Wavefront Modulator

Operating Guide

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

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

All trademark and registered trademarks are the property of their respective owners.

Copyright © 2008, Red Rapids, Inc. All rights reserved.

Table of Contents

List of Figures

List of Tables

1.0 Introduction

1.1 Contents and Structure

This manual describes the features and application interface for the Wavefront Quadrature Modulator product. The scope includes functional details, software API, example code, and configuration registers. This publication supplements the supporting documents listed in section 1.2 that focus on the hardware characteristics and product installation.

Section	Description
Section 1	Introductory information about the manual.
Section 2	Overview of functions provided in the application programming interface.
Section 3	Functional description of the product and the available operating modes.
Section 4	Summary of the memory map and detailed register descriptions.

The latest product documentation and software is available for download from the Red Rapids web site (www.redrapids.com) by following the Wavefront Quadrature Modulator Docs & Software link..

1.2 Supporting Documents

Author	Number	Title
Red Rapids	REF-150-000	Model 150 Wavefront Hardware Reference Manual
Red Rapids	REF-150-001	Wavefront Installation Guide
PCI SIG	PCIe Base Spec 2.0	PCI Express Base Specification Revision 2.0
PCI SIG	PCIe CEM Spec 2.0	PCI Express Card Electromechanical Specification Revision 2.0

1.3 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.

- **API** Application Program Interface

- **BAR** Base Address Register
- **HAL** Hardware Abstraction Layer
- **ISR** Interrupt Service Routine
- **PCI** Peripheral Component Interconnect
- **PCIe** PCI Express
- **SPI** Serial Peripheral Interconnect

1.4 Manual Compatibility

The applicable hardware part numbers are defined as follows:

- Model 150-XXX *Wavefront Modulator*

1.5 Revision History

Version	Date	Description
R00	11/12/2008	Initial release.

2.0 Application Program Interface

The application program interface (API) provides C functions to access all hardware resources from the host. These functions are the basis for sample programs provided by Red Rapids that can be used as a foundation for custom software development.

Features of the API include:

- Unified Windows and Linux driver structure.
- Support for up to eight cards in a single system.
- Support for DMA transfers (not used on Wavefront Products).
- Interrupt processing for error and status indicators (not used on Wavefront Products).

A single API is shared across several Red Rapids I/O adapter products. Consequently, the "adapter" naming convention is adopted throughout the Wavefront Modulator software distribution.

> ☞ DMA transfers and interrupt handling are not directly applicable to the Wavefront Modulator family. References are included due to the use of a shared API.

2.1 General API Usage

A hardware abstraction layer (HAL) link must be established prior to accessing the hardware using the API functions. The HAL link is created by successfully calling the Adapter_Open() function. The Adapter_Close() function must be called prior to exiting any program that has issued an Adapter_Open() command to de-allocate system memory assigned to the hardware. Failure to do so will result in memory leakage and eventual system crash. Be sure to include the file adapter.h in any application that calls an Adapter_ function. This file contains the prototypes for all functions in the API.

2.2 API C Library Versions

The adapterlib directory includes a C library containing the API for each supported development environment and host platform. The supported platforms and library locations are detailed below.

Development Platform	
Visual Studio 2005 (x86–32)	vs2005/win_x86-32/adapter.lib
Visual Studio 2005 (x86–64)	vs2005/win_x86-64/adapter.lib
Linux (x86-32)	linux_x86-32/libadapter.a
Linux (x86-64)	linux_x86-64/libadapter.a

2.2.1 Visual Studio 2005 (x86-32)

Add the following library to the project:

c:\<extract_path>\DSK-806-900-RXX\adapterlib\RXX\ vs2005\win_x86-32\adapter.lib

Add the following directories to the include path:

c:\<extract_path>\DSK-806-042-RXX\lib

c:\<extract_path>\DSK-806-042-RXX\memmap

c:\<extract_path>\DSK-806-900-RXX\adapterlib\RXX

Add the following preprocessor directives:

VS2005; X86; WIN32;

The project must be set to generate multithreaded code since the interrupts are handled in a separate thread.

> **!** Programs will crash if multithreaded code generation is not turned on.

2.2.2 Visual Studio 2005 (x86-64)

Add the following library to the project:

c:\<extract_path>\DSK-806-900-RXX\adapterlib\RXX\ vs2005\win_x86-64\adapter.lib

Add the following directories to the include path:

c:\<extract_path>\DSK-806-042-RXX\lib

c:\<extract_path>\DSK-806-042-RXX\memmap

c:\<extract_path>\DSK-806-900-RXX\adapterlib\RXX

Add the following preprocessor directives:

VS2005; X86; WIN32; X64;

The project must be set to generate multithreaded code since the interrupts are handled in a separate thread.

> **!** Programs will crash if multithreaded code generation is not turned on.

2.2.3 Linux (x86-32)

Link the application to the static file:

</extract_path>/DSK-806-900-RXX/adapterlib/RXX/linux_x86-32/libadapter.a

The project must be compiled with the pthread library since the interrupts run in a separate thread from the main program.

The following should be added to the include path and the symbol LINUX should be defined for any object that includes the adapter.h file:

</extract_path>/DSK-806-900-RXX/adapterlib/RXX/

</extract_path>/DSK-806-042-RXX/lib

</extract_path>/DSK-806-042-RXX/memmap

2.2.4 Linux (x86-64)

Link the application to the static file:

</extract_path>/adapterlib/RXX/linux_x86-64/libadapter.a

The project must be compiled with the pthread library since the interrupts run in a separate thread from the main program.

The following should be added to the include path and the symbols LINUX and X64 should be defined for any object that includes the adapter.h file:

</extract_path>/DSK-806-900-RXX/adapterlib/RXX/

</extract_path>/DSK-806-042-RXX/lib

</extract_path>/DSK-806-042-RXX/memmap

2.3 API Functions

The API consists of a memory map to all local registers, a handle to access the primary hardware features, and software functions to initialize the card and perform data transfers.

2.3.1 Handle

The s_Adapter handle provides the mechanism to access the hardware.

Table 2-1 s_Adapter Handle Structure Members

Member Name	Data Type	Definition
DevNum	integer	Detection order for multiple boards. Starts at zero.
Asy	char[]	Assembly number assigned to the product.
Bus	UINT32	Number of the Bus where the product is installed.
Slot	UINT32	Number of the Slot where the product is installed.
Function	UINT32	Number of the Function where the product is installed.
UnitsFound	integer	Number of all Red Rapids products installed in the system.
RevCode	UINT32	Hardware revision of the PCI bridge.
DMA []	s_DMAConfig	Structure that contains DMA configuration information.
BrgIntrMask	UINT32	Stores a copy of the interrupt mask for re-enabling interrupts after processing.
DevStatus	UINT32	Status flags for the s_Adapter handle.
ReturnStatus	UINT32	Stores function return codes.
BrgJtagProgAdr	UINT32	Stores bridge address used for JTAG programming.
FILE * debugFP	FILE *	Output location for messages in debug enabled version of the API

The API function described below is used to initialize the handle.

Prototype	UINT32 Adapter_Zero(s_Adapter *pCA);
Arguments	s_Adapter * pCA – Pointer to a s_Adapter handle.
Description	Zeros the members of the s_Adapter handle.
Return	0 if OK Non-zero if error

2.3.2 API Functions

The API includes several software functions to access the hardware features. Most of these are provided in source code on the software distribution disk discussed in the Wavefront Modulator Installation Guide. However, there are a few functions listed below that are not available in source form. These functions are included in the adapter.h header file.

Prototype	char * Adapter_GetAPIVerInfo(UINT32 *Version);
Arguments	s_Adapter * pCA – Pointer to a s_Adapter handle. UINT32 * - Returns a 32bit API revision level in the form: [31:24] Major Rev [23:16] Minor Rev [15:0] Sub Rev
Description	Returns text and numeric values providing the API revision level and target platform
Return	char * - Pointer to text string.

Prototype	void Adapter_uSleep(UINT32 uSec);
Arguments	UINT32 uSec – Number of microseconds to sleep.
Description	Causes the processor to perform a busy sleep.
Return	None.

Prototype	UINT32 Adapter_Open(s_Adapter *pCA, UINT32 BrgRevAdr);
Arguments	s_Adapter * pCA – Pointer to a s_Adapter handle. UINT32 BrgRevAdr – Address in the bridge where the revision code is stored.
Description	This function opens an instance of s_Adapter handle. It returns an unsigned integer based on the mapping status. It stores information about the mapped unit in the s_Adapter handle.
Return	0 if OK Non-zero if error

Prototype	UINT32 Adapter_Close(s_Adapter *pCA);
Arguments	s_Adapter * pCA – Pointer to a s_Adapter handle.
Description	This function closes an open s_Adapter handle instance and de-allocates any system memory allocated to the hardware.
Return	0 if OK Non-zero if error

Prototype	UINT32 Adapter_Read32(s_Adapter *pCA, BAR Bar, UINT32 Offset, UINT32 *pData); UINT32 Adapter_Read64(s_Adapter *pCA, BAR Bar, UINT32 Offset, QWORD *pData);
Arguments	s_Adapter * pCA – Pointer to a s_Adapter handle. BAR – Memory area to read from, BRIDGE or V4. UINT32 Offset – Address offset in the memory area. UINT32 * pData – Returns data for 32-bit read. QWORD * pData – Returns data for 64-bit read.
Description	This function performs a bus read to the specified BAR and offset.
Return	0 if OK Non-zero if error

Prototype	UINT32 Adapter_Write32(s_Adapter *pCA, BAR Bar, UINT32 Offset, UINT32 Data); UINT32 Adapter_Write64(s_Adapter *pCA, BAR Bar, UNIT32 Offset, QWORD Data);
Arguments	s_Adapter * pCA – Pointer to a s_Adapter handle. BAR – Memory area to read from, BRIDGE or V4. UINT32 Offset – Address offset in the memory area. UINT32 Data – 32-bit value to write to offset. QWORD Data – 64-bit value to write to offset.
Description	This function performs a bus write to the specified BAR and offset.
Return	0 if OK Non-zero if error

Prototype	UINT32 Adapter_DMABufAllocate(s_Adapter *pCA);
Arguments	s_Adapter * pCA – Pointer to a s_Adapter handle.
Description	Allocates DMA buffers specified by the DMA structure in the s_Adapter handle.
Return	0 if OK Non-zero if error

2.4 Interrupt Service Routine

The following ISR function must be included in all C programs using the API:

void Adapter_ISR(s_Adapter *pCA).

It does not need to have a body if interrupts are not used by the application.

Interrupts are not used on the Wavefront Modulator.
References are included due to the use of a shared API.

Each Red Rapids card with an open handle is associated with an interrupt handler in the API. When the API detects an interrupt, it first determines whether it originated from a Red Rapids product. The Adapter_ISR() is called only in the application that has an open handle to the hardware producing the interrupt.

3.0 Functional Description

A block diagram of the Model 150 Quadrature Modulator hardware is shown in Figure 3-1. The hardware is made up of four major functional blocks: baseband inputs, carrier LO generation, quadrature modulator/RF output chain and board controller. The primary use of the Model 150 is as a quadrature modulator where users apply analog data to the baseband inputs to modulate a carrier LO. A secondary mode of operation allows the user to bypass the quadrature modulator and transmit the I baseband input directly out the TX port. This secondary mode of operation, called direct RF mode, provides support for the transmission of user generated RF waveforms. The primary purpose for the direct RF mode is to provide frequency coverage below the lower operating frequency limit of the on-board modulator.

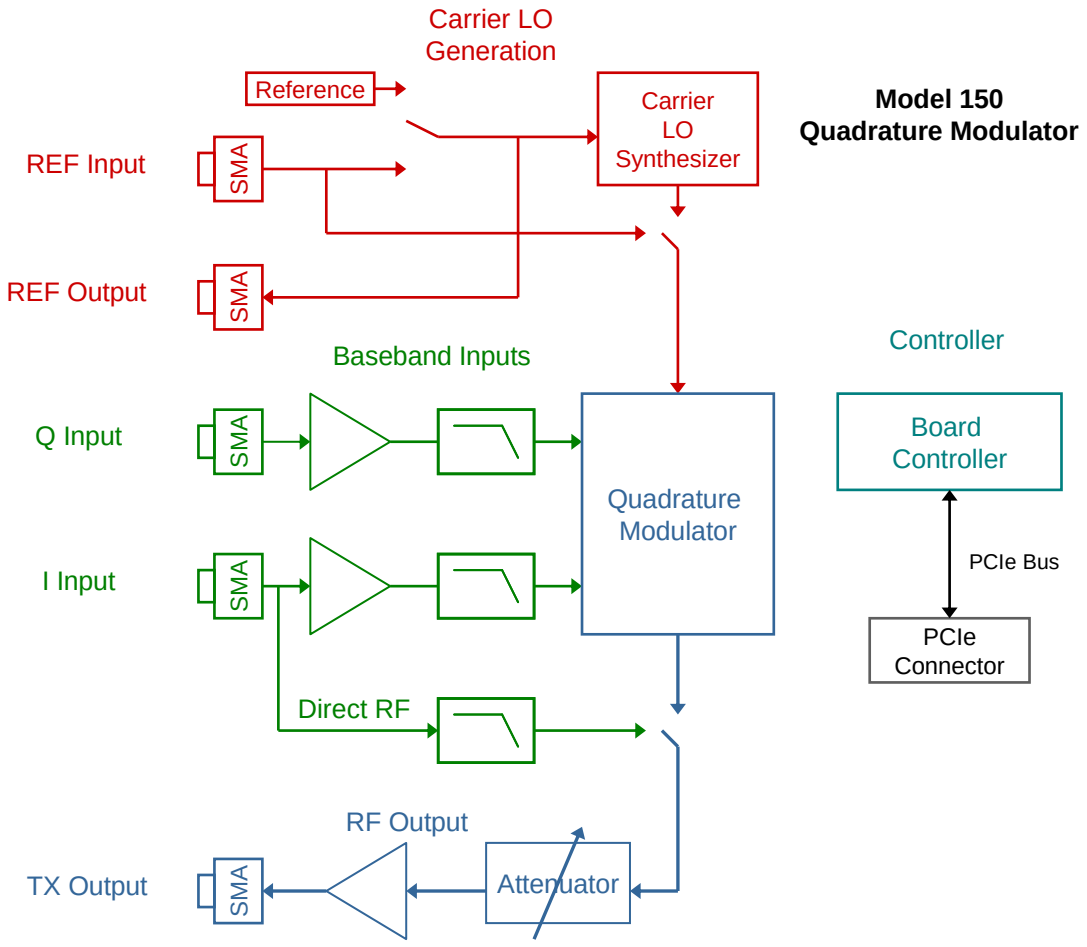

Figure 3-1 Model 150 Block Diagram

There are two analog inputs to the quadrature modulator labeled I and Q. Each dc-coupled input is amplified, offset adjusted and applied to the quadrature modulator input. The broadband inputs feature a software controlled offset adjustment for improving carrier rejection.

The Model 150 modulator receives its carrier from one of two sources; an on-board synthesizer or a user supplied external carrier. The user also has the choice of supplying their own reference source or using one of the built-in references for the on-board synthesizer.

The modulator board features a broadband RF section consisting of a digitally controlled step attenuator followed by a wideband amplifier. The attenuator provides coarse level control while the amplifier boosts the modulator output across the operating band.

The Model 150 is an industry standard PCI express card that supports the Windows and Linux operating systems. Control of the 150 is handled by simple software commands interpreted by a central board controller. The board controller converts software commands issued over the express bus into to discrete control signals to various board functions.

The following paragraphs describe the hardware functions of the Model 150 in detail.

3.1 Operating Modes

Wavefront is designed to operate as a quadrature modulator or a pass through RF channel. The quadrature modulator mode is shown in Figure 3-2. The figure depicts a Wavefront unit installed in a host system connected to a baseband generator and transmitter channel of some kind. The I and Q outputs of the baseband system are connected to the Wavefront I and Q inputs respectively while the Wavefront TX output is connected to the transmitter channel RF input. Also shown are the optional reference clock input and output connections.

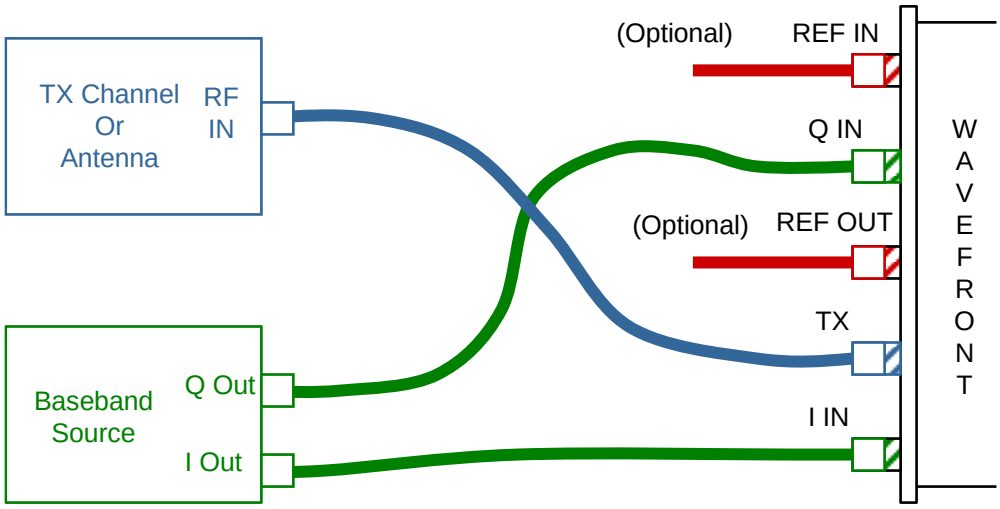

Figure 3-2 Quadrature Modulator Mode

In quadrature mode the user creates an analog signal that is applied to the Wavefront inputs. The user issues software commands to set the transmit port output level and carrier frequency.

In direct RF mode the user bypasses the quadrature modulator which results in the TX output port being connected directly to the I input port inside the Wavefront unit. A diagram of the direct RF mode is shown in Figure 3-3. The Q port input is ignored and may remain connected to the baseband source. There is no upconversion in this operating mode. The I input is passed through a filter-amplifier-attenuator chain and out the TX port. The direct RF mode is intended to extend the low end operating frequency range of the system by enabling modulated signals to be directly output from the baseband source without the need to change cabling. The idea is that carrier frequencies below the operating range of the Wavefront synthesizer can be created and transmitted directly by the baseband source.

The mode is activated by setting the directrf mode flag true via software command. User also has command over the attenuator level. All other control parameters are ignored. See the Wavefront Quadrature Modulator hardware reference manual for more information.

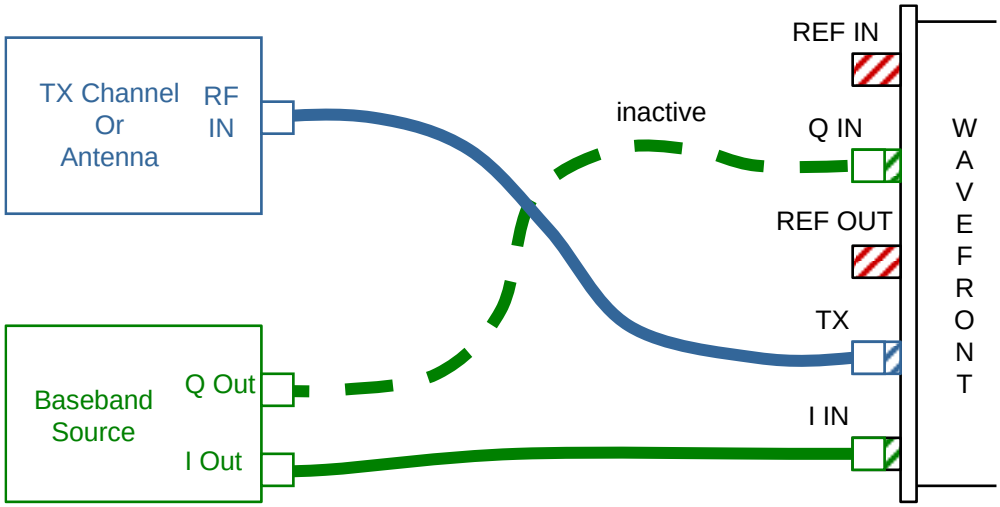

Figure 3-3 Direct RF Mode

3.2 Software Control

The Wavefront Quadrature modulator is an industry standard PCI express card that is controlled via software command over the PCI express bus. The Wavefront controller provides a gateway from the express bus interface to the command/status functions of the board hardware. The controller resides on the PCI express bus as a single lane endpoint and provides connectivity to the hardware as shown in Figure 3-4.

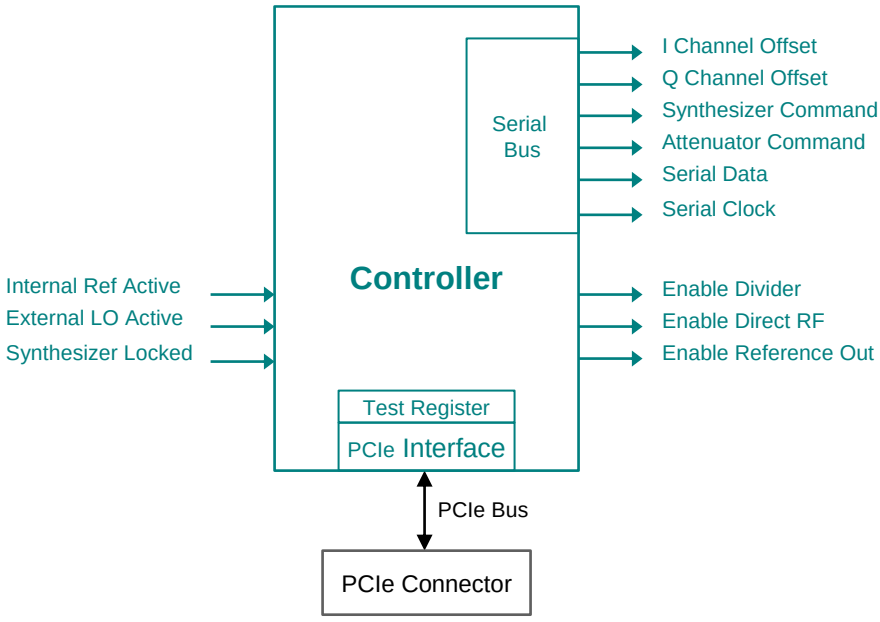

Figure 3-4 Wavefront PCI Express Controller

The controller receives user command/status requests over the PCI express bus and routes them to the serial bus interface or converts them into discrete signal

command/status requests. Red Rapids provides an API that allows users to communicate via software with Wavefront using simple read and write commands.

There is a single software application supplied for the Wavefront Modulator that demonstrates all of the control registers required to operate the hardware. The main program is called Wavefront.

The main.c source code can be used as a template to quickly integrate the product into custom user applications. The code is commented to explain what actions are necessary to enable specific hardware features.

The wavefront.exe executable offers a quick demonstration of the hardware features as a standalone application. This utility can be used to run a quick diagnostic as described in the Installation Guide.

There are several switches available to exercise different features of the Wavefront Modulator. A complete list of options can be obtained by executing the compiled application code from a command line as follows:

wavefront.exe -help

A list of commands used with the Wavefront Modulator software utility can be found in Table 3-1. These commands comprise the complete command set for the Wavefront unit. Command functions are explained in detail in the hardware reference manual. Operation of the software utility is described in the installation guide.

Table 3-1 Wavefront Software Utility Commands

Command Line Option (switch)	Description
-help	Help option, lists command line menu.
-stat	Reads status register and displays contents.
-v <n>	Verbose output, n = 1:5 for debug. Default is off.
-asy <n>	Assembly number, defaults to 150.
-dev <n>	PCI device number, default is 0. Device number required when multiple units installed using Jungo driver.
-div_sel <n>	Carrier divider mode enable. 0 = disable, 1 =enable. Default is 0.
-direct_rf <n>	Direct RF mode enable. 0 = disable, 1 =enable. Default is 0.
-ref_out <n>	Reference clock output enable. 0 = disable, 1 =enable. Default is 0.
-attn <n>	Attenuator value. n = 0:31.5 in 0.5 dB increments. Default is 0.
-synthR <n>	Synthesizer reference clock divider value. n = 10. Default is 10.
-synthN <n>	Synthesizer carrier divider value. n = 250:4000 in 1 MHz steps. Default is 1500.
-idacpos <n>	I Channel positive dc offset value n = 0x00:0x3FF. Default = 0.
-idacneg <n>	I Channel negative dc offset value n = 0x00:0x3FF. Default = 0.
-qdacpos <n>	Q Channel positive dc offset value n = 0x00:0x3FF. Default = 0.
-qdacneg <n>	Q Channel negative dc offset value n = 0x00:0x3FF. Default = 0.

4.0 Memory Map

The Wavefront Modulator hardware includes configuration and status registers that are memory mapped to the address space of the host computer. The memory map of each card is referenced to base address registers (BARs) that separate the PCIe bridge control functions from the application controls. The memory allocation for the Wavefront Modulator is shown in Table 4-1.

Table 4-1 Wavefront Modulator Memory Allocation

Name	Type	Byte Allocation
BAR0	Memory	512
BAR1	I/O	256
BAR2	Memory	4k
BAR3	Memory	1M

BAR0 and BAR1 are assigned to the command/status registers that control the various functions on the PCI Express Bridge. These registers are initialized at start-up or when the card is opened by an application.

BAR2 is assigned to the command and status registers that control various Wavefront Modulator operating functions. Many of these registers will be updated or monitored during normal operation.

BAR3 is reserved for future use.

The memory map is not contiguous, any memory location not called out in the memory map should be considered reserved and should not be accessed. Writing to locations not called out in the memory map may cause erroneous operation.

The following convention is used to describe valid register operations:

(W) Write

(R) Read

(cl) Read clears contents

BAR0 PCI Bridge Memory Map		
Byte Address Offset	**Access**	**Register (Group) Name**
0x000 – 0x1FF	Reserved	PCI Express Bridge Registers (Memory)

BAR1 PCI Bridge Memory Map		
Byte Address Offset	**Access**	**Register (Group) Name**
0x000 – 0x0FF	Reserved	PCI Express Bridge Registers (I/O)

BAR2 Wavefront Command/Status		
Byte Address Offset	**Access**	**Register (Group) Name**
0x0000	User	Wavefront Modulator Serial Data Bus Control Register
0x0004	User	Wavefront Modulator Firmware Revision Register
0x0014	User	Wavefront Modulator Command/Status Register

Address	Register Name			
0x00	Wavefront Modulator Serial Data Bus Control Register			
Bit(s)	**R/W/cl**	**POR**	**Function**	**Description**
31:7			Reserved	
6	W	0x0	Attenuator Enable	Attenuator SPI bus enable (enabled = 1).
5	W	0x0	IDAC Enable	I port dc offset DAC SPI bus enable (enabled = 0).
4	W	0x0	QDAC Enable	Q port dc offset DAC SPI bus enable (enabled = 0).
3	W	0x0	Synth Enable	Carrier synthesizer SPI bus enable (enabled = 1).
2	W	0x0	Serial Clock	SPI bus clock.
1	W	0x0	Serial Data	SPI bus data.
0			Reserved	

Address	Register Name			
0x04	Wavefront Modulator Firmware Revision Register			
Bit(s)	**R/W/cl**	**POR**	**Function**	**Description**
31:3			Reserved	
2:0	R		Firmware Rev	Firmware revision number

Address	Register Name			
0x14	Wavefront Modulator Command/Status Register			
Bit(s)	**R/W/cl**	**POR**	**Function**	**Description**
31:6			Reserved	
5	R	0x0	Internal REF Active	Internal reference clock active (active = 1).
4	R	0x0	External LO Active	External carrier LO active (active = 1).
3	R	0x0	Internal LO Active	Internal carrier LO active (active = 1).
2	W	0x0	Direct RF	Direct RF mode enable (enable =1).
1	W	0x0	Ref Out Enable	Enable reference clock output (enable = 1).
0	W	0x0	Divider Enable	Enable carrier divider mode (enable = 1).

BAR3 Wavefront Modulator Memory		
Byte Address Offset	**Access**	**Register (Group) Name**
0x00000 – 0xFFFFF	Reserved	Wavefront Memory (Reserved).