Universal Driver Software User Manual

Vega EMX Basic SBC

With Configurable COM Express CPU and On-board Data Acquisition

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 Vega SBC 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 rugged Vega EMX Basic SBC family includes interchangeable COM Express COMs for scalability and long product life, high feature density in a compact size, integrated high-quality data acquisition, expandable I/O, conduction cooled thermal solution for improved reliability, and rugged construction. Designed in the COM Express Basic form factor (125 x 95mm / 4.92 x 3.74 in), the Vega family offers performance scalability with three COM Express processor options: 2.1GHz Intel Core i7-3612QE, 1.7GHz Intel Core i7-3517UE, and 1.4GHz Intel Celeron 827E CPU. The use of interchangeable COM Express modules helps extend product lifecycles or keep up with new market needs by making it easy to replace an obsolete CPU or increase processing performance simply by replacing the COM module. Vega is therefore an excellent choice for applications with expected lifetimes of 10 or more years.

Vega's core embedded-PC functionality is implemented with a COM Express CPU module mounted on the bottom side of an I/O baseboard. This approach results in several benefits including enhanced thermal management, increased space for I/O functions and interface connectors, and scalable processing power. Accordingly, Vega integrates the equivalent functions of five embedded boards — CPU, system I/O, industry-leading data acquisition, Gigabit Ethernet, and a wide-input DC-to-DC power supply — all within the compact and modularly-expandable EMX Basic single board computer form-factor.

Vega is available with Intel 2.1GHz Intel Core i7 or 1.4GHz Intel Celeron processors. Vega's on-board EMX stackable I/O facilitates the addition of both custom and off-the-shelf I/O expansion modules to tune system functionality to the application's precise requirements.

Vega is offered in a range of models that vary according to the choice of COM Express CPU module, on-board SO-DIMM SDRAM capacity, variable-input DC/DC supply, and optional data acquisition circuitry.

2.2 Specifications

Processor and I/O

- 2.1GHz Intel Core i7-3612QE, 1.7GHz Intel Core i7-3517UE, or 1.4GHz Intel Celeron 827E CPU
- Up to 8GB SO-DIMM DDR3 SDRAM
- 2 Gigabit Ethernet ports
- 1 SATA port
- 4 USB 2.0 ports
- 4 RS-232/422/485 serial ports
- VGA, LVDS, DVI or HDMI video output
- HD audio
- mSATA and USB flash-disk socket supports up to 64GB

On-board Data Acquisition

- 16 16-bit A/D inputs usable as 16 single-ended or 8 differential
- Up to 250KHz maximum aggregate sampling rate
- Programmable input ranges: +/-10V, +/-5V, +/-2.5V, +/-1.25V, 0-10V, 0-5V, 0-2.5V
- 2048 A/D FIFO
- 8 16-bit D/A outputs
- Programmable output ranges: +/-10V, 0-10V
- 30 programmable bidirectional digital I/O lines with 3.3V logic
- 4 24-bit pulse width modulators
- 8-channel waveform generator
- 8 32-bit counter/timers
- On-board EEPROM storage of auto-calibration values

General

- 7-36V DC/DC power supply on-board
- Power consumption:
  - VEGA-3612QE-4GA -- 14.52W at 12V typical
  - VEGA-3517UE-4GA -- 13.93W at 12V typical
  - VEGA-827E-2GA -- 14.7W at 12V typical
- EMX stackable I/O expansion
- Shared PCIe MiniCard and mSATA socket
- Operating temperature: -40°C to +85°C
- Operating humidity: 5% to 95% non-condensing
- MIL-STD 202G shock and vibration compatible
- EMX Basic form factor 4.92” x 3.74” (125mm x 95mm)
- RoHS compliant

3. GENERAL PROGRAMMING GUIDELINES

3.1 Initialization and Exit Function Calls

All demo applications begin with the following functions. These should be called in sequence to initialize the Universal Driver and the Vega SBC. These functions should be called prior to any other Vega specific functions.

- dscInit ( ) : This function initializes the Universal Driver
- VEGAInitBoard () : This function initializes the Vega SBC
- DSCGetBoardInfo() : This function collects the board information from the Universal Driver and returns boardinfo structure to be used in the board specific functions

At the termination of the demo application the user should call the dscfree() function to close the file handler which is opened in dscInit() function.

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

```
#include "DSCUD_demo_def.h"
#include "vega.h"
ERRPARAMS errorParams; //structure for returning error code and error string
DSCCB dsccb; // structure containing board settings
BoardInfo *bi=NULL; //Structure containing board base address
VEGAINIT Init; // Structure containing board FPGA ID, Revision ID etc.

int main()
{
if ( (dscInit ( DSC_VERSION ) != DE_NONE) )
{
        dscGetLastError ( &errorParams );
        printf    (    "dscInit    error:    %s    %s\n",    dscGetErrorString    (
        errorParams.ErrCode), errorParams.errstring );
        return 0;
}
dsccb.boardtype = DSC_VEGA;
dsccb.base_address = 0x280;
dsccb.int_level = 5;

if ( VEGAInitBoard ( &dsccb,&Init) != DE_NONE )
{
        dscGetLastError (&errorParams);
        printf("VEGAInitBoard              error:              %s              %s\n",
        dscGetErrorString(errorParams.ErrCode),errorParams.errstring );
        return 0;
}
bi = DSCGetBoardInfo ( dsccbp.boardnum );

/* Application code goes here */
dscFree ( );
return 0;
}
```

In the example above, DSC_VERSION, DSC_VEGA, and DE_NONE are macros defined in the included header file, dscud.h file.

3.2 Error handling

All 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, the last error can be checked by calling the function dscGetLastError(). This function takes an ERRPARAMS structure pointer as its argument.

Nearly all of 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 as to whether or not the function call was successful. Users should always check if the result returns a DE_NONE value (signifying that no errors were reported), as the code below illustrates:

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

In the above 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). Anytime a function is not successfully executed, an error code other than DE_NONE will be generated and the current API function will terminate. The function dscGetErrorString() provides a description of the error that occurred.

4. UNIVERSAL DRIVER API DESCRIPTION

4.1 VEGAADSetSettings

Function Definition
BYTE VEGAADSetSettings (BoardInfo* bi, VEGAADSETTINGS* settings);

Function Description
This function configures the A/D input range, channel register, and scan settings. It can optionally recall the A/D calibration settings for the selected input range.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGAADSETTINGS	Gain	int	0-3: 0 = 1, 1 = 2, 2 = 4, 3 = 8
	Polarity	int	0-1; 0 = bipolar, 1 = unipolar
	Sedi	int	0-1; 0 = single-ended, 1 = differential
	Lowch	int	low channel, 0-15
	Highch	int	high channel, 0-15
	LoadCal:	bool	True/False
	ScanEnable	int	0 = disable, 1 = enable
	ScanInterval	int	0 = 10us, 1 = 5us, 2 = 8us, 3 = programmable
	ProgInt	int	125-255, used if Interval = 3

Return Value
Error code or 0.

Usage Example
To configure channel zero in unipolar 5V range single ended input mode and scan disabled,

```
VEGAADSETTINGS vegaadsettings;
vegaadsettings.Polarity = 1;
vegaadsettings.Gain= 1;
vegaadsettings.Sedi = 0;
vegaadsettings.Highch = 0;
vegaadsettings.Lowch = 0;
vegaadsettings.ADClock = 0;
vegaadsettings.ScanEnable = 0;
VEGAADSetSettings (bi, vegaadsettings);
```

4.2 VEGAADSetRange

Function Definition
BYTE VEGAADSetRange(BoardInfo* bi, VEGAADSETTINGS* settings);

Function Description
This function configures the A/D input range. All other settings remain the same.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGAADSETTINGS	Gain	int	0-3: 0 = 1, 1 = 2, 2 = 4, 3 = 8
	Polarity	int	0-1; 0 = bipolar, 1 = unipolar
	Sedi	int	0-1; 0 = single-ended, 1 = differential

Return Value
Error code or 0.

Usage Example
To configure channel 0 in 0-5v single ended and leave the other A/D settings untouched,

```
VEGAADSETTINGS vegaadsettings;
vegaadsettings.Gain = 1;
vegaadsettings.Polarity = 1;
vegaadsettings.Sedi = 0;
VEGAADSetSettings (bi, vegaadsettings);
```

4.3 VEGAADSetChannel

Function Definition
BYTE VEGAADSetChannel (BoardInfo* bi, VEGAADSETTINGS* settings);

Function Description
This function configures the A/D input channel range. All other settings remain the same.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGAADSETTINGS	Lowch Highch	int int	low channel, 0-15 high channel, 0-15

Return Value
Error code or 0.

Usage Example
To configure low and high channels as 5 and 6 with all other settings remaining the same,

```
VEGAADSETTINGS vegaadsettings;
vegaadsettings.Highch = 3;
vegaadsettings.Lowch = 0;
VEGAADSetSettings (bi, vegaadsettings);
```

4.4 VEGAADSetScan

Function Definition
BYTE VEGAADSetScan (BoardInfo* bi, VEGAADSETTINGS* settings);

Function Description
This function configures the A/D logic for scan operation. It does not configure any other settings on the board.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGAADSETTINGS	Scan Enable	int	0 = disable, 1 = enable
	ScanInterval	int	0 = 10us, 1 = 5us, 2 = 8us, 3 = programmable
	ProgInt	int	125-255, used if Interval = 3

Return Value
Error code or 0.

Usage Example
To configure a 5us scan interval with all other settings the same,

```
VEGAADSETTINGS vegaadsettings;
vegaadsettings.ScanEnable = 1;
vegaadsettings.ScanInterval= 1;
vegaadsettings.ProgInt= 0;
VEGAADSetSettings (bi, vegaadsettings);
```

4.5 VEGAADSetClock

Function Definition
BYTE VEGAADSetClock(BoardInfo* bi, BYTE ADclk);

Function Description
This function configures the clock source for A/D conversions.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
VEGAADSETTINGS	ADclk int 0-3

Return Value
Error code or 0.

Usage Example
To configure clock source as 2 i.e. counter 0 output for A/D conversions,

```
int ADclk=2;
VEGAADSetClock (bi, ADclk);
```

4.6 VEGAADEnableClock

Function Definition
BYTE VEGAADEnableClock (BoardInfo* bi);

Function Description
This function enables the clock source for A/D conversions function parameters.

Return Value
Error code or 0.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Usage Example
To enable the clock which is configured using VEGAADSetClock() function

```
VEGAADEnableClock (bi);
```

4.7 VEGAADStopClock

Function Definition
BYTE VEGAADStopClock (BoardInfo* bi);

Function Description
This function configures the clock source for A/D conversions.

Return Value
Error code or 0.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Usage Example
To stop the clock which is configured using VEGAADSetClock() function,

```
VEGAADStopClock (bi);
```

4.8 VEGAADSample

Function Definition

BYTE VEGAADSample (BoardInfo* bi, unsigned* Sample);

Function Description

This function executes one A/D conversion using the current board settings. It does not perform any configuration of the board or the FPGA but rather uses the current settings. It assumes that the board is configured for sample mode, not scan mode.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
VEGAADSample	Return value, 16 bits; may be interpreted as unsigned integer 0 – 65535, or signed integer -32768 – 32767, depending on the A/D configuration and software expectations

Return Value

Error code or 0.

Usage Example

To do an A/D conversion with the preconfigured A/D settings using the VEGAADSetSettings() function,

```
VEGAADSAMPLE sample;
VEGAADSample (bi, & sample);
printf ("A/D sample value = %d ",*sample.ADsample );
```

4.9 VEGAADScan

Function Definition
BYTE VEGAADScan (BoardInfo* bi, unsigned* Sample);

Function Description
This function executes one A/D scan (sample on all channels between Lowch and Highch). It uses all existing A/D settings on the board (input range).

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
VEGAADSample	sample unsigned int 0 - 65535 or -32768 – 32767 depending on the A/D configuration

Return Value
Error code or 0.

Usage Example
To do A/D scan operation with preconfigured A/D settings using VEGAADSetSettings() function,

```
int index = 0 ;
VEGAADSAMPLE sample [16];
VEGAADScan (bi, sample);
for (index =  lowch ; index <=Highch ; index++)
{
      printf ("A/D CH%d sample value = %d\n",*sample[index].ADsample );
}
```

4.10 VEGAADInt

Function Definition
BYTE VEGAADInt (BoardInfo* bi, VEGAADINT* vegaadint);

Function Description
This function enables the A/D interrupt operation using the current analog input settings. It configures the FIFO and the clock source on the board.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGAADInt	FIFOEnable	int	0 = disable; interrupt occurs after each sample / scan; 1 = enable; interrupt occurs on FIFO threshold flag
	FIFOThreshold	int	0-2048, indicates level at which FIFO will generate an interrupt if FIFOEnable = 1
	Cycle	int	0 = one shot operation, interrupts will stop when the selected no. of samples / scans are acquired; 1 = continuous operation until terminated
	ADClk	int	0-3 selects A/D clock source
	NumConversions	int	if Cycle = 0 this is the number of samples / scans to acquire; if Cycle = 1 this is the size of the circular buffer in samples / scans
	ADBuffer	unsigned *	pointer to A/D buffer to hold the samples; the buffer must be greater than or equal to NumConversions x 1 if scan is disabled or NumConversions x Scansize if scan is enabled (Scansize is Highch – Lowch + 1)

Return Value
Error code or 0.

Usage Example
To perform A/D sampling in interrupt mode,

```
char buffer[];
VEGAINIT vegaadint;
vegaadint.FIFOEnable=1;
vegaadint.FIFOThreshold=1600;
vegaadint.Cycle=1;
vegaadint.ADClk=0;
vegaadint.NumConversions=1600;
vegaadint.ADBuffer= (SWORD*) malloc (sizeof
(SWORD)*dscIntSettings.NumConversions);
VEGAADInt (bi, &vegaadint);
```

4.11 VEGAADIntStatus

Function Definition
BYTE VEGAADIntStatus (BoardInfo* bi, VEGAADINTSTATUS* intstatus);

Function Description
This function returns the interrupt routine status including, running / not running, number of conversions completed, cycle mode, FIFO status, and FIFO flags.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGAADIntStatus	OpStatus	int	0 = not running, 1 = running
	NumConversions	int	Number of conversions since interrupts started
	Cycle	int	0 = one-shot operation, 1 = continuous operation
	FIFODepth	int	Current FIFO depth pointer
	UF, OF, FF, TF, EF	int	FIFO flags

Return Value
Error code or 0.

Usage Example
```
VEGAADINTSTATUS intstatus;
VEGAADIntStatus (bi, & intstatus);
printf ("No of A/D conversions completed %d\n",intstatus.NumConversions);
```

4.12 VEGAADIntPause

Function Definition
BYTE VEGAADIntPause (BoardInfo* bi);

Function Description
This function pauses A/D interrupts by turning off the interrupt enable and stopping the A/D clock. This holds the A/D channel counter and FIFO at their current positions. Interrupts may be resumed from the point at which they were stopped with the Resume function.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
To pause A/D interrupts,

```
VEGAADIntPause(bi);
```

4.13 VEGAADIntResume

Function Definition
BYTE VEGAADIntResume (BoardInfo* bi);

Function Description
This function resumes A/D interrupts from the point at which they were paused. This function cannot be used to initiate interrupt operations because it does not set up the board or the driver to handle interrupts.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
To resume A/D interrupts,

```
VEGAADIntResume (bi);
```

4.14 VEGAADIntCancel

Function Definition
BYTE VEGAADIntCancel (BoardInfo* bi);

Function Description
This function stops A/D interrupts by turning off the interrupt enable, stopping the A/D clock, and removing the A/D interrupt handler. This function is the same as VEGAADIntPause () except that the interrupt handler is removed.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
To cancel A/D interrupts,

```
VEGAADIntCancel (bi);
```

4.15 VEGADASetSettings

Function Definition
BYTE VEGADASetSettings (BoardInfo* bi, VEGADASETTINGS* settings);

Function Description
This function works with the AD5360 / AD5362 D/A converter. This function sets the D/A output ranges for channel group A (channels 0-3) and B (channels 4-7), the data format, and the update mode. Each group of 4 channels will have the same output range.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGADASetSettings	RangeA	int	0-3
	RangeB	int	0-3
	PolarityA	int	0 = bipolar, 1 = unipolar
	PolarityB	int	0 = bipolar, 1 = unipolar
	Mode	int	0 = binary, 1 = 2s complement
	Simultaneous	int	0 = single update, 1 = simultaneous mode

Return Value
Error code or 0.

Usage Example
To configure D/A Group A in +/-10V range and Group B in 0-10V range,

```
VEGADASETTINGS dasettings;
settings.RangeA=0;
settings.RangeB=0;
settings.PolarityA=0;
settings.PolarityB=1;
settings.Mode =0;
settings.Simultaneous =0;
VEGADASetSettings (bi, & dasettings);
```

4.16 VEGADAConvert

Function Definition
BYTE VEGADAConvert(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 DASetSettings(). If the board is not set for simultaneous update mode (DASIM = 0), the channel will be updated immediately. 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		
VEGADAConvert	Channel DACode	int unsigned	0-7 0-65535 if D/A is set for binary (B2C = 0) or -32768 – 32767 if D/A is set for 2s complement (B2C = 1).

Return Value
Error code or 0.

Usage Example
To generate 10V on D/A channel 0 using the VEGADASetSettings() function,

```
int channel=0;
unsigned int DACode=65535;
VEGADAConvert (bi,channel, DACode );
```

4.17 VEGADAConvertScan

Function Definition
BYTE VEGADAConvertScan(BoardInfo* bi, VEGADASCAN* dacs);

Function Description
This function outputs multiple values to multiple D/A channels. The output ranges must be previously set with DASetSettings(). 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.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGADAConvertScan	ChannelSelect	int *	Array of 8 values: 0 = don't update channel, 1 = update channel
	DACodes	unsigned *	Array of 8 values: 0-65535 if D/A is set for binary (B2C = 0) or -32768 – 32767 if D/A is set for 2s complement (B2C = 1).

Return Value
Error code or 0.

Usage Example
To update channel 0, 1 with dacode 65535, 32768 respectively and rest of the channel unchanged from the existing voltage level,

```
VEGADASCAN Scan;
Scan.ChannelSelect = (int *) malloc (sizeof (int) *16);
Scan.DACodes         = (int *) malloc (sizeof (int) * 16);
Scan.ChannelSelect [0] = 1;
Scan.ChannelSelect [1] = 1;
Scan.DACodes [0] = 65535;
Scan.DACodes [1] = 32768
VEGADAConvertScan (bi,&Scan);
```

4.18 VEGADAFunction

Function Definition
BYTE VEGADAFunction (BoardInfo* bi, unsigned DAData, int DACommand);

Function Description
This function enables the user to control the D/A chip directly to implement special functions that are not supported by other Universal Driver functions. These special functions are listed in Table 16 of the AD5362 datasheet. This function is general purpose and does not take any special action other than to write the command and data to the chip.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGADAFunction	DAData DACommand	unsigned int	16-bit value in straight binary 0000-FFFF 8-bit value in straight binary 00-FF

Return Value
Error code or 0.

Usage Example
To perform D/A convert operation by using VEGADAFunction,

```
DACode = 65535;
Channel = 2;
DAcommand = 0x10 + (1<<channel); //DA update command
VEGADAFunction (bi,DACode,DAcommand);
```

4.19 VEGADAUpdate

Function Definition
BYTE VEGADAUpdate(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 DAConvertScan function. In this case the programmer is using DAConvert to configure channels 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
To update D/A channel 0 and 1 with 65535 and 32768 respectively at the same time,

```
VEGADASETTINGS dasettings;
settings.RangeA=0;
settings.RangeB=0;
settings.PolarityA=0;
settings.PolarityB=1;
settings.Mode =0;
settings.Simultaneous =1;
VEGADASetSettings (bi, & dasettings);
VEGADAConvert (bi, 0, 65535);
VEGADAConvert (bi, 1, 32768);
VEGADAUpdate (bi);
```

4.20 VEGADAMonitorSelect

Function Definition
BYTE VEGADAMonitorSelect(BoardInfo* bi, int MonEnable, int DAChannel);

Function Description
This function configures the A/D circuit and the D/A chip for the readback of D/A outputs. The D/A chip has a built-in monitor function that enables the readback of any of its analog outputs plus two analog inputs via a separate analog output “monitor” channel. This monitor output is fed into the A/D circuit via the SE/Diff selection circuit. The two monitor inputs are tied to analog ground, enabling them to be used to measure the built-in offset of the monitor circuit which can help during D/A calibration.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGADAMonitorSelect	MonEnable	int	0 = disable the function and return to normal A/D operation; 1 = enable the function
	DAChannel	int	valid range 0-9; 0-7 select the analog outputs 0-7; 8-9 select the monitor inputs 0-1

Return Value
Error code or 0.

Usage Example
To enable read back of DA channel 1 to an A/D circuit,

```
int MonEnable=1;
int DAChannel=1;
VEGADAMonitorSelect (bi, MonEnable,  DAChannel);
```

4.21 VEGADARead

Function Definition
BYTE VEGADARead (BoardInfo* bi, unsigned Register, unsigned* Value);

Function Description
This function enables the readback of internal registers in the AD5362 D/A converter.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGADARead	Value Register	unsigned int	16-bit return value in straight binary 0000-FFFF Selects which D/A register to read back; see AD5362 datasheet Table 17 for list of registers

Return Value
Error code or 0.

Usage Example
To read M register of channel 0,

```
int value = 0 ;
VEGADARead (bi, 0x6400, & value);
Printf (value = 0x%x ", value);
```

4.22 VEGAWaveformBufferLoad

Function Definition
BYTE VEGAWaveformBufferLoad(BoardInfo* bi, VEGAWAVEFORM vegawaveform);

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

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGAWaveformBufferLoad	Waveform	unsigned *	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	int	total number of frames in the array
	Framesize	int	number of channels to be driven = frame size
	Channels	int *	List of channels to be driven by the waveform generator; the number of values in this array must be equal to Framesize; the channels can be in any sequence

Return Value
Error code 0.

Usage Example
To generate a square wave on channel zero with four D/A values,

```
VEGAWAVEFORM waveform;
waveform.Frames        = 4;
waveform.Waveform = (unsigned int*) malloc (waveform. Frames * sizeof (unsigned
                      int)*4);
waveform.Waveform [0] = 0;
waveform.Waveform [1] = 0;
waveform.Waveform [2] = 65535;
waveform.Waveform [3] = 65535;
waveform.Channels [0] =0;
VEGAWaveformBufferLoad (bi, & waveform);
```

4.23 VEGAWaveformConfig

Function Definition
BYTE VEGAWaveformConfig (BoardInfo* bi, VEGAWAVEFORM* VEGAwaveform);

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 frame size is needed because it is stored in the same board register as the clock source and cycle mode.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGAWaveformConfig	Frames	int	total number of frames in the array
	Framesize	int	no. of channels to be driven = frame size
	Clock	int	0 = software increment; 1 = counter/timer 0 output; 2 = counter/timer 1 output; 3 = DIO pin D0
	Rate	float	frame update rate, Hz (only used if Clock = 1 or 2)
	Cycle	int	0 = one-shot operation; 1 = repetitive

Return Value
Error code or 0.

Usage Example
To configure the waveform generator with 500 frames using the clock as the software increment,

```
VEGAWAVEFORM waveform;
waveform.Frames = 500;
waveform.Framesize = 1
waveform.Clock = 0;
VEGAWaveformConfig (bi, &waveform);
```

4.24 VEGAWaveformStart

Function Definition
BYTE VEGAWaveformStart (BoardInfo* bi);

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

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
To start or restart the waveform generator,

```
VEGAWaveformStart (bi);
```

4.25 VEGAWaveformDataLoad

Function Definition
BYTE VEGAWaveformDataLoad(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	int	Address in buffer at which to store the data; 0-2047;
Channel	int	Channel number, 0-3
Value	unsigned	Data value, 0-65535

Return Value
Error code or 0.

Usage Example
To load value 65535 into the address 2000 of channel 0 of waveform generator,

```
Address=2000;
Channel=0;
Value=65535;
VEGAWaveformDataLoad (bi, Address, Channel, Value);
```

4.26 VEGAWaveformPause

Function Definition
BYTE VEGAWaveformPause (BoardInfo* bi);

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

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
To stop the waveform generator,

```
VEGAWaveformPause (bi);
```

4.27 VEGAWaveformReset

Function Definition
BYTE VEGAWaveformReset (BoardInfo* bi);

Function Description
This function resets the waveform generator.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
To reset the waveform generator,

```
VEGAWaveformReset (bi);
```

4.28 VEGAWaveformInc

Function Definition
BYTE VEGAWaveformInc (BoardInfo* bi);

Function Description
This function increments the waveform generator by one frame. The current frame of data is output to the selected channels associated with 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
To increment wave generator by one frame,

```
VEGAWaveformInc (bi);
```

4.29 VEGADIOConfig

Function Definition
BYTE VEGADIOConfig (BoardInfo* bi, int Port, int Config);

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

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGADIOConfig	Port Config	int int	0-3 for port A, B, C, or D 8-bit configuration values for selected port

Return Value
Error code or 0.

Usage Example
To set Port A in input mode,

```
Port = 0;
Config = 0;
BYTE VEGADIOConfig (bi, Port, Config);
```

4.30 VEGADIOConfigAll

Function Definition
BYTE VEGADIOConfigAll (BoardInfo* bi, int* Config);

Function Description
This function sets the digital I/O port directions for all ports at once.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
VEGADIOConfigAll	Config int * pointer to 4 8-bit configuration values for ports A, B, C, D

Return Value
Error code or 0.

Usage Example
To set all the digital I/O port's direction to input mode,

```
int *config;
config = (int *) malloc (sizeof (int)* 4);
config [0] = 0;
config [1] = 0;
config[2]  = 0;
config [3] = 0;
VEGADIOConfigAll (bi, config);
```

4.31 VEGADIOOutputByte

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

Function Description
This function outputs the specified data to the specified port. The data is 8 bits for ports A, B, and C and 6 bits for port D.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGADIOOutputByte	Port Data	int int	0 = A, 1 = B, 2 = C, 3 = D 6 or 8 bit value to write to the port; for port D only the lower 6 bits are used, the upper 2 are ignored by the FPGA

Return Value
Error code or 0.

Usage Example
To set Port 0 output as 0x77,

```
port=0;
Data=0x77;
VEGADIOOutputByte (bi, port, Data);
```

4.32 VEGADIOInputByte

Function Definition
BYTE DSCUDAPICALL VEGADIOInputByte (BoardInfo* bi, int port, BYTE* data);

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

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGADIOInputByte	Port Data	int int *	0 = A, 1 = B, 2 = C, 3 = D pointer to receive the data read from the port

Return Value
Error code or 0.

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

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

4.33 VEGADIOOutputBit

Function Definition
BYTE DSCUDAPICALL VEGADIOOutputBit (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		
VEGADIOOutputBit	Port	int	Port to write bit to: 0 = A, 1 = B, 2 = C, 3 = D
	Bit	int	0-7 indicates the bit position in the port (for port D the value should be 0-5)
	Value	int	0 or 1

Return Value
Error code or 0.

Usage Example
To set Port 0, bit 6 to 1,

```
int port=0;
int bit=6;
int digital_val =1;
VEGADIOOutputBit (bi, port, bit, digital_val);
```

4.34 VEGADIOInputBit

Function Definition
BYTE VEGADIOInputBit (BoardInfo* bi, int Port, int Bit, int* Value);

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

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGADIOInputBit	Port	int	0 = A, 1 = B, 2 = C, 3 = D
	Bit	int	0-7 for ports A, B, or C, 0-5 for port D
	Value	int *	pointer to receive the bit data; return data is always

Return Value
Error code or 0.

Usage Example
To read the value at port 0 bit 5,

```
int Port=0;
int Bit=5;
VEGADIOInputBit (bi, port, bit, &value);
printf ("The value at Port 0 Bit 6 is %d", value);
```

4.35 VEGACounterSetRate

Function Definition
BYTE VEGACounterSetRate (BoardInfo* bi, VEGACOUNTER *Ctr);

Function Description
This function programs a counter for timer mode with down counting. The output may be used for A/D or D/A timing. The output may also be enabled on a DIO pin. The counter is started immediately.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGACounterSetRate	CtrNo	int	Counter number, 0-7
	CtrData	uns. long	Initial load data, 32-bit straight binary
	CtrClk	int	Clock source, Must be 2 or 3 (see FPGA specification for usage)
	CtrOutEn	int	1 = enable output onto corresponding I/O pin; 0 = disable output
	CtrOutPol	int	1 = output pulses high, 0 = output pulses low; only used if CtrOutEn = 1
	CtrOutWidth	int	0 = 1 clock, 1 = 10 clocks, 2 = 100 clocks, 3 = 1000 clocks; only used if CtrOutEn = 1 and CtrClk = 2 or 3

Return Value
Error code or 0.

Usage Example
To configure counter 0 with 100Hz, output enabled, polarity high and an output pulse width of 1000 clocks,

```
VEGACOUNTER counter;
counter.CtrNo = 0;
counter.Rate = 100;
counter.CtrOutEn = 1;
counter.CtrOutPol = 1;
counter.ctrOutWidth =3;
VEGACounterSetRate (bi, &counter);
```

4.36 VEGACounterConfig

Function Definition
BYTE VEGACounterConfig (BoardInfo* bi, VEGACTR *Ctr);

Function Description
This function programs a counter for up or down counting and starts the counter running. The output is not enabled on a DIO pin, but a DIO pin may be selected as the input source.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGACounterConfig	Ctrno	int	Counter number, 0-7
	CtrData	uns. long	Initial load data, 32-bit straight binary
	CtrClk	int	Clock source, 0-3 (see FPGA specification for usage)
	CtrCountDir	int	0 = down counting, 1 = up counting
	CtrReload	int	0 = one-shot counting, 1 = auto-reload (repetitive counting, only works in countdown mode)

Return Value
Error code or 0.

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

```
VEGACOUNTER counter;
counter.Ctrno = 0;
counter.CtrClock = 2; //50MHz
counter.CtrData = 50000000/100;
counter.CtrCountDir = 0;
counter.CtrReload = 1;
counter.CtrOutEn = 0;
VEGACounterConfig (bi, &counter);
```

4.37 VEGACounterRead

Function Definition
BYTE VEGACounterRead (BoardInfo* bi, int Ctrno, Unsigned Long * CtrData);

Function Description
This function latches a counter and reads the value. The counter does not stop counting. This command can be executed at any time on any counter.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
VEGACounterRead	This function latches a counter and reads the value. The counter does not stop counting. This command can be executed at any time on any counter.

Return Value
Error code or 0.

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

```
unsigned long CtrData;
Ctrno = 0;
VEGACounterRead (bi, &counter);
printf ("Counter Data %ld \r", counter.CtrData);
```

4.38 VEGACounterReset

Function Definition
BYTE VEGACounterReset (BoardInfo* bi, int CtrNum);

Function Description
This function can be used to reset the counter.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
CtrNum	int Counter number, 0-7

Return Value
Error code 0.

Usage Example
To reset counter 0 only,

```
CtrNum=0;
VEGACounterReset (bi, CtrNum)
```

4.39 VEGACounterFunction

Function Definition
BYTE VEGACounterFunction (BoardInfo* bi, VEGACTR* Ctr);

Function Description
This function can be used to program any desired function into a counter, except reading, which is done by VEGACounterRead. The counters have a set of commands used to configure them. Configuration generally requires the execution of multiple commands. Each call to this function can execute a single command.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGACounterFunction	Ctrno	int	Counter number, 0-7
	CtrData	uns. long	Initial load data, 32-bit straight binary
	CtrCmd	int	Counter command, 0-15 (see FPGA specification for available commands)
	CtrCmdData	int	Auxiliary data for counter command, 0-3 (see FPGA specification for usage)

Return Value
Error code 0.

Usage Example
To reset counter 0 only,

```
VEGACOUNTER counter;
counter.Ctrno = 0;
counter.CtrCmd = 0;
counter.CtrCmdData = 0xF;
VEGACounterFunction (bi, &counter);
```

4.40 VEGAPWMConfig

Function Definition
BYTE VEGAPWMConfig (BoardInfo* bi, VEGAPWM* vegapwm);

Function Description
This function configures a pulse width modulator (PWM) for operation. It can optionally start the PWM running.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGAPWMConfig	Num	int	PWM number, 0-3
	Rate	float	output frequency in Hz
	Duty	float	initial duty cycle, 0-100
	Polarity	int	0 = pulse high, 1 = pulse low
	OutputEnable	int	0 = disable output, 1 = enable output on DIO pin
	Run	int	0 = don't start PWM, 1 = start PWM

Return Value
Error code or 0.

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

```
VEGAPWM pwm;
PWM.Num = 0;
PWM.Rate = 100;
PWM.Duty = 50;
PWM.Polarity = 1;
PWM.OutputEnable   = 1;
VEGAPWMConfig (bi, &pwm);
```

4.41 VEGAPWMStart

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

Function Description
This function starts a PWM running.

Function Parameters

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

Return Value
Error code or 0.

Usage Example
To start PWM 0,

```
pwm.Num = 0;
VEGAPWMStart (bi, pwm. Num);
```

4.42 VEGAPWMStop

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

Function Description
This function stops a PWM.

Function Parameters

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

Return Value
Error code or 0.

Usage Example
To stop PWM 0,

```
 pwm.Num = 0;
VEGAPWMStop (bi, pwm.Num)
```

4.44 VEGAPWMReset

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

Function Description
This function resets a PWM.

Function Parameters

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

Return Value
Error code or 0.

Usage Example
To reset PWM 0,

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

4.45 VEGAPWMCommand

Function Definition
BYTE VEGAPWMCommand (BoardInfo* bi, VEGAPWM* vegapwm);

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

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGAPWMCommand	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	int	24-bit value for use with period and duty cycle commands

Return Value
Error code or 0.

Usage Example
To reset all PWMs,

```
VEGAPWM pwm;
pwm.Command = 0x4; //Stop PWM
pwm.CmdData = 0x01; //Stop all PWM
BYTE VEGAPWMCommand (bi, &pwm);
```

4.46 VEGAUserInterruptSet

Function Definition
BYTE VEGAUserInterruptSet (BoardInfo* bi, VEGAUSERINT* vegauserint);

Function Description
This function installs a pointer to a user function that runs when an interrupt occurs. It configures the interrupt handler to run the user function either before or after the standard interrupt function or in standalone mode.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGAUserInterruptSet	IntFunc	void *	pointer to user function to run when interrupts occur
	Mode	int	0 = alone, 1 = before standard function, 2 = after standard function
	Source	int	0 = counter/timer 2 output, 1 = digital input D1; used only if mode = 0
	Enable	int	0 = disable interrupts, 1 = enable interrupts

Return Value
Error code or 0.

Usage Example
To configure userfun () as the user defined function which gets invoked on interrupt raised,

```
void userfun (void *param)
{
      count ++;
}
inter.IntFunc=userfun;
inter.Mode=0;
inter.Enable=1;
inter.Source=0;
VEGAUserInterruptSet (bi, &inter);
```

4.47 VEGAUserInterruptStop

Function Definition
BYTE VEGAUserInterruptStop (BoardInfo* bi, VEGAUSERINT* vegauserint);

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. This function is identical to VEGAUserInterruptSet with Enable = 0.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGAUserInterruptStop	IntFunc	void *	pointer to user function to run when interrupts occur
	Enable	int	0 = disable interrupts, 1 = enable interrupts

Return Value
Error code or 0.

Usage Example
To stop a user interrupt,

```
inter.Enable=0;
VEGAUserInterruptStop (bi, &inter);
```

4.48 VEGAInitBoard

Function Definition
BYTE VEGAInitBoard (DSCCB* dsccb);

Function Description
This function initializes the board.

Function Parameters

Name	Description
dsccb	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
To initialize the board with I/O address 0x280 and interrupt number 5,

```
DSCCB dsccb; // structure containing board settings
dsccb.boardtype = DSC_VEGA;
dsccb.base_address = 0x280;
dsccb.int_level = 5;
VEGAInitBoard (DSCCB* dsccb);
```

4.49 VEGAFreeBoard

Function Definition
BYTE VEGAFreeBoard (DSCB board);

Function Description
This function stops any active interrupt processes and frees the interrupt resources assigned to a board. It then decrements the driver's count of the number of active I/O boards under its control. Next it calls DSCFreeBoardSubSys () to remove the board handle from the driver's list of boards. Finally this function can execute any specific code needed for this board.

Function Parameters

Name	Description
board	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
```
VEGAFreeBoard (DSCB board);
```

4.50 VEGAFIFOStatus

Function Definition
BYTE VEGAFIFOStatus (BoardInfo* bi, VEGAFIFO* vegafifo);

Function Description
This function returns the current FIFO depth and status flags in the parameter array.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGAFIFO	Threshold	int	Current FIFO programmed threshold
	Depth	int	Current FIFO depth pointer
	Enable	int	0 = FIFO is not currently enabled, 1 = FIFO is currently enabled
	UF	int	0 = no underflow, 1 = FIFO underflow (attempt to read was made when FIFO was empty)
	OF	int	0 = no overflow, 1 = FIFO overflow (attempt to write into FIFO when FIFO was full)
	FF	int	0 = FIFO not full, 1 = FIFO is full
	TF	int	0 = number of A/D samples in FIFO is less than the programmed threshold, 1 = number of A/D samples in FIFO is equal to or greater than the programmed threshold
	EF	int	0 = FIFO has unread data in it, 1 = FIFO is empty

Return Value
Error code or 0.

Usage Example
To read current FIFO status,

```
VEGAFIFO vegafifo;
VEGAFIFOStatus (bi, &vegafifo);
Printf ("Depth = %d \n", vegafifo. Depth);
printf ("Overflow flag = %d \n", vegafifo.OF);
```

4.51 VEGAEEPROMRead

Function Definition
BYTE VEGAEEPROMRead (BoardInfo* bi, int Address, int* Data);

Function Description
This function reads the 8-bit data from the data acquisition EEPROM at the given address.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGAEEPROMRead	Address Data	int int	EEPROM address, 0-511 EEPROM data read from the EEPROM

Return Value
Error code or 0.

Usage Example
To read the value at EEPROM address 32,

```
int Data = 0
int Address=32;
VEGAEEPROMRead (bi, Address, & Data);
Printf ("The value at address %d is %d \n",Address,Data);
```

4.52 VEGAEEPROMWrite

Function Definition
BYTE VEGAEEPROMWrite (BoardInfo* bi, int Address, int Data);

Function Description
This function writes the given 8-bit data to the data acquisition EEPROM at the given address. It does not allow access to the protected area that holds the backup calibration information.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
VEGAEEPROMRead	Address Data	int int	EEPROM address, 0-127 or 256-511 EEPROM data to write to the EEPROM

Return Value
Error code or 0.

Usage Example
To write 0x55 at EEPROM address 32,

```
int Address=32;
int Data=0x55;
VEGAEEPROMWrite (bi, Address, Data);
```

4.53 VEGA Monitor

Function Definition
BYTE VEGAMonitor (BoardInfo* bi, float* Status)

Function Description
This function reads the main input voltage and the on-board temperature sensor. These signals are routed into the calmux circuit.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
VEGAMonitor	Status float * Array of float values to store return data; array size = 2

Return Value
Error code or 0.

Usage Example
To read input voltage,

```
BYTE VEGAMonitor (bi, & Status);
```

4.54 VEGALED

Function Definition
BYTE VEGALED (BoardInfo* bi, int enable);

Function Description
This function turns the on-board LED on or off.

Function Parameters

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

Return Value
Error code or 0.

Usage Example
To turnoff blue LED on the board,

```
Enable=0;
VEGALED (bi, Enable);
```

5. UNIVERSAL DRIVER DEMO APPLICATION DESCRIPTION

The Universal Driver supports the following applications on the Vega SBC:

- DA Convert
- DA Convert Scan
- DA Waveform
- DIO
- Counter Function
- Counter Set Rate
- PWM
- User Interrupt
- AD Sampling
- AD Sample Scan
- AD Trigger
- AD Interrupt
- LED

5.1 DA Convert

This function outputs a value to a single D/A channel. The output range can be selected from the user input. When a D/A conversion is being performed, it is essentially taking a digital value and sending it out to the specified analog output channel as a voltage.

Once a D/A conversion have been generated 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 Convert Scan

D/A scan conversion is similar to D/A conversion except that it performs several conversions on multiple, specified output channels with each function call.

5.3 DA Waveform

This function generates a desired waveform on a selected DA channel. It configures a D/A channel by copying the waveform values to the board waveform buffer. The waveform generator 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 including a previously acquired waveform or a calculated or a simulated waveform.

5.4 DIO

This function supports four types of direct digital I/O operations: input bit, input byte, output bit, and output byte.

5.5 Counter Function

Generally the counter is used as rate generator. The counter also can be configured in count-up or count-down direction. The counter function can be used to program any desired function into a counter except reading.

5.6 Counter Set Rate

This function programs a counter for timer mode with down counting and continuous operation (reload enabled). The output may be used for waveform generator control, interrupt generation, or for a general programmable frequency output pulse train. The output may also be enabled on a DIO pin.

5.7 PWM

This application generates pulse width modulation (PWM) signals. PWM is 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 taken 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 rate, and with a certain duty cycle, the output will appear to behave like a constant voltage analog signal when providing power to the devices. This program gets duty cycle and frequency values from the user and generates PWM signals from these.

5.8 User Interrupt

The user interrupt feature enables the user to run their 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 the code at regular fixed intervals. Universal Driver includes example programs for each board with user interrupt capability to illustrate how to use the feature. This application gets interrupt frequency as a user input and calls the user function periodically at the rate of interrupt frequency.

5.9 AD Sample

Performs a single A/D conversion on the currently selected channel and returns a digital reading of an analog voltage signal applied to the currently selected A/D board's analog input channel. The A/D board uses a device called an analog-to-digital (A/D) converter to convert the real-world analog signal (temperature, pressure, tank level, speed, etc.) into a digital value that the digital computer electronics can process.

The function first waits for the board to be ready for a conversion. Then it starts the conversion and waits to finish before reading data from the board. A built-in timer is used to check for board failure. If the timer times out before the conversion completes, the board returns the error code.

5.10 AD Sample Scan

A/D scan is similar to an A/D sample except that it performs several conversions on a specified range of input channels with each function call.

5.11 AD Trigger

A/D trigger is similar to A/D sample. It performs a single A/D conversion on the currently selected channel and returns a digital reading of the analog voltage signal applied to the currently selected A/D board's analog input channel.

5.12 AD Interrupt

The AD interrupt application performs A/D scans using interrupt-based I/O with one scan per A/D clock tick. Note that calling this function only starts the interrupt operations in a separate system thread. The function call does not result in an atomic transaction with immediate results. Rather, it starts a real-time process that terminates only when the number of conversions reaches the maximum specified (one-shot mode) or if a call to VEGAADIntCancel() is made.

The behavior of A/D interrupt operations depends on three parameters. Some affect the behavior of the hardware, and some affect the behavior of the interrupt routine software. Together they determine the overall characteristics of the interrupt operation.

5.13 LED

This function is used to turn the on-board LED on or off.

6. UNIVERSAL DRIVER DEMO APPLICATION USAGE INSTRUCTIONS

The following section illustrates how to enter the Vega SBC console applications and to confirm the output which is obtained through the application.

6.1 DA Convert

This application gets input from the user as follows:

- Enter D/A polarity (0 for bipolar, 1 for unipolar, default: 0):
- Enter D/A range value (1-3, 1 = 10V, 2 = 5V, 3 = 2.5V,default 1):
- Enter Channel Number(0 – 7,Default 0):
- Enter output code (0-65535, default:32768):

To set channel 0 with -10V, run the DA Convert application and provide input as follows,

D/A polarity = 0
D/A range = 0
Channel Number=0
Output code=0

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

6.2 D/A Scan Conversion

This application gets input from user as follows:

- Enter D/A polarity A and polarity B for Channel 0-3 (0 for bipolar, 1 for unipolar)
- Enter D/A range A and range B value (1-3, 1 = 10V, 2 = 5V, 3 = 2.5V)
- Enter Sim value (0 =Individual Update 1=Simultaneous update):
- Enter the channel enable flag for channel 0
  (0 for FALSE, 1 for TRUE):
- Enter the output code for channel 0 (0-65535):
- Enter the channel enable flag for channel 1
  (0 for FALSE, 1 for TRUE ;):
- Enter the output code for channel 1 (0-65535):
- Enter the channel enable flag for channel 2
  (0 for FALSE, 1 for TRUE ;):
- Enter the output code for channel 2 (0-65535):
- Enter the channel enable flag for channel 3
  (0 for FALSE, 1 for TRUE ;):
- Enter the output code for channel 3 (0-65535):

To set channel 0 and 2 with 5V and 2.5V respectively with 0-5V range and leaving other channels existing voltages are untouched, run DA Convert scan application and provide input as follows:

- Enter D/A polarity A and polarity B for Channel 0-3 (0 for bipolar, 1 for unipolar) 1
- Enter D/A range A and range B value (1-3, 1 = 10V, 2 = 5V, 3 = 2.5V) 2
- Enter the channel enable flag for channel 0
  (0 for FALSE, 1 for TRUE ;) 1
- Enter the output code for channel 0 (0-65535): 65535
- Enter the channel enable flag for channel 1
  (0 for FALSE, 1 for TRUE ;) 0
- Enter the channel enable flag for channel 2
  (0 for FALSE, 1 for TRUE; default: 1): 1
- Enter the output code for channel 2 (0-65535): 32768
- Enter the channel enable flag for channel 3
  (0 for FALSE, 1 for TRUE; default: 1): 0

Measure the voltage on channel 0 and 2 using a multi meter. It should show 5V and 2.5V respectively, the application generates the expected voltage.

6.3 D/A Waveform Application

This application gets input from user as follows:

- Enter D/A Channel Number (0-3):
- Enter D/A polarity (0 for bipolar, for unipolar, default: 0):
- Enter D/A range value (1-3, 1 = 10V, 2 = 5V, 3 = 2.5V,default 1):
- Enter waveform size (1-2048) <Default =500>:
- Enter Clock source (0-3 , 0 -Manual, 1-Counter 0 output, 2-Counter 1 output , 3-External trigger on DIO pin 20, Default =1 :
- Select waveform type (0-Sine wave 1-Sawtooth Wave) <Default =0>:
- Enter waveform frequency <Default =100>:
- Enter waveform mode(0-1, 0-One shot 1-Repeat mode, default = 1:

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

Channel Number=0

Polarity =1

Range =1

Waveform size = 500

Clock source = 1

Waveform type = 0; //sine wave

Waveform frequency = 100

Waveform mode = 1

Place an oscilloscope probe on D/A channel 0 and set voltage division to 1V range and second division to 1ms. It should show a sine wave with 100Hz frequency, the application generated expected waveform.

6.4 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 DIO operation needed to be performed.

Output Byte:

- Select Write a Byte to port option from main menu
- Enter port number (0-3):
- Enter value 0-255 or q to quit

To set all pins in Port 0 high except pin 3 and pin 7, run the DIO application and provide input as follows:

- Select Write a Byte to port option from main menu :4
- Enter port number (0-3): 0
- Enter value 0-255 or q to quit: 119

The Byte value 119 is sent to port 0.

Measure voltage on Port 0 pins using a multi meter. It should show 3.3/5V on all the pins except pin 3 and pin 7, the application generated expected voltages.

Input Byte:

- Select Read a Byte from port option from main menu:
- Enter port number (0-3):
- Press ENTER key to stop reading ...

Provide 3.3V 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 Read a Byte from port option from main menu:1
- Enter port number (0-2):0

The application should show 0x01 on the screen.

6.5 Counter Function Application

This application gets input from the user as follows:

- Enter counter number (0-7):
- Enter Counter Direction (0 = down counting, 1 = up counting):
- Select Clock source (0=External ,2=Internal clock 50MHz,3=Internal clock 1MHz):
- Enter Output Enable (0=disable, 1=Enable):
- Enter Output Polarity (0=negative, 1= positive):
- Enter output pulse width (0-3 ,0 = 1 clocks, 1 = 10 clocks, 2 = 100 clocks, 3 = 1000 clocks, default = 3) :
- Enter Counter Frequency :
- Press any key to stop counting.

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

- Enter counter number (0-7): 0
- Enter Counter Direction (0 = down counting, 1 = up counting): 0
- Select Clock source (0=External ,2=Internal clock 50MHz,3=Internal clock 1MHz): 2
- Enter Output Enable (0=disable, 1=Enable): 1
- Enter Output Polarity (0=negative, 1= positive): 1
- Enter output pulse width (0-3 ,0 = 1 clocks, 1 = 10 clocks, 2 = 100 clocks, 3 = 1000 clocks, default = 3) : 3
- Enter Counter Frequency :100

Place an oscilloscope probe on counter 0 output pin (Port C 0^{th} pin is output pin for counter0) and the set voltage division to 1V range and second division to 1ms. It should show a periodic pulse with 100Hz frequency, the application generated expected rate.

- Press any key to automatically stop the application's counter output

6.6 Counter Set Rate Application

This application gets input from the user as follows:

- Enter counter number (0-7):
- Enter Counter frequency rate (1-50MHz):
- Enter Output Enable (0=disable, 1=Enable):
- Enter Output Polarity (0=negative, 1= positive):
- Enter output pulse width (0-3 ,0 = 1 clocks, 1 = 10 clocks, 2 = 100 clocks, 3 = 1000 clocks, default = 3) :
- Press any key to stop counting

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

- Enter counter number (0-7):0
- Enter Counter frequency rate (1-50MHz):100
- Enter Output Enable (0=disable, 1=Enable):1
- Enter Output Polarity (0=negative, 1= positive):1
- Enter output pulse width (0-3, 0 = 1 clocks, 1 = 10 clocks, 2 = 100 clocks, 3 = 1000 clocks, default = 3):0

Place an oscilloscope probe on counter 0 output pin (Port C 0^{th} pin is output pin for counter0) and set the voltage division to 1V range and second division to 1ms. It should show a periodic pulse with 100Hz frequency, the application generated expected rate.

- Press any key to automatically stop the counter output

6.7 PWM Application

This application gets input from the user as follows:

- Select PWM no (0-3):
- Select Output Frequency (1-50MHz):
- Select Duty cycle value (1-100):
- Select Polarity (0 = pulse high, 1 = pulse low):
- Waits for key press
- Output is generated
- If any key is pressed, application Stops PWM output.

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

- Select PWM no (0-3): 0
- Select Output Frequency (1-50MHz): 100
- Select Duty cycle value (1-100): 50
- Select Polarity (0 = pulse high, 1 = pulse low): 0
- Press any key to start PWM

Place an oscilloscope probe on PWM channel 0 output pin (Port D 2nd pin is output pin for PWM0) and set the voltage division to 1V range and 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 to automatically stop the PWM output

6.8 User Interrupt function

This application gets input from user as follows:

- Enter Interrupt source (0-1 ;0 = counter/timer 2 , 1 = DIO input) default =0 ");
- Enter Interrupt source frequency rate (1-50MHZ) (default = 1000)
- It calls user function based interrupt rate
- Press any key to cancel the application:

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 interrupt rate, run the UserInt application and provide input as follows:

- Enter Interrupt source (0-1; 0 = counter/timer2, 1=DIO input):0
- Enter Interrupt source frequency rate (1-50MHZ): 100

It calls the user function based interrupt rate and prints the count value every second. Since it is configured for 100Hz the display count value is displayed 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.

6.9 A/D Sample Application

This application gets input from the user as follows:

- Enter channel number(0-15):
- Enter A/D polarity (0 for bipolar, 1 for unipolar):
- Enter A/D Gain value (0-3, 0 = 1, 1 = 2, 2 = 4, 3 = 8,default 0):
- Enter input mode (0 = single-ended, 1 = differential; default 0):

To perform this operation for channel 0 with 5V range, run the AD Sample application, provide an external voltage (5v) to channel 0 and provide inputs as follows:

- Enter channel number (0-15):0
- Enter A/D polarity (0 for bipolar, 1 for unipolar):1
- Enter A/D Gain value (0-3, 0 = 1, 1 = 2, 2 = 4, 3 = 8,default 0): 1
- Enter input mode (0 = single-ended, 1 = differential; default 0): 0

It will perform the sampling operation and display the output as follows:

Sample readout: 32768, Actual voltage: 5.0V.

6.10 A/D Sample Scan Application

This application gets input from the user as follows:

- Enter the low channel (0-15, default: 0):
- Enter the High channel (0-15, default:7):
- Enter A/D polarity (0 for bipolar, 1 for unipolar, default: 0):
- Enter A/D Gain value (0-3, 0 = 1, 1 = 2, 2 = 4, 3 = 8,default 0):
- Enter input mode (0 = single-ended, 1 = differential; default 0):
- Enter Scan Interval (0 = 10us, 1 = 5us, 2 = 8us, 3 = programmable, default 0):
- Press any key to stop scanning A/D channel

To perform the AD scan application for channel 0-4 with bipolar 5V range, run the ADScan application and provide the inputs as follows:

- Enter the low channel (0-7, default: 0):0
- Enter the High channel (0-7, default:7):4
- Enter A/D polarity (0 for bipolar, 1 for unipolar, default: 0):0
- Enter A/D Gain value (0-3, 0 = 1, 1 = 2, 2 = 4, 3 = 8,default 0): 1
- Enter input mode (0 = single-ended, 1 = differential; default 0):0
- Enter Scan Interval (0 = 10us, 1 = 12.5us, 2 = 20us, 3 = programmable, default 0):0

The application scans AD channels from 0 to 4 and displays the AD sample values.

6.11 A/D Trigger Application

This application gets input from the user as follows:

- Enter channel number(0-15):
- Enter A/D polarity (0 for bipolar, 1 for unipolar):
- Enter A/D Gain value (0-3, 0 = 1, 1 = 2, 2 = 4, 3 = 8,default 0):
- Enter input mode (0 = single-ended, 1 = differential; default 0):

To perform this operation for channel 0 with a 5V range, run the ADSample application, provide an external voltage (5v) to channel 0 and provide inputs as follows:

- Enter channel number (0-15):0
- Enter A/D polarity (0 for bipolar, 1 for unipolar):1
- Enter A/D Gain value (0-3, 0 = 1, 1 = 2, 2 = 4, 3 = 8, default 0):1
- Enter input mode (0 = single-ended, 1 = differential; default 0):0

It will perform the sampling operation and display the output as follows:

Sample readout: 32768, Actual voltage: 5.0V.

6.12 A/D Interrupt Application

This application gets input from the user as follows:

- Enter low channel value (0-15, default: 0):
- Enter High channel value(0-15, default:7):
- Enter A/D polarity (0 for bipolar, 1 for unipolar, default: 0):
- Enter A/D Gain value (0-3, 0 = 1, 1 = 2, 2 = 4, 3 = 8,default 0):
- Enter input mode (0 = single-ended, 1 = differential; default 0):
- Enter Scan Interval (0 = 10us, 1 = 5us, 2 = 8us, 3 = programmable, default 0):
- Enter A/D clock source (1-3, 1= External trigger 2=Counter 0, 3=Counter 1 default=2) :
- Enter A/D sampling Rate (Default 1000) :
- Enter number of A/D conversions (must be multiple of FIFO threshold value default 1600):
- Enter the cycle flag (0 = one shot operation, 1 = continuous operation, default 1):
- Enter FIFO Enable bit value (0 = disable, 1 = enable, default: 1) :
- Enter FIFO Threshold value (0-2048 default: 1600) :
- Press Space bar key to Pause/Resume A/D Int or Press any other key to stop A/D Interrupt

To perform the AD interrupt application for channel 0-4 with bipolar 5V range and sampling rate as 1000Hz, run the A/D interrupt application and provide inputs as follows:

- Enter low channel value (0-15, default: 0):0
- Enter High channel value(0-15, default:7):4
- Enter A/D polarity (0 for bipolar, 1 for unipolar, default: 0):0
- Enter A/D Gain value (0-3, 0 = 1, 1 = 2, 2 = 4, 3 = 8, default 0):0
- Enter input mode (0 = single-ended, 1 = differential; default 0):0
- Enter Scan Interval (0 = 10us, 1 = 5us, 2 = 8us, 3 = programmable, default 0):
- Enter A/D clock source (1-3, 1= External trigger 2=Counter 0, 3=Counter 1 default=2) :2
- Enter A/D sampling Rate (Default 1000) :1000
- Enter number of A/D conversions (must be multiple of FIFO threshold value default 1600):1600
- Enter the cycle flag (0 = one shot operation, 1 = continuous operation, default 1):1
- Enter FIFO Enable bit value (0 = disable, 1 = enable, default: 1) :1
- Enter FIFO Threshold value (0-2048 default: 1600) :1600

The application scans AD channel from 0 to 4 at the 1000Hz interrupt rate and displays the AD sample values.

6.13 LED Application

This application gets input from the user as follows:

- Press space bar key to toggle LED or Press 'q' to quit

To toggle the on-board LED, press the space bar key to disable the LED, press the key again to enable the LED.

7. COMMON TASK REFERENCE

7.1 Data Acquisition Feature Overview

I/O Connectors

Vega SBC provides two I/O connectors for the attachment of all user I/O signals. The connectors are JST model no. SM30B-ZPDSS-TF. Diamond's I/O cable number 6981504 (2 per board) may be used to connect the user's signals to these connectors. This cable comes as a part of the Vega Cable Kit, part number CK-VEGA-01. This cable has a common 1.5mm 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 inputs

There are 16 analog input channels, numbered 0-15, located on connector J8 pins 1-16. In single-ended mode, each pin acts as an individual channel. The voltage on each channel is measured relative to the reference analog ground, which must be connected to any of pins 17 or 18. In differential mode, each consecutive pair of pins form a high-low pair; for example pins 0/8, 1/9 , 2/10 , 3/11, 4/12 ,5/13 ,6/14 ,7/15 are treated as the high and low inputs of a single channel. The input voltage is measured as the difference between these two pins

The maximum difference between any voltage on any analog input pin and the analog ground pins (common mode voltage) is 10V. In single-ended mode, as long as the input signal is within the range of +/-10V, the A/D will be able to measure the signal accurately. In differential mode, both inputs must be within this range. If the input voltages exceed this range, errors will occur, since the A/D will treat anything outside this range as the limit voltage, i.e. either +10V or -10V. This limit is not precise and should not be used in any normal operation. Furthermore any long term excursion by an input voltage beyond +/-10V may cause damage to the A/D chip or the on-board analog power supply.

Analog outputs

There are 8 analog output channels, numbered 0-7, located on connector J8 pins 19-26. Analog outputs operate in single-ended mode only and are always referenced to the analog ground, which must be connected to any of pins 17 or 18.

Waveform generator

The Vega SBC contains four analog waveform generators using the analog outputs. One to four 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 J8 pin 28 (Digital I/O port D1), 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	Max frame rate
1	387 Hz
2	410 Hz
3	405 Hz
4	415 Hz

Digital I/O signals

There are 30 digital I/O signals, divided into four groups as DIOA0-DIOA7, DIOB0-DIOB7, DIOC0-DIOC7 and DIOD0-DIOD5. Port A-C and Port D (DIOD3, DIOD4 and DIOD5) are on I/O connector J10 pins 1-27 and port D (DIOD0-DIOD2) is on I/O connector J8 pins 27-29.

Digital I/O signals use 3.3V signaling only. Each signal's direction is independently programmable. On system startup or reset, all signals are automatically set to input mode.

All digital I/O signals have programmable pull-up/down resistors and are divided into two groups for this purpose. Group A includes I/O port A, port B and Group B includes I/O port C, port D. All signals within each group have the same pull direction. The default configuration for Vega is for all DIO signals to be pulled low.

Counter/timers

The board provides 8 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 J10 Pin	PortC Bit	Counter/Timer
17	0	0
18	1	1
19	2	2
20	3	3
21	4	4
22	5	5
23	6	6
24	7	7

Pulse Width Modulators (PWMs)

The board offers four 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 J8 Pin	DIO D Bit	PWM
27	2	0

Connector J10 Pin	DIO D Bit	PWM
27	3	1
26	4	2
25	5	3

7.2 Data Acquisition Software Task Reference

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

- Program entry / exit sequence
- A/D conversions
- A/D interrupts
- 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 VEGAInitBoard() This function must be called prior to any other function involving the Vega SBC.
2. At the termination of the program the programmer may use VEGAFreeBoard(), but this is not required. This function is normally used in a development environment where the program is being repeatedly modified and rerun.

A/D Conversion Operation

Each A/D conversion is triggered by a clock event. The clock event may be a software command, an external digital signal on I/O connector J8 pin 29 (digital I/O port D0), or the output of either counter 0 or counter 1. Generally, low-speed conversions and “on-demand” conversions are triggered by software or by an external signal, and high speed conversions (where a precise time interval is required between samples) are driven by a counter/timer. If an external signal is used, it is connected to I/O connector J8 pin 29.

High-speed conversions are generally controlled with an interrupt-based A/D function in order to reduce the software overhead. The application program sets up the A/D circuit as needed and then calls the A/D interrupt function. The sampling and data transfer to memory occur in a background process. The application can monitor the progress of A/D conversions and retrieve data from the memory buffer as needed.

A/D conversions fall into two basic modes, sample and scan. In sample mode, a single channel is sampled on each clock. If the user is sampling multiple channels, then each clock will cause a single A/D conversion to occur on the currently selected channel, and then the channel counter will increment to the next channel in the list to be ready for the next clock. In scan mode, each clock causes one A/D conversion to occur on each channel in the user-selected channel range. The channel range can consist of 1 to 16 channels and must be consecutive, for example 0-8 or 9-15. Note that a scan operation on a single channel is equivalent to a sample operation on that channel.

In A/D scan operations, the time interval between samples (scan interval) is selectable from a range of options, as described in the function descriptions. Three fixed intervals and one user-programmable interval are provided. Generally the programmer will want to use the fastest available scan interval to complete all samples as closely together in time as possible. In cases where the input signals are using high input ranges such as 0-10V or +/-10V, using a longer interval may result in higher accuracy, since the input circuit has more time to swing from the current channel to the next. Most Diamond A/D boards are designed to provide accuracy close to the specified performance using the smallest scan interval for A/D input ranges of 0-5V or less.

The full sequence of operations is the same for A/D sample and A/D scan operations

1. VEGAADSetSettings () is used to select the input type, input voltage range, and input channel range.
2. VEGAADSetClock () is used to select the A/D clock source and the scan mode if desired.

3. If software A/D clocking is selected, then VEGAADEnableClock() is used to enable the selected clock. VEGAADSample () is used to generate A/D conversions for A/D sample mode and VEGAADScan()() is used to generate A/D conversions for A/D sample mode . The function must be called once for each A/D sample. The board auto-increments within the selected input channel range, starting with the lowest numbered channel in the range. When the highest numbered channel has been sampled, the board will reset to the lowest numbered channel. The function will return the A/D value from the currently sampled channel in A/D counts. This number must be converted to a voltage using the formulas described in the VEGA hardware user manual. The programmer may also convert this number to whatever engineering units are appropriate for the application.
4. If external clocking or counter/timer clocking is selected, the A/D conversions will start as soon as the selected clock source becomes active. In this case, the A/D FIFO will start to fill up with samples, and the program should use the A/D interrupt functions described below to acquire the data from the FIFO.

A/D Interrupt Operations

For high speed or externally clocked A/D conversions, interrupts should be used. This method installs an interrupt handler as a background task to read data from the board and store it in the program's data buffer. The A/D conversions can be triggered by a software command, a falling edge on I/O connector J8 pin 29 (digital I/O port D0), or the output of either counter 0 or counter 1.

A/D data is stored in a FIFO on the board, incrementing the FIFO depth counter each time. When the depth in the FIFO reaches the user-selected threshold, an interrupt will occur, and the interrupt handler will read out the data in the FIFO, decrementing the FIFO depth counter. This process occurs repeatedly until stopped.

The program must select the FIFO threshold based on two opposing parameters: The waiting time until the first data is available (threshold divided by sample rate) and the desired interrupt rate (sample rate divided by threshold). Generally the FIFO threshold should be selected to avoid exceeding a 1KHz interrupt rate. Higher interrupt rates consume more processor time, so applications may wish to reduce the interrupt rate even lower, to 100-200Hz, by using a deeper threshold if the sample rate permits it.

Note that A/D data will only be transferred from the board to the user's data buffer when the FIFO reaches the selected threshold. This means that once starting the interrupt operation, the program will have no data until the first interrupt occurs.

Generally the sample rate for interrupt operations is high, so the initial waiting time is not significant, and the desired interrupt rate will drive the selection of the FIFO threshold value.

Interrupt-based A/D conversions can be done in two modes, one-shot or continuous. One-shot means that a fixed number of A/D conversions is done and then the operation automatically stops. Continuous means that the A/D sampling continues until the program stops the operation.

The sequence of operations for interrupt-based A/D conversions is as follows:

1. VEGAADSetSettings () is used to select the input type, input voltage range, and input channel range.
2. VEGAADSetClock () is used to select the clock source.
3. VEGAADInt () is used to install the interrupt handler.
4. If a counter/timer is being used to control A/D timing, then VEGACounterSetRate () is used to program the counter/timer for the desired sample rate.
5. VEGAADEnableClock () is used to enable selected clock source and data will automatically be transferred to the user-specified data buffer based on the selected FIFO threshold.
6. To monitor A/D operations, use VEGAADIntStatus ().

The interrupt operation can be monitored with the VEGAADIntStatus () function. This function will report whether the interrupt operation is running, how many samples have been taken since the start, the current FIFO depth, and the type of operation – single or recycle.

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

For multi-channel output with individual channel update use the following sequence:

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

For multi-channel output with simultaneous update use the following sequence:

1. VEGADASetSettings () to select the output range; set Sim = 1
2. VEGADAConvertScan () to load the given data into the selected channels
3. VEGADAUpdate () 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 as shown in the following table.

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 / <number of channels> for multiple channels. The term <number of channels> is also referred to as the frame size.

Once the data buffer is built, use VEGAWaveformBufferLoad () 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 <number of frames> must be less than or equal to 2048.

VEGAWaveformDataLoad () 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 VEGAWaveformConfig () 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 VEGAWaveformStart (). 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 point of time in its current position use VEGAWaveformPause (). The waveform may be restarted in its current position by using VEGAWaveformStart () again.

To reset the waveform generator use VEGAWaveformReset (). 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 VEGAWaveformConfig () should be called followed by VEGAWaveformStart ().

If software increment is selected, the waveform is incremented with the function VEGAWaveformInc (). 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 VEGADIOConfig () configures all 8 bits of a selected port
BYTE VEGADIOConfigAll () configures all 30 bits at once.
```

Then execute whichever I/O function is desired. Byte read/write enables 5 or 8 bits of digital I/O to be updated at once. Bit operation enables a single bit to be updated.

```
BYTE VEGADIOOutputByte () outputs 8 bits of data
BYTE VEGADIOInputByte (BoardInfo* bi, int Port, byte* Data);
BYTE VEGADIOOutputBit (BoardInfo* bi, int Port, int Bit, int Value);
BYTE VEGADIOInputBit (BoardInfo* bi,int Port, int Bit, int* Value);
```

To configure the digital I/O pull-up/down resistors, use the programmed value is stored in a small flash device on the board, so that the board will retain the latest configuration the next time it is powered up.

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 VEGACounterSetRate (). 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 VEGACounterConfig () 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 VEGACounterRead () 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.
3. When the counting function is no longer needed, use VEGACounterReset () to reset the counter and return any assigned Digital I/O pin to normal digital I/O operation.

Detailed Programming

To program a counter/timer using individual commands, use VEGACounterFunction (). 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.
9	Select the desired output pulse width.

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.

Alternatively, the function VEGACounterReset () can be used to reset the counter/timer.

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 pulse.
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 VEGACounterRead () function.
When the counting function is no longer needed, the function VEGACounterReset () can be used to reset the counter/timer.

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

To stop a PWM:

VEGAPWMStop () 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 VEGAPWMStart ().
To reset a PWM and return its assigned digital I/O output pin to normal operation use VEGAPWMReset ().

To implement special functions, such as changing the duty cycle or frequency of a PWM while it is running, use VEGAPWMCommand (). 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 from a variety of sources. The interrupt can run as the only procedure when the interrupt occurs (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 number	Modes supported
A/D interrupts	0	1 Before, 2 After
D/A interrupts	1	Not supported on Vega
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:

VEGAUserInterruptSet () 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 VEGACounterSetRate ().
2. If a digital input is being used to drive interrupts, it is configured with VEGAUserInterruptSet ().

Run (before / after modes):
Call the standard interrupt setup function, such as VEGAADInt ().

Stop (alone mode):
Use VEGAUserInterruptStop () to stop user interrupts.

Stop (before / after modes):
Use the standard interrupt cancel function such as VEGAADIntCancel ().

LED Control

The Vega SBC 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 VEGALED ().

7.3 Performing D/A Conversion

Description

When a D/A conversion is performed, 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.

The Universal Driver function for performing a D/A conversion is VEGADAConvert (bi, channel, DACode)

Step-By-Step Instructions

Call VEGADAConvert (bi, channel, DACode) 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 is set with the output code.

NOTE: Once a D/A conversion is generated 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 = 0;
unsigned int DACode = 65535;
VEGADASETTINGS dasettings;

dasettings.PolarityA = 0;
dasettings.RangeA = 0;
dasettings.LoadCalA = 1;
dasettings.PolarityB = 0;
dasettings.RangeB = 0;
dasettings.LoadCalB = 1;
dasettings.Mode = 0;
dasettings.Simultaneous = 0;

if (VEGADASetSettings (bi, &dasettings ) !=DE_NONE) // To D/A settings for +/-10V
range
{
      dscGetLastError (&errorParams);
      printf ("VEGADASetSettings error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode),   errorParams.errstring );
      return 0;
}/* Step 2 */
if ( VEGADAConvert(bi,channel, DACode ) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ("VEGADAConvert error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}
```

7.4 Performing D/A Scan Conversion

Description

A D/A scan conversion is similar to a D/A conversion except that it performs several conversions on multiple specified output channels with each function call.

The Universal Driver function for performing a D/A conversion scan is VEGADAConvertScan ().

Step-By-Step Instructions

Pass the values for ChannelSelect and DACodes pointer to this function.
Call VEGADAConvertScan() and pass it to a pointer to the scanned array values. This generates a D/A conversion for each channel that is set to enable.

Example of Usage for D/A Conversion Scan

To update channel 0 and 2 with DA code 65535 and 32768 respectively and the rest of the channels remain unchanged from their existing voltage level:

```
VEGADASCAN Scan;
Scan.ChannelSelect = (int*) malloc (sizeof (int) * 8);
Scan.DACodes = (unsigned int*) malloc (sizeof (unsigned int) * 8);
ChannelSelect [0] = 1;
DACodes [0] = 65535;
Scan.ChannelSelect [2] = 1;
Scan.DACodes [2] = 32768;
if ( (VEGADAConvertScan (bi,&Scan) ) != DE_NONE )
{
      dscGetLastError (&errorParams);
      printf ("VEGADAConvertScanerror: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}
```

7.5 Performing Digital IO Operations

Description

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, provide a pointer to the storage variable and indicate the port number and bit number if relevant. To perform digital output, provide the output value and the output port and bit number, if relevant.

The six Universal Driver functions used are VEGADIOConfig () , VEGADIOConfigAll (), VEGADIOOutputByte (), VEGADIOInputByte (), VEGADIOOutputBit(), VEGADIOInputBit ().

Step-By-Step Instructions

If digital input is being performed, 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 VEGADIOConfigAll () to set the direction if necessary or use function VEGADIOConfig () to set the direction for a single bit.

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 *config;
int port;
BYTE input_byte;  // the value ranges from 0 to 255
BYTE output_byte;
Int digital_value;
config = (int *)malloc(sizeof(int)* 3);

/* 1. Configure Port 0 in output mode */
Config [0] = 0; //0 = Input, 1 = Output
Config [1] = 1; //0 = Input, 1 = Output
config [2] = 0;  //0 = Input, 1 = Output
if (result = VEGADIOConfigAll(bi,config) != 0)
      return result;

/* 2.  input bit - read bit 6 (port 0 is in input mode) */
Config [0] = 0;
VEGADIOConfig (bi,config);
port = 0;
bit = 6;
if ( (VEGADIOInputBit ( bi, port, bit, &digital_value ) != DE_NONE) )
{
      dscGetLastError (&errorParams);
      printf ("VEGADIOInputBit error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}

/* 3. input byte - read all 8 bits of port 0 (port 0 is in input mode) */
port = 0;
config [0] = 0;
VEGADIOConfig (bi,config);
if ( (VEGADIOInputByte ( bi, port, &input_byte ) != DE_NONE) )
```

```
{
      dscGetLastError (&errorParams);
      printf ("VEGADIOInputByte error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}

/* 4. output bit – set the bit 6 of port 1 (assumes port 1 is in output mode) */
digital_val = 1;
port=1;
config [1] = 1;
VEGADIOConfig (bi,config);
if ( (VEGADIOOutputBit(bi, port, bit,digital_val) != DE_NONE) )
{
      dscGetLastError (&errorParams);
      printf ("VEGADIOOutputBit error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}

/* 5. output byte – set the port 1  to "0xFF"(assumes port 1 is in output mode) */
port =1;
output_byte = 0xFF;
config [1] = 1;
VEGADIOConfig (bi,config);
if( (VEGADIOOutputByte( bi, port, output_byte ) != DE_NONE) )
{
      dscGetLastError (&errorParams);
      printf ("VEGADIOOutputByte error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}
```

7.6 Performing PWM Operations

Description:

The PWM operation generates a PWM signal with desired frequency and duty cycle value. The following functions are used for PWM operation: VEGAPWMConfig () , VEGAPWMStart() and VEGAPWMStop().

Step-By-Step Instructions

Create a VEGAPWM structure variable to hold the PWM settings and initialize the PWM structure variables, then call VEGAPWMConfig () to configure the PWM. Call VEGAPWMStart () to start the PWM, and finally call VEGAPWMStop () to stop the running PWM signal.

Example of Usage for PWM Operations

```
VEGAPWM pwm; // structure to hold the PWM 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 = 0;

//The following function configures PWM circuit
If (VEGAPWMConfig (bi, &pwm)! =DE_NONE)
{
      dscGetLastError(&errorParams);
      printf           ("VEGAPWMConfig           error:           %s           %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}

//The following function start the PWM
if (VEGAPWMStart (bi,pwm.Num) !=DE_NONE)
 {
      dscGetLastError (&errorParams);
      printf           ("VEGAPWMStart           error:           %s           %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}
printf ("Press any key to stop PWM \n");
getch ();

//The following function stop the PWM
if (VEGAPWMStop (bi,pwm.Num) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ("VEGAPWMStop error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
      errorParams.errstring );
      return 0;
}
```

7.7 Performing Counter Function Operations

Description:

Generally the counter is used as rate generator. The counter also can be configured in count-up or count-down direction. The following functions are used for counter function operation: VEGACounterConfig () and VEGACounterFunction ().

Step-By-Step Instructions

Create a VEGACOUNTER structure variable to hold the counter settings and initialize the counter structure variables, then call VEGACounterConfig () to configure the counter, and finally the VEGACounterFunction () to stop the running counter.

Example of Usage for Counter Operations

```
VEGACOUNTER Ctr;
Ctr.CtrNo = 0;
Ctr.CtrCountDir=0;
Ctr.CtrClock = 2; //50MHz
Ctr.CtrData = 50000000/100;
Ctr.CtrOutEn=1;
Ctr.CtrOutPol = 1;
Ctr.CtrReload     = 1;
//The following function configures the counter with desired rate
If (VEGACounterConfig (bi, &Ctr)! =DE_NONE)
{
      dscGetLastError (&errorParams);
      printf          ("VEGACounterConfig          error:          %s          %s\n",
      dscGetErrorString(errorParams.ErrCode),
      errorParams.errstring);
      return 0;
}
Printf ("Press any key to stop counting \n");
while ( !kbhit())
{
      dscSleep (1000);
      VEGACounterRead (bi, &Ctr);
      Printf ("Counter Data %ld \r", Ctr.CtrData);
      fflush (stdout);
}
VEGACounterReset (bi, Ctr. CtrNo);
```

7.8 Performing Counter Set Rate Operation

Description:

Generally the counter is used as rate generator. The counter also can be configured in count-up or count-down direction. The following functions are used for counter set rate operation: VEGACounterSetRate (), VEGACounterRead (), and VEGACounterFunction() .

Step-By-Step Instructions

Create a VEGACOUNTER structure variable to hold the counter settings and initialize the counter structure variables. Call VEGACounterSetRate () to program a counter for timer mode with down counting and continuous operation (reload enabled), then call VEGACounterRead () to read the counter value. Finally call VEGACounterFunction() to stop the running counter.

Example of Usage for Counter Set Rate Operations

```
VEGACOUNTER counter;
counter.CtrNo = 0;
counter.Rate = 1000;    // Counter frequency
counter.CtrOutEn = 1;
counter.CtrOutPol = 1;
counter.ctrOutWidth = 3;

if(VEGACounterSetRate (bi,&counter) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ("VEGACounterSetRate error: %s %s\n",
      dscGetErrorString (errorParams.ErrCode), errorParams.errstring );
      return 0;
}
Printf ("Press any key to stop counting \n");
While ( !kbhit())
{
      dscSleep (1000);
      VEGACounterRead (bi,&counter);
      Printf ("Counter Data %ld \r", counter.ctrData);
      Fflush (stdout);
}
VEGACounterReset (bi, counter. CtrNo);
```

7.9 Performing User Interrupt Operations

Description:

The User Interrupt application configures counter 0 to generate a desired interrupt rate and at the same interrupt rate, a registered user interrupt function is called. To perform the User Interrupt application the following functions are used: VEGAUserInterruptSet(), VEGACounterSetRate(), and VEGAUserInterruptStop ().

Step-By-Step Instructions

Create a VEGAUSERINT structure variable to hold the interrupt settings and initialize the interrupt structure variables, then call VEGAUserInterruptSet to run the user interrupt, VEGACounterSetRate() to set the counter, and finally call VEGAUserInterruptStop () to stop the user interrupt.

Example of Usage for User Interrupt Operations

```
int count =0;
VEGAUSERINT inter;
VEGACOUNTER counter;

Void intfunction ()
{
      count ++;
}
Void main()
{
      inter.IntFunc = intfunction;
      inter.Mode = 0;
      inter.Source = 0;
      inter.Enable = 1;

if(VEGAUserInterruptSet(bi,&inter) !=DE_NONE)
{
      dscGetLastError ( &errorParams );
      printf ( "VEGAUserInterruptSet error: %s %s\n", dscGetErrorString (
      errorParams.ErrCode ),errorParams.errstring );
      return 0;
}
counter.Rate = rate;
counter.CtrNo = 2;//inter.Source;
counter.CtrOutEn = 0;
counter.CtrOutPol = 0;

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

while( !kbhit())
{
      dscSleep (1000);
      printf ("Count value %d \r",count);
      fflush (stdout);
}
inter.Enable= 0;
if (VEGAUserInterruptStop(bi,&inter) !=DE_NONE)
{
      dscGetLastError ( &errorParams );
```

```
      printf  (  "VEGAUserInterruptStop  error:  %s  %s\n",  dscGetErrorString  (
      errorParams.ErrCode ), errorParams.errstring );
      return 0;
}
Printf ("Press any key to terminate the application \n");
While ( !kbhit())
{
      dscSleep (1000);
      printf ("Count value %d \r",count);
      fflush (stdout);
}

}
```

7.10 Generating D/A Waveform

Description:

This function generates a desired waveform on selected DA channel. It configures a D/A channel by copying the waveform values to the board's waveform buffer.

The following functions are used to generate a waveform: VEGAWaveformReset (), VEGAWaveformConfig(), VEGAWaveformBufferLoad(), VEGAWaveformStart(), VEGAWaveformPause(), and VEGAWaveformReset ().

Step-By-Step Instructions

Create a VEGAWAVEFORM structure variable to hold waveform settings and initialize the waveform structure variables, then call VEGAWaveformReset () to reset the D/A waveform buffer. Call VEGAWaveformConfig () to configure the operating parameters of the waveform generator, call VEGAWaveformBufferLoad () to load D/A conversion value into the D/A waveform buffer, call VEGAWaveformStart () to start the waveform generator, and finally call VEGAWaveformPause () and VEGAWaveformReset () to stop the waveform generator.

Example of Usage for D/A Waveform Generator:

```
VEGAWAVEFORM waveform;
VEGADASETTINGS dasettings;
waveform.Frames = 500;
waveform.Clock =1;
waveform.FrameSize=1;
waveform.Cycle =1;
frequency = 100;
waveform.Rate = waveform.Frames     * frequency;
channel = 0;
range = 1; //0-5v
if (VEGAWaveformReset (bi) != DE_NONE )
{
      dscGetLastError (&errorParams);
      printf          ("VEGAWaveformReset          error:          %s          %s\n",
      dscGetErrorString(errorParams.ErrCode),errorParams.errstring );
      return 0;
}
if (VEGAWaveformConfig (bi, &waveform ) != DE_NONE )
{
      dscGetLastError (&errorParams);
      printf      (      "      VEGAWaveformConfig      error:      %s      %s\n",
      dscGetErrorString(errorParams.ErrCode),errorParams.errstring );
      return 0;
}

/* Loading D/A conversion values to waveform buffer */
waveform.Waveform = (unsigned int*) malloc(waveform.Frames * sizeof(unsigned int));
waveform.Channels= (int *) malloc (4 * sizeof (int *));
for (index=0; index <waveform.Frames; index++)
{

/* generating sine wave */
      waveform.Waveform [index] = (int) ((sin (index * (360.0/waveform. Frames) *
      PI/180) + 1) * (0xFFFF/2));
}
for ( index=0;index<4;index++)
{
      waveform.Channels [index] = channel;
}
dasettings.PolarityB = 0;
dasettings.RangeB = 0;
```

```
dasettings.Mode = 0;
dasettings.Simultaneous = 0;
dasettings.LoadCalA=1;
dasettings.LoadCalB=1;
if (VEGADASetSettings(bi, &dasettings ) != DE_NONE )
{
      dscGetLastError (&errorParams);
      printf          ("VEGADASetSettings          error:          %s          %s\n",
      dscGetErrorString(errorParams.ErrCode),
      errorParams.errstring);
      return 0;
}

//the following loads the D/A conversion value into the D/A buffer.
if (VEGAWaveformBufferLoad (bi,&waveform) != DE_NONE )
{
      dscGetLastError (&errorParams);
      printf        ("VEGAWaveformBufferLoad        error:        %s        %s\n",
      dscGetErrorString(errorParams.ErrCode),errorParams.errstring );
      return 0;
}

//start D/A conversions
Printf ("Starting DA wave form generator \n");
if (VEGAWaveformStart (bi) != DE_NONE )
{
      dscGetLastError (&errorParams);
      printf ("VEGAWaveformStart error: %s %s\n",
      dscGetErrorString (errorParams.ErrCode), errorParams.errstring );
      return 0;
}
printf("Press any key to stop wave form Generator \n");
fgets(input_buffer,20,stdin);
VEGAWaveformPause (bi);
VEGAWaveformReset (bi);
free(waveform.Waveform);
free(waveform.Channels);
```

7.11 Performing A/D Sample

Description:

When an A/D conversion is being performed, a digital reading is provided of an analog voltage signal applied to one of the A/D board's analog input channels. The following function is used for A/D conversion: VEGAADSample ().

Step-By-Step Instructions

Create a VEGAADSETTINGS structure variable to hold the A/D settings and initialize the structure variables. Call VEGAADSample () to configure the A/D settings as requested by the user. It takes a single A/D sample value of a single channel.

Example of Usage for A/D Sample:

```
VEGAADSETTINGS dscadsettings;
unsigned int sample;       // sample reading
int channel;
channel=0;
dscadsettings.Polarity = 0;
dscadsettings.Gain = 0;
dscadsettings.Sedi = 0;
dscadsettings.Highch = channel;
dscadsettings.Lowch = channel;
dscadsettings.ScanEnable = 0;
//dscadsettings.ADClock = 0;
if ( VEGAADSample ( bi,&sample)  != DE_NONE )
{
      dscGetLastError (&errorParams);
      printf         (         "VEGAADSample         error:         %s         %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}
```

7.12 Performing A/D Scan

Description:

A/D scan is similar to an A/D sample except that it performs several conversions on a specified range of input channels with each function call. The functions for performing a A/D scan are VEGAADSetSettings (), VEGAADEnableClock (), and VEGAADScan ().

Step-By-Step Instructions

Create a VEGAADSETTINGS structure variable to hold A/D settings and initialize the structure variables. Call VEGAADSetSettings () to configure the A/D settings as requested by the user. Then call VEGAADEnableClock () to configure the turbo mode, A/D clock source, and scan settings. Finally call VEGAADScan () to execute the A/D conversion scan using the current board settings.

Example of Usage for A/D Scan:

```
VEGAADSETTINGS dscadsettings;
VEGAADSAMPLE sample [8];       // sample reading
dscadsettings.Lowch=0;
dscadsettings.Highch=10;
dscadsettings.Polarity = 0;
dscadsettings.Gain = 0;
dscadsettings.Sedi = 0;
dscadsettings.ScanInterval = 0;
dscadsettings.ADClock = 0;
dscadsettings.ScanEnable = 1;
if ( (VEGAADSetSettings ( bi, &dscadsettings ) ) != DE_NONE )
{
      dscGetLastError (&errorParams);
      printf          ("VEGAADSetSettings          error:          %s          %s\n",
      dscGetErrorString(errorParams.ErrCode),errorParams.errstring );
      return 0;
}
if ( (VEGAADSetClock (bi,&dscadsettings) ) != DE_NONE )
{
      dscGetLastError (&errorParams);
      printf       (       "       VEGAADSetClock       error:       %s       %s\n",
      dscGetErrorString(errorParams.ErrCode),errorParams.errstring );
      return 0;
}
if ( VEGAADEnableClock ( bi)  != DE_NONE )
{
      dscGetLastError (&errorParams);
      printf       (       "VEGAADEnableClock       error:       %s       %s\n",
      dscGetErrorString(errorParams.ErrCode),errorParams.errstring );
      return 0;
}
while (!kbhit() )
{
      if ( (VEGAADScan ( bi, sample ) ) != DE_NONE )
      {
            dscGetLastError (&errorParams);
            printf       (       "       VEGAADScan       error:       %s       %s\n",
            dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
            return 0;
      }
}
```

7.13 Performing A/D interrupts

Description:

The AD interrupt application performs A/D scans using interrupt-based I/O with one scan per A/D clock tick. The following functions are used for A/D interrupt: VEGAADSetSettings (), VEGAADSetClock(), VEGAADInt(), VEGAADIntResume(), VEGAADIntStatus (), and VEGAADIntCancel ().

Step-By-Step Instructions:

Create a VEGAADSETTINGS structure variable to hold A/D settings, create a VEGAADINT structure variable to hold A/D conversion interrupt settings, and create a VEGAADINTSTATUS structure variable to hold A/D conversion interrupt status. Initialize the A/D settings structure variables, then use VEGAADSetSettings () to configure the A/D settings as requested by the user. Call VEGAADSetClock() to configure the turbo mode, A/D clock source, and scan settings. Initialize the A/D interrupt status structure variable, then call VEGAADInt() to enable A/D interrupt operation using the current analog input settings. Call VEGAADIntStatus() to get the interrupt routine status including, running / not running, number of conversions completed, cycle mode, FIFO status, and FIFO flags. Call VEGAADIntResume() to resume the interrupt. Call VEGAADIntPause () to pause the interrupt, and finally call VEGAADIntCancel() to stop the A/D interrupt.

Example of Usage for A/D interrupt:

```
VEGAADSETTINGS dscadsettings; // structure containing A/D conversion settings
VEGAADINT    dscIntSettings  //  structure  containing  A/D  conversion  interrupt
settings
VEGAADINTSTATUS intstatus; // structure containing A/D conversion interrupt status
int sample[8];       // sample reading
int key = 0;
int Pause = 0;
/* Initializing A/D settings */
dscadsettings.Lowch = 0;
dscadsettings.Highch = 7;
dscadsettings.Polarity = 0; //bipolarity
dscadsettings. Gain= 0; //5v
dscadsettings.Sedi = 0;
dscadsettings.ScanEnable = 0;
dscadsettings.ScanInterval = 0;
dscadsettings.ADClock = 2;
if ( (VEGAADSetSettings ( bi, &dscadsettings ) ) != DE_NONE )
{
      dscGetLastError(&errorParams);
      printf          ("VEGAADSetSettings          error:          %s          %s\n",
      dscGetErrorString(errorParams.ErrCode),errorParams.errstring );
      return 0;
}
if ( (VEGAADSetClock (bi,&dscadsettings.ADClk) ) != DE_NONE )
{
      dscGetLastError (&errorParams);
      printf      (      "      VEGAADSetClock            error:      %s      %s\n",
      dscGetErrorString(errorParams.ErrCode),errorParams.errstring );
      return 0;
}

/* Initializing Interrupt settings */

dscIntSettings.NumConversions = 1000;
dscIntSettings.Cycle = 1;
dscIntSettings.FIFOEnable = 1;
dscIntSettings.FIFOThreshold = 1600;
```

```
dscIntSettings.ADBuffer = (SWORD*) malloc (sizeof(SWORD)*
dscIntSettings.NumConversions );
if (VEGAADInt (bi,&dscIntSettings ) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ( "VEGAADInterror: %s %s\n", dscGetErrorString(errorParams.ErrCode),
      errorParams.errstring);
      return 0;
}
if (VEGACounterSetRate (bi,dscadsettings.ADClock-2,rate,0,0) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ( "VEGACounterSetRateerror: %s %s\n",
      dscGetErrorString(errorParams.ErrCode),errorParams.errstring );
      return 0;
}
dscSleep (1000);
printf ("\nPress Space bar key to Pause/Resume A/D Int or Press any other key to
stop A/D Interrupt \n");
while (1)
{
      key = kbhit();
      if (key==0)
      {
            If (VEGAADIntStatus (bi,&intstatus) !=DE_NONE)
            {
                  dscGetLastError (&errorParams);
                  printf ( "VEGAADIntStatuserror: %s %s\n", dscGet
                  ErrorString(errorParams.ErrCode),
                  errorParams.errstring);
                  return 0;
            }
            Printf("No of A/D conversions completed
            %d\n",intstatus.NumConversions);
            dscSleep (1000);
            if( (dscIntSettings.Cycle == 0) && (intstatus.NumConversions
            >=dscIntSettings.NumConversions) )
            {
                  break;
            }
            else
            {
                  key = getchar();
                  if(key ==32 )
                  {
                        if(Pause )
                        {
                              if(VEGAADIntResume (bi) !=DE_NONE)
                              {
                                    dscGetLastError (&errorParams);
                                    printf ( "VEGAADIntResume error: %s %s\n",
                                    dscGetErrorString(errorParams.ErrCode),
                                    errorParams.errstring );
                                    return 0;
                              }
                              Pause = 0;
                        }
                        else
                        {
```

```
                                    if(VEGAADIntPause (bi) !=DE_NONE)
                                    {
                                          dscGetLastError (&errorParams);
                                          printf ( "VEGAADIntPause error: %s %s\n",
                                          dscGetErrorString (errorParams.ErrCode),
                                          errorParams.errstring);
                                          return 0;
                                    }
                                    Pause = 1;
                              }
                        }
                        else
                        {
                              break;
                        }
                  }
            }
}
If(VEGAADIntCancel (bi)!=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf        (       "VEGAADIntCancel        error:        %s        %s\n",
      dscGetErrorString(errorParams.ErrCode),
      errorParams.errstring);
      return 0;
}
```

7.14 Performing LED operations

Description:

The application toggles the on-board LED whenever the space bar key pressed. The VEGALED () function is used to toggle the on-board LED.

Example of Usage for LED toggle:

```
int key=0
int LEDOn = 1;
printf("\nPress space bar key to toggle LED or Press 'q' to quit  \n");
while (1)
{
      key =getch();

      if (key == 32)
      {
            If (LEDOn)
            {
                  VEGALED (bi, 0);
                  LEDOn = 0;
            }
            else
            {
                  VEGALED (bi, 1);
                  LEDOn = 1;
            }
      }
      else if(key == 'q')
      {
            break;
      }
}
```

8. INTERFACE CONNECTOR DETAILS

8.1 VEGA Digital GPIO Connector (J10)

This connector carries the digital I/O from the DAQ circuitry. The signals originate with the FPGA and have ESD protection diodes. Most of the signals are in three groups (A, B, and C) of eight signals each.

DAQ 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 C3
DIO C4	**21**	**22**	DIO C5
DIO C6	**23**	**24**	DIO C7
DIO D4	**25**	**26**	DIO D5
DIO D3	**27**	**28**	GND
+5V	**29**	**30**	GND

Description: CONN, 2x15P, 1.5MM, LOCK, SH, R/A, SMT

Part Number: JST, SM30B-ZPDSS-TF

8.2 VEGA Analog GPIO Connector (J8)

This connector carries the analog I/O from the DAQ circuitry. The input signals are routed to the ADS8509 ADC. The output signals originate from the AD5362 DAC. There are no ESD protection diodes for these signals.

DAQ Analog GPIO Connector Pinout

VIN0	**1**	**2**	VIN8
VIN1	**3**	**4**	VIN9
VIN2	**5**	**6**	VIN10
VIN3	**7**	**8**	VIN11
VIN4	**9**	**10**	VIN12
VIN5	**11**	**12**	VIN13
VIN6	**13**	**14**	VIN14
VIN7	**15**	**16**	VIN15
AGND_DAQ	**17**	**18**	AGND_DAQ
VOUT0	**19**	**20**	VOUT1
VOUT2	**21**	**22**	VOUT3
VOUT4	**23**	**24**	VOUT5
VOUT6	**25**	**26**	VOUT7
DIO D2	**27**	**28**	DIO D1
DIO D0 / Ext Trig	**29**	**30**	GND

Description: CONN, 2x15P, 1.5MM, LOCK, SH, R/A, SMT

Part Number: JST, SM30B-ZPDSS-TF

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 are required to configure a counter/timer for operation. See the counter/timer usage instructions in this 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. This command is not implemented on the Vega SBC.

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 VEGACounterRead() 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 and 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 table on the following page. 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 J10 Pin	Port C Bit	Counter/Timer
17	0	0
18	1	1
19	2	2
20	3	3
21	4	4
22	5	5
23	6	6
24	7	7

9 Select counter output pulse width. Only has effect when counter output is enabled with command 1000 and the clock source selected with command 0110 is internal 50MHz or 1MHz. The counter output pulse width is defined according to the table below.

If the selected pulse width is equal to or greater than the clock period, the output will stay at its active state indefinitely.

If external clock is selected, the output pulse is always 1 clock wide, meaning that it will transition to active on the terminal count clock pulse and then transition to inactive on the next clock pulse.

Option	Output pulse width
0	1 clock (default if command is not executed)
1	10 clocks
2	100 clocks
3	1000 clocks

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 and 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 are 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 J8 Pin	**DIO D Bit**	**PWM**
27	2	0

Connector J10 Pin	**DIO D Bit**	**PWM**
27	3	1
26	4	2
25	5	3