Publication No. REF-005-000-R00

Application Programming Interface (API) Reference Manual

Red Rapids

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

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 five separate libraries:

- Common 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.
- Device functions to communicate with various peripheral chips.
- Utility functions that provide performance monitoring and debug capabilities.

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
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
ISR	Interrupt Service Routine
LED	Light Emitting Diode
PCI	Peripheral Component Interconnect
QDR	Quad Data Rate
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**
R00	11/14/2014	Initial release.

2.0 Common Functions (common_functions.h)

The common 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 BusClose()

Close an open handle instance and de-allocates any system memory allocated to the handle. This function should be called before exiting an application that has previously opened a PCI adapter device.

Name	Type	Description
BusClose()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.

2.2 BusOpen()

Open and store a handle to the driver's kernel module. This function should be called to initiate communication with a PCI adapter device.

Name	Type	Description
BusOpen()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.

2.3 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 long	Number of channels available matching the type requested by the RxTxSelect flag.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
RxTxSelect	size_t	Flag to indicate whether to return the number of receiver channels (0) or transmitter channels (1).

2.4 ClockInit()

Command the device to synchronize all on-board clocks. 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()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.

2.5 ClockStatus()

Report the current 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()	unsigned long	Numeric status code indicating clock state: 0 External sample clock active, digital clocks are locked. 1 External sample clock active, digital clocks are not locked. 2 Sample clock locked to internal reference, digital clocks are locked. 3 Sample clock locked to internal reference, digital clocks are not locked. 4 Sample clock locked to external reference, digital clocks are locked. 5 Sample clock locked to external reference, digital clocks are not locked. >5 Sample clock is not locked.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.

2.6 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.

The SigFPGA (Model 37x) product assigns the following four types of clock errors:

Clock[0]: IDELAY out of lock condition detected.

Clock[1]: Digital clock initialization failed to complete.

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

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

Name	**Type**	**Description**
ErrorMask()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ErrorIntMask	s_ErrorStatus	Variable assigned to the s_FaultStatus structure consisting of the following members: unsigned long Bar[4] Interrrupt mask assigned to the four BAR fault status flags. Zero disables interrupt generation when the corresponding fault is detected and one enables interrupts. unsigned long Clock[8] Interrrupt mask assigned to the eight availalbe clock fault status flags. Zero disables interrupt generation when the corresponding fault is detected and one enables interrupts.

2.7 FirmwareRevision()

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

Name	**Type**	**Description**
FirmwareRevision()	unsigned long	Firmware revision date in hex format, but it is not a hex value (e.g. 04/27/2012 returns 0x04272012).
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.

2.8 GetErrorStatus()

Report the state of individual fault status flags. This function returns the total number of flags that are curently 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.

The SigFPGA (Model 37x) product assigns the following four types of clock errors:

Clock[0]: IDELAY out of lock condition detected.

Clock[1]: Digital clock initialization failed to complete.

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

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

Name	**Type**	**Description**
GetErrorStatus()	unsigned long	Total number of active error status flags recorded.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
*p_ErrorStatus	s_ErrorStatus	Pointer to the s_FaultStatus structure consisting of the following members: unsigned long Bar[4] Conveys the current state of four BAR fault status flags. Zero indicates that no fault has been detected. unsigned long Clock[8] Conveys the current state of three clock fault status flags. Zero indicates that no fault has been detected.

2.9 GetGlobalStatus()

Report the state of the global channel and fault status flags. There are up to eight channel status flags and one fault flag that can be monitored by software. This function returns the total number of flags that are curently 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. Further details about what type of event set the flag can be obtained using the channel specific GetChannelStatus() function.

A single global fault flag is assigned to the device. Further details about what type of event set the flag can be obtained using the GetFaultStatus() function.

Name	**Type**	**Description**
GetGlobalStatus()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
*p_GlobalStatus	s_GlobalStatus	Pointer to the s_GlobalStatus structure consisting of the following members: unsigned long Channel[8] Conveys the current state of each channel status flag (8 max). Zero indicates that the channel does not currently require service. unsigned long Error Conveys the current state of the device error status flag. Zero indicates that no faults have been detected.

2.10 GetPower()

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**
GetPower()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
*p_Power	s_Power	Pointer to the s_Power structure consisting of the following members: double Supply1[3] Conveys the bus voltage (0), current (1), and power dissipation (3) of supply #1. double Supply2[3] Conveys the bus voltage (0), current (1), and power dissipation (3) of supply #2. double Supply3[3] Conveys the bus voltage (0), current (1), and power dissipation (3) of supply #3.

2.11 GetTemperature()

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
GetTemperature()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
*p_Temperature	s_Temperature	Pointer to the s_Temperature structure consisting of the following members: double Sensor[5] Conveys the temperature recorded by five sensors. double Vcc Conveys the voltage measurment of the power supply to the temperature sensor.

2.12 GetTODSeconds()

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

Name	Type	Description
GetTODSeconds()	unsigned long	The current value of the time of day (TOD) seconds counter.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.

2.13 GlobalSync()

Issue a global synchronization strobe that is synchronous to all channels. A global synchronization is useful when multiple datapaths need to be initialized to the same state. For example, it can be used to ensure that a decimator function operating in two different channels will always pass data on the same clock tick instead of anytime within the decimation window.

Name	Type	Description
GlobalSync()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.

2.14 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 faults using the FaultMask() function and specific channel events using the ChannelMask() function.

Name	Type	Description
InterruptMask()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
GlobalIntMask	unsigned long	Zero disables hardware interrupts from the device and one enables hardware interrupts.

2.15 InterruptResponse()

Report the response time for the last interrupt that was serviced by software. The integer value returned by the function has to be multiplied by 4 ns to arrive at a time value. For example, a return value of 2,000 equates to 8 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()	unsigned long	The maximum interrupt response time measured in increments of 4 ns to a maximum value of (2^28)-1, or 1.074 sec
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.

2.16 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()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
IntTimeout	unsigned long long	The interrupt timeout value in increments of 4 ns to a maximum value of (2^28)-1, or 1.074 sec

2.17 LED()

Select the conditions that will illuminate LED-A and LED-B.

The four flags assigned to LED-A select the following conditions:

LEDA[0]: Flash LED A when the software trigger is detected.

LEDA[1]: Flash LED A when the 1 PPS trigger is detected.

LEDA[2]: Flash LED A when the GPIO rising trigger is detected.

LEDA[3]: Flash LED A when the GPIO falling trigger is detected.

The four flags assigned to LED-A select the following conditions:

LEDB[0]: Illuminate LED B when Channel #1 is active.

LEDB[1]: Illuminate LED B when Channel #2 is active.

LEDB[2]: Illuminate LED B when Channel #3 is active.

LEDB[3]: Illuminate LED B when Channel #4 is active.

Please note that LED-B will not respond to very short bursts of activity on a channel.The BAR mask bits are assigned to the following four types of errors:

Name	Type	Description
LED()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
LEDSource	s_LEDSource	Variable assigned to the s_ LEDSource structure consisting of the following members: unsigned long LEDA[4] Flags to select which of four possible conditions will illuminate LED A. Each condition is enable by setting the flag to a one or disabled by setting it to zero. Multiple conditions can be selected. unsigned long LEDB[4] Flags to select which of four possible conditions will illuminate LED B. Each condition is enable by setting the flag to a one or disabled by setting it to zero. Multiple conditions can be selected.

2.18 LoadGPIOSettings()

Load the GPIO settings to configure each connector pin as an input or outpt. This function also loads the logic state of any pin configured as an output.

Name	Type	Description
LoadGPIOSettings()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	Variable containing the channel number to target with the function.
GPIO	s_GPIOSettings	The number and function of s_GPIOSettings structure members will be unique to each Red Rapids product.

2.19 LoadPinDirection()

Set the direction of signals on external connector pins. Most Red Rapids products include connectors that allow the device to communicate with external signals. These external pins frequently need to support voltage levels that could damage internal components.

Bidirectional translator chips are used to form a bridge between incompatible internal and external voltages. The direction of each translator must be set as an input (0) or output (1) depending on the function of the connector pin.

Name	Type	Description
LoadPinDirection()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
Translator	s_TranslatorDirecti on	Variable assigned to the s_TranslatorDirection structure consisting of the following members: unsigned GPIODirection[6] Sets the direction of pins on the GPIO connector to input (0) or output (1). unsigned GPIODirection[12] Sets the direction of pins on the USER connector to input (0) or output (1).

2.20 LoadTODSettings()

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
LoadTODSettings()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	Variable containing the channel number to target with the function.
TODClock	s_TODSettings	Variable assigned to the s_ TODSettings structure consisting of the following members: unsigned long TODSeconds Value used to set the device clock current time of day (TOD) in seconds (max = (2^32)-1). unsigned long ClockFrequency 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).

2.21 MicrosecondTimer()

Pause for the specified time in microseconds.

Name	Type	Description
MicrosecondTimer()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
Time	unsigned long	The length of time to pause in microseconds.

2.22 ModelNumber()

Report the three digit model number of the device.

Name	Type	Description
ModelNumber()	unsigned long	Device model number in hex format, but it is not a hex value (e.g. Model 266 returns 0x266).
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.

2.23 Read()

Perform a 64-bit register read from the device at the specified BAR and address offset. Although this function performs a 64-bit read, it can be executed by either a 32-bit or 64-bit operating system. The operation is performed in two bus transactions if executed in a 32-bit environment.

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

Name	**Type**	**Description**
Read()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
Bar	unsigned long	Base address register (BAR) of the device.
Off	unsigned long	Address offset within the BAR.
*p_Value	unsigned long long	Pointer to a variable that will contain the value read by the operation.

2.24 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()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
BusSelect	unsigned long	Code to select a specific type of serial bus.
MulitFunction	unsigned long	Code that selects either the active edge of serial clock or the I2C transaction type depending on context.
Instruction	unsigned long	Instruction in the format supported by a particular serial bus, usually including an address.
InstructionSize	size_t	Size in bits of the instruction segment of a payload.
Data	unsigned long	Variable containing value to be written for write transactions, not used for read transactions.
DataSize	size_t	Size in bits of the data segment of a payload.
*p_ReadValue	unsigned long long	Pointer to a variable that will contain the value retrieved from a read transaction.

2.25 SoftwareReset()

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

Name	Type	Description
SoftwareReset()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.

2.26 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()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.

2.27 Write()

Perform a 64-bit register write to the device at the specified BAR and address offset. Although this function performs a 64-bit write, it can be executed by either a 32-bit or 64-bit operating system. The operation is performed in two bus transactions if executed in a 32-bit environment.

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

Name	Type	Description
Write()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
Bar	unsigned long	Base address register (BAR) of the device.
Off	unsigned long	Address offset within the BAR.
Value	unsigned long long	Variable containing the value that will be written by the operation.

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()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	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: unsigned long DMAMarker Interrrupt mask assigned to the channel DMA marker status flags. Zero disables interrupt generation when the DMA buffer requires service and one enables interrupts. unsigned long Error[2] Interrrupt 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 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()	unsigned long	Zero indicates the channel is off and one indicates the channel is on.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	Variable containing the channel number to target with the function.

3.3 ChannelReset()

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

Name	**Type**	**Description**
ChannelReset()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	Variable containing the channel number to target with the function.

3.4 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 actually begin signal acquisition or generation. The start command basically 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()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	Variable containing the channel number to target with the function.
OnOff	unsigned long	Zero stops channel processing and one enables channel processing.

3.5 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()	unsigned long	Zero indicates the channel is off and one indicates the channel is on.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	Variable containing the channel number to target with the function.

3.6 GetChannelStatus()

Report the state of individual channel status flags. This function returns the total number of flags that are curently 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**
GetChannelStatus()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	Variable containing the channel number to target with the function.
*p_ChannelStatus	s_ChannelStatus	Pointer to the s_ChannelStatus structure consisting of the following members: unsigned long DMAMarker Interrrupt mask assigned to the channel DMA marker status flags. Zero disables interrupt generation when the DMA buffer requires service and one enables interrupts. unsigned long Error[2] Interrrupt mask assigned to the two channel error status flags. Zero disables interrupt generation when the corresponding error is detected and one enables interrupts.

3.7 LoadChannel()

Load all of the configuration settings to a specific channel. This function basically calls a sequence of finer grain configuration functions that address individual features of a channel. The number of configuration settings to be loaded will vary across different Red Rapids products.

Name	Type	Description
LoadChannel()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	Variable containing the channel number to target with the function.
Channel	s_Channel	The number and function of s_Channel structure members will be unique to each Red Rapids product.

3.8 LoadDatapathControl()

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
LoadDatapathControl()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	Variable containing the channel number to target with the function.
DatapathControl	s_DatapathControl	The number and function of s_DatapathControl structure members will be unique to each Red Rapids product.

3.9 LoadEventDuration()

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
LoadEventDuration()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	Variable containing the channel number to target with the function.
EventDuration	s_EventDuration	The number and function of s_EventDuration structure members will be unique to each Red Rapids product.

3.10 LoadIFdpktFormat()

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
LoadIFdpktFormat()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	Variable containing the channel number to target with the function.
IFdpktFormat	s_IFdpktFormat	The number and function of s_IFdpktFormat structure members will be unique to each Red Rapids product.

3.11 LoadPayloadFormat()

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
LoadPayloadFormat()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	Variable containing the channel number to target with the function.
PayloadFormat	s_PayloadFormat	The number and function of s_PayloadFormat structure members will be unique to each Red Rapids product.

3.12 LoadSwitchSelect()

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
LoadSwitchSelect()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	Variable containing the channel number to target with the function.
SwitchSelect	s_SwitchSelect	The number and function of s_SwitchSelect structure members will be unique to each Red Rapids product.

3.13 LoadSynchronizer()

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 products.

Name	Type	Description
LoadSynchronizer()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	Variable containing the channel number to target with the function.
Synchronizer	s_Synchronizer	The number and function of s_Synchronizer structure members will be unique to each Red Rapids product.

3.14 LoadTODTriggers()

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
LoadTODTriggers()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	Variable containing the channel number to target with the function.
TODTriggers	s_TODTriggers	The number and function of s_TODTriggers structure members will be unique to each Red Rapids product.

3.15 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()	unsigned long	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_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	Variable containing the channel number to target with the function.

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 DMAAllocate()

Allocate DMA buffer space in host memory for the requested channel number. The DMA channel number does not have to be associated with a hardware channel number, but that is frequently the case. Buffers that are allocated by this function should be deallocated with the DMAFree() function before an application is closed.

Name	**Type**	**Description**
DMAAllocate()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	Variable containing the channel number to target with the function.
DMABuffer	s_DMABuffer	Variable assigned to the s_ChannelIntMask structure consisting of the following members: unsigned long BurstSizeB Size of a single DMA burst transaction in bytes. unsigned long BurstCount Number of bursts within a page. unsigned long PageCount Number of pages that form the circular buffer. unsigned long PagesPerMark Number of pages between DMA markers.

4.2 DMAConfigCheck()

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

Name	**Type**	**Description**
DMAConfigCheck()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
DMABuffer	s_DMABuffer	Variable assigned to the s_ChannelIntMask structure consisting of the following members: unsigned long BurstSizeB Size of a single DMA burst transaction in bytes. unsigned long BurstCount Number of bursts within a page. unsigned long PageCount Number of pages that form the circular buffer. unsigned long PagesPerMark Number of pages between DMA markers.
SettingNumber	size_t	Number assigned to the settings that will be checked: 1. Burst Size 2. Burst Count 3. Page Count 4. Pages per Marker

4.3 GetDMAStatus()

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**
DMAAllocate()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	Variable containing the channel number to target with the function.
*p_DMAStatus	s_DMAStatus	Pointer to the s_DMAStatus structure consisting of the following members: unsigned long CurrentPage Page number that is currently active. Page numbering begins with zero. unsigned long CurrentBurst Burst number that is currently active. Burst numbering begins with zero. unsigned long Engine[2] Current status of the DMA Engine controls.

4.4 DMAFree()

Release DMA buffer space from host memory that was previously protected by the DMAAllocate() functions.

Name	**Type**	**Description**
DMAFree()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	Variable containing the channel number to target with the function.

5.0 Device Functions (device_functions.h)

The device functions are used to access the configuration and status registers of individual chips present on the device. 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 specifc chip for a description of the available registers. A list of chips used on a specific Red Rapids product can be found in the hardware manuals.

5.1 AD9512Read()

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

Name	**Type**	**Description**
AD9512Read()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChipSelect	unsigned long	Code to select the desired chip on the serial bus.
Address	unsigned long	Internal address of the register to read from the chip.

5.2 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()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChipSelect	unsigned long	Code to select the desired chip on the serial bus.
Address	unsigned long	Internal address of the register to write to the chip.
Data	unsigned long	Data value written to the selected register address.

5.3 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()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChipSelect	unsigned long	Code to select the desired chip on the serial bus.

5.4 ADC12D1600Read()

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

Name	**Type**	**Description**
ADC12D1600Read()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChipSelect	unsigned long	Code to select the desired chip on the serial bus.
Address	unsigned long	Internal address of the register to read from the chip.

5.5 ADC12D1600Write()

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

Name	**Type**	**Description**
ADC12D1600Write()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChipSelect	unsigned long	Code to select the desired chip on the serial bus.
Address	unsigned long	Internal address of the register to write to the chip.
Data	unsigned long	Data value written to the selected register address.

5.6 ADL5566Write()

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

Name	**Type**	**Description**
ADL5566Write()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChipSelect	unsigned long	Code to select the desired chip on the serial bus.
PortNumber	unsigned long	Port number assigned to each amplifier by Red Rapids.
Data	unsigned long	Must be either zero (disable) or one (enable).

5.7 ADS42LB69Read()

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

Name	Type	Description
ADS42LB69Read()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChipSelect	unsigned long	Code to select the desired chip on the serial bus.
Address	unsigned long	Internal address of the register to read from the chip.

5.8 ADS42LB69Write()

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

Name	Type	Description
ADS42LB69Write()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChipSelect	unsigned long	Code to select the desired chip on the serial bus.
Address	unsigned long	Internal address of the register to write to the chip.
Data	unsigned long	Data value written to the selected register address.

5.9 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()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChipSelect	unsigned long	Code to select the desired chip on the serial bus.
Address	unsigned long	Internal address of the register to read from the chip.

5.10 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()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChipSelect	unsigned long	Code to select the desired chip on the serial bus.
Address	unsigned long	Internal address of the register to write to the chip.
Data	unsigned long	Data value written to the selected register address.

5.11 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()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChipSelect	unsigned long	Code to select the desired chip on the serial bus.
Address	unsigned long	Internal address of the register to read from the chip.

5.12 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()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChipSelect	unsigned long	Code to select the desired chip on the serial bus.
Address	unsigned long	Internal address of the register to write to the chip.
Data	unsigned long	Data value written to the selected register address.

5.13 LTC1661Write()

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

Name	Type	Description
LTC1661Write()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChipSelect	unsigned long	Code to select the desired chip on the serial bus.
Address	unsigned long	Internal address of the register to write to the chip.
Data	unsigned long	Data value written to the selected register address.

5.14 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()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChipSelect	unsigned long	Code to select the desired chip on the serial bus.
Address	unsigned long	Internal address of the register to read from the chip.

5.15 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()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChipSelect	unsigned long	Code to select the desired chip on the serial bus.
Address	unsigned long	Internal address of the register to write to the chip.
Data	unsigned long	Data value written to the selected register address.

6.0 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.

6.1 MxxxIntialize()

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

Name	Type	Description
MxxxInitialize()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
MxxxDevices	s_MxxxDevices	Variable assigned to the s_MxxxDevices structure that contains a member for every chip on the product.
*p_RXChannel	s_Channel	The number and function of s_Channel structure members will be unique to each Red Rapids product.
*p_TXChannel	s_Channel	The number and function of s_Channel structure members will be unique to each Red Rapids product.

7.0 Utility Functions (dma_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 long	0: Idle 1: Initialize/Prime 2: Start Trigger 4: Start Delay 8: Synch/Stop Trigger 16: Stop Delay 32: Purge/Residue 64: Cycle 128: Scheduler/Stop Bench 255: Undefined
ChannelNumber	unsigned long	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()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.

7.3 DataSave()

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. The function will work on any buffer accessible through a virtual address, it does not have to be a DMA buffer.

Name	Type	Description
DataSave()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
**p_PageAddresses	void	Array containing the virtual address of each DMA page.
RXChannel	s_Channel	The number and function of s_Channel structure members will be unique to each Red Rapids product.
*p_file	char	Name of the output text file.
PageCount	size_t	Number of DMA pages to write to the file.
DataItemSize	size_t	Size in bits of the data items stored in the DMA buffer.

7.1 DataLoad()

Loads sine wave data samples at the requested frequency into a DMA buffer. The buffer is typically used to supply a DAC to demostrate transmitter capability.

Name	Type	Description
DataLoad()	unsigned long	Zero indicates that the buffer contains an integer number of sine wave periods. Non-zero indicates a fractional number of sine wave periods in the buffer.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
**p_PageAddresses	void	Array containing the virtual address of each DMA page.
TXChannel	s_Channel	The number and function of s_Channel structure members will be unique to each Red Rapids product.
FsDivisor	double	Frequency of the sine wave expressed as Fs/FsDivisor.
DataItemSize	size_t	Size in bits of the data items stored in the DMA buffer.

7.1 FIFOResidue()

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

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

7.1 InterruptResponse()

Report the maximum interrupt service response time recorded since the register was last cleared. The integer value returned by the function has to be multiplied by 4 ns to arrive at a time value. For example, a return value of 2,000 equates to 8 microseconds.

A timer internal to the device is started when a hardware interrupt is issued to the host. The response is measured as the time elapsed until the device detects that the interrupt mask bit has been set by software as part of the interrupt service routine. The measured value is recorded only if it exceeds the current maximum value.

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()	unsigned long	The maximum interrupt response time measured in increments of 4 ns to a maximum value of (2^28)-1, or 1.074 sec
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.

7.2 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 long	Address size in bits.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.

7.3 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()	unsigned long	Zero indicates successful completion. Non-zero indicates the number of errors detected.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
cycles	size_t	Number of test cycles to run. A zero value will produce a shorter test by running a single cycle through one thirty-second of the available addresses.

7.1 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()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.

7.1 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 reciever 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 reciever channels (DMA writes) and transmitter channels (DMA reads).

Name	Type	Description
PciPerformance()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
*p_Performance	unsigned long	Four element array to store PCI RX throughput (0), TX throughput (1), RX latency (2), and TX latency (3).

7.2 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 trasferred 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()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.

7.1 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()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	Variable containing the channel number to target with the function.
*p_file	char	Name of the output text file.

7.2 UserInitialize()

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 actually begin signal acquisition or generation. The start command basically arms a channel to react when the requested start event is detected (trigger, time of day, 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, other events.

Name	**Type**	**Description**
UserInitialize()	unsigned long	Zero indicates successful completion. Non-zero indicates an error condition.
*p_Adapter	s_Adapter	Pointer to the s_Adapter handle. Refer to the Adapter Device Driver Rerence Manual for further information.
ChannelNumber	unsigned long	Variable containing the channel number to target with the function.
DMABuffer	s_DMABuffer	*TBD*
*p_DataBuffer	unsigned long	*TBD*