Publication No. REF-004-002-R00

SigStream Operating Guide

Red Rapids

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

Table of Contents

List of Figures

List of Tables

1.0 Introduction

The SigStream application demonstrates a complete signal acquisition and generation thread performed on a Red Rapids signal converter or up/down converter product. The software follows the API demonstration flow described in document REF-005-002. The behavior of the application is controlled through several structures that are grouped into two categories; hardware configuration and operating configuration.

The hardware configuration structures tend to be channel independent and frequently apply to specific chips that are not present on every product. The variables assigned to these structures are used to perform the following operations:

- Set the time of day (TOD) clock.
- Select the action that will illuminate status LEDs.
- Set the direction and logic state of GPIO or USER ports.
- Load internal registers of all chips that have a serial bus interface (SPI, I2C, etc).

The operating configuration structures target the signal conditioning and data manipulation features available to each channel. The variables assigned to these structures are used to perform the following operations:

- Select the source or sink attached to each datapath.
- Format the data payload.
- Define the behavior of the channel.
- Format the IF data packets.
- Set DMA operating parameters.
- Set interrupt processing options.

The SigStream application automatically scales to any number of transmitter or receiver channels supported by the target product. It obtains this product specific information by reading a feature identification register from the Red Rapids device.

The latest product documentation and software is available for download from the Red Rapids web site (www.redrapids.com).

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.
- Active low signals are followed by ‘#’, For example, TRST#.

☞	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 used in this manual.

• **ADC**	Analog to Digital Converter
• **DAC**	Digital to Analog Converter
• **PPS**	Pulse per Second
• **RX**	Receiver
• **TOD**	Time of Day
• **TSF**	TimeStamp-Fractional (VITA 49)
• **TSI**	TimeStamp-Integer (VITA 49)
• **TX**	Transmitter
• **VITA**	VME International Trade Association
• **VRT**	VITA Radio Transport

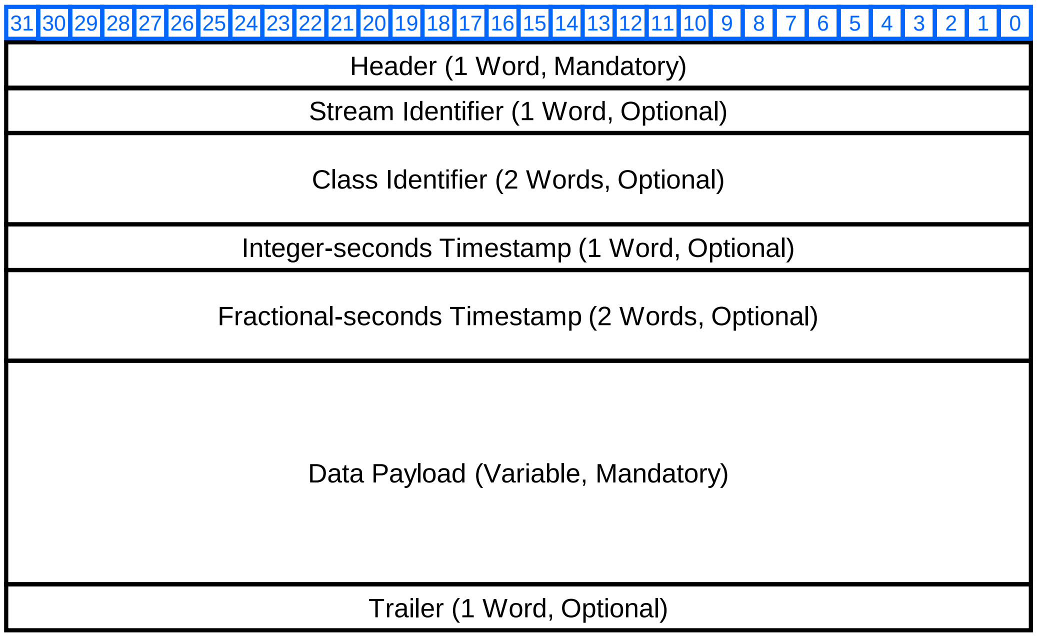
1.2 Revision History

Version	**Date**	**Description**
R00	1/12/2016	Initial release.

2.0 VITA Radio Transport Standard (ANSI/VITA 49.0)

Red Rapids has adopted the IF Data Packet defined by the VITA Radio Transport (VRT) Standard (ANSI/VITA 49.0) as the data format for all products supported by the SigStream application. Consequently, much of the nomenclature used throughout the code is derived from this standard. A detailed description of the IF Data Packet can be found in Section 6 of the VRT document available from VITA. A summary of the specification is provided here to assist in the explanation of the SigStream variables.

The template for an IF Data Packet is show in Figure 2-1, all fields are aligned to a 32-bit boundary. The packet consists of a mandatory Header and Data Payload along with five optional fields that convey additional information about the payload. Optional fields are completely absent from the packet if there is no information to convey. Remaining fields simply move up toward the header with no padding.



31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Header (1 Word, Mandatory)
Stream Identifier (1 Word, Optional)
Class Identifier (2 Words, Optional)
Integer-seconds Timestamp (1 Word, Optional)
Fractional-seconds Timestamp (2 Words, Optional)
Data Payload (Variable, Mandatory)
Trailer (1 Word, Optional)

Figure 2-1 VRT (VITA 49.0) IF Data Packet Format

2.1 Header

The format of the 32-bit Data Packet Header is shown in Figure 2-2.

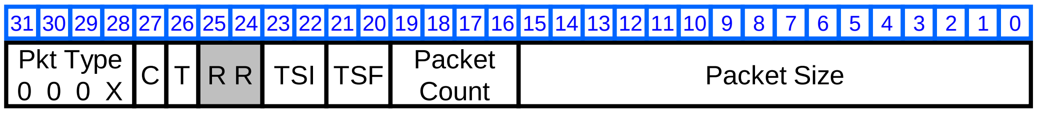


Figure 2-2 Data Packet Header Format

There are only two valid packet type codes assigned to an IF Data Packet. The value "0000" designates a packet without the optional Stream Identifier and "0001" indicates that a Stream Identifier is present in the packet.

The "C" bit field is set to one if a Class Identifier is present in the packet or set to zero if it is not included.

The “T” bit field is set to one if a Trailer is included at the end of the packet or set to zero if it is not.

The TSI and TSF codes indicate whether the timestamps are included in the packet. If timestamps are present, these codes also convey the origin of each value as shown in Table 2-1.

Table 2-1 TSI/TSF Code Meaning

Code	TSI	TSF
0	Integer Timestamp Not Present	Fractional Timestamp Not Present
1	Coordinated Universal Time (UTC)	Sample Count
2	GPS Time	Real Time (Not Supported)
3	Other	Free Running Count

The Packet Count field increments with each new IF Data Packet produced for a specific stream. The least significant bit of the modulo-16 count is the right-most bit in the field.

The Packet Size field indicates the total number of 32-bit words in the current IF Data Packet; including header, payload, and optional fields. The 16-bit field size limits the maximum packet size to 65,535 words.

2.2 Stream and Class Identifier

The Stream Identifier is an optional 32-bit number assigned to a packet stream. This designation can help back-end processes identify the source of IF data in a multi-stream environment.

The optional 64-bit Class Identifier field makes it possible to identity the Information Class used for the application and the Packet Class used to create each packet. Figure 6.1.3-1 shows the the three subfields included in the Class ID field. The first subfield contains the Organizationally Unique Identifier (OUI), an IEEE-assigned 24-bit number which indicates the identity of the company that created the Information Class and the Packet Class generating the IF Data Packet. The second field contains the Information Class code indicating which of that company’s Information Classes defines the Information Stream containing the Packet Stream. The third field contains the Packet Class code that identifies which of the company’s Packet Classes was used to make the packet.

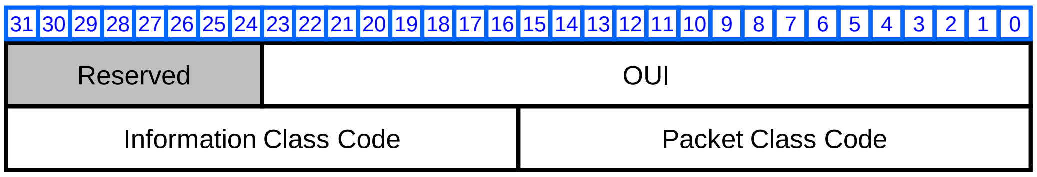


Figure 2-3 Class Identifier Contents

2.3 Timestamps

The optional Timestamp in an IF Data Packet is divided into an integer-seconds component and a fractional-seconds component. Together these two components precisely specify a reference time point for the first data sample contained in the packet.

The integer-seconds part consists of a single 32-bit word that specifies the reference time point only to one-second resolution. It may be used to convey UTC time, GPS time, or some user-specified time-code.

The fractional-seconds part consists of an unsigned 64-bit integer which occupies two consecutive 32-bit words. The first 32-bit word in the packet contains the most significant bits of the overall 64-bit value. It may be expressed as a number of sample count periods, a real-time value in picoseconds, or a free running modulo-N counter where N can be any positive integer up to 64.

2.4 Data Payload

The Data Payload contains a contiguous sequence of data samples encapsulated in a virtual container called the Item Packing Field. The Item Packing Field, shown in Figure 2-4, can occupy any size from one to sixty-four bits.

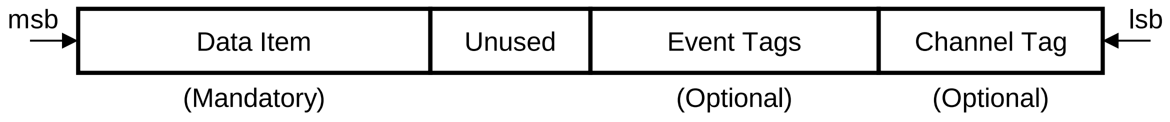


Figure 2-4 Item Packing Field

An Item Packing Field must contain exactly one Data Item that can occupy the entire field if no Event Tags or Channel Tags are present. The Data Item can be a real-valued sample, the real or imaginary component of a complex Cartesian sample, or the amplitude or phase component of a complex polar sample. The Data Item is always left justified in the Item Packing Field.

A Channel Tag is a label associating a Data Item with a particular signal conveyed by an IF Data Packet. Only one optional channel tag may be included in the Item Packing Field. It is always right justified in the Item Packing Field and must be no larger than fifteen bits.

An Event Tag is a bit used to indicate that a signal-related or processing-related event has occurred coincident with a Data Item in the payload. The Event Tag field can be no larger than seven bits and is located immediately to the left of the Channel Tag if it is present.

The size of the Item Packing Field may be larger than required to contain the included fields. In this case, the unused bits shall reside immediately to the right of the Data Item.

There are two methods of organizing the Item Packing Fields within the 32-bit payload words. Figure 2-5 illustrates the processing efficient method using a 15-bit Item Packing Field as an example. This approach simply left justifies the maximum number of Item Packing Fields that will fit in a 32-bit word without truncation, any remaining bits are set to zero.

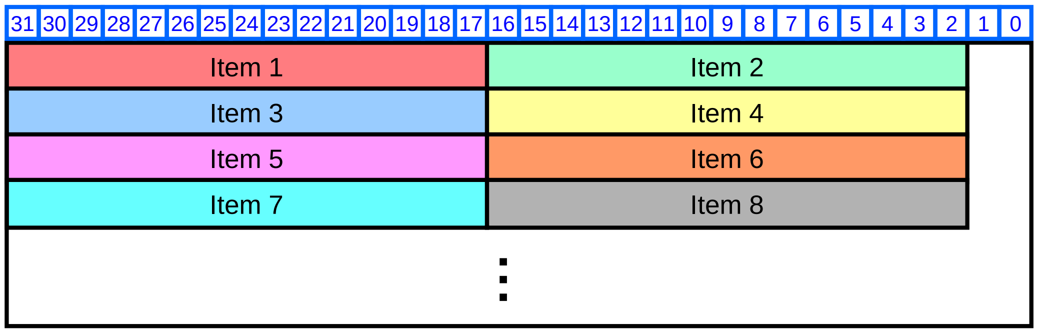


Figure 2-5 Processing Efficient Payload Organization

Figure 2-6 illustrates the link efficient method using the same 15-bit Item Packing Field as an example. This approach utilizes all of the available payload space by wrapping Item

Packing Fields between 32-bit word boundaries. This eliminates the zero padding of the processing efficient organization. The payload always ends with a complete Item Packing Field, so there may be some zero padding at the end of the final 32-bit word.

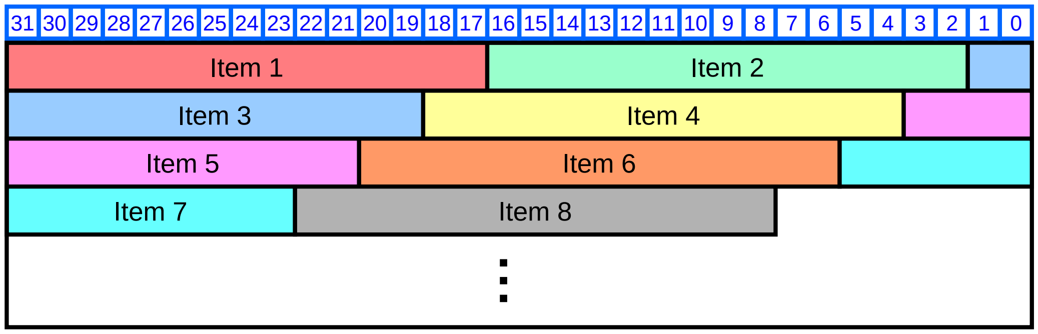


Figure 2-6 Link Efficient Payload Organization

When the Item Packing Fields contain complex samples, the in-phase value always precedes the quadrature value for Cartesian coordinates and the magnitude always precedes phase for polar coordinates.

2.5 Trailer

The trailer, shown in Figure 2-7, contains fields that indicate the validity of the data and the status of the processes producing that data. It also contains a field that indicates whether related context is being sent in one or more separate Context Packets.

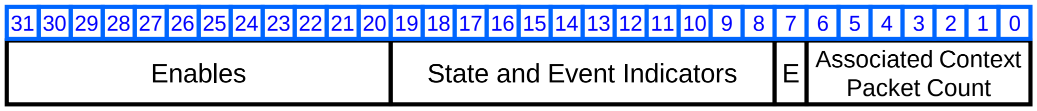


Figure 2-7 Trailer

The bits assigned to the Enables field act as a mask to the corresponding bit location in the State and Event Indicators field. An enable bit is set to zero when there is nothing to report in the associated State and Event Indicators bit. The enable bit is set to one when there is information available in one of the indicator bits listed in Table 2-2.

Table 2-2 State and Event Indicators

Enables Bit	Indicators Bit	Description
31	19	Calibrated Time Indicator
30	18	Valid Data Indicator
29	17	Reference Lock Indicator
28	16	AGC/MGC Indicator
27	15	Detected Signal Indicator
26	14	Spectral Inversion Indicator
25	13	Over-range Indicator
24	12	Sample Loss Indicator
23…20	11…8	User Defined Indicators

When the “E” bit is set to one the Associated Context Packet Count filed contains a count of all of transmitted Context packets that are directly or indirectly associated with the IF Data Packet. When the “E” bit is set to zero, the Associated Context Packet Count is undefined.

3.0 Hardware Configuration

Default variable settings are assigned to the hardware configuration structures contained in the default_hardware.config.c source file. The exact contents of this file will vary between product families due to the characteristics of specific chips used in the design. There may also be model number specific structures within a product family.

3.1 Time of Day (TOD) Clock Settings

```
typedef struct
{
   unsigned long TODSeconds;
   unsigned long ClockFrequency;
   unsigned long TSFCode;
   unsigned long TSICode;
} s_TODSettings;
```

An embedded time of day (TOD) clock is used to produce IF data packet timestamps and start/stop triggers for the synchronizer. Time is measured in two components; integer seconds and fractional seconds. The TODSeconds variable is used to set the integer seconds value. This value is often synchronized to an external time source such as GPS or UTC. The absolute time of day is read from the external source and then written to the TOD clock. The entire operation must be completed before the external source increments to the next second to achieve synchronization.

An external source usually supplies a 1 PPS strobe that can be connected to the Red Rapids device to increment the integer seconds value. The ClockFrequency variable must be set to zero if an external 1 PPS signal is active. If an external 1 PPS is not available, the ClockFrequency variable should be set to a value equivalent to the frequency of the ADC/DAC sample clock. An internal counter will then be used to increment the seconds counter autonomously when the specified number of periods have passed. The fractional seconds value will be reset to zero coincident with every increment of the integer seconds value.

It is impossible to achieve exact alignment between an external source and the TOD clock. If a 1 PPS strobe is used, there is no way to predict the phase relationship between the leading edge of the strobe and the leading edge of the clock that will be used to increment the integer seconds. In general, the TOD clock will lag the external source by two sample clock periods with an additional ambiguity of plus or minus one sample clock period. If a 1 PPS strobe is not used, the fractional seconds counter will start from zero when the integer seconds value is loaded. This will cause the TOD clock to lag the external source by the time it took to read and write the value, which is likely some fraction of a second. This latency will remain constant if the sample clock is phase locked to the external time source and there will be no ambiguity if the sample clock is set to an integer value. If the sample clock is non-integer, then the TOD clock will accumulate a small error every second. The error will be equivalent to the difference between the non-integer sample clock frequency and the integer value assigned to the ClockFrequency variable.

The integer seconds component of the TOD clock can be read from software but the fractional seconds component cannot. It is impractical to supply a fractional seconds value since it transitions within a clock domain that is not coherent to the bus clock. The nondeterministic nature of the bus timing also contributes an ambiguity that is much greater than the fractional seconds resolution.

Timestamps are only available when data is received from the hardware in IF Data Packets as discussed in Section 2.0. This is accomplished by setting the PacketSize variable to a non-zero value in the IF Data Packet structure (see section TBD). The integer-seconds and fractional-seconds timestamps will only be included in the IF Data Packet if their corresponding codes (TSICode/TSFCode) are also non-zero.

Setting the TSICode variable to one indicates that the TOD clock integer seconds value is synchronized to Coordinated Universal Time (UTC). Setting the variable to two indicates that GPS Time is the source. The TSICode should be set to three if the TOD clock is synchronized to any other source.

Setting the TSFCode variable to one will report fractional-second timestamps measured in periods of the sample clock. This value will never exceed the sample clock frequency since that is the one second roll-over value. Setting the TSFCode to three will produce a free running 64-bit fractional seconds count that increments with each period of the sample clock. The counter will reset back to zero when the maximum 64-bit value is reached. The integer seconds timestamp is automatically suppressed when the TSFCode is set to three since it has no meaning in the context of a free running count.

3.2 LED Source Selection

```
typedef struct
{
   unsigned long LEDA[ ];
   unsigned long LEDB[ ];
} s_LEDSource;
```

There are two status LED indicators available to visually monitor trigger and channel activity. Trigger activity is displayed on LED A and channel activity is displayed on LED B.

There are multiple triggers available to synchronize processing through a channel. The LEDA[] array allows any selected trigger shown in Table 3-1 to flash the light. Setting a variable in the array to zero disables monitoring of the trigger assigned to that index. Setting a variable to one causes the corresponding trigger to flash the LED when an active edge is detected. Multiple trigger monitors can be enabled simultaneously, but there will be no way to determine which specific trigger produced a flash.

Table 3-1 LED A Trigger Monitor Assignment

Index	Trigger Source
0	GPIO #1 Rising
1	GPIO #2 Rising
2	GPIO #3 Rising
3	GPIO #4 Rising
4	GPIO #1 Falling
5	GPIO #2 Falling
6	GPIO #3 Falling
7	GPIO #4 Falling
8	Software
9	Coax

The LEDB[ ] array is used to enable activity monitoring in each channel. A channel is considered active when the synchronizer state machine is operating between start and stop events. Setting a variable in the array to zero disables monitoring of the channel assigned to that index, as shown in Table 3-2. Setting a variable to one causes the assigned channel to flash the LED when the channel is actively processing data. Multiple channel monitors can be enabled simultaneously, but there will be no way to determine which specific channel produced a flash.

Table 3-2 LED B Channel Monitor Assignment

Index	Trigger Source
0	Channel #1
1	Channel #2
2	Channel #3
3	Channel #4

3.3 GPIO Settings

```
typedef struct
{
   unsigned long Direction[ ];
   unsigned long OutputValue[ ];
   unsigned long DebounceEnable[ ];
   unsigned long Disconnect;
   unsigned long OnePPSDebounceEnable;
} s_GPIOSettings;
```

The GPIO connector is a bidirectional interface that can be used to accept external input triggers or drive an external output signal. The Direction[ ] array sets each signal assigned to the connector as an input when the variable is set to one or an output when the variable is set to zero. Table 3-3 shows the mapping of array index to GPIO signal.

Table 3-3 GPIO Setting Assignment

Index	Connector Pin
0	GPIO #1
1	GPIO #2
2	GPIO #3
3	GPIO #4
4	GPIO #5
5	GPIO #6

The OutputValue[ ] array only applies to signals that have been configured as outputs. This variable is used to drive the corresponding output high with a one or low with a zero using the mapping shown in Table 3-3.

The DebounceEnable[ ] array is used to activate the optional debounce circuit on the GPIO trigger inputs. A hysteresis equivalent to four sample clock periods is applied to the input when the variable is set to one using the mapping shown in Table 3-3. This prevents ringing on the input from producing multiple unintended trigger crossings. Setting the variable to zero disables the debounce circuit.

Setting the Disconnect variable to one will electrically isolate (tristate) all of the pins on the GPIO connector. The variable must be set to zero if any of the GPIO signals are going to be used as inputs or outputs.

The CoaxDebounceEnable variable activates the optional debounce circuit on the coax input trigger. It behaves the same as the GPIO debounce circuit described above.

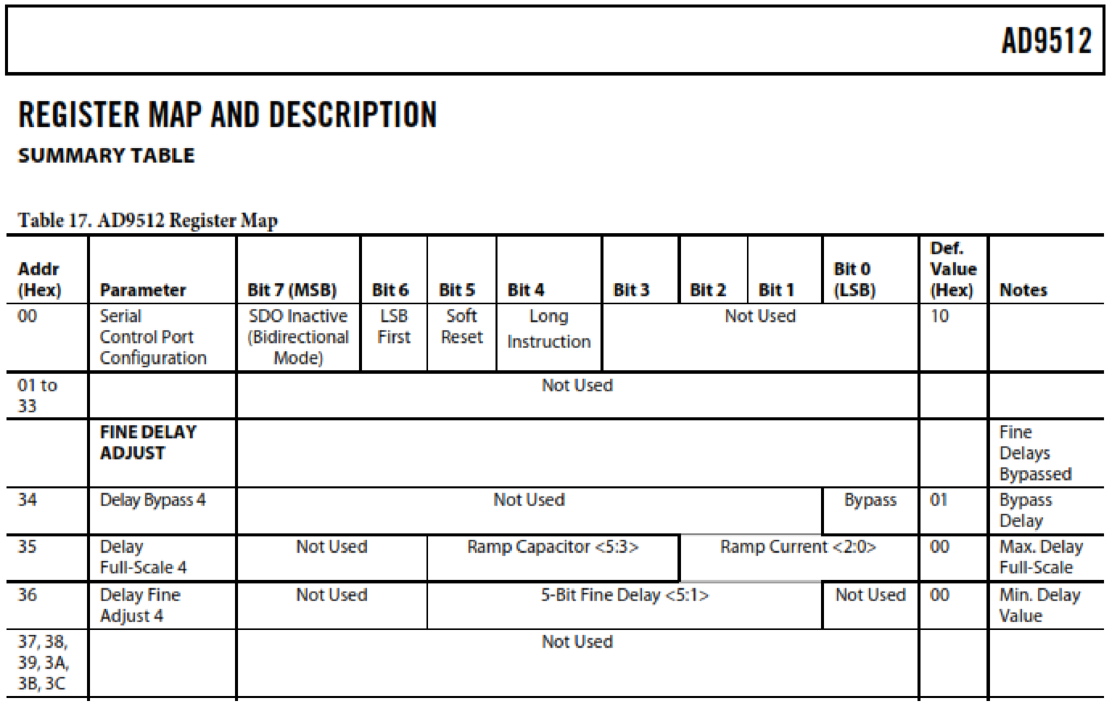
3.4 Device Configuration Settings

```
typedef struct
{
   chip_type chip_function;
   .
   .
   .
   chip_type chip_function;
} s_MxxxDevices;
```

Every Red Rapids product is assigned a three digit model number that uniquely identifies the chips used on that product. A device configuration structure that lists all of the chips (chip_type) is assigned to each model number (s_MxxxDevices). The name of each chip_type corresponds to the vendor part number for that device. The chip_function variable assigns values to the configuration registers internal to each chip_type. The following is an example of the structure that would be used for a design that includes the AnalogDevices AD9512 clock distribution chip and two Texas Instruments ADS42LB69 dual channel ADC chips.

```
typedef struct
{
   s_AD9512 ClockDist;
   s_ADS42LB69 RX1TO2_ADC;
   s_ADS42LB69 RX3TO4_ADC;
} s_MxxxDevices;
```

Please refer to the device vendor datasheet for a list of configuration registers specific to the part number. These registers are typically accessed through a serial port (SPI, I2C, etc.) provided on the device. Figure 3-1 contains a partial listing of the configuration registers internal to the Analog Devices AD9512 clock distribution chip that is used on several Red Rapids products. Each register is assigned an address and configuration variable, which are both eight bits in this example.



AD9512

REGISTER MAP AND DESCRIPTION

SUMMARY TABLE

Table 17. AD9512 Register Map

Addr (Hex)	Parameter	Bit 7 (MSB)	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 (LSB)	Def. Value (Hex)	Notes
00	Serial Control Port Configuration	SDO Inactive (Bidirectional Mode)	LSB First	Soft Reset	Long Instruction	Not Used				10	
01 to 33		Not Used									
	FINE DELAY ADJUST										Fine Delays Bypassed
34	Delay Bypass 4	Not Used							Bypass	01	Bypass Delay
35	Delay Full-Scale 4	Not Used		Ramp Capacitor <5:3>			Ramp Current <2:0>			00	Max. Delay Full-Scale
36	Delay Fine Adjust 4	Not Used		5-Bit Fine Delay <5:1>					Not Used	00	Min. Delay Value
37, 38, 39, 3A, 3B, 3C		Not Used									

Figure 3-1 Analog Devices AD9512 Configuration Registers

The chip_function variable is a two dimensional array that stores the address and configuration value for every internal register of the device. There are a total of twenty-two registers in the AD9512 example, but the addresses are not contiguous. Referring to Figure 3-1, notice that addresses 0x01 through 0x33 are not used. This address range is skipped in the chip_function array, so index #1 in the first dimension of the array is assigned to address 0x34 in the AD9512 register map.

The SigStream application code will include a default device configuration structure for every model number available in a product family, but only one will be used to configure the hardware based on the model number returned by a query to the feature identification register.

4.0 Operating Configuration

Default variable settings are assigned to the operating configuration structures contained in the default_operating_config.c source file. There are separate default values assigned to receiver (RX) and transmitter (TX) channels, but both use the same s_Channel structure type. Some products are only equipped with one type of channel, so the default values assigned to the other type may be ignored. The SigStream application will query a feature identification register in the Red Rapids device to determine which channel types are available.

Although there may be multiple RX or TX channels in a device, there is only one set of default settings defined for each channel type. This means that all channels of the same type (RX or TX) will be configured identically for demonstration purposes. This is not a limitation of the hardware or software, each channel can be configured with unique settings.

4.1 Channel Settings

```
typedef struct
{
   s_SwitchSelect SwitchSelect;
   s_PayloadFormat PayloadFormat;
   s_DatapathControl DatapathControl;
   s_Synchronizer Synchronizer;
   s_TODTriggers TODTriggers;
   s_EventDuration EventDuration;
   s_IFdpktFormat IFdpktFormat;
   s_DMABuffer DMABuffer;
} s_Channel;
```

Each member of the channel structure is another structure that defines a specific channel characteristic. The channel structure is identical for RX or TX channels, although some variables may have slightly different meaning.

The RXChannel[ ] array declared in main( ) stores up to eight unique sets of RX channel configuration settings using the s_Channel structure. The TXChannel[ ] array provides the equivalent storage for up to eight TX channels. Not all products will be equipped with eight RX inputs and eight TX outputs. This is simply the maximum number supported by the SigStream application code.

The following sections explain the function of each channel structure member and the differences in how they are applied to an RX channel versus a TX channel.

4.2 Switch Settings

```
typedef struct
{
   unsigned long long ChannelSelect;
   unsigned long long Amplitude;
} s_SwitchSelect;
```

Any RX channel can be connected to any RX input (ADC) and any TX output (DAC) can be connected to any TX channel. There are also special settings available to assist debug and hardware initialization. For demonstration purposes, there is one RX channel connected to each RX input and one TX channel connected to each TX output in the SigStream application.

The ChannelSelect variable determines the mapping between ADC/DAC hardware resources and the available RX/TX channels. The setting has different meaning when applied within the context of an RX channel versus a TX channel.

All of the valid RX ChannelSelect settings are shown in Figure 4-1. The function of the switch is to select which source should supply data to a particular RX channel. Not all products will be equipped with the eight RX inputs shown in the drawing. The number of available RX channels will always match the number of RX inputs on the product.

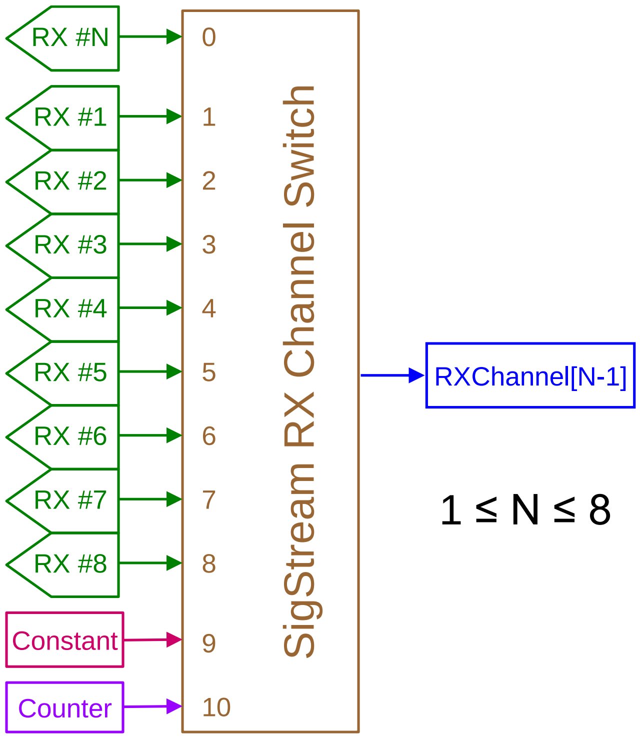


Figure 4-1 RX Switch Select Settings

The zero setting is the default used to demonstrate the SigStream application. This value connects the RX #N input to RXChannel[N-1], where N is any integer between one and the total number of RX channels available on the product. For example, RX #1 connects to RXChannel[0] and RX #2 connects to RXChannel[1].

Setting the ChannelSelect variable between one and eight will connect any RX channel to a specific RX input. For example, setting ChannelSelect to one in both RXChannel[0] and RXChannel[1] will route the RX #1 input to both RX channels. This will leave one input stranded since the number of available RX channels matches the number of inputs.

There are two ChannelSelect settings that will produce a defined input pattern for test and debug purposes. Setting the ChannelSelect variable to nine will connect the RX channel to the constant value set by the Amplitude variable. The ten setting will supply a binary count to the channel.

All of the valid TX ChannelSelect settings are shown in Figure 4-2. The function of the switch is to select which source should supply data to a particular TX output. Not all products will be equipped with the eight TX channels shown in the drawing. The number of available TX channels will always match the number of TX outputs on the product. If the product also contains RX channels, TX channel numbering will begin after the last RX channel. For example, a board with two RX channels will begin with TX #3.

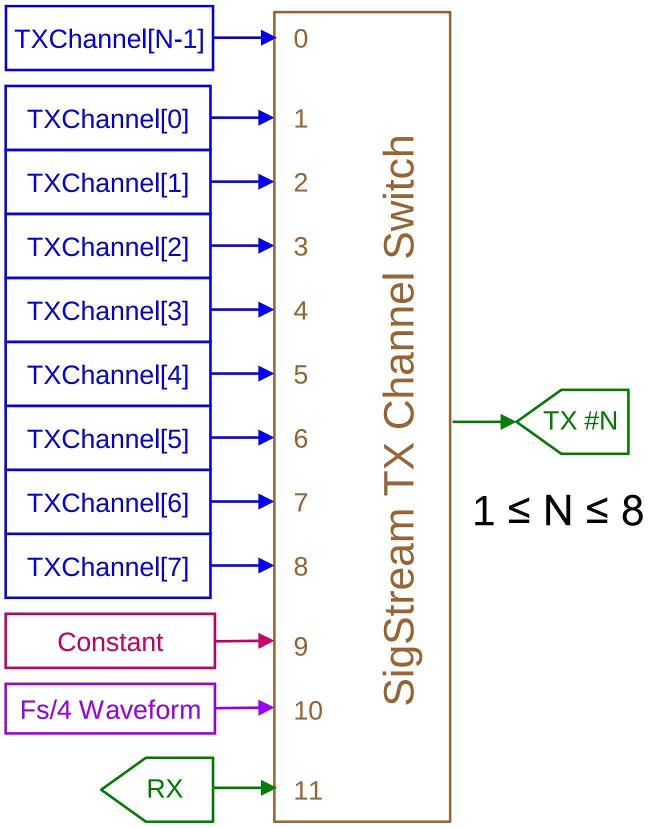


Figure 4-2 TX Switch Select Settings

The zero setting is the default used to demonstrate the SigStream application. This value connects TXChannel[N-1] to the TX #N output. The range of N depends on whether RX channels are also available on the product. If there are no RX channels, N is any integer between one and the total number of TX channels available. For example, TXChannel[0] connects to TX #1 and TXChannel[1] connects to TX #2. If RX channels are present, N is any integer between the total number of RX channels plus one and the total number of RX channels plus the total number of TX channels. So a product with two RX channels would connect TXChannel[2] to TX #3 and TXChannel[3] to TX #4. Any configuration settings assigned to TXChannel[0] and TXChannel[1] would be ignored since they are not physically present in the product.

Setting the ChannelSelect variable between one and eight will connect any TX output to a specific TX channel. For example, setting ChannelSelect to zero in both TXChannel[0] and TXChannel[1] will route TXChannel[0] to both the TX #1 and TX #2 outputs. The remaining TXChannel[1] settings will be ignored since the TX #1 output will be supplied from the TXChannel[0] datapath.

There are two ChannelSelect settings that will produce a defined output pattern for test and debug purposes. Setting the ChannelSelect variable to nine will connect the TX output to the constant value set by the Amplitude variable. The ten setting will supply a sinusoid waveform of frequency Fs/4 to the TX output.

Setting the ChannelSelect variable to eleven is only valid on products that have both RX and TX channels. This setting will route signal data directly from each RX input to each TX output, pairing the channels in sequential order. For example, a product with two RX channels and two TX channels will route RX #1 to TX #3 and RX #2 to TX #4. The pairing continues until either channel type is exhausted.

4.3 Payload Format Settings

```
typedef struct
{
   unsigned long long PayloadPacking;
   unsigned long long Endianness;
   unsigned long long DataItemSize;
   unsigned long long EventTagOrder;
   unsigned long long EventTagMask;
   unsigned long long EventTagIndex;
   unsigned long long ItemPackingFieldSize;
   unsigned long long ChannelTag;
} s_PayloadFormat;
```

The VRT Data Payload consists of a virtual container called the Item Packing Field as described in Section 2.4. The contents of the Item Packing Field are determined by the settings in the s_PayloadFormat structure. The VRT standard limits the size of an Item Packing Field to sixty-four bits, but the SigStream hardware imposes a further restriction of thirty-two bits since that is twice the size of the largest Data Item produced by a Red Rapids products.

The PayloadPacking variable determines how the Item Packing Fields are arranged within the 32-bit payload words. Setting the variable to zero results in processing efficient organization (Figure 2-5) and setting the variable to one results in link efficient organization (Figure 2-6).

The Endianness variable selects big endian or little endian data order, but this feature is not yet supported.

The DataItemSize variable sets the number of bits allocated to the Data Item that appears in each Item Packing Field. The default value matches the number of bits produced by the ADC in an RX channel or the number of bits consumed by the DAC in a TX channel. If the Data Item is set to a number less than the default, then the values produced by an RX channel will be rounded to the requested number of bits and values consumed by a TX channel will be sign extended to match the DAC input. If the Data Item is set to a number greater than the default, then values produced by an RX channel will be sign extended to the requested number of bits and values consumed by a TX channel will be rounded to match the DAC input. When a Data Item is rounded, fractional values less than 0.5 will be rounded toward zero to the nearest integer and fraction values greater than or equal to 0.5 will be rounded to the nearest integer away from zero.

Optional Event Tags can be included in the Item Packing Field. There are two Event Tags produced by Red Rapids hardware; ADC over-range and channel synchronization. The ADC over-range tag is set to one when a Data Item is produced while the ADC is driven beyond the specified analog input range. The channel synchronization tag is set to one when a sync strobe is issued to initialize elements of the datapath.

The EventTagOrder variable is used to select the position of the tags in the Event Tags field. When set to zero, the channel synchronization tag is located in the LSB of the 2-bit field and the ADC over-range tag is located in the MSB. Setting the value to one will swap the positions.

The EventTagMask will suppress specific tags based on the settings listed in Table 4-1.

Table 4-1 Event Tag Mask Settings

Value	Result
0	Suppress both tags.
1	Suppress the ADC over-range tag.
2	Suppress the channel synchronization tag.
3	Both tags are active.

Suppressing a tag does not remove it from the Item Packing List, it will simply report zero at all times.

The EventTagIndex defines the location of the Event Tags field within the Item Packing Field as an offset from the LSB. The Event Tags field is located to the immediate left of the optional Channel Tag field in the Item Packing Field. Since the Channel Tag field is right-justified within the Item Packing Field, the EventTagIndex effectively defines the size of the Channel Tag. The SigStream hardware limits the EventTagIndex to any even integer between zero and fourteen, which limits the size of the optional Channel Tag to fourteen bits even though the VRT standard allows fifteen bits.

The ChannelTag variable can be any integer that fits within the Channel Tag field size as defined by the EventTagIndex.

4.4 Datapath Control Settings

```
typedef struct
{
   unsigned long long Transform[];
   unsigned long long Residue[];
   unsigned long long DataItemSize;
   unsigned long long SyncInitMask[];
   unsigned long long SyncCycleMask[];
} s_Datapath Control;
```

The Transform[ ] array activates optional signal conditioning functions that can be applied to the sample data. The array stores six variables as defined in Table 4-2.

Table 4-2 Transform Array Definition

Index	Function	Integer Range	Transform
0	Bandwidth Select	$0 \le n \le 11$	n = 0: BW = Fs/2 n > 0: BW = Fs/2^n
1	Real Resampler	$0 \le n \le 2^{16} - 1$	n = 0: Product default setting n > 0: Channel rate = Fs / n
2	Frequency Translation	$0 \le n \le 2^{22}$	Fcenter = (n * Fs) / 2^{23}
3	Phase Offset	$0 \le n \le 2^{23} - 1$	Phase = (n * π) / 2^{23}
4	Reserved		

5	Reserved		

The first index in the array, Bandwidth Select, selects an optional digital filter and resampler at the output of the ADC in RX channels or the input to the DAC in TX channels. Setting this variable to zero bypasses all filters and streams real samples through the datapath. The data flows at a rate equal to the ADC or DAC sample clock (Fs) and the signal bandwidth is defined by the Nyquist theorem (Fs/2).

Setting Bandwidth Select to an integer greater than zero will stream data in complex (I/Q) format instead of real samples. The signal bandwidth is set to Fs/2^n and the complex pairs flow at a rate of Fs/2^n. For example, setting the variable to one will still pass the maximum available Nyquist bandwidth (Fs/2) but the complex samples will flow at half the rate of real samples carrying the same bandwidth.

The second index in the array, Real Resampler, will discard samples out of the ADC in RX channels or replicate samples into the DAC of TX channels. The variable simply defines the ratio of samples to discard or replicate. There is no filtering applied to this operation, so the signal bandwidth of the channel is effective reduced. For example, setting the variable to two will produce a result equivalent to the ADC or DAC operating at half the actual sample rate. The Real Resampler can only be used when Bandwidth Select is set to zero.

The third index in the array, Frequency Translation, programs a complex mixer that will down convert an RX spectrum or up convert a TX spectrum. This variable will only be recognized if Bandwidth Select is set to a non-zero value. The translation operation will effectively map a center frequency of (n * Fs) / 2^{23} to zero in the complex baseband spectrum.

The fourth index in the array, Phase Offset, controls the starting phase of the digital local oscillator connected to the complex mixer. Each count adds a phase increment offset of (n * π) / 2^{23} from zero.

Figure 4-3 illustrates the primary components involved in the flow of data from source to destination through a TX or RX channel. It is important to realize that the ADC or DAC is always producing or consuming sample data, there is no way to turn them off. Instead, the flow control function effectively isolates the ADC or DAC from the datapath when the channel is inactive. It is possible to strand samples already in the pipeline when the flow of data is suspended.

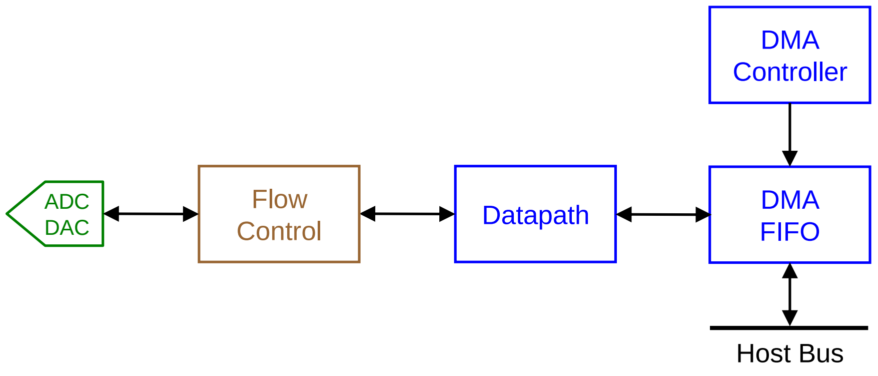


Figure 4-3 TX/RX Data Flow

The Residue[ ] array determines the disposition of data remaining in the pipeline when a channel is stopped or a cycle completes (CycleCount ≠ 0). It also determines the state of the DMA controller when the channel is started or each new cycle begins. The array stores two variables as listed in Table 4-3.

Table 4-3 Residue Array Definition

Index	Function	Integer Range
0	Channel Stop	$0 \leq n \leq 7$
1	Channel Cycle	$0 \leq n \leq 7$

The settings listed in Table 4-4 can be applied to both the Channel Stop and Channel Cycle variables. Each value represents a unique combination of actions that are applied to the datapath registers, DMA FIFO, and DMA controller.

Table 4-4 Residue Variable Settings

Value	Datapath	DMA FIFO	DMA Controller
0	Clear	Clear	Reset
1	Clear	Clear	Hold
2	Hold	Hold	Reset
3	Hold	Hold	Hold
4	Flush	Flush	Reset
5	Flush	Flush	Hold
6	Flush	Hold	Reset
7	Flush	Hold	Hold

There are three actions that can be applied to residue. First, the pipeline can be cleared to remove any remaining sample data. This action ensures that the pipeline will be primed with new samples the next time the channel is started. Notice that the datapath and FIFO are always cleared together. The second option is to simply hold the data samples in place. As the name implies, the residue will remain in the pipeline and will be processed ahead of fresh data when the channel is restarted. Finally, the pipeline can be flushed. Residue will be forced to a destination instead of pushed along by new samples entering the pipeline. The details of how each setting will affect operation depends on whether the channel is RX or TX.

Samples cannot be stranded in an RX datapath, data will flow to the FIFO even after the channel is shut off. However if the data is packetized, there is always a possibility that the channel will stop on an incomplete packet boundary. The datapath residue setting determines how an incomplete packet is handled. Clearing the datapath will reset the packetizer and eliminate any partial packet already stored in the FIFO. This may result in a partial packet stored in the host DMA buffer. Holding the datapath will allow the packet to be completed with data produced when the channel restarts. This will obviously result in a waveform discontinuity within the packet. Flushing the datapath will force the packet to complete by simply replicating the final payload data. The completed packet will enter the FIFO where it can either be held or flushed along with other residue. Flushing the FIFO will append a repeating pattern of data to the end of the last DMA burst after the final sample or packet has exited the FIFO.

Stopping a TX channel halts the flow of data from the FIFO to the DAC, potentially stranding samples anywhere in between. Clearing the datapath and FIFO will remove all residue and reset the packet stripper if it is enabled. Holding the datapath and FIFO will allow the residue to eventually reach the DAC when the channel is restarted. It will be followed by new data once the residue has been depleted. Flushing both the datapath and FIFO will force the entire pipeline to empty. The packet stripper will be reset when the pipeline flush is complete, even if the last sample was part of a packet fragment. Flushing the datapath while holding the FIFO will force any packet fragment to complete. Only enough data from the FIFO will be removed to complete the current packet. If the remainder of the packet is not available from the FIFO, the operation will behave as though both the datapath and FIFO were flushed.

The DMA controller continuously cycles through DMA buffer pages as data is transferred. Each page consists of a programmed number of bursts. The state of the controller can be set to hold the current page and burst pointer when a channel is stopped or reset both to zero. Reference the DMA on Demand Operating Guide for further details.

The SyncInitMask[ ] array determines which elements of the datapath will respond to the synchronization strobe immediately preceding the first start event. The SyncCycleMask[ ] array performs the same function with respect to the synchronization strobe immediately preceding subsequent start events (CycleCount ≠ 0). The arrays stores two variables as listed in Table 4-5.

Table 4-5 SyncInitMask Array Definition

Index	RX Function	TX Function	Integer Range
0	Binary Data Counter	Reserved	0 (no sync) or 1 (sync)
1	Resampler or DDC	Resampler or DUC	0 (no sync) or 1 (sync)
2	Reserved	Reserved	
3	Reserved	Reserved	

A binary counter can be selected as the input to an RX channel in place of ADC sample data as described in section 4.3. This feature is provided as a debug function to analyze channel behavior with a known input pattern. The first index of the two synchronization arrays is used to select when this counter is reset to zero.

The second index of each array determines when the resampler or up/down converter of any channel will be initialized. The resampler either discards or replicates samples depending on whether it is associated with an RX or TX channel, respectively. Initializing the resampler establishes a starting point for this process. For example, the RX resampler will pass the sample aligned with the synchronization strobe and start the discard process with the following sample. The TX resampler will behave the same, passing the sample coincident to the strobe and duplicating it on the next clock. The digital up/down converter behaves much like the resampler in each channel except that it processes complex pairs instead of real samples. It also has a mixer that is reset to the programmed starting phase when the synchronization strobe is applied.

4.5 Synchronizer

```
typedef struct
{
   unsigned long long StartEvent;
```

```
    unsigned long long StartOffset;
    unsigned long long StopEvent;
    unsigned long long StopOffset;
    unsigned long long CycleSelect;
    unsigned long long CycleStop;
} s_Synchronizer;
```

The synchronizer supervises all activity associated with data movement through a channel. It manages flow control and ensures that DMA transfers occur only when expected by the host. The s_Synchronizer structure defines the specific actions that will initiate and terminate channel processing.

The StartEvent variable selects which action will begin the flow of data through a channel. First the channel is enabled using the FlowControl() API function. This places the synchronizier in an armed state waiting for the selected start event. Once the event occurs, data will begin to flow after the time interval specified by the StartOffset variable.

The StopEvent variable selects which action will halt the flow of data through a channel. The synchronizer begins monitoring for the StopEvent immediately following a StartEvent. Once the StopEvent event occurs, data flow will be shut off after the time interval specified by the StopOffset variable.

The StartEvent and StopEvent variables share almost identical definitions as listed in Table 4-6. Most of the events are associated with external signals supplied through the coaxial or GPIO hardware connectors. Each hardware input can be set to detect the rising or falling edge of the signal. The SoftwareTrigger() API function can also be chosen as an event. This option allows multiple channels to respond synchronously to a single software command instead of commanding each channel individually.

Table 4-6 StartEvent and StopEvent Variable Settings

Value	Start Event	Stop Event
0	Immediate	Software Command
1	Software Trigger	Software Trigger
2	Coax Trigger Rising	Coax Trigger Rising
3	Coax Trigger Falling	Coax Trigger Falling
4	Reserved	Sample Count
5	Time of Day	Time of Day
6	GPIO Trigger Rising	GPIO Trigger Rising
7	GPIO Trigger Falling	GPIO Trigger Falling
8	GPIO #1 Rising	GPIO #1 Rising
9	GPIO #1 Falling	GPIO #1 Falling
10	GPIO #2 Rising	GPIO #2 Rising
11	GPIO #2 Falling	GPIO #2 Falling
12	GPIO #3 Rising	GPIO #3 Rising
13	GPIO #3 Falling	GPIO #3 Falling
14	GPIO #4 Rising	GPIO #4 Rising
15	GPIO #4 Falling	GPIO #4 Falling

The zero setting shown in Table 4-6 produces a different result when it is applied to a StartEvent versus a StopEvent. Data will begin flowing immediately after the channel is enabled if the StartEvent is set to zero. The same value will cause the channel to wait for software to issue a command through the FlowControl() API function when it is assigned to a StopEvent. Setting the StopEvent variable to four will activate a sample counter that terminates data flow when the requested number of samples have transferred. It is important to note that the samples are counted at the output of the ADC in an RX channel or at the input to the DAC in a TX channel and not at the lower rates that may be selected through the Transform[ ] array. The time of day may also be used as a StartEvent or StopEvent. See Section 3.1 for more information about the time of day clock operation.

The StartOffset and StopOffset variables perform identically. They both measure time in increments of sample clock periods to establish a delay interval. The range can be set from 0 to 2^{16}-1. The computed time interval will be referenced to the ADC sample clock in an RX channel and the DAC sample clock in a TX channel.

Each channel can be programmed to cycle on and off automatically. The CycleSelect variable selects what criteria, if any, should mark the beginning of each new cycle. The three available options are listed in Table 4-7. The zero setting disables the channel immediately after the first StopEvent is detected, resulting in a single cycle. Setting the variable to one causes each new cycle to begin with whatever action is selected by the StartEvent variable. The final option, setting two, commands the synchronizer to start each new cycle on a periodic frame interval.

Table 4-7 CycleSelect Variable Settings

Value	Cycle Start Event
0	None / Single Cycle
1	StartEvent
2	Frame

The CycleStop variable selects the criteria that will terminate cycling when CycleSelect is set to a non-zero value. The three available options are listed in Table 4-8. The zero setting instructs the synchronizer to continue cycling until software issues a command through the FlowControl() API function. Setting the variable to one activates a counter that terminates cycling when the requested number of cycles have completed. The final option, setting two, terminates cycling based on a time of day. See Section 3.1 for more information about the time of day clock operation.

Table 4-8 CycleStop Variable Settings

Value	Cycle Start Event
0	Software Command
1	Cycle Count
2	Time of Day

4.6 Time of Day Triggers

```
typedef struct
{
   unsigned long long IntegerSecondsStart;
   unsigned long long FractionalSecondsStart;
   unsigned long long IntegerSecondsStop;
   unsigned long long FractionalSecondsStop;
} s_TODTriggers;
```

The s_TODTriggers structure stores the time of day start value and stop value that can trigger events in the synchronizer as described in Section 4.5. Each value consists of an integer seconds component and fractional seconds component. All of the variables will accept an input range from 0 to 2^{32}-1. See Section 3.1 for more information about the time of day clock operation.

4.7 Event Duration

```
typedef struct
{
    unsigned long long SampleCount;
    unsigned long long BurstCount;
    unsigned long long CycleCount;
    unsigned long long FrameCount;
} s_EventDuration;
```

The s_EventDuration structure stores count values that are used to time various events in the synchronizer as described in Section 4.5. The SampleCount and BurstCount variables are associated with two of the StopEvent options listed in Table 4-6. The CycleCount variable is associated with one of the CycleStop options listed in Table 4-8. The FrameCount variable is used to define the length of a periodic frame that can be used as one of the CycleSelect options listed in Table 4-7. All of the variables will accept an input range from 0 to 2^{32}-1.

4.8 IF Data Packet Format

```
typedef struct
{
    unsigned long long PacketSize;
    unsigned long long TSFSelect;
    unsigned long long TSISelect;
    unsigned long long TrailerSelect;
    unsigned long long ClassIDSelect;
    unsigned long long ClassID;
    unsigned long long StreamIDSelect;
    unsigned long long StreamID;
    unsigned long long ContextPktCntEnable;
    unsigned long long IndicatorEnables;
    unsigned long long SpectralInversion;
    unsigned long long UserIndicators[];
} s_IFdpktFormat;
```

The s_IFdpktFormat structure selects optional information to be included in the VRT IF Data Packet.

The PacketSize variable conveys the total number of 32-bit words necessary to form the complete IF Data Packet, not just the number of payload words. The value must include the header, payload, and all optional fields. A value of zero will result in a continuous stream of payload fields with no actual packet information. This option is not allowed if payload packing is enabled (PayloadPacking = 1). Setting the variable to one is impractical since the packet could only consist of only a header and no payload data. The maximum PacketSize value is 2^{16}-1, which is a limit imposed by the number of bits allocated to the header field as discussed in Section 2.1.

The TSFSelect and TSISelect variables determine whether the fractional and integer seconds components of the timestamp are included in the packet, respectively. Setting the variable to zero suppresses the designated field while setting to one inserts the field. These values are not equivalent to the code displayed in the TSF and TSI header fields described in Section 2.1.

The TrailerSelect variable determines whether a trailer is included in the packet. Setting the variable to zero suppresses the trailer while setting to one inserts the field. All of the bits contained in the trailer are assigned to optional fields as discussed in Section 2.5. The hardware does not generate context packets, so the ContextPktCntEnable variable is ignored and the lower eight bits of the trailer are always zero. The remaining bits are assigned to the State and Event Indicators listed in Table 2-2. There are two indicators produced by Red Rapids hardware; ADC over-range (bit 13) and channel synchronization (bit 11). The remaining User Defined Indicators (bits 10:8) can be set to zero or one through the UserIndicators[] array. The Spectral Inversion Indicator (bit 14) can be set to zero or one though the SpectralInversion variable. The IndicatorEnables variable determines which of the indicator bits should be masked as explained in Section 2.5.

The ClassIDSelect variable determines whether the value stored in the ClassID variable is included in the packet. Setting the variable to zero suppresses the Class Identifier while setting to one inserts the field. Refer to Section 2.2 for additional information about the Class Identfier.

The StreamIDSelect variable determines whether the value stored in the StreamID variable is included in the packet. Setting the variable to zero suppresses the Stream Identifier while setting to one inserts the field. Refer to Section 2.2 for additional information about the Stream ID.

4.9 DMA Buffer

```
typedef struct
{
   unsigned long long BurstSizeB;
   unsigned long long BurstCount;
   unsigned long long PageCount;
   unsigned long long PagesPerMark;
} s_DMABuffer;
```

The s_DMABuffer structure sets the operating parameters for DMA transfers. Refer to the DMA on Demand Operating Guide (REF-004-000-Rxx) for a detailed description of each variable.

The BurstSizeB variable sets the size of each DMA burst in bytes.

The BurstCount variable sets the number of bursts in a page.

The PageCount variable sets the number of pages available in the buffer.

The PagesPerMark variable sets the number of pages between marker flags.