Start Trigger<br>4: Start Delay<br>8: Synch/Stop Trigger<br>16: Stop Delay<br>32: Purge/Residue<br>64: Cycle<br>128: Scheduler/Stop Bench<br>255: Undefined |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |

## 7.2 ClearLatency()

Clears the PCI latency performance measurement register. This register stores the largest value recorded since the last clear operation.

| Name | Type | Description |
|---|---|---|
| ClearLatency() | void | Nothing returned |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |

## 7.3 DataFile()

Create a file of sine wave data samples that can be used to load a DMA buffer with the DMALoad() function. This file is used to demonstrate transmitter channels.

| Name | Type | Description |
|---|---|---|
| DataFile() | int | Non-zero indicates a failure opening the sine wave text file or a warning that the samples do not end on zero phase. |
| *p_file | char | Full path name to the sine wave data text file. |
| PageSize | unsigned | Size of the DMA page in bytes. |
| FsDivisor | double | The frequency of the sine wave is calculated as a fraction of the DAC sample clock (Fs/FsDivisor). |
| DataItemSize | unsigned | Sample data size in bits. |

## 7.4 DataLoad()

Copy data samples from a text file to the designated data buffer buffer. This function is used to populate a DAC data buffer with the same samples preloaded into the DMA buffer to demonstrate continuous transmitter capability.

| Name | Type | Description |
|---|---|---|
| DataLoad() | int | Non-zero indicates a failure opening the sample data text file to read. |
| *p_file | char | Full path name to the sample data text file. |
| **p_Address | void | Array containing the virtual address of each data buffer returned by malloc(). |
| PageSize | unsigned | Size of the data page in bytes. |
| NumPages | unsigned | Number of data pages to write to the file. |
| DataItemSize | unsigned | Sample data size in bits. |

## 7.5 DataSave()

Writes the data buffer contents to a text file. The save operation starts at the first page in the buffer and sequences through the requested number of pages. There are two different implementations of the function depending on product features.

The following description applies to the streamlined DataSave() function that does not include packet processing features.

| Name | Type | Description |
|---|---|---|
| DataSave() | int | Return of non-zero indicates a failure opening the output data text file to write. |
| *p_file | char | Name of the output text file. |
| **p_Address | void | Array containing the virtual address of each data buffer returned by malloc(). |
| PageSize | unsigned | Size of the data page in bytes. |
| NumPages | unsigned | Number of data pages to write to the file. |
| DataItemSize | unsigned | Sample data size in bits. |

The following description applies to the full featured DataSave() function that can strip packet information included with the raw data.

| Name | Type | Description |
|---|---|---|
| DataSave() | int | Return of non-zero indicates a failure opening the output data text file to write. |
| *p_file | char | Name of the output text file. |
| **p_Address | void | Array containing the virtual address of each data buffer returned by malloc(). |
| RXChannel | s_Channel | The number and function of s_Channel structure members will be unique to each Red Rapids product. |
| TODClock | s_TODSettings | Variable assigned to the s_TODSettings structure consisting of the following members:<br>unsigned long TODSeconds<br>Value used to set the device clock current time of day (TOD) in seconds (max = (2^32)-1).<br>unsigned long ClockFrequency<br>Clock frequency used to increment the time of day fractional seconds, usually the ADC/DAC sample clock frequency, if there is no 1 PPS external source (max = (2^32)-1).<br>unsigned long TSFCode<br>TSF code defined by the VITA 49 specification.<br>unsigned long TSICode<br>TSI code defined by the VITA 49 specification. |
| PageSize | unsigned | Size of the data page in bytes. |
| NumPages | unsigned | Number of data pages to write to the file. |
| PacketInfo | unsigned | A non-zero value causes packet information to be printed to the destination file along with sample data when packets are enabled. |

## 7.6 FIFOResidue()

Report the number of bytes currently stored in the DMA FIFO of the requested channel.

| Name | Type | Description |
|---|---|---|
| FIFOResidue() | unsigned | Number of bytes in the FIFO. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |

## 7.7 FreeResources()

Free all memory resources allocated to the device and close the device.

| Name | Type | Description |
|---|---|---|
| FreeResources() | int | Non-zero indicates an error occurred closing the device or releasing memory resources. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| Channel | s_Channel | The s_Channel structure carries all configuration variables assigned to a channel. |
| ChannelCount | unsigned | Number of channels assigned to the device. |
| UIO | S_UIOScan | Device identification in Linux. |
| OpenStatus | int | Zero indicates the device was successfully opened in Windows. |

## 7.8 InterruptResponse()

Report the response for the last interrupt that was serviced by software. The time is reported in units of microseconds.

A timer internal to the device is started when a hardware interrupt is issued to the host. The elapsed time is recorded in a register when the device detects that the interrupt mask bit has been set by software as part of the interrupt service routine. The value is held in the register until a new interrupt cycle is completed. This function can be used to read the most recent value recorded at any time.

It may be necessary to use the InterruptTimeout() function if the interrupt latency through the host is so long that multiple interrupts get processed. This function provides some insight into what timeout value may be needed.

| Name | Type | Description |
|---|---|---|
| InterruptResponse() | float | The maximum interrupt response time measured in microseconds. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |

## 7.9 InterruptTimeout()

Set a maximum time interval that the device can issue interrupt requests to the host. The IntTimeout parameter is multiplied by 4 ns to arrive at a time value. For example, an input value of 2,000 equates to 8 microseconds on the device.

This function is used to prevent the device from overwhelming the host with interrupt requests. Some operating systems will disable a device if the interrupt frequency exceeds an established threshold. This is a defensive measure to protect against malfunctioning hardware. Unfortunately, the threshold may be exceeded simply due to an excessive interrupt latency through the host. Latency is defined as the time interval from the device initiating the interrupt to the time it is serviced by the software application.

Setting an interrupt timeout does not necessarily mean that the requested interrupt will never be serviced. It simply prevents a single event from holding the interrupt active while waiting for the system to respond.

| Name | Type | Description |
|---|---|---|
| InterruptTimeout() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| IntTimeout | unsigned | The interrupt timeout value in increments of 4 ns to a maximum value of (2^28)-1, or 1.074 sec |

## 7.10 MemSize()

Report the size of the QDR II+ SRAM address bus in bits. This function is called by MemTest() to set the address range of a test.

| Name | Type | Description |
|---|---|---|
| MemSize() | unsigned | Address size in bits. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |

## 7.11 MemTest()

Exercise every address of the QDR II+ SRAM with an alternating binary pattern to verify functionality and performance. Note, setting the cycles variable to zero produces a much shorter test that does not touch every available address.

| Name | Type | Description |
|---|---|---|
| MemTest() | int | Number of errors detected. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| SRAMSelect | unsigned | Select SRAM-A (1) for test, SRAM-B (2) or both simultaneously (3). |

## 7.12 MicrosecondTimer()

Pause for the specified time in microseconds.

| Name | Type | Description |
|---|---|---|
| MicrosecondTimer() | int | Zero indicates successful completion.<br>Non-zero indicates an error condition. |
| uSeconds | long | The length of time to pause in microseconds. |

## 7.13 PciBenchmark()

Maximize PCIe bus traffic by continuously requesting DMA transactions on all available DMA channels. The benchmark condition is maintained for just over a quarter second.

| Name | Type | Description |
|---|---|---|
| PciBenchmark() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |

## 7.14 PciPerformance()

Report the PCIe bus throughput that was measured over the last quarter second interval and the maximum DMA latency since the last clear.

The PCI throughput is continuously measured every quarter second regardless of traffic. The maximum throughput can be measured by calling the PciBenchmark() function immediately before reading the performance measurements to maximize DMA traffic. Separate performance values are reported for receiver channels (DMA writes) and transmitter channels (DMA reads).

A timer internal to the device is started when a DMA request is issued to the host. The latency is measured as the time elapsed until the device detects that the DMA transfer has initiated. The measured value is recorded only if it exceeds the current maximum value. Separate latency values are reported for receiver channels (DMA writes) and transmitter channels (DMA reads).

| Name | Type | Description |
|---|---|---|
| PciPerformance() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| *p_Performance | float | Four element array to store PCI RX throughput (0), TX throughput (1), RX latency (2), and TX latency (3). |

## 7.15 ProbeSave()

Writes the contents of the transmitter snapshot memory to a text file. Each transmitter channel includes a small memory at the interface between the datapath output and the DAC. The memory captures a snapshot of sample data sent to the DAC when the datapath is first enabled. This information is useful for debugging transmitter datapath configuration settings.

| Name | Type | Description |
|---|---|---|
| ProbeSave() | int | Non-zero indicates a failure opening the output data text file to write. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |
| *p_file | char | Name of the output text file. |

## 7.16 ProcessChannels()

Query all available channels for any status change and update the user defined structure of the hardware handle. This function demonstrates one method of servicing DMA buffers based on an interrupt or software polling technique. Data is transferred between individual pages of the circular DMA buffer and another buffer that was allocated by the application software and attached to the hardware handle.

This function was created only for demonstration purposes and would probably not be used in applications that need to process data in real time directly from the DMA buffers.

| Name | Type | Description |
|---|---|---|
| ProcessChannels() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |

## 7.17 UserInit()

Initialize all variables in the s_User structure to default values.

| Name | Type | Description |
|---|---|---|
| UserInit() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| *p_User | s_User | Set all s_User structure values to zero and pointers to NULL. |

# 8.0 DSP Functions (dsp_functions.h or filter_functions.h)

The DSP functions are used to configure various types of signal conditioning elements that are available in some Red Rapids products. A product may contain up/down converters that can be uniquely configured within each channel. The exact number of channels depends on the configuration of a specific product.

## 8.1 ChannelizerInit()

Issue a command to initialize individual datapaths within a channelizer. Each datapath of a channelizer consists of tuners and filters that must be loaded and primed with data before the channel is operational.

| Name | Type | Description |
|---|---|---|
| ChannelizerInit() | unsigned | Zero indicates successful completion.<br>Non-zero indicates an error condition. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the IF channel number to target with the function. |
| Channelizer | unsigned | Variable containing the channelizer datapath number to target with the function. |

## 8.2 ChannelizerSelect()

Select the datapath within a channelizer that will be the target of any subsequent configuration commands. The configuration registers of each channelizer datapath are accessed through this index so that a single address range in the memory map can be used to load each datapath with unique configuration parameters.

| Name | Type | Description |
|---|---|---|
| ChannelizerSelect() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the IF channel number to target with the function. |
| Channelizer | unsigned | Variable containing the channelizer datapath number to target with the function. |

## 8.3 CICSet()

Set the CIC filter resample (decimation or interpolation) value. This value will be applied to the channelizer datapath number designated in the most recent ChannelizerSelect() call.

| Name | Type | Description |
|---|---|---|
| CICSet() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the IF channel number to target with the function. |
| Select | unsigned | Variable designating the filter number in the datapath to apply the configuration values. |
| Resample | unsigned | Variable containing the resample (decimation or interpolation) setting for the filter. |

## 8.4 FilterCoefficients()

Load the filter coefficients to a specific channel.

| Name | Type | Description |
|---|---|---|
| FilterCoefficients() | int | Non-zero indicates an unknown filter type or command error. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |
| Select | unsigned | Filter number to load. |
| Type | unsigned | Type number of filter to load. |
| IQ | unsigned | Zero if the filter is real, one if it is complex. |
| *p_Coefficients | unsigned | Array of coefficient values. |

## 8.5 FilterConfigure()

Load the coefficients, gain setting, and resampler setting to the selected filter within a specific channel. All configuration values are stored in a single text file separated by the following keywords:

| | |
|---|---|
| Coefficients | This keyword marks the beginning of a list of coefficients that define the characteristics of the filter. There must be one coefficient listed for each available tap, even if the filter is symmetric. Complex filter taps occupy two lines in the file with the in-phase (I) value preceding the quadrature (Q) value. |
| Log2Gain | The value immediately following this keyword conveys the gain through the filter expressed as a binary logarithm (log base2). The filter uses this gain along with the coefficient size to select the most significant bits available from each calculation. |
| Resample | The value immediately following this keyword defines the ratio of input to output samples processed through the filter. This can be either a downsample (decimation) ratio or upsample (interpolation) ratio depending on the filter. |

| Name | Type | Description |
|---|---|---|
| FilterConfigure() | int | Non-zero indicates a memory allocation error. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |
| Select | unsigned | Filter number to configure in the datapath. |
| *p_file | char | Full path name to the filter configuration text file. |

## 8.6 FilterCount()

Report the number of filters available in the selected channel.

| Name | Type | Description |
|---|---|---|
| FilterCount() | unsigned | Number of filters available. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |

## 8.7 FilterFilename()

Create a configuration file name of the form filter<M>_type<N>.txt, where M is the filter number in the channel and N is the type number. The character string includes the full directory path to the descriptor file as follows:

Windows: ..\filters\filterM_typeN.txt
Linux: ../filters/filterM_typeN.txt

| Name | Type | Description |
|---|---|---|
| FilterFilename() | void | Nothing returned. |
| *p_file | char | Full path name to the frame descriptor text file. |

## 8.8 FilterGain()

Load the log base2 filter gain to a specific channel.

| Name | Type | Description |
|---|---|---|
| FilterGain() | int | Non-zero indicates requested gain is out of range. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |
| Select | unsigned | Filter number to load. |
| Type | unsigned | Type number of filter to load. |
| Log2Gain | unsigned | Log base2 gain value. |

## 8.9 FilterKeyword()

Identify one of the three keywords that are expected in a filter configuration.

1. Coefficient
2. Log2Gain
3. Resample

| Name | Type | Description |
|---|---|---|
| FilterKeyword() | unsigned | Unique numeric identification for each keyword. |
| *Keyword | char | Character string to compare against possible keywords. |

## 8.10 FilterParser()

Extract filter configuration settings from the designated text file.

| Name | Type | Description |
|---|---|---|
| FilterParser() | int | Non-zero indicates file access error. |
| Type | unsigned | Filter type number. |
| *p_real | unsigned | Array of coefficient values for a real filter or the real part of a complex filter. |
| *p_imag | unsigned | Array of coefficient values for the imaginary part of a complex filter. |
| *p_Log2Gain | unsigned | Log base 2 gain setting. |
| *p_Resample | unsigned | Resampler setting. |
| *p_file | char | Full path name to the filter configuration text file. |

## 8.11 FilterResampler()

Load the resampler setting to a specific channel.

| Name | Type | Description |
|---|---|---|
| FilterResampler() | int | Non-zero indicates requested resampler value is out of range. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |
| Select | unsigned | Filter number to load. |
| Type | unsigned | Type number of filter to load. |
| Resample | unsigned | Resample integer to indicate the downsample ratio in a receiver channel or upsample ratio in a transmitter channel. |

## 8.12 FilterType()

Report the filter type number of the selected filter number in the selected channel.

| Name | Type | Description |
|---|---|---|
| FilterType() | unsigned | Type number of selected filter. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the channel number to target with the function. |
| Select | unsigned | Filter number to report. |

## 8.13 GainSet()

Set the automatic gain control (AGC) reference and step size values. These values will be applied to the channelizer datapath number designated in the most recent ChannelizerSelect() call.

| Name | Type | Description |
|---|---|---|
| GainSet() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the IF channel number to target with the function. |
| Reference | unsigned | Target reference value to be used by the AGC expressed as signal amplitude: $\sqrt{(I^2 + Q^2)}$ |
| StepSize | unsigned | AGC attack and decay factor (stepsize/256) from 0 (AGC off) to 255 (99.6%) |

## 8.14 RSSIMeasurement()

Report the received signal strength as the amplitude of a complex value.

| Name | Type | Description |
|---|---|---|
| RSSIMeasurement() | unsigned | Received signal strength expressed as signal amplitude: $\sqrt{(I^2 + Q^2)}$ |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the IF channel number to target with the function. |

## 8.15 RSSISet()

Set the number of samples to average in each RSSI calculation along with the low and high thresholds that will trigger an alarm. These values will be applied to the channelizer datapath number designated in the most recent ChannelizerSelect() call.

| Name | Type | Description |
|---|---|---|
| RSSISet() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the IF channel number to target with the function. |
| Average | unsigned | Number of samples to average in each RSSI calculation. The range of values is restricted to $2^n$, where n is any integer between zero (instantaneous) and ten. |
| Low | unsigned | Lower RSSI alarm threshold expressed as signal amplitude: $\sqrt{(I^2 + Q^2)}$ |
| Hi | unsigned | Upper RSSI alarm threshold expressed as signal amplitude: $\sqrt{(I^2 + Q^2)}$ |

## 8.1 SquelchSet()

Set the lower (T0) and upper (T1) thresholds that define the hysteresis of the squelch control. These values will be applied to the channelizer datapath number designated in the most recent ChannelizerSelect() call.

| Name | Type | Description |
|---|---|---|
| RSSISet() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the IF channel number to target with the function. |
| T0 | unsigned | Lower squelch threshold expressed as signal amplitude: $\sqrt{(I^2 + Q^2)}$ |
| T1 | unsigned | Lower squelch threshold expressed as signal amplitude: $\sqrt{(I^2 + Q^2)}$ |

## 8.1 TunerSet()

Set the operating rate and frequency translation values of the tuner. These values will be applied to the channelizer datapath number designated in the most recent ChannelizerSelect() call.

| Name | Type | Description |
|---|---|---|
| TunerSet() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the IF channel number to target with the function. |
| ComplexFs | unsigned | Complex sample rate of data entering the tuner expressed in samples per second. |
| Frequency | signed | Positive frequency value to up convert the baseband spectrum or negative frequency to down convert the baseband spectrum. |

## 8.1 WaveformSet()

Command the transmitter to generate a tone or frequency modulated test signal. These values will be applied to the channelizer datapath number designated in the most recent ChannelizerSelect() call.

| Name | Type | Description |
|---|---|---|
| WaveformSet() | void | Nothing returned. |
| *p_Handle | s_Handle | Pointer to the s_Handle device handle. Refer to the device driver reference manual for further details. |
| ChannelNumber | unsigned | Variable containing the IF channel number to target with the function. |
| ComplexFs | unsigned | Complex sample rate of data entering the tuner expressed in samples per second. |
| Waveform | signed | Optional data source for the transmitter to generate a test signal. Zero passes receiver data for normal operation while a non-zero value becomes the amplitude of a tone if either the FMCarrier or FMDeviation variables are set to zero. A non-zero value becomes the amplitude of a frequency modulated test signal if both the FMCarrier and FMDeviation variables are set to non-zero values. |
| FMCarrier | unsigned | FM test signal carrier frequency expressed in Hertz with a range of 1 to 100,000. |
| FMDeviation | float | FM test signal frequency deviation with a range of 0.000015 to 3. |