Universal Driver Software User Manual

RUBY-MM-1616-A/AP

PC/104 / PC/104-Plus I/O Modules with 4, 8 or 16 16-Bit Analog Outputs

For Version 7.0.0 and later

Revision A.0 May 2015

Revision	Date	Comment
A.0	5/13/2015	Initial release

FOR TECHNICAL SUPPORT PLEASE CONTACT:

support@diamondsystems.com

Diamond Systems Corporation
555 Ellis Street
Mountain View, CA 94043 USA
Tel 1-650-810-2500
Fax 1-650-810-2525
www.diamondsystems.com

CONTENTS

1. INTRODUCTION

This user manual contains all essential information about the Universal Driver 7.0 RMM-1616 demo applications, programming guidelines and usage instructions. This manual also includes the Universal Driver API descriptions with usage examples.

2. HARDWARE OVERVIEW

2.1 Description

The RMM-1616A/AP is a family of PC/104 and PC/104-*Plus* I/O modules featuring 4, 8 or 16 16-bit analog voltage and current outputs, and 48 digital I/O lines. A 50-pin connector provides access to the 16 analog outputs, and another 50-pin connector provides access to the 48 digital I/O lines. The board operates over the extended temperature range of -40°C to +85°C and is supported by Diamond Systems' Universal Driver software.

2.2 Specifications

- PC/104 (ISA) or PC/104-*Plus* (ISA + PCI) models
- 4, 8 or 16 analog outputs with 16-bit D/A resolution
- Programmable voltage output ranges: 0-5V, 0-10V, ±5V, ±10V, 0-20mA, 4-20mA, 0-24mA
- Independent output range for each channel
- Waveform generator on up to 16 channels
- Simultaneous update of any combination of channels
- Multi-channel simultaneous waveform output capability with up to 100KHz waveform update rate
- Output current limit +/-5mA per channel in voltage mode
- Auto-calibration of D/A circuits using the internal offset and gain registers for each channel
- 40 digital I/O lines with byte-wide and 8 with bit-wide programmable direction
- 2 32-bit programmable counter / timers
- 4 24-bit pulse width modulators (PWMs)

3. GENERAL PROGRAMMING GUIDELINES

3.1 Initialization and Exit Function Calls

All the demo applications begin with the following functions and should be called in sequence to initialize the Universal Driver and the board. These functions should be called prior to calling any other RMM-1616 board specific functions.

1. dscInit ( ), this function initializes the Universal Driver
2. RMM1616InitBoard(), this function initializes the RMM-1616 module
3. DSCGetBoardInfo(), this function collects the board information from the Universal Driver and returns the boardinfo structure to be used in the board specific functions

At the termination of any demo application, the user should call the dscfree() function to close the file handles which are opened by the dscInit() function.

These function calls are important in initializing and freeing resources used by the driver. Following is an example of the framework for an application using the driver:

```
#include "DSCUD_demo_def.h"
#include "rmm1616.h"

ERRPARAMS errorParams; //structure for returning error code and error string
DSCCBP dsccbp; //structure containing PCI board settings
BoardInfo *bi=NULL; //Structure containing board base address

int main()
{
        if ( (dscInit ( DSC_VERSION ) != DE_NONE) )
        {
                dscGetLastError ( &errorParams );
                printf ( "dscInit error: %s %s\n", dscGetErrorString (
                errorParams.ErrCode ), errorParams.errstring );
                return 0;
        }
        dsccbp.boardtype = DSC_ RMM1616;
        dsccbp.pci_slot  = 0;
        if(RMM1616InitBoard ( &dsccbp  ) !=DE_NONE)
        {
                dscGetLastError(&errorParams);
                printf("RMM1616InitBoard        error:        %s        %s\n",
                dscGetErrorString(errorParams.ErrCode),   errorParams.errstring
                );
                return 0;
        }
        bi = DSCGetBoardInfo ( dsccbp.boardnum );
        /* Application code goes here */

        dscFree ( );
        return 0;
}
```

In the above example, DSC_VERSION, DSC_RMM1616, and DE_NONE are macros defined in the included header file, *Universal Driver.h file.*

3.2 Error Handling

All the Universal Driver functions provide a basic error handling mechanism that stores the last reported error in the driver. If the application is not behaving properly, check for the last error by calling the function dscGetLastError(). This function takes an ERRPARAMS structure pointer as its argument.

Nearly all the available functions in the Universal Driver API return a BYTE value upon completion. This value represents an error code that will inform the user whether the function call was successful or not. The user should always check if the result returns a DE_NONE value (signifying that no errors were reported), as the following code illustrates:

```
BYTE result;
ERRPARAMS errparams;
if ((result = dscInit(DSC_VERSION)) != DE_NONE)
{
        dscGetLastError(&errparams);
        fprintf(stderr,        "dscInit        failed:        %s        (%s)\n",
        dscGetErrorString(result), errparams.errstring);
        return result;
}
```

In this code snippet, the BYTE result of executing a particular driver function (dscInit() in this case) is stored and checked against the expected return value (DE_NONE). At any point of time, if a function does not complete successfully, an error code other than DE_NONE will be generated, and the current API function will be terminated. The function dscGetErrorString() provides a description of the error that occurred.

4. UNIVERSAL DRIVER API DESCRIPTION

4.1 RMM1616DASetSettings

Function Definition

BYTE RMM1616DASetSettings (BoardInfo* bi, int Channel, int Range, int ClearEnable);

Function Description

This function is designed to work with the AD5755 / AD5755-1 D/A converter. This function configures the following settings for a single output channel.

- Output range set via input parameter
- Overrange is enabled/disabled via input parameter
- Clear enable/disable set via input parameter

All D/A channels are configured for 0-5V during board initialization, and the unipolar calibration settings are loaded into the D/A chips at power-up or reset. If the application requires any other ranges, then each channel that uses a different range must have its settings configured independently by calling this function prior to using the channel.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Channel	Channel number 0-15
Range	Output range set via input parameter(0 – 6) 0 = 0-5V, 1 = 0-10V, 2 = +/5V, 3 = +/-10V, 4 = 4-20mA,5 = 0-20mA , 6 = 0-24mA
Overrange	0 = disable overrange, range is as indicated above 1 = enable 20% overrange (for voltage ranges only, invalid for current ranges)
ClearEnable	0 = disable clear on chip clear; 1 = enable clear on chip clear
LoadCal	0 = don't load settings from EEPROM 1 = load Gain/Offset setting for this Range for this Channel, from EEPROM

Return Value

Error code or 0.

Usage Example

To configure channel zero in 0-5v with over range and clear disabled:

```
Channel=0;
Range=0;
Overrange=0;
ClearEnable=0;
RMM1616DASetSettings (bi, Channel, Range, overrange, ClearEnable);
```

4.2 RMM1616DASetSim

Function Definition
BYTE RMM1616DASetSim (BoardInfo* bi, int Simup);

Function Description
This function configures the board for simultaneous update or regular single-channel update mode. Updating a DAC may be done in either single-channel mode or simultaneous update mode. The mode is set with register bit DASIM. DASIM = 0 sets single-channel mode, while DASIM = 1 sets simultaneous update mode. After DASIM has been set high, an update command to the DAC will then be required for all channels to change. This can be done with RMM1616DAUpdate ().

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Simup	0 = single-channel update mode, 1 = simultaneous update mode

Return Value
Error code or 0.

Usage Example
To set all DA channel in regular single-channel update mode, i.e. when DA code is written to DAC data register, the DA channel will be updated immediately without further needed code:

```
int Simup=0;
RMM1616DASetSim (bi, Simup);
```

To update all the DA desired channels at once, set DASIM to 1, update all the DA channels, and then run the update command:

```
int Simup=1;
RMM1616DASetSim (bi, Simup);
RMM1616DAConvert (bi, 0, 65535);
RMM1616DAConvert (bi, 1, 65535);
RMM1616DAConvert (bi, 2, 65535);
RMM1616DAConvert (bi, 3, 65535);
RMM1616DAUpdate (bi);
```

4.3 RMM1616DAConvert

Function Definition
BYTE RMM1616DAConvert (BoardInfo* bi, int Channel, unsigned DACode);

Function Description
This function outputs a value to a single D/A channel. The output range must be previously set with RMM1616DASetSettings(). If the board is not set for simultaneous update mode (DASIM = 0), the channel will be updated immediately after the DAC internal calibration circuit is finished. If the board is set for simultaneous update mode (DASIM = 1), the channel will not be updated, and a separate update command must be executed.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Channel	Channel number 0-15
DACode	Code to convert (0 – 65535)

Return Value
Error code or 0.

Usage Example
To set channel zero with 5V

```
Channel=0;
DACode=65535
RMM1616DAConvert (bi, Channel, DACode);
```

4.4 RMM1616DAConvertScan

Function Definition

BYTE RMM1616DAConvertScan (BoardInfo* bi, int* ChannelSelect, unsigned int* DACodes);

Function Description

This function outputs multiple values to multiple D/A channels. The output ranges must be previously set with RMM1616DASetSettings(). If the board is not set for simultaneous update mode (DASIM = 0), each channel will be updated immediately by the hardware after being written to. If the board is set for simultaneous update mode (DASIM = 1), the channel will not be updated, and a separate update command must be executed. This function simply calls RMM1616DAConvert () once for each channel to be updated. It is mostly used for simultaneous mode operation where all channels will be updated at the same time after all data is written.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
ChannelSelect	0 = don't update channel xx, 1 = update channel xx
DACodes	0-65535 for each channel to be updated

Return Value

Error code or 0.

Usage Example

To update channel 0,4,6,8 with da code 65535, 32768, 4568 and 2637 respectively and rest of the channel not to be changed from existing voltage level.

```
ChannelSelect = (int *) malloc (sizeof (int) *16);
DACodes = (int *) malloc (sizeof (int) *16);
ChannelSelect[0]=1;
DACodes[0]=65535;
ChannelSelect[4]=1;
DACodes[4]= 32768;
ChannelSelect[6]=1;
DACodes[6]= 4568;
ChannelSelect[8]=1;
DACodes[8]= 2637;
RMM1616DAConvertScan (bi, ChannelSelect, DACodes);
```

4.5 RMM1616DAUpdate

Function Definition
BYTE RMM1616DAUpdate (BoardInfo* bi);

Function Description
This function is used to update the D/A when it is set for simultaneous mode (DASIM = 1) and the programmer is not using the RMM1616DAConvertScan () function. In this case the programmer is using RMM1616DAConvert () multiple times to write channel data individually. At the end of all these functions, the update command is needed to update all channels at once.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
```
RMM1616DAUpdate (bi);
```

4.6 RMM1616WaveformBufferLoad

Function Definition
BYTE RMM1616WaveformBufferLoad (BoardInfo* bi, RMM1616WAVEFORM* RMM1616waveform);

Function Description
This function configures a D/A waveform by copying the waveform data to the board waveform buffer and programming the number of frames into the board.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
RMM1616waveform	Structure with following member variables *Waveform - pointer to array of 16-bit unsigned data; If multiple channels are being set up at the same time, the data must be previously interleaved by the program, i.e. ch. 0, ch. 1, ch. 2, ch. 0, ch. 1, ch. 2, …; the array size must be equal to Frames x FrameSize; the array size must be no greater than 2048. Frames - total number of frames in the array FrameSize - number of channels to be driven 1-16 Channels - List of channels to be driven by the waveform generator

Return Value
Error code or 0.

Usage Example
To generate square wave on channel zero with four DA values ranges from 0- 5V.

```
RMM1616WAVEFORM RMM1616waveform;
RMM1616waveform.Waveform = (int *) malloc (sizeof (int) *4);
RMM1616waveform.Waveform [0] = 0;
RMM1616waveform.Waveform [1] = 0;
RMM1616waveform.Waveform [2] = 65535;
RMM1616waveform.Waveform [3] = 65535;
RMM1616waveform.Frames        = 4;
RMM1616waveform.FrameSize     = 1;
RMM1616waveform.Channels [0] = 0;
Channel = 0;
Range = 0;
Overrange = 0;
ClearEnable = 0;
RMM1616DASetSettings (bi, Channel, Range, overrange, ClearEnable);
RMM1616WaveformBufferLoad (bi, &RMM1616waveform);
```

4.7 RMM1616WaveformDataLoad

Function Definition

BYTE DSCUDAPICALL RMM1616WaveformDataLoad (BoardInfo* bi, int Address, int Channel, unsigned int Value)

Function Description

This function loads a single data point into the D/A waveform buffer. It can be used to update a waveform in real time while the waveform generator is running.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Address	Address in buffer at which to store the data; 0-2047;
Channel	Channel number, 0-15
Value	Data value, 0-65535

Return Value

Error code or 0.

Usage Example

To load the Data 65535 at address 258 on channel 0:

```
Address = 258;
Channel = 0;
Value = 65535;
RMM1616WaveformDataLoad (bi, 258, 0, 65535);
```

4.8 RMM1616WaveformConfig

Function Definition
BYTE RMM1616WaveformConfig (BoardInfo* bi, RMM1616WAVEFORM* RMM1616waveform);

Function Description
This function configures the operating parameters of the waveform generator, including the clock source, the output frequency if being controlled by a timer, and one-shot / continuous mode. The values in the buffer need to be loaded with RMM1616WaveformBufferLoad () / RMM1616WaveformDataLoad () before starting the waveform generator with RMM1616WaveformStart ().

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
RMM1616waveform	Structure with following member variables FrameSize - number of channels to be driven 1-16 Clock - 0 = software increment; 1 = counter/timer 0 output; 2 = counter/timer 1 output; 3 = DIO pin D0 Rate - frame update rate, Hz (only used if Clock = 1 or 2) Cycle - 0 = one-shot operation; 1 = repetitive operation

Return Value
Error code or 0.

Usage Example
To configure waveform generator with 100 Hz frequency

```
RMM1616WAVEFORM RMM1616waveform;
RMM1616waveform. FrameSize = 1;
RMM1616waveform. Clock = 1;
RMM1616waveform. Rate = 100;
RMM1616waveform. Cycle = 1;
RMM1616WaveformConfig (bi, & RMM1616waveform);
```

4.9 RMM1616WaveformStart

Function Definition
BYTE RMM1616WaveformStart (BoardInfo* bi);

Function Description
This function starts or restarts the waveform generator running based on its current configuration.

Prerequisites:
RMM1616WaveformConfig (), RMM1616WaveformBufferLoad (), and RMM1616WaveformDataLoad () have to have been run already in order to start a predictable waveform.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
```
RMM1616WaveformStart (bi);
```

4.10 RMM1616WaveformPause

Function Definition
BYTE RMM1616WaveformPause (BoardInfo* bi);

Function Description
This function stops the waveform generator. It can be restarted with RMM1616WaveformStart ().

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
```
RMM1616WaveformPause (bi);
```

4.11 RMM1616WaveformReset

Function Definition
BYTE RMM1616WaveformReset (BoardInfo* bi);

Function Description
This function resets the waveform generator and stops all waveform output.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
```
RMM1616WaveformReset (bi);
```

4.12 RMM1616WaveformInc

Function Definition
BYTE RMM1616WaveformInc (BoardInfo* bi);

Function Description
This function increments the waveform generator by one frame. The current frame of data is output to the channels associated to the each DAC output code in the buffer frame.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
```
RMM1616WaveformInc (bi);
```

4.13 RMM1616DIOConfig

Function Definition
BYTE RMM1616DIOConfig (BoardInfo* bi, int Port, int Dir, int Mode);

Function Description
This function sets the digital I/O port direction and mode for the selected port.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Port	0 = A, 1 = B, 2 = C, 3 = D, 4 = E, 5 = F
Dir	For ports 0-4 : 0 = input, 1 = output For port 5 (F) : 0-255 where 1 = out and 0 = in for each bit
Mode	For ports 0-3 : 0 = normal mode, 1 = latched mode For ports 4-5 : not used

Return Value
Error code or 0.

Usage Example
To set port 0 in output mode

```
Port=0;
Dir=1;
Mode=0;
RMM1616DIOConfig (bi, Port, Dir, Mode);
```

4.14 RMM1616DIOOutputByte

Function Definition
BYTE RMM1616DIOOutputByte (BoardInfo* bi, int Port, byte Data);

Function Description
This function outputs the specified data to the specified port's output register.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Port	0 = A, 1 = B, 2 = C, 3 = D, 4 = E, 5 = F
Data	8 bit value to write to the port

Return Value
Error code or 0.

Usage Example
To set Port 0 output as 0x77,

```
PORT=0;
Data=0x77;
RMM1616DIOOutputByte (bi, Port, Data);
```

4.15 RMM1616DIOInputByte

Function Definition
BYTE RMM1616DIOInputByte (BoardInfo* bi, int Port, byte* Data);

Function Description
This function reads in the data from the specified port and returns it in the location specified by the pointer passed.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Port	0 = A, 1 = B, 2 = C, 3 = D, 4 = E, 5 = F
Data	pointer to location to store 8 bit value read from the port

Return Value
Error code or 0.

Usage Example
To read port 0 input values and display on the screen

```
Port=0;
RMM1616DIOInputByte (bi, Port, &Data);
printf ("The PORT 0: 0x%x",Data);
```

4.16 RMM1616DIOOutputBit

Function Definition

BYTE RMM1616DIOOutputBit (BoardInfo* bi, int Port, int Bit, int Value);

Function Description

This function outputs a single bit to an output port. The other bits remain at their current values.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Port	0 = A, 1 = B, 2 = C, 3 = D, 4 = E, 5 = F
Bit	Bit position 0-7
Value	0 or 1

Return Value

Error code or 0.

Usage Example

To set Port 0, bit 6 to 1:

```
Port=0;
Bit=6;
Value=1;
RMM1616DIOOutputBit (bi, Port, Bit, Value);
```

4.17 RMM1616DIOInputBit

Function Definition
BYTE RMM1616DIOInputBit (BoardInfo* bi, int port, int bit, int* value);

Function Description
This function reads in the specified bit from the specified port and returns it in the location specified by the pointer passed.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Port	0 = A, 1 = B, 2 = C, 3 = D, 4 = E, 5 = F
Bit	Bit position 0-7
Value	pointer to location to receive the bit data; return data is always 0 or 1

Return Value
Error code or 0.

Usage Example
To read Port 0, bit 7 and display on the screen:

```
Value=0;
Port=0
Bit=7;
RMM1616DIOInputBit (bi, port, bit, &value);
printf ("The PORT0 7th bit value %d : ",value);
```

4.18 RMM1616CounterSetRate

Function Definition
BYTE RMM1616CounterSetRate (BoardInfo* bi, int Num, float Rate, int OutEn, int OutPol);

Function Description
This function programs a counter for timer mode with down counting and continuous operation (reload enabled).

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Num	Counter number, 0-1
Rate	Desired output rate, Hz
OutEn	1 = enable output onto corresponding I/O pin; 0 = disable output
OutPol	1 = positive output pulse; 0 = negative output pulse; only valid if CtrOutEn = 1

Return Value
Error code or 0.

Usage Example
To configure counter 0 with 100Hz,

```
Num=0;
Rate=100;
OutEn = 1;
OutPol = 1;
RMM1616CounterSetRate (bi, Num, Rate, OutEn, OutPol);
```

4.19 RMM1616CounterConfig

Function Definition
BYTE RMM1616CounterConfig (BoardInfo *bi, RMM1616CTR *Ctr);

Function Description
This function performs a general counter configuration operation with all options and starts the counter.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Ctr	Structure with following member variables Num - Counter number, 0-1 Data - Initial load data, 32-bit unsigned binary Clock - Clock source, Must be 0, 2, or 3 (see Hardware User Manual for usage) CountDir - 0 = down counting, 1 = up counting Reload - 0 = one-shot counting, 1 = auto-reload OutEn - 1 = enable output onto corresponding I/O pin; 0 = disable output OutPol - 1 = positive output pulse; 0 = negative output pulse; only valid if OutEn = 1 GateEn - 0 = no external gate; 1 = external gate enabled

Return Value
Error code or 0.

Usage Example
To configure counter 0 with 100Hz, counter direction down, auto reload and output enabled

```
RMM1616CTR Ctr;
Ctr.Num = 0;
Ctr.clock = 2; //50MHz
Ctr.data = 50000000/100;
Ctr.Reload = 1;
Ctr.OutEn = 1;
BYTE RMM1616CounterConfig (bi, &Ctr);
```

4.20 RMM1616CounterRead

Function Definition
BYTE RMM1616CounterRead (BoardInfo* bi, int CtrNo, unsigned long * CtrData);

Function Description
This function latches a counter and reads the value.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
CtrNo	Counter number, 0-1
CtrData	pointer to storage location of return data, unsigned 32-bit value

Return Value
Error code or 0.

Usage Example
To read current value of counter 0 when it is running and display on the screen:

```
Unsigned long CtrData;
ctrNo = 0;
RMM1616CounterRead (bi, CtrNo, &CtrData);
printf ("The counter value %d ",CtrData);
```

4.21 RMM1616CounterFunction

Function Definition
BYTE RMM1616CounterFunction (BoardInfo* bi, RMM1616CTR* Ctr);

Function Description
This function can be used to program any single command into a counter, except reading which is done by RMM1616CounterRead ().

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
RMM1616CTR	Structure with following member variables Ctrno - Counter number, 0-1 CtrData - Initial load data, 32-bit straight binary CtrCmd - Counter command, 0-15 (see Hardware User Manual for usage) CtrCmdData - Auxiliary data for counter command, 0-3 (see Hardware User Manual for usage)

Return Value
Error code or 0.

Usage Example
To reset the counter 0:

```
Ctr. Ctrno = 0;
Ctr. CtrData = 10000;
Ctr. CtrCmd = 0xF0; // RMM1616_COUNTER_CMD_RESET_ONE
Ctr. CtrCmdData = 0;
RMM1616CounterFunction (bi, &Ctr);
```

4.22 RMM1616PWMConfig

Function Definition
BYTE RMM1616PWMConfig (BoardInfo* bi, RMM1616PWM* RMM1616pwm);

Function Description
This function configures a PWM for operation.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
RMM1616pwm	Structure with following member variables Num - PWM number, 0-3 Rate - output frequency in Hz Duty - initial duty cycle, 0.0-100.0 Polarity - 0 = pulse high, 1 = pulse low OutputEnable - 0 = disable output, 1 = enable output on DIO pin Run - 0 = don't start PWM, 1 = start PWM

Return Value
Error code or 0.

Usage Example
To configure PWM 0 with 100 Hz, 50% duty cycle and output enabled:

```
RMM1616PWM RMM1616pwm;
RMM1616pwm.Num=0;
RMM1616pwm.Rate=100;
RMM1616pwm.Duty=50.0;
RMM1616pwm.Outputenable=1;
RMM1616PWMConfig (bi, &RMM1616pwm);
```

4.23 RMM1616PWMStart

Function Definition
BYTE RMM1616PWMStart (BoardInfo *bi, int Num);

Function Description
This function starts a PWM running.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Num	PWM number, 0-3

Return Value
Error code or 0.

Usage Example
To start PWM 0:

```
Num=0;
RMM1616PWMStart (bi,Num);
```

4.24 RMM1616PWMStop

Function Definition
BYTE RMM1616PWMStop (BoardInfo* bi, int Num);

Function Description
This function stops a PWM.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Num	PWM number, 0-3 for single PWM or 0xFF / 255 to stop all PWMs

Return Value
Error code or 0.

Usage Example
To stop PWM 0:

```
Num=0;
RMM1616PWMStop (bi, Num);
```

4.25 RMM1616PWMCommand

Function definition
BYTE RMM1616PWMCommand (BoardInfo* bi, RMM1616PWM* RMM1616pwm);

Function Description
This function is used to modify a PWM configuration. For example, it can be used to modify the duty cycle or frequency while the PWM is running.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
RMM1616pwm	Num	int	PWM number, 0-3
	Command	int	0-15 = PWM command
	CmdData	int	0 or 1 for auxiliary PWM command data (used for certain commands)
	Divisor	uns. long	24-bit value for use with period and duty cycle commands

Return Value
Error code or 0.

Usage Example
To stop PWM 0,

```
RMM1616PWM RMM1616pwm;
RMM1616pwm. Num=0;
RMM1616pwm. Command =0x0000;
RMM1616pwm. CmdData =1;
RMM1616pwm. Divisor =0;
RMM1616PWMCommand (bi, & RMM1616pwm);
```

4.26 RMM1616UserInterruptSet

Function Definition
BYTE RMM1616UserInterruptSet (BoardInfo* bi, RMM1616USERINT* RMM1616userint);

Function Description
This function installs a pointer to a user function that runs when interrupts occur. It configures the interrupt handler to run the user function either before or after the standard interrupt function or in stand-alone mode.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
RMM1616userint	Structure with following member variables IntFunc - pointer to user function to run when interrupts occur Mode - 0 = alone, 1 = before standard function, 2 = after standard function Source - 0 = counter/timer 2 output, 1 = digital input D1; (used only if mode = 0) Enable - 0 = disable interrupts, 1 = enable interrupts

Return Value
Error code or 0.

Usage Example
To install a function to be called whenever interrupt occurs:

```
Void intfunction ()
{
      printf ("My function called ");
}
Void main ()
{
      RMM1616USERINT RMM1616userint;
      RMM1616userint.IntFunc = intfunction;
      RMM1616userint.Mode=0;
      RMM1616userint.Source=0;
      RMM1616userint.Enable=1;
      RMM1616UserInterruptSet (bi, & RMM1616userint);
}
```

4.27 RMM1616UserInterruptStop

Function Definition
BYTE RMM1616UserInterruptStop (BoardInfo* bi, RMM1616USERINT* RMM1616userint);

Function Description
This function stops user interrupts. If the interrupt mode is before or after, the main interrupt operation may continue. If mode is Alone, the interrupt operation is stopped.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
RMM1616userint	Structure with following member variables IntFunc - pointer to user function to run when interrupts occur Mode - 0 = alone, 1 = before standard function, 2 = after standard function Source - 0 = counter/timer 2 output, 1 = digital input D1; (used only if mode = 0) Enable - 0 = disable interrupts, 1 = enable interrupts

Return Value
Error code or 0.

Usage Example
```
RMM1616USERINT RMM1616userint;
RMM1616userint.disable=1;
RMM1616UserInterruptStop (bi, & RMM1616userint);
```

4.28 RMM1616InitBoard

Function definition
BYTE RMM1616InitBoard (DSCCB* dsccb);

Function Description
This function Initialize the board in PCI mode.

Function Parameters

Name	**Description**
dsccb	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
```
DSCCB dsccb;
dsccb.boardtype = DSC_RMM1616;
RMM1616InitBoard (& dsccb);
```

4.29 RMM1616PCIInitBoard

Function Definition
BYTE RMM1616PCIInitBoard (DSCCBP* dsccbp);

Function Description
This function initializes the board in PCI mode.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
```
DSCCBP dsccbp;
dsccbp.boardtype = DSC_RMM1616;
RMM1616PCIInitBoard (& dsccbp);
```

4.30 RMM1616ISAInitBoard

Function Definition
BYTE RMM1616ISAInitBoard (DSCCB* dsccb);

Function Description
This function Initialize the board in ISA mode.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
```
DSCCB dsccb;
dsccb.boardtype = DSC_RMM1616;
dsccb.base_address=0x300;
dsccb.int_level=5;
BYTE RMM1616ISAInitBoard (& dsccb);
```

4.31 RMM1616FreeBoard

Function Definition
BYTE RMM1616FreeBoard (BoardInfo *bi);

Function Description
This function stops any active interrupt processes and frees the interrupt resources assigned to a board.

Function Parameters

Name	Description
Board	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
```
RMM1616FreeBoard (bi);
```

4.32 RMM1616LED

Function Definition
BYTE RMM1616LED (BoardInfo* bi, int Enable);

Function Description
This function turns the LED on or off.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Enable	0 = off, 1 = on

Return Value
Error code or 0.

Usage Example
To turnoff blue LED on the board

```
Enable=0;
BYTE RMM1616LED (bi, Enable);
```

5. UNIVERSAL DRIVER DEMO APPLICATION DESCRIPTION

The Universal Driver supports the following applications for the RMM-1616 board:

- DA Convert
- DA Waveform
- DIO
- Counter
- PWM
- User Interrupt

5.1 DA Convert

This function outputs a value to a single D/A channel. The output range can be selected from user input. When a D/A conversion is performed, it is essentially taking a digital value and sending it out to the specified analog output channel as a voltage. This output code can be translated to an output voltage using one of the formulas described in the hardware specification.

NOTE: Once you generate a D/A conversion on a specific output channel, that channel will continue to maintain the specified voltage until another conversion is done on the same channel or the board is reset or powered down. For a 16-bit DAC, the range of output code is from 0 to 65535.

5.2 DA Waveform

This function generates a desired waveform on selected DA channel. It configures a D/A channel by copying the waveform values to the board waveform buffer. This Waveform Generators replays a loaded waveform from the internal FPGA memory. The digital signal is converted into an analog output signal with a defined offset and amplitude based on the D/A values sent. Any waveform can be replayed be it a previously acquired waveform or a calculated or simulated waveform.

5.3 DIO

Digital I/O is fairly straightforward. This function supports five types of direct digital I/O operations: input bit, input byte, output bit, output byte and DIO Loopback.

5.4 Counter

Generally the counter is used as rate generator, Also the counter can be configured in count-up or count-down direction.

5.5 PWM

This application generates pulse width modulation (PWM) signals. PWM signals are a method for generating an analog signal using a digital source. A PWM signal consists of two main components that define its behavior: duty cycle and frequency. The duty cycle describes the amount of time the signal is in a high (on) state as a percentage of the total time it takes to complete one cycle. The frequency determines how fast the PWM completes a cycle (i.e. 1000Hz would be 1000 cycles per second), and therefore how fast it switches between high and low states. By cycling a digital signal off and on at a fast enough rate, and with a certain duty cycle, the output will appear to behave like a constant voltage analog signal when providing power to devices. This program gets duty cycle and frequency value from User and generates PWM signals.

5.6 User Interrupt

The user interrupt feature of the Universal Driver enables you to run your own code when a hardware interrupt is generated by an I/O board. This is useful for applications that require special operations to be performed in conjunction with the interrupt or applications where you want to run your code at regular fixed intervals. Universal Driver includes example programs for each board with user interrupt capability to illustrate how to use this feature. This application gets interrupt frequency as user input and calls the user function periodically at the rate of interrupt frequency.

6. UNIVERSAL DRIVER DEMO APPLICATION TEST PROCEDURE

The following section details about how to input to the RMM-1616 application and to confirm the output which is obtained through application.

6.1 DA Convert Application

This application gets input from user as follows:

- Enter channel number (0-15):
- Enter output range (0-6, 0 = 0-5V, 1 = 0-10V, 2 = +/5V, 3 = +/-10V, 4 = 4-20mA, 5 = 0-20mA,6 = 0-24mA) :
- Enter output code (0-65535 or q to quit):

To set channel 0 with 10 v using range 1, run DA Convert application and provide input as follows:

Channel Number=0
Range=1
Output code=65535

Measure the voltage on channel 0 using a multi-meter and it should show 10V, the application generated expected voltage.

6.2 DA Waveform Application

This application gets input from user as follows:

- Enter output channel (0-15):
- Enter output range(0-4, 0 = 0-5V, 1 = 0-10V, 2 = +/5V, 3 = +/-10V):
- Select waveform type, Sine or saw tooth
- Select waveform size and sample rate

To generate a sine wave form on channel zero with 100Hz frequency and a range from 0-5V, run the wave form application and provide input as follows:

Channel Number=0
Range=0
Select waveform type as Sine

Place an oscilloscope probe on D/A channel zero, set the voltage division to 1V range and the second division to 1ms. It should show a sine wave with 100Hz frequency, the application generated expected wave form.

6.3 DIO Application

The DIO application provides various operations on a DIO channel i.e. input byte, output byte, input bit, output bit and DIO loopback. This section describes the input byte and output byte DIO operation. The DIO port must be configured in either input or output mode based on the DIO operation need to be performed.

To configure a DIO port in output mode for output byte:

- Select configure option from main menu
- Enter port number
- Configure the port in output mode:
- Exit from current menu
- Select output byte option from main menu
- Enter port number
- Enter desired value in Hex to be sent on DIO port

To set Port 0 on all the pins to 5V, except pin 3 and pin 7, run the DIO application and provide input as follows:

- Select configure option from main menu : 1
- Enter port number : 0
- Configure the port in output mode: 1
- Exit from current menu : -1
- Select output byte option from main menu : 2
- Enter port number: 0
- Enter desired value in Hex to be sent on DIO port : 0x77

Measure the voltage on Port 0 pins using a multi-meter. It should show 5V on all the pins except pin 3 and pin 7, the application generated expected voltages.

To configure a DIO port in input mode for input byte:

- Select configure option from main menu
- Enter port number
- Configure the port in input mode:
- Exit from current menu
- Select input byte option from main menu
- Enter port number
- Reads and displays current voltage levels on the screen

Provide 5V to Port 0 pin 0 from VCC and it should read and display 0x01. To see the output, run the DIO application and provide input as follows:

- Select configure option from main menu : 1
- Enter port number : 0
- Configure the port in input mode: 0
- Exit from current menu : -1
- Select output byte option from main menu : 3
- Enter port number: 0

The application should shows 0x01 on the screen as expected.

6.4 Counter Application

This application gets input from user as follows:

- Enter counter number (0-1)
- Enter output frequency(1-50MHZ)
- Waits for key press
- If any key is pressed , application stops rate generator

To generate a 100Hz rate generator using counter 0, run the counter application and provide input as follows:

- Enter counter number (0-1):0
- Enter output frequency(1-50MHZ):100

Place an oscilloscope probe on counter 0 output pin (Port F2 is the output pin for Counter 0), set the voltage division to 1V range and the second division to 1ms. It should show a periodic pulse with 100Hz frequency, the application generated expected rate.

- Press any key and automatically the application stops counter output

6.5 PWM Application

This application gets input from user as follows:

- Enter PWM Number (0-3)
- Enter output frequency (1-50MHZ)
- Enter polarity value (0-1)
- Enter duty cycle (0-100)
- Output is generated
- Waits for key press
- If any key is pressed, application Stops PWM output.

To generate a 100Hz PWM waveform with a 50% duty cycle on PWM channel 0, run the PWM application and provide input as follows:

- Enter PWM Number (0-3):0
- Enter output frequency (1-50MHZ):100
- Enter polarity value (0-1):0
- Enter duty cycle (0-100):50

Place an oscilloscope probe on PWM channel 0 output pin (PORTF0 is the output pin for PWM 0), set the voltage division to 1V range and the second division to 1ms. It should show a PWM wave form with 50 % duty cycle and 100Hz frequency, the application generated expected rate.

- Press any key and the application automatically stops the PWM output

6.6 User Interrupt function

This application gets input from user as follows:

- Enter clock source (0=Counter 0 , 1= Counter 1)
- Enter interrupt rate (1-50MHZ)
- It calls user function based interrupt rate
- Press any key to cancel interrupt.

This application installs a function where a count value is incremented by one whenever the function gets called. To confirm the user function is getting called as per the interrupt rate, run the UserInt application and provide input as follows:

- Enter clock source (0=Counter 0, 1= Counter 1):0
- Enter interrupt rate (1-50MHZ) :100

It calls user function based interrupt rate and prints the count value every second. Since it configured for 100Hz it should display count value as follows:

```
UserInt count value = 0
UserInt count value = 100
UserInt count value = 200
UserInt count value = 300
```

Press any key to cancel the interrupt.

7. COMMON TASK REFERENCE

1. Performing D/A Conversion
2. Performing D/A Scan Conversion
3. Performing D/A Simultaneous Update
4. Performing D/A Waveform Generation
5. Performing Digital IO Operations
6. Performing PWM
7. Performing Counter Operations
8. Performing User Interrupt Operations

7.1 Data Acquisition Feature Overview

I/O Connectors

RMM-1616 provides 2 I/O connectors for the attachment of all user I/O signals. These connectors are shown below. The connectors are JST model no. BM20B-GHDS-G-TF. Diamond's I/O cable no. 6980501 (2 per board) may be used to connect the user's signals to these connectors. This cable comes in a kit of 2 as part number C-50-18. This cable has a common 2mm pitch IDC connector at the opposite end which may be plugged into a custom board or may be cut off and the wiring then used as needed. Unused signals do not need to be terminated. However if the cable wiring is cut, care should be taken to avoid shorting any unused wires to any other voltages in the system, in order to prevent possible damage to the board or incorrect analog I/O readings.

Analog outputs

There are 16 analog voltage and current output channels, numbered 0-15, located on connector J4 pins 1-48. Analog outputs operate in single-ended mode only and are always referenced to the analog ground, which must be connected to the respective analog ground pins.

Waveform generator

The board contains a 2-channel analog waveform generator using the analog outputs of channel 0 and 1. Two channels may operate simultaneously. The waveforms are stored in an on-board data buffer that holds 2048 samples. Any number of samples can be used up to the 2048 limit. If more than one channel is being used, then each channel's waveform must have the same number of samples, and the maximum length of each waveform is 2048 divided by the number of channels in use.

The waveform generator can be clocked in multiple ways, including: software command, external digital signal on I/O connector J3 pin 19 (Digital I/O port 20), or on-board counter/timer 0 or 1. On each clock, one "frame" of data will be output, consisting of one data point for each channel being used.

The maximum frame output rate depends on the number of channels in use and is shown in the table below:

Channels (same chip)	Max frame rate
1	166KHz
2	83KHz
3	55KHz
4	41KHz

Digital I/O signals

There are 48 digital I/O signals, divided into five groups as DIOA0-DIOA7, DIOB0-DIOB7, DIOC0-DIOC7, DIOD0-DIOD7, DIOE0-DIOE7 and DIOF0-DIOF7. Ports A-F are on I/O connectors J5. Digital I/O signals use 3.3V signaling only. Port A-E, directions of all bits are set in unison and Port F signal direction is independently programmable. On system startup or reset, all signals are automatically set to input mode.

All digital I/O lines are tied to individual pull-up/down resistors which may be jumpered to either the DIO logic level voltage selected above or ground using locations UP or DN on jumper block J8, respectively. All DIO lines are pulled up or down together; it is not possible to configure some for pull-up and some for pull-down. The default is pull-up.

Counter/timers

The board provides 2 32-bit counter/timers. Counter mode means the circuit will count external events that are connected via one of the digital I/O lines. Timer mode means the circuit will generate output pulses at a user-specified rate. The pulse width is programmable for both polarity (high or low pulse) and width (1, 10, 100, or 1000 clock pulses). The clock for timer mode is provided internally from an on-board 50MHz oscillator. This oscillator is divided by 50 to provide a 1MHz clock for very low pulse rates. The available range of output rates is 50MHz / 20ns (50MHz / 1) to .0002328Hz / 4295 sec (1MHz / 2^32).

The counter/timers use the digital I/O lines for their I/O signals. When a counter/timer is programmed to use external clock or output, the associated digital I/O line is taken over for the counter/timer and its direction is set as needed to support the selected function. The I/O pin may be assigned as either an input (clock) to the counter/timer or an output. The I/O pin assignment is as follows:

Connector J5 Pin	DIO Bit	Counter/Timer
43	F2	0
44	F3	1

Pulse Width Modulators (PWMs)

The board offers 4 24-bit PWMs. Each PWM may be programmed for output frequency, duty cycle, and output polarity. Duty cycle is defined as the percentage of time the output signal will have the indicated polarity during each period. For example, a 1KHz output frequency (1ms period) with 20% duty cycle and positive output polarity will exhibit a repetitive waveform that is high for 0.2ms at the start of the period and low for 0.8ms during the remainder of the period. Each PWM contains 2 counters. Counter C0 controls the output frequency, and counter C1 controls the duty cycle.

The PWMs use the digital I/O lines for their output signals. When a PWM is running and its output is enabled, the associated digital I/O line is taken over to be used as the output for the PWM, and its direction is forced to output. The I/O pin assignment is as follows:

Connector J5 Pin	DIO Bit	PWM
41	F0	0
42	F1	1
43	F2	2
44	F3	3

7.2 Data Acquisition Software Task Reference

This section describes the various data acquisition tasks that may be performed with the RMM-1616 and gives step by step instructions on how to achieve them using the Universal Driver functions. Tasks include:

- Program entry / exit sequence
- D/A conversions
- Waveform generator
- Digital I/O
- Counter/timer operation
- PWM operation
- User interrupts

Program Entry/Exit sequence

1. All driver usage begins with the function RMM1616InitBoard() This function must be called prior to any other function involving the RMM-1616.
2. At the termination of the program the programmer may use RMM1616FreeBoard(), but this is not required. This function is normally used in a development environment where the program is being repeatedly modified and rerun.

D/A conversion operation

This section discusses single D/A conversions on one or more channels. For waveform generator operations, see the separate D/A waveform generator section.

D/A conversions can be performed on one or more channels at a time. When operating on multiple channels, the program has the option of selecting single channel update or multi-channel simultaneous update. Simultaneous update is useful in certain applications where two or more parameters need to change simultaneously, for example when driving a laser from point (x1, y1) to point (x2, y2). In this type of application it is obviously preferable to move the laser from point 1 to point 2 in a straight line rather than in two orthogonal lines, one in the X direction and one in the Y direction.

For single channel output, use the following sequence:

1. RMM1616DASetSettings() to select the output range and simultaneous update mode if desired
2. RMM1616DAConvert() to update a single channel with the given value

For multi-channel output with individual channel update:

1. RMM1616DASetSettings() to select the output range; set Sim = 0
2. RMM1616DAConvertScan() to update the selected channels with the given data

For multi-channel output with simultaneous update:

1. RMM1616DASetSettings() to select the output range; set Sim = 1
2. RMM1616DAConvertScan() to load the given data into the selected channels
3. RMM1616DAUpdate() to update all channels at the same time

Waveform Generator

Using the waveform generator involves a series of operations:

1. Create the waveform data buffer and download it to the board
2. Configure the waveform generator

3. Start (and restart) the waveform generator
4. Pause or reset the waveform generator

To create the waveform buffer, first compute the waveform or load the data from a file. The data is in binary form using numbers in the range 0 – 65535 (0 – 2^16-1). If more than one channel will be used for waveform generation, each waveform must be the same length, and the data for all channels must be interleaved in the buffer, so that each consecutive group of data values represents one "frame" of data. For example, a 2-channel waveform generator using D/A channels 0 and 1 would have its data buffer organized like this:

Buffer address	Channel
0	0
1	1
2	0
3	1
4	0
5	1
…	…

The total number of samples cannot exceed 2048 for a single channel or 2048 / <no. of channels> for multiple channels. The term <no. of channels> is also referred to as the frame size.

Once the data buffer is built, use RMM1616WaveformBufferLoad() to download the entire buffer to the board at one time. The buffer must not be larger than 2048 samples, and the formula <frame size> x <no. of frames> must be less than or equal to 2048.

RMM1616WaveformDataLoad() can be used to update a single data point in the waveform buffer at any time, including while the waveform generator is running.

Once the buffer is downloaded, use RMM1616WaveformConfig() to configure the clock source, clock rate (for internal clocking), and one-shot or continuous operation. In One-shot operation, the waveform generator will make a single pass through the data buffer and output the data one time, then automatically stop. In continuous operation, the waveform generator will output the waveform(s) repeatedly until stopped with a software command.

To start the waveform, use RMM1616WaveformStart(). After this function is called, the waveform generator will run in response to clocks from the selected source.

To pause the waveform generator at any time in its current position, use RMM1616WaveformPause(). The waveform may be restarted in its current position by using RMM1616WaveformStart() again.

To reset the waveform generator, use RMM1616WaveformReset(). This stops the waveform generator function, however the data buffer still retains its data. After the reset function is called, to restart the waveform generator you must RMM1616WaveformConfig() followed by RMM1616WaveformStart().

If software increment is selected, the waveform is incremented with the function RMM1616WaveformInc(). Each time this function is called, the waveform generator will output one frame of data, i.e. one data value to each channel in use.

Digital I/O operations

Digital I/O operation is relatively simple. First configure the DIO port direction with one of the below functions:

BYTE RMM1616DIOConfig () configures a single port for input or output.

Then execute whichever I/O function is desired. Byte read/write enables 8 bits of digital I/O to be updated at once. Bit operation enables a single bit to be updated.

BYTE RMM1616DIOOutputByte(BoardInfo* bi, int port, BYTE data);
BYTE RMM1616DIOInputByte(BoardInfo* bi, int Port, byte* Data);
BYTE RMM1616DIOOutputBit(BoardInfo* bi, int port, int Bit, int Value);
BYTE RMM1616DIOInputBit(BoardInfo* bi, int port, int Bit, int* Value);

Counter/timer operations

The counter/timers are configured using a series of commands to control individual features. Driver functions provide shortcuts to quickly configure the counters for common counting and timing operations. For non-standard or specialized operations, the individual commands can be used to configure the counter/timers exactly as desired.

Simplified Programming

To program a counter/timer as a rate generator with a specific frequency, use RMM1616CounterSetRate(). This function is also used to set the sampling rate for interrupt-based A/D conversions. The counter is programmed for down counting, and an external clock is selected. The counter output may optionally be enabled onto a digital I/O pin, with programmable polarity and pulse width.

To program a counter/timer for counting operation, use the following functions:

1. Use RMM1616CounterConfig() to configure the counter for either up or down counting and start the counter running. A Digital I/O pin may be selected for either the input or the output (but not both). This function is typically used to count external events.
2. Use RMM1616CounterRead() to read the current contents of the counter. This function can be used repeatedly to monitor the operation. This is normally used with event counting.

Detailed Programming

To program a counter/timer using individual commands, use RMM1616CounterFunction(). This function must be used multiple times to execute each command needed to configure the counter. All commands take effect immediately upon execution. The typical command sequences for the most common operations are provided below. See the full list of counter/timer commands in the appendix.

For a rate generator:

Command	Function
15	Reset the counter/timer. This function should be used first to reset any previous configurations to ensure the counter/timer operates exactly as desired.

1 Load counter with desired divisor to select the desired output pulse rate. The output rate is the selected clock frequency divided by the divisor.

2 Select the count direction. For a rate generator the direction should be down.

6 Select clock source. Normally an internal clock (50MHz or 1MHz) will be selected.

7 Enable auto-reload. This means that the counter will operate continuously.

If the rate generator output is desired, use the following two commands:

8 Enable counter output on the associated digital I/O pin. The desired output polarity is also selected with this command.

Finally, enable the counter/timer with the following command:

4 Start the counter/timer running. (This function is also used to stop the counter/timer.)

When the rate generator is no longer needed, either of the following commands can be used:

4 Stop the counter/timer running. The existing settings are maintained so the counter can be restarted later if desired. If it was assigned for the output pulse, the digital I/O line is still tied to the counter/timer and cannot be used for normal digital I/O operations.

15 Reset the counter/timer. This stops the counter/timer and releases the digital I/O line back to normal digital I/O operation.

For an event counter:

15 Reset the counter/timer. This function should be used first to reset any previous configurations to ensure the counter/timer operates exactly as desired. This will reset the counter data register to 0.

2 Select the count direction. For an event counter the direction should be up.

6 Select clock source. Normally the associated digital I/O pin will be selected to enable counting external pulses.

7 Enable or disable auto-reload. If auto-reload is enabled, the counter will operate continuously, meaning that when it reaches 2^32-1 it will roll over to zero. In most cases auto-reload will be disabled for event counting.

4 Start the counter/timer running. (This function is also used to stop the counter/timer.)

While the counter is operating, its current count can be read by using the RMM1616CounterRead() function.

PWM operations

The PWMs are configured using a series of commands to control individual features. Driver functions provide shortcuts to quickly configure them for common operations. For non-standard or specialized operations, the individual commands can be used to configure the PWMs exactly as desired.

To configure and start a PWM:

1. RMM1616PWMConfig() configures the selected PWM for output frequency, duty cycle, and polarity. The PWM may optionally be started as well.
2. RMM1616PWMStart(). can be used to start the PWM running if the config function did not start it.

To stop a PWM:

RMM1616PWMStop() stops a PWM from running. The output is driven to the inactive state. For a PWM with positive output polarity, the output will go low.

To restart a PWM that has been stopped: use RMM1616PWMStart().

To implement special functions, such as changing the duty cycle or frequency of a PWM while it is running, use RMM1616PWMCommand(). This function must be executed multiple times, once for each command, to carry out the desired configuration. The available commands are listed in the appendix. All commands take effect immediately upon execution.

User interrupts

Universal Driver enables the installation of user-defined code to be run when an interrupt occurs. The interrupt can be triggered by a variety of sources. The interrupt can run as the only procedure when the interrupt occurs (standalone or alone mode) or it can run before or after the driver's built-in interrupt function (Before and After modes). The available modes depend on the source of the interrupt:

Source	Source no.	Modes supported
A/D interrupts	0	Not supported on RMM1616
D/A interrupts	1	Not supported on RMM1616
Counter/timer interrupts	2, 3	0 Alone
Digital input interrupts	4	0 Alone

User interrupts are very easy to use. Just 3 steps are required: Configure, run, and stop.

Configure:

RMM1616UserInterruptSet() selects the source for the user interrupts and also installs a pointer to the user's code to run when the interrupt occurs. If user interrupts are being run in Before or After mode, this function must be called before the function that initiates the standard interrupt function (e.g. before the sequence described in the A/D interrupts section).

Run (alone mode):

1. If a counter/timer is being used to drive interrupts, then configure it with RMM1616CounterSetRate().
2. If a digital input is being used to drive interrupts, it is configured with RMM1616UserInterruptSet () and it will start the interrupt operation.

Stop (alone mode):

Use RMM1616UserInterruptStop () to stop user interrupts.

Stop (before / after modes):

Use the standard interrupt cancel function such as RMM1616UserInterruptStop ().

LED control

The RMM-1616 contains a blue LED that is user-programmable. This can be used as a visual indication that the board is responding to commands. Turn the LED on and off, use RMM1616LED().

7.3 Performing D/A Conversion

When you perform a D/A conversion, you are essentially taking a digital value and converting it to a voltage value that you send out to the specified analog output. This output code can be translated to an output voltage using one of the equations described in the hardware manual. The Universal Driver function for performing a D/A conversion is RMM1616DAConvert ().

Step-By-Step Instructions

Call RMM1616DAConvert () and pass the channel number and output code value. This will generate a D/A conversion on the selected channel that will output the new voltage that you set with the output code.

Once you generate a D/A conversion on a specific output channel, that channel will continue to maintain the specified voltage until another conversion is done on the same channel or the board is reset or powered down. For a 12 bit DAC, the range of output code is from 0 to 4095. For a 16-bit DAC, the range of output code is from 0 to 65535.

Example of Usage for D/A Conversion

```
...
BoardInfo *bi;
int channel;
unsigned DACode;
int Range;
int OverRange;
int ClearEnable;
int LoadCal;
...
channel = 0;
DACode = 4095;
Range    = 0;           /* 0 = 0-5V */
OverRange = 0;
ClearEnable = 0;
LoadCal = 0;

/* Step 1 */
if ((result = RMM1616DASetSettings(bi, channel, Range, OverRange,
ClearEnable, LoadCal )) != DE_NONE)
    return result;
/* Step 2 */
if ((result = RMM1616DAConvert(bi, channel, DACcode)) != DE_NONE)
    return result;
...
```

7.4 Performing D/A Scan Conversion

A D/A scan conversion is similar to a D/A conversion except that it performs several conversions on a multiple, specified output channels with each function call. The Universal Driver function for performing a D/A conversion scan is RMM1616DAConvertScan ().

Step-By-Step Instructions

Pass the values for ChannelSelect and DACodes pointer to this function.

Call RMM1616DAConvertScan () and pass it a pointer to your scan array values. This will generate a D/A conversion for each channel you set to enable.

Example of Usage for D/A Conversion Scan

To update channel 0, 4, 6, 8 with DA code 65535, 32768, 4568 and 2637 respectively and the rest of the channels not to be changed from existing voltage level:

```
        ...
ChannelSelect = (int *) malloc (sizeof (int) *16);
DACodes = (int *) malloc (sizeof (int) *16);
ChannelSelect[0]=1;
DACodes[0]=65535;
ChannelSelect[4]=1;
DACodes[4]= 32768;
ChannelSelect[6]=1;
DACodes[6]= 4568;
ChannelSelect[8]=1;
DACodes[8]= 2637;
If ((result = RMM1616DAConvertScan (bi, ChannelSelect, DACodes))! = DE_NONE)
    return result;
```

7.5 Performing D/A Simultaneous Update

Sometimes it becomes necessary to have DA voltages occur on multiple channels at as near to exactly the same time as possible. For this, DAC chips have simultaneous modes. Once DASIM is set to 1, as many of any of the channels can be prepared to trigger voltage as desired; only when RMM1616DAUpdate() is called will the output voltages change on those channels that had DASIM set to 1 during their preparation.

The Universal Driver functions described here are: RMM1616DASetSim (), RMM1616DAConvert (), RMM1616DAUpdate ().

Step-By-Step Instructions

First the DASIM bit in the FPGA must be set to 1. This can be accomplished using RMM1616DASetSim(). Second, the DA values for all channels desired need to be sent to the DAC chip, using RMM1616DAConvert(). Third, RMM1616DAUpdate() must be run in order to trigger the DAC chip to simultaneously change all relevant DA outputs.

Example Usage for DA Simultaneous Update

```
RMM1616DASetSim (bi, 1);
RMM1616DAConvert (bi, 0, 65535);
RMM1616DAConvert (bi, 1, 65535);
RMM1616DAConvert (bi, 2, 65535);
RMM1616DAConvert (bi, 3, 65535);
RMM1616DAUpdate (bi);
RMM1616DAConvert (bi, 0, 0);
RMM1616DAConvert (bi, 1, 0);
RMM1616DAConvert (bi, 2, 0);
RMM1616DAConvert (bi, 3, 0);
RMM1616DAUpdate (bi);
```

On the oscilloscope, all the channels should instantaneously change from the full scale voltage (e.g.: 10V) to the low end of the scale voltage (e.g.: 0V or -10V) once the second RMM1616DAUpdate() call is performed. To see this precisely, configuring a single sequence capture function on the oscilloscope is useful.

7.6 Performing D/A Waveform Generation

Creating an analog voltage in freeform at high speeds with on-the-fly changes can be extremely useful. The driver supports configuring the Waveform Generator feature, loading it will values, functionality to change individual buffer values on-the-fly, using different incrementing clock sources, software incrementing, pausing, resetting, and in general starting D/A waveform generation.

The Universal Driver functions described here are: RMM1616WaveformBufferLoad(), RMM1616WaveformDataLoad(), RMM1616WaveformConfig(), RMM1616WaveformStart(), RMM1616WaveformPause(), RMM1616WaveformReset(), RMM1616WaveformInc()

Step-By-Step Instructions

First, it's recommended to configure the Waveform Generator; this can be done with RMM1616WaveformConfig(). If a counter/timer is selected as the clock, RMM1616WaveformConfig() automatically calls RMM1616CounterSetRate(). Then, load the buffer full of DA packets that contain a channel and DA value it should have, this can be done with RMM1661WaveformBufferLoad() and the associated structure, or RMM1616WaveformDataLoad() loading one packet at a time. After the buffer is loaded, one may wish to set up a counter/timer or input signal to be inputted to the board, so that the waveform can increment automatically. Otherwise, RMM1616WaveformInc() should be used to single-increment the waveform manually via software. After loading the buffer, and selecting input clock, one must start the Waveform Generator, with RMM1616WaveformStart(). If for some reason it is desirable to pause the waveform, use RMM1616WaveformPause(), followed by RMM1616WaveformStart() to resume. To start from the beginning of the buffer, run RMM1616WaveformReset(), followed by RMM1616WaveformStart().

Example for Setting up Waveform Generator with Step Wave Using Counter/Timer

```
RMM1616DASetSettings (bi, 0, 0, 0, 0);          // Set up D/A with desired
configuration
RMM1616WAVEFORM RMM1616waveform;
RMM1616waveform.Frames = 4;          // Four frames of size:
RMM1616waveform.FrameSize = 1;             // 1 value per frame
RMM1616waveform.Clock = 1;           // Input clock to use: Counter/Timer 0
RMM1616waveform.Rate = 100;          // 100Hz, uses RMM1616CounterSetRate
RMM1616waveform.Cycle = 1;
RMM1616WaveformConfig (bi, & RMM1616waveform);
RMM1616waveform.Waveform = (int*) malloc (sizeof (int) *4);
RMM1616waveform.Waveform [0] = 0;          // Position 0 data
RMM1616waveform.Waveform [1] = 16384;            // Position 1 data
RMM1616waveform.Waveform [2] = 32768;            // Position 2 data
RMM1616waveform.Waveform [3] = 0;          // Position 3 data
RMM1616waveform.Channels [0] = 0;                // Position 0 channel
RMM1616waveform.Channels [1] = 0;          // Position 1 channel
RMM1616waveform.Channels [2] = 0;          // Position 2 channel
RMM1616waveform.Channels [3] = 0;          // Position 3 channel
RMM1616WaveformBufferLoad (bi, &RMM1616waveform);     // Loads into D/A
                                                      //buffer
```

If in real-time, it's decided that position 3 will have 65535 instead:

```
RMM1616WaveformDataLoad (bi, 3, 0, 65535);

RMM1616WaveformStart (bi);                        // Waveform Generator
triggers using Counter0
```

Attaching an oscilloscope probe to DA channel 0 shows a step-wave function: low scale, 1/4 full-scale, 1/2 full-scale, then full-scale Voltage, repeating. The wave should increment between DA codes at 100Hz.

7.7 Performing Digital IO Operations

The driver supports four types of direct digital I/O operations: input bit, input byte, output bit, and output byte. Digital I/O is fairly straightforward - to perform digital input, you provide a pointer to the storage variable and indicate the port number and bit number if relevant. To perform digital output, you provide the output value and the output port and bit number, if relevant.

The five Universal Driver functions described here are RMM1616DIOConfig (), RMM1616DIOOutputByte (), RMM1616DIOInputByte (), RMM1616DIOOutputBit () and RMM1616DIOInputBit ().

Step-By-Step Instructions

If you are performing digital input, create and initialize a pointer to hold the returned value of type byte*.

Some boards have digital I/O ports with fixed directions, while others have ports with programmable directions. Make sure the port is set to the required direction, input or output. Use function RMM1616DIOConfig () to set the direction if necessary.

Call the selected digital I/O function. Pass it an int port value and either a pointer to or a constant byte digital value. If you are performing bit operations, then you will also need to pass in an int value telling the driver which particular bit (0-7) of the DIO port you wish to operate on.

Example of Usage for Digital I/O Operations

```
...
BoardInfo *bi;
int port, bit;
int digital_value;      // the value ranges from 0 to 255

/* 1. Configure Port 0 in output mode */
port = 0;
dir = 1;    //    0 = Input, 1 = Output
mode = 0;   //    0 = Normal, 1 = Latched
if ((result = RMM1616DIOConfig(bi, port, dir, mode)) != DE_NONE)
    return result;

/* 2. input bit - read bit 3 of port 0 (port 0 is in input mode) */
port = 0;
bit = 3;
if ((result = RMM1616DIOInputBit(bi, port, bit, &digital_value)) != DE_NONE)
    return result;

/* 3. input byte - read all 8 bits of port 0 (port 0 is in input mode) */
port = 0;
if ((result = RMM1616DIOInputByte(bi, port, &digital_value)) != DE_NONE)
    return result;

/* 4. output bit - 3 examples (assumes port 2 is in output mode) */
port = 2;
bit = 7;

/* 4a. RMM1616DIOOutputBit(), set bit 7 of port 2 to 1, leave other bits
alone */
if ((result = RMM1616DIOOutputBit(bi, port, bit, 1)) != DE_NONE)
    return result;
```

```
/* 5. output byte - set port 2 to "01010101" (port 2 is in output mode) */
port = 2;
if ((result = dscDIOOutputByte(dscb, port, 0x55)) != DE_NONE)
    return result;
...
```

7.8 Performing PWM Operations

The PWM operation generates a PWM signal with desired frequency and duty cycle value. The following functions are used for PWM operation: RMM1616PWMConfig () and RMM1616PWMStop ().

Step-By-Step Instructions

Create RMM1616PWM structure variable to hold PWM settings and initialize PWM structure variables, then call RMM1616PWMConfig () to configure the PWM and finally call RMM1616PWMStop ().

Application to stop the running PWM signal:

Example of Usage for PWM Operations

```
RMM1616PWM pwm; //structure to hold OWM settings
pwm.Num = 0; //select PWM channel
pwm.Rate = 100; // Select Output Frequency
pwm.Duty = 50; // Select Duty cycle value
pwm.Polarity = 0; // Select Polarity value
pwm.OutputEnable = 1; //Enable PWM output
pwm.Run = 1;
//The following function configures PWM circuit

if (RMM1616PWMConfig(bi,&pwm) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ("MPEGPIOPWMConfig error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}
printf ("Press any key to stop PWM \n");
while ( !kbhit()) //detects any key pressed
{
}
//The following function stops the selected PWM.

if (RMM1616PWMStop(bi,pwm.Num) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ("MPEGPIOPWMStop error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode),     errorParams.errstring );
      return 0;
}
```

7.9 Performing Counter Operations

The counter operation makes the counter to run in specified rate. The counter application uses following Universal driver function RMM1616CounterSetRate () and RMM1616CounterFunction ().

Step-By-Step Instructions

Create RMM1616CTR structure variable to hold Counter settings and initialize counter structure variables, then call RMM1616CounterSetRate () to configure the counter and finally call RMM1616CounterFunction () to stop the running counter.

Example of Usage for Counter Operations

```
RMM1616CTR counter; //Structure to hold counter settings
Int CtrNo =0 //Select counter Number
Float rate =1000 //Select counter rate
//The following function configures the counter with desired rate

if (RMM1616CounterSetRate(bi,counter.CtrNo,Rate,0,0) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ("RMM1616CounterConfig error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}
printf ("Press any key to stop counting \n");
while ( !kbhit())
{
}
counter.CtrCmd =RMM1616_COUNTER_CMD_RESET_ONE; //Reset the counter
if (RMM1616CounterFunction(bi,&counter) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ("RMM1616CounterFunction error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}
```

7.10 Performing User Interrupt Operations

The user interrupt application configures counter 0 to generate desired interrupt rate and at the same interrupt rate, a registered user interrupt function is called.

Step-By-Step Instructions

Create RMM1616USERINT structure variable to hold Interrupt settings and initialize Interrupt structure variables, then call RMM1616UserInterruptSet () to configure the counter and finally call RMM1616UserInterruptStop () to stop the running interrupt with Enable variable is equal to zero.

Example of Usage for User Interrupt Operations

```
int count =0;
void userfun (void *param)
{
   count ++;
}
Void main ()
{
      RMM1616USERINT inter; // structure to hold Interrupt settings
      inter.IntFunc=userfun;// Set user function as function pointer
      inter.Mode=0; // only mode 0is supported by RMM1616
      inter.Enable=1; // Enable interrupt
      inter.Source=0; // Select interrupt source 0-Counter o input 1=DIO
      //input
      //The following function configures user Interrupt.
      if (RMM1616UserInterruptSet(bi,&inter) !=DE_NONE)
      {
            dscGetLastError ( &errorParams );
            printf ( "RMM1616UserInterruptSet error: %s %s\n",
            dscGetErrorString ( errorParams.ErrCode ), errorParams.errstring
            );
            return 0;
      }
      Float rate =1000//Select Interrupt source frequency rate
      //Start counter with desired rate if counter is chosen
      if (RMM1616CounterSetRate(bi,0,rate,0,0) !=DE_NONE)
      {
            dscGetLastError ( &errorParams );
            printf ( "RMM1616CounterSetRate error: %s %s\n",
            dscGetErrorString ( errorParams.ErrCode ), errorParams.errstring
            );
            return 0;
      }
      printf("Press any key to terminate the application \n");
      while ( !kbhit())
      {
            printf("Count value %d \n ",count);
            dscSleep(1000);
      }
      inter.Enable=0; // Disable interrupt
      if (RMM1616UserInterruptSet(bi,&inter) !=DE_NONE)
      {
            dscGetLastError ( &errorParams );
            printf ( "RMM1616UserInterruptSet error: %s %s\n",
            dscGetErrorString ( errorParams.ErrCode ), errorParams.errstring
            );
```

```
            return 0;
        }
}
```

8. INTERFACE CONNECTOR DETAILS

8.1 RMM1616 Digital GPIO Connector – J5

Connector J5 brings the 48 digital I/O signals to a pin header. These lines have 3.3V logic levels with 5V input tolerance.

RMM1616 Digital GPIO Connector Pinout

DIO A0	**1**	**2**	DIO A1
DIO A2	**3**	**4**	DIO A3
DIO A4	**5**	**6**	DIO A5
DIO A6	**7**	**8**	DIO A7
DIO B0	**9**	**10**	DIO B1
DIO B2	**11**	**12**	DIO B3
DIO B4	**13**	**14**	DIO B5
DIO B6	**15**	**16**	DIO B7
DIO C0	**17**	**18**	DIO C1
DIO C2	**19**	**20**	DIO C4
DIO C4	**21**	**22**	DIO C6
DIO C6	**23**	**24**	DIO C7
DIO D0	**25**	**26**	DIO D1
DIO D2	**27**	**28**	DIO D3
DIO D4	**29**	**30**	DIO D5
DIO D6	**31**	**32**	DIO D7
DIO E0	**33**	**34**	DIO E1
DIO E2	**35**	**36**	DIO E3
DIO E4	**37**	**38**	DIO E5
DIO E6	**39**	**40**	DIO E7
DIO F0	**41**	**42**	DIO F1
DIO F2	**43**	**44**	DIO F3
DIO F4	**45**	**46**	DIO F5
DIO F6	**47**	**48**	DIO F7
+5VDC	**49**	**50**	DGND

Description: 20 position 1.25mm pitch right angle latching

8.2 RMM1616 Analog GPIO Connector – J4

Connector J4 brings the analog output signals to a pin header

Rmm1616 Analog GPIO Connector Pinout

VOUT 0	**1**	**2**	IOUT 0
AGND 3	**3**	**4**	VOUT 1
IOUT 1	**5**	**6**	AGND
VOUT 2	**7**	**8**	IOUT 2
AGND	**9**	**10**	VOUT 3
IOUT 3	**11**	**12**	AGND
VOUT 4	**13**	**14**	IOUT 4
AGND	**15**	**16**	VOUT 5
IOUT 5	**17**	**18**	AGND
VOUT 6	**19**	**20**	IOUT 6
AGND	**21**	**22**	VOUT 7
IOUT 7	**23**	**24**	AGND
VOUT 8	**25**	**26**	IOUT 8
AGND	**27**	**28**	VOUT 9
IOUT 9	**29**	**30**	AGND
VOUT 10	**31**	**32**	IOUT 10
AGND	**33**	**34**	VOUT 11
IOUT 11	**35**	**36**	AGND
VOUT 12	**37**	**38**	IOUT 12
AGND	**39**	**40**	VOUT 13
IOUT 13	**41**	**42**	AGND
VOUT 14	**43**	**44**	IOUT 14
AGND	**45**	**46**	VOUT 15
IOUT 15	**47**	**48**	AGND
EXT TRIG	**49**	**50**	DGND

Description: 0.1” pitch 50-pin (2x25) dual row right-angle pin header with gold flashing.

APPENDIX: REFERENCE INFORMATION

Counter/timer commands

The counter/timers are programmed with a series of commands. Each command is a 4 bit value. A command may have an associated option. A series of commands is required to configure a counter/timer for operation. See the counter/timer usage instructions in the manual for more information.

0 Clear the selected counter. If count direction is up the counter register is cleared to 0. If count direction is down the counter register is set to the reload value. All other counter settings are preserved. If the counter is running it continues running.

1 Load the selected counter with data in registers 0-3. This is used for down counting operations only.

2 Select count direction, up or down

3 Enable / disable external gate.

4 Enable / disable counting

5 Latch selected counter. A counter must be latched before its contents can be read. Latching can occur while the counter is counting. The latched data can be read with the RMM1616CounterRead() function.

6 Select counter clock source according to the table below:

Option	Clock source
0	External input pin, active low
1	Reserved
2	Internal clock 50MHz
3	Internal clock 1MHz

If an external DIO pin is selected as the counter input, the DIO pin's direction is automatically set for input mode. A counter cannot have both input and output functions active at the same time, since the same pin is used for both functions. If both are selected, the input function will prevail.

A counter must be enabled for the external input function to override the normal DIO operation. When one or more counters are reset with command 1111, any I/O pins tied to the counter or counters are released to normal DIO operation.

7 Enable / Disable Auto-Reload When auto-reload is enabled, then when the counter is counting down and it reaches 1, on the next clock pulse it will reload its initial value and keep counting. Otherwise on the next clock pulse it will count down to 0 and stop.

8 Enable counter output and select the output pulse polarity. The initial logic level of the output pin will be the inactive state (the opposite state of the selected polarity). The output pulse always

comes at the end of each clock period. The counter outputs are enabled on DIO pins according to the following table.

Enabling a counter output automatically sets the corresponding DIO pin's direction to output. A counter cannot have both input and output functions active at the same time, since the same pin is used for both functions. If both are selected, the input function will prevail, and the requested output function will be ignored.

Connector J5 Pin	DIO Bit	Counter/Timer
43	F2	0
44	F3	1

15 Reset one or all counters. When any counter is reset, all its registers are cleared to zero, and any DIO lines assigned to that counter for input or output are released to normal DIO operation. A command of 0xFF will reset all counters.

Each counter's output operates as follows. When disabled or during normal counting operation, the output is 0. When count direction is up, the output is always 0. When count direction is down, then when the counter reaches the selected pulse width, the output will go high. When the counter reaches 1, it will reload to its initial value on the next clock pulse. Thus a counter value of n will result in a divide by n output pulse rate. If a counter latch command is requested during this process, the command will be delayed until the reload is completed.

PWM commands

The PWMs are programmed with a series of commands. A command may have an associated parameter, referred to as PWMCD in the descriptions below. A series of commands is required to configure a PWM for operation. See the PWM usage instructions in the manual for more information.

0 Stop all PWMs / selected PWM

0 = stop all PWMs (opposite polarity for "all" compared to other PWM commands)

1 = stop single PWM

When a PWM is stopped, its output returns to its inactive state, and the registers are reloaded with their initial values. If the PWM is subsequently restarted, it will start at the beginning of its waveform, i.e. the start of the active output pulse.

1 Load counter C0 (period counter) or C1 (duty cycle counter)

0 = load C0 / period counter

1 = load C1 = duty cycle counter

2 Set output polarity. The pulse occurs at the start of the period.

0 = pulse high

1 = pulse low

3 Enable/disable pulse output

0 = disable pulse output; output = opposite of polarity setting from command 0010

1 = enable pulse output

4 Reset all PWMs / selected PWM

0 = reset PWM selected with PWM2-0

1 = reset all PWMs

When a PWM is reset, it stops running, and any DIO line assigned to that PWM for output is released to normal DIO operation. The direction of the DIO line will revert to its value prior to the PWM operation.

5 Enable/disable PWM outputs on DIO port F

0 = disable output

1 = enable output on DIO pin; this forces the DIO pin to output mode

6 Select clock source for a PWM

0 = 50MHz

1 = 1MHz

7 Start all PWMs / selected PWM

0 = start PWM selected with PWM2-0

1 = start all PWMs

If a PWM output is not enabled, its output is forced to the inactive state, which is defined as the opposite of the value selected with command 2. The PWM may continue to run even though its output is disabled.

PWM outputs may be made available on I/O pins according to the table below using command 5. When a PWM output is enabled, the corresponding DIO pin is forced to output mode. To make the pulse appear on the output pin, command 3 must additionally be executed, otherwise the output will be held in inactive mode (the opposite of the selected polarity for the PWM output).

Connector J5 Pin	DIO Bit	PWM
41	F0	0
42	F1	1
43	F2	2
44	F3	3