Universal Driver Software User Manual

Helix PC/104 SBC

For Version 8.0.0 and later

Revision A.0 January 2017

Revision	Date	Comment
A.0	01/03/2017	Initial release

FOR TECHNICAL SUPPORT
PLEASE CONTACT:

support@diamondsystems.com

www.diamondsystems.com

Contents

1. INTRODUCTION

This user manual contains all essential information about the Universal Driver 8.0.0 Helix 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

Helix SBCs use the Vortex86DX3 SoC from DMP Electronics. It is a 32-bit x86 architecture dual-core 1GHz microprocessor designed for ultra-low power consumption, combining both the North and South bridges with a rich set of integrated features including a 32KB write through 8-way L1 cache, 512KB write through/write back 4-way L2 cache, PCIe bus at 2.5 GHz, DDR3 controller, ISA, I2C, SPI, IPC (includes Internal Peripheral Controllers with DMA and interrupt timer/counter), Fast Ethernet, FIFO UART, USB2.0 Host, and an IDE/SATA controller. The SBC supports up to 2GB of DDR3 memory soldered on-board.

The SBC provides four serial ports: two with fixed RS-232 capability using SP211EHEA-L, and a second two having RS-232/422/485 capability using a SP336. The built-in UARTs from the VortexDX3 are used. In RS-232 mode, only the signals TX, RX, RTS, and CTS are provided. Protocol selection for serial ports 3-4 is controlled using GPIO pins from the SoC and is configurable via BIOS configuration screens as well as via application software. Jumpers are used to enable termination resistors (121 Ohm) for RS-422 and RS-485 protocols. Console redirection, using a serial port

The SBC provides an optional data acquisition circuit containing analog input, analog output, and additional digital I/O features. This circuit is controlled by an FPGA interfaced to the processor via SPI.
The data acquisition features include 16 single-ended / 8 differential analog inputs with 16-bit resolution, programmable input ranges, and a 100KHz maximum sample rate; 4 analog outputs with 16-bit resolution and programmable output ranges; and 11 additional digital I/O lines with selectable 3.3V logic levels, selectable pull-up/down resistors, programmable direction, buffered I/O, and capability for use as counter/timer and PWM circuits.

2.2 Specifications

- 1GHz DMP Vortex86DX3 dual core CPU Up to 2GB DDR3 SDRAM soldered on board
- Up to 2GB DDR3 SDRAM soldered on board
- I/O Support:
    1. 3 or 6 USB2.0 ports (model dependent)
    2. 2 RS-232/422/485 & 2 RS-232 ports
    3. 1 10/100Mbps Ethernet port
    4. 1 Gigabit Ethernet port
    5. 1 SATA port for disk-on-module or external drive
    6. 24-bit dual channel LVDS LCD display
    7. VGA CRT
    8. HD audio
    9. PCIe MiniCard socket shared with mSATA
    10. 16 digital I/O lines with programmable direction
- Data Acquisition:
    1. 16 16-bit analog inputs
    2. 100KHz max sample rate
    3. 4 16-bit analog outputs
    4. 11 additional digital I/O lines with programmable direction
    5. 8 32-bit counter/timers
    6. 4 24-bit pulse width modulators
- PC/104 stackable I/O expansion capability

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 HELIX SBC. These functions should be called prior to any other HELIX specific functions.

- dscInit ( ) : This function initializes the Universal Driver
- HelixInitBoard() : This function initializes the HELIX 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 "Helix.h"
ERRPARAMS errorParams; //structure for returning error code and error string
DSCCBP dscCbp; // structure containing board settings for PCI Express and
FeaturePak boards
BoardInfo *bi=NULL; //Structure containing board base address

int main ()
{
if ( (dscInit ( DSC_VERSION )!= DE_NONE) )
{
        dscGetLastError (&errorParams);
        printf ("dscInit error: %s %s\n", dscGetErrorString (errorParams.ErrCode),
        errorParams.errstring );
        return 0;
}
dsccbp.boardtype = DSC_HELIX;
dsccbp.pci_slot = 0;
dsccbp.io_address[0] =0xEE00;
dsccbp.irq_level[0] = 7;

if (HelixInitBoard (&dsccbp)! = DE_NONE )
{
        dscGetLastError (&errorParams);
        printf ("HelixInitBoard 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_HELIX, 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 HelixADSetSettings

Function Definition
BYTE HelixADSetSettings (BoardInfo* bi, HelixADSETTINGS* settings);

Function Description

This function configures the A/D input range, channel register, and scan settings.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
HelixADSETTINGS	Range	int	0-1; 0 = 5V, 1 = 10V
	Polarity	int	0-1; 0 = bipolar, 1 = unipolar
	Sedi	int	0-1; 0 = single-ended, 1 = differential
	Lowch	int	low channel, 0-15 for SE mode or 0-7 for Diff mode
	Highch	int	high channel,0-7 for SE mode or 0-7 for Diff mode
	ScanEnable	int	0 = disable, 1 = enable
	ScanInterval	int	0 = 10us, 1 = 12.5us, 2 = 20us, 3 = programmable; used only if ScanEnable = 1
	ProgInt	int	100-255, used if Interval = 3
	ADClock	int	0-3: 0 = command bit ADSTART = 1; 1 = falling edge of DIO bit D2; 2 = rising edge of output of counter 0; 3 = rising edge of output of counter 1
	Sign	int	0-1; in Diff mode only: 0 = even channel is high; 1 = odd channel is high

Return Value
Error code or 0.

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

```
HelixADSETTINGS settings;
settings.Polarity = 1;
settings.Range=0;
settings.Sign=0;
settings.Sedi = 0;
settings.Highch = 0;
settings.Lowch = 0;
settings.ADClock = 0;
settings.ScanEnable = 0;
HelixADSetSettings (bi, &settings);
```

4.2 HelixADSetChannelRange

Function Definition

BYTE HelixADSetChannelRange (BoardInfo* bi, HelixADSETTINGS* 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		
HelixADSETTINGS	Lowch	int	low channel, 0-15 for SE mode or 0-7 for Diff mode
	Highch	int	low channel, 0-15 for SE mode or 0-7 for Diff mode

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,

```
HelixADSETTINGS settings;
settings.Lowch = 5;
settings.Highch = 6;
HelixADSetChannelRange (bi, &settings);
```

4.3 HelixADSetChannel

Function Definition
BYTE HelixADSetChannel (BoardInfo* bi, int Channel);

Function Description

This function configures the A/D circuit for a single channel. All other settings remain the same.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Channel	int 0-15 for SE mode or 0-7 for Diff mode

Return Value
Error code or 0.

Usage Example
To configure channels as 5 with all other settings remaining the same,

```
Channel=5;
HelixADSetChannel (bi, Channel);
```

4.4 HelixADConvert

Function Definition

BYTE HelixADConvert (BoardInfo* bi, HelixADSETTINGS* settings, unsigned* Sample);

Function Description

This function configures the A/D input range, channel register, and scan settings as requested by the user, and then sets the timing to be non-turbo, single-channel (non-scan), and software trigger. It then takes a single A/D sample of a single channel. It then triggers the A/D and reads back the A/D sample value.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
Sample	Return value; may be interpreted as unsigned or signed int depending on the input range		
HelixADSETTINGS	Range	int	0-1; 0 = 5V, 1 = 10V
	Polarity	int	0-1; 0 = bipolar, 1 = unipolar
	Sedi	int	0-1; 0 = single-ended, 1 = differential
	Lowch	int	low channel, 0-15 for SE mode or 0-7 for Diff mode
	Highch	int	high channel,0-15 for SE mode or 0-7 for Diff mode
	ScanEnable	int	0 = disable, 1 = enable
	ScanInterval	int	0 = 10us, 1 = 12.5us, 2 = 20us, 3 = programmable
	ProgInt	int	100-255, used if Interval = 3
	ADClock	int	0-3: 0 = command bit ADSTART = 1; 1 = falling edge of DIO bit D2; 2 = rising edge of output of counter 0; 3 = rising edge of output of counter 1
	Sign	int	0-1; in Diff mode only: 0 = even channel is high; 1 = odd channel is high

Return Value

Error code or 0.

Usage Example

To read a single sample value from channel 0, by configuring the A/D input range as 0-5v and single ended,

```
HelixADSETTINGS settings;
unsigned int sample;       // sample reading
Settings.Polarity = 1;
settings.Range = 0;
settings.Sedi = 0;
settings.Highch = 0;
settings.Lowch = 0;
settings.ADClock = 0;
settings.ScanEnable = 0;
HelixADConvert (bi, &settings, &Sample);
printf ("A/D sample value = %d ", Sample);
```

4.5 HelixADSetTiming

Function Definition

BYTE DSCUDAPICALL HelixADSetTiming (BoardInfo* bi, HelixADSETTINGS* settings)

Function Description

This function configures the A/D clock source, and scan settings.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
HelixADSETTINGS	ScanEnable	int	0 = disable, 1 = enable
	ScanInterval	int	0 = 10us, 1 = 12.5us, 2 = 20us, 3 = programmable; used only if ScanEnable = 1
	ProgInt	int	100-255, used if Interval = 3
	ADClock	int	0-3: 0 = command bit ADSTART = 1; 1 = falling edge of DIO bit D2; 2 = rising edge of output of counter 0; 3 = rising edge of output of counter 1

Return Value

Error code or 0.

Usage Example

To configure the A/D clock source to manual and scan enable with programmable 100us,

```
HelixADSETTINGS* dscadsettings;
dscadsettings.ADClock = 0;
dscadsettings.ScanEnable = 1;
dscadsettings.ScanInterval = 3;
dscadsettings.ProgInt = 100;
HelixADSetTiming (bi, &dscadsettings);
```

4.6 HelixADTrigger

Function Definition

BYTE HelixADTrigger (BoardInfo* bi, unsigned int* Sample);

Function Description

This function executes one A/D conversion or scan using the current board settings. It does not perform any configuration of the board but rather uses the current settings. If the board is configured for sample mode (SCANEN = 0), then one A/D conversion will be performed and stored in the sample buffer. If the board is configured for scan mode (SCANEN = 1), then one scan of all channels between 'Highch' and 'Lowch' will be performed and all samples will be stored in the sample buffer.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Sample	pointer to an array of size 16 to hold the return values

Return Value

Error code or 0.

Usage Example

To read samples from 0th channel,

```
unsigned int sample;       // sample reading
HelixADSETTINGS* dscadsettings;
dscadsettings.Polarity = 1;
dscadsettings.Range = 0;
dscadsettings.Sedi = 0;
dscadsettings.Highch = 0;
dscadsettings.Lowch = 0;
dscadsettings.ADClock = 0;
ScanEnable = 0;
HelixADSetTiming (bi, dscadsettings);
HelixADTrigger (bi, &sample);
printf ("The samples from channel 0: 0x%X\n", sample);
```

4.7 HelixADInt

Function Definition
BYTE HelixADInt (BoardInfo* bi, HelixADINT* Helixadint);

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		
HelixADINT	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
	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	SWORD *	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 disabled or NumConversions x Scansize if scan is enabled (Scansize is Highch \96 Lowch + 1)

Return Value
Error code or 0.

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

```
HelixADINT Helixadint;
Helixadint.FIFOEnable=1;
Helixadint.FIFOThreshold=1000;
Helixadint.Cycle=1;
Helixadint.NumConversions=1000;
Helixadint.ADBuffer=(SWORD*) malloc (sizeof (SWORD)*dscIntSettings.NumConversions);
HelixADInt (bi, &Helixadint);
```

4.8 HelixADIntStatus

Function Definition
BYTE HelixADIntStatus (BoardInfo* bi, HelixADINTSTATUS* 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		
HelixADINTSTATUS	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
	BufferPtr;	int	Position in A/D storage buffer, used in continuous mode when buffer is being repetitively overwritten
	OF	int	0 = no overflow, 1 = FIFO overflow (attempt to write to 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 the interrupt routine status,

```
HelixADINTSTATUS intstatus;
HelixADIntStatus (bi, & intstatus);
printf ("No of A/D conversions completed %d\n", intstatus.NumConversions);
```

4.9 HelixADIntPause

Function Definition
BYTE HelixADIntPause (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 pauses the A/D interrupts by turning off the interrupt enable and stopping the A/D clock,

```
HelixADIntPause (bi);
```

4.10 HelixADIntResume

Function Definition
BYTE HelixADIntResume (BoardInfo* bi);

Function Description
This function resumes A/D interrupts from the point at which they were paused.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
To resumes the A/D interrupts from the point at which they were paused,

```
HelixADIntResume (bi);
```

4.11 HelixADIntCancel

Function Definition
BYTE HelixADIntCancel (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.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
To stops A/D interrupts by turning off the interrupt,
HelixADIntCancel `(bi);`

4.12 HelixDASetSettings

Function Definition
BYTE HelixDASetSettings (BoardInfo* bi, int Range, int Sim);

Function Description

This function works with the AD5686R D/A converter. This function sets the D/A output ranges.

Function Parameters

Name	Description	
BoardInfo	The handle of the board to operate on	
Range	int	0-1, 0 = 0-2.5V,1 = 0-5V
Sim	int	0 = single update, 1 = simultaneous mode

Return Value
Error code or 0.

Usage Example
To configure D/A of 0-5V range,

```
int Range;
int Sim;
Range=1;
Sim=0;
HelixDASetSettings (bi, Range, Sim);
```

4.13 HelixDAConvert

Function Definition
BYTE HelixDAConvert (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 HelixDASetSettings ().

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
HelixDAConvert	Channel DACode	int unsigned	0-3 0-65535

Return Value
Error code or 0.

Usage Example
To set channel zero with 5V,

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

4.14 HelixDAConvertScan

Function Definition

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

Function Description

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

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
ChannelSelect int*	Array of 4 flags: 0 = don't update channel n, 1 = update channel n
DACodes unsigned int *	Array of 4 D/A values, 0-65535; values where ChannelSelect[n] = 0 are ignored

Return Value

Error code or 0.

Usage Example

To update channel 0 and 2 with DA code 65535 and 32768 respectively and rest of the channels will to be changed from existing voltage level,

```
ChannelSelect = (int*) malloc (sizeof (int) * 4);
DACodes = (unsigned int*) malloc (sizeof (unsigned int) * 4);
ChannelSelect [0] = 1;
DACodes [0] = 65535;
ChannelSelect [1] = 1;
DACodes [1] = 32768;
HelixDAConvertScan (bi, ChannelSelect, DACodes);
```

4.15 HelixDAFunction

Function Definition
BYTE HelixDAFunction (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.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
HelixDAFunction	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 HelixDAFunction,

```
DAData = 0;
DACommand = 0x10 + (1<<channel); //DA update command
HelixDAFunction (bi, DAData, DACommand);
```

4.16 HelixDAUpdate

Function Definition
BYTE HelixDAUpdate (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 HelixDAConvertScan () function.

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,

```
Range = 1;
Sim = 1;
HelixDASetSettings (bi, Range, sim);
HelixDAConvert (bi, 0, 65535);
HelixDAConvert (bi, 1, 32768);
HelixDAUpdate (bi);
```

4.17 HelixWaveformBufferLoad

Function Definition
BYTE HelixWaveformBufferLoad (BoardInfo* bi, HelixWAVEFORM* waveform);

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		
HelixWAVEFORM	Waveform	unsigned *	pointer to array of 16-bit unsigned data;
	Frames	int	total number of frames in the array
	Framesize	int	no. of channels to be driven = frame size
	Channels	int *	List of channels to be driven by the waveform generator;
	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 operation

Return Value
Error code 0.

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

```
HelixWAVEFORM waveform;
waveform.Waveform = (int *) malloc (sizeof (int) *4);
waveform.Waveform [0] = 0;
waveform.Waveform [1] = 0;
waveform.Waveform [2] = 65535;
waveform.Waveform [3] = 65535;
waveform.Frames= 4;
waveform.FrameSize= 1;
waveform.Channels [0] = 0;
Channel = 0;
Range = 0;
Overrange = 0;
Sim = 0;
HelixDASetSettings (bi, 0, 1);
HelixWaveformBufferLoad (bi, &waveform);
```

4.18 HelixWaveformDataLoad

Function Definition

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

Function Description

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

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Address	Address in buffer at which to store the data; 0-2047;
Channel	Channel number, 0-3
Value	Data value, 0-65535

Return Value

Error code or 0.

Usage Example

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

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

4.19 HelixWaveformConfig

Function Definition
BYTE HelixWaveformConfig (BoardInfo* bi, HelixWAVEFORM* Helixwaveform);

Function Description

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

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
HelixWAVEFORM	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 operation

Return Value
Error code or 0.

Usage Example
To configure waveform generator with 100 Hz frequency,

```
HelixWAVEFORM Helixwaveform;
Helixwaveform.FrameSize = 1;
Helixwaveform. Frames = 500;
Helixwaveform.Clock = 1;
Helixwaveform.Rate = 100;
Helixwaveform.Cycle = 1;
HelixWaveformConfig (bi, & Helixwaveform);
```

4.20 HelixWaveformStart

Function Definition
BYTE HelixWaveformStart (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,

```
HelixWaveformStart (bi);
```

4.21 HelixWaveformPause

Function Definition
BYTE HelixWaveformPause (BoardInfo* bi);

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

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,

```
HelixWaveformPause (bi);
```

4.22 HelixWaveformReset

Function Definition
BYTE HelixWaveformReset (BoardInfo* bi);

Function Description
This function resets the waveform generator and stops all waveform output. The data buffer is not affected.

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,

```
HelixWaveformReset (bi);
```

4.23 HelixWaveformInc

Function Definition
BYTE HelixWaveformInc (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,

```
HelixWaveformInc (bi);
```

4.24 HelixDIOConfig

Function Definition
BYTE HelixDIOConfig (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		
HelixDIOConfig	Port	int	0-3 for port A, B, C, or D
	Config	int	8-bit configuration values for selected port; 0-1: 0 =input,1 = output. For Port 2: 0-255 where 1 = out and 0 = in for each bit. For Port 3:0-7 where 1 = out and 0 = in for each bit

Return Value
Error code or 0.

Usage Example
To set Port A in input mode,

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

4.25 HelixDIOOutputByte

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

Function Description
This function outputs the specified data to the specified port.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
HelixDIOOutputByte	Port	int	0 = A, 1 = B, 2 = C, 3 = D
	Data	int	The data is 8 bits for ports A, B, and C and 3 bits for port D.

Return Value
Error code or 0.

Usage Example
To set Port 0 output as 0x77,

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

4.26 HelixDIOInputByte

Function Definition
BYTE DSCUDAPICALL HelixDIOInputByte (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		
HelixDIOInputByte	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;
HelixDIOInputByte (bi, Port, &Data);
printf ("The PORT 0: 0x%x", Data);
```

4.27 HelixDIOOutputBit

Function Definition
BYTE HelixDIOOutputBit (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		
HelixDIOOutputBit	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 0-2)
	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;
HelixDIOOutputBit (bi, port, bit, digital_val);
```

4.28 HelixDIOInputBit

Function Definition
BYTE HelixDIOInputBit (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		
HelixDIOInputBit	Port	int	0 = A, 1 = B, 2 = C, 3 = D
	Bit	int	0-7 for ports A, B, or C, 0-2 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;
HelixDIOInputBit (bi, port, bit, &value);
printf ("The value at Port 0 Bit 5 is %d", value);
```

4.29 HelixCounterSetRate

Function Definition
BYTE HelixCounterSetRate (BoardInfo* bi, HelixCOUNTER * counter);

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		
HelixCOUNTER	CtrCmd	int	Counter command, 0-15
	Ctrs	int	Counter number, 0-7
	CtrData	unsigned long	Initial load data, 32-bit straight binary
	CtrClk	int	Clock source, must be 2 or 3
	CtrCountDir	int	1=Up direction and 0-Down direction
	CtrReload	int	0 = one-shot counting, 1 = auto-reload
	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	0-3 0 = 1 clocks, 1 = 10 clocks, 2 = 100 clocks, 3 = 1000 clocks , only used if CtrOutEn = 1 and CtrClock = 2 or 3
	Rate	float	Desired output rate, Hz
	ActRate	float	Actual rate resulting from the closest divisor available for the desired rate
	Start	int	0 = don't start counter after configuration; 1 = start counter after configuration
	CtrCmdData	int	Auxiliary data for counter command, 0-3

Return Value
Error code or 0.

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

```
HelixCOUNTER counter;
counter.Ctrs = 1 << 0;
counter.Rate = 100;
counter.CtrOutEn = 1;
counter.CtrOutPol = 1;
counter.ctrOutWidth = 3;
counter.Start = 1;
HelixCounterSetRate (bi, &counter);
```

4.30 HelixCounterConfig

Function Definition
BYTE HelixCounterConfig (BoardInfo* bi, HelixCOUNTER *Ctr);

Function Description
This function programs a counter for up or down counting and starts the counter running. A DIO pin may be selected as the input source. If a DIO pin is not selected as the input, and the counter is counting in down mode, the output may be enabled on the DIO pin. Using a DIO pin for input or output causes the FPGA to automatically set the DIO pin direction as needed.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
HelixCOUNTER	Ctrs	int	Counter number, 0-7
	CtrCmd	int	Counter command, 0-15
	CtrData	unsigned long	Initial load data, 32-bit straight binary
	CtrClk	int	Clock source, must be 2 or 3
	CtrCountDir	int	1=Up direction and 0-Down direction
	CtrReload	int	0 = one-shot counting, 1 = auto-reload
	Start	int	0 = don't start counter after configuration; 1 = start counter after configuration

Return Value
Error code or 0.

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

```
HelixCOUNTER Ctr;
Ctr.Ctrno = 1<<0;
Ctr.CtrClock = 2; //50MHz
Ctr.CtrData = 50000000/100;
Ctr.CtrCountDir = 0;
Ctr.CtrReload = 1;
Ctr.CtrOutEn = 0;
counter.Start = 1;
HelixCounterConfig (bi, & Ctr);
```

4.31 HELIXCounterStart

Function Definition
BYTE DSCUDAPICALL HELIXCounterStart (BoardInfo* bi, int Ctrs);

Function Description
This function starts the counters selected by the mask value Ctrs.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Ctrs	int

Return Value
Error code or 0.

Usage Example
To stop the counter 0 only,

```
int CtrNum;
CtrNum = 1<<0;
```
HELIXCounterStart (bi, `CtrNum`);

4.32 HELIXCounterLatch

Function Definition
BYTE DSCUDAPICALL HELIXCounterLatch (BoardInfo* bi, int Ctrs)

Function Description
This function latches a counter. The data can then be read back with HELIXCounterRead ().

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Ctrs	int

Return Value
Error code or 0.

Usage Example
To latch the current value of counter 0 when it is running,

```
int Ctrs;
Ctrs = 1<<0;
HELIXCounterLatch (bi, Ctrs);
```

4.33 HelixCounterRead

Function Definition
BYTE HelixCounterRead (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
HelixCounterRead	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<<1;
HELIXCounterLatch (bi, Ctrno);
HelixCounterRead (bi, Ctrno, & CtrData);
printf ("Counter Data %ld \r", CtrData);
```

4.34 HelixCounterReset

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

Function Description
This function resets a counter. When a counter is reset, it stops running, all its registers are cleared to 0, and any DIO line used for input or output is released back to normal DIO operation and its direction returns to its previous setting.

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,

```
int CtrNum;
CtrNum = 1<<0;
HelixCounterReset(bi, CtrNum)
```

4.35 HelixCounterFunction

Function Definition
BYTE HelixCounterFunction (BoardInfo* bi, HelixCOUNTER * Ctr);

Function Description
This function can be used to program any desired function into a counter, except reading, which is done by HELIXCounterRead. 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		
HelixCOUNTER	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 set counter 0 as up counting,

```
HelixCOUNTER Ctr;
Ctr.Ctrno = 1<<0;
Ctr.CtrData=0;
Ctr.CtrCmd = 0x02;
Ctr.CtrCmdData = 0x01;
HelixCounterFunction (bi, & Ctr);
```

4.36 HelixPWMConfig

Function Definition
BYTE HelixPWMConfig (BoardInfo* bi, HelixPWM* Helixpwm);

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		
HelixPWM	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
	Command	int	0-15 = PWM command
	CmdData	int	0 or 1 for auxiliary PWM command data (used for certain commands)
	Divisor	unsigned long	24-bit value for use with period and duty cycle commands

Return Value
Error code or 0.

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

```
HelixPWM Helixpwm;
Helixpwm.Num = 0;
Helixpwm.Rate = 100;
Helixpwm.Duty = 50;
Helixpwm.Polarity = 1;
Helixpwm.OutputEnable   = 1;
HelixPWMConfig (bi, & Helixpwm);
```

4.37 HelixPWMStart

Function Definition
BYTE HelixPWMStart (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
HelixPWMStart	Num int PWM number, 0-3

Return Value
Error code or 0.

Usage Example
To start PWM 0,

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

4.38 HelixPWMStop

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

Function Description
This function stops a PWM.

Function Parameters

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

Return Value
Error code or 0.

Usage Example
To stop PWM 0,

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

4.39 HelixPWMReset

Function Definition
BYTE HelixPWMReset (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;
HelixPWMReset (bi, Num)
```

4.40 HelixPWMCommand

Function Definition
BYTE HelixPWMCommand (BoardInfo* bi, HelixPWM* Helixpwm);

Function Description

This function is used to modify a PWM configuration.

Function Parameters

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

Return Value
Error code or 0.

Usage Example
To reset all PWMs,

```
HelixPWM Helixpwm;
Helixpwm.Command = 0x0000; //Stop PWM
Helixpwm.CmdData = 0x0; //Stop all PWM
Helixpwm.Divisor=0;
HelixPWMCommand (bi, & Helixpwm);
```

4.41 HelixUserInterruptConfig

Function Definition
BYTE HelixUserInterruptConfig (BoardInfo* bi, HelixUSERINT* Helixuserint);

Function Description

This function installs a pointer to a user function that runs when interrupts occur.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
HelixUSERINT	Enable	int	0 = disable interrupts, 1 = enable interrupts; if Enable = 0 then no other parameters are required
		void (*IntFunc) (void *parameter);	//pointer to user function to run when interrupts occur
	Mode	int	0 = alone, 1 = before standard function, 2 = after standard function
	Source	int	Selects interrupt source:0 = A/D, 1 = unused, 2 = counter 2 output, 3 = counter 3 output, 4 = digital I/O
	BitSelect	int	0-20 selects which DIO line to use to trigger interrupts; only used if Source = 4
	Edge	int	1 = rising edge, 0 = falling edge; only used if Source = 4

Return Value
Error code or 0.

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

```
void intfunction()
{
      printf ("My function called ");
}
void main()
{
      HelixUSERINT Helixuserint;
      Helixuserint.IntFunc = intfunction;
      Helixuserint.Mode = 0;
      Helixuserint.Source = 2;
      Helixuserint.Enable = 1;
      HelixUserInterruptConfig (bi, &Helixuserint);

}
```

4.42 HelixUserInterruptRun

Function Definition
BYTE HelixUserInterruptRun (BoardInfo* bi, int Source, int Bit, int Edge);

Function Description

This function is used to start user interrupts when they are running in Alone mode.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Source	int 0 = A/D, 1 = unused, 2 = counter 2 output, 3 = counter 3 output, 4 = digital I/O
Bit	int 0-20 selects which DIO line to use to trigger interrupts; only used if Source = 4
Edge	int 1 = rising edge, 0 = falling edge; only used if Source = 4

Return Value
Error code or 0.

Usage Example
To start the user interrupt,

```
Source = 2;
Bit = 0;
Edge = 0;
HelixUserInterruptRun (bi, Source, Bit, Edge);
```

4.43 HelixUserInterruptCancel

Function Definition

BYTE HelixUserInterruptCancel (BoardInfo* bi, int Source);

Function Description

This function is used to start user interrupts when they are running in Alone mode.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Source	int 0 = A/D, 1 = unused, 2 = counter 2 output, 3 = counter 3 output, 4 = digital I/O

Return Value

Error code or 0.

Usage Example

To cancel the user interrupt,

```
Source = 2;
HelixUserInterruptCancel (bi, Source);
```

4.44 HelixInitBoard

Function Definition
BYTE HelixInitBoard (DSCCBP* dsccbp);

Function Description

This function initializes the board.

Function Parameters

Name	Description
dsccbp	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
To initialize the board with I/O address 0xEE00 and interrupt number 7,

```
DSCCB dsccbp; // structure containing board settings
dsccbp.boardtype = DSC_HELIX;
dsccbp.pci_slot = 0;
dsccbp.io_address[0] =0xEE00;
dsccbp.irq_level[0] = 7;

HelixInitBoard (&dsccbp);
```

4.45 HelixFreeBoard

Function Definition
BYTE HelixFreeBoard (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
```
HelixFreeBoard (board);
```

4.46 HelixSPIInit

Function Definition
BYTE HelixSPIInit (BoardInfo* bi);

Function Description
This function initializes SPI to communicate with FPGA. The SPI clock is set to 25MHZ, Full duplex mode is disabled.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example

```
HelixSPIInit (bi);
```

4.47 HelixWrite

Function Definition

```
BYTE DSCUDAPICALL HelixWrite (BYTE address, BYTE value);
```

Function Description

This function writes the 8-bit data at the given address.

Function Parameters

Name	Description	
BoardInfo	The handle of the board to operate on	
Address	int	address
Data	int	data

Return Value

Error code or 0.

Usage Example

To write the value at 0xFF address 0x2A,

```
int Data = 0xFF;
int Address=0x2A;
HelixWrite (bi, Address, Data);
```

4.48 HelixRead

Function Definition

```
BYTE HelixRead (BYTE address, BYTE *value);
```

Function Description

This function reads the 8-bit data from the given address

Function Parameters

Name	Description	
BoardInfo	The handle of the board to operate on	
Address	int	address
Data	int	data read

Return Value

Error code or 0.

Usage Example

To read data from address 0x7D,

```
BYTE Address=0x7D;
BYTE Data;
HelixRead (bi, Address, &Data);
printf ("Data read from address 0x%x", Data);
```

4.49 HelixLED

Function Definition
BYTE HelixLED (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 turn off user LED on the board,

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

4.50 HELIXDX3GPIOPortConfiguration

Function Definition
Int HELIXDX3GPIOPortConfiguration (unsigned char *direction);

Function Description

This function can configure input and or output direction of GPIO ports. Before using any GPIO port we must have to set direction using this function On Helix models without FPGA, the GPIO ports 0 and 1 of DX3 are connected to Digital IO ports A and B.

Function Parameters

Name	Description
*direction	Set direction of GPIO port. 0 – Input, 1 – Output

Return Value
On error 0 or 1.

Usage Example
To configure GPIO port as an input or output.

```
direction [0] = 0;
direction [1] = 1;
HELIXDX3GPIOPortConfiguration (direction);
```

4.51 HELIXDX3GPIOInputByte

Function Definition
Int HELIXDX3GPIOInputByte (int port, unsigned char *data);

Function Description
This function will read data from particular GPIO port. That port should be in input mode.

Function Parameters

Name	Description
port	Port number (0- PortA, 1- PortB)
*data	it will store 8 bit data from particular port address

Return Value
On error 0 or 1.

Usage Example
read data from particular GPIO port.

```
port = 0;
unsigned char data = 0;
HELIXDX3GPIOInputByte (port, &data);
```

4.52 HELIXDX3GPIOOutputByte

Function Definition
Int HELIXDX3GPIOOutputByte (unsigned char OutputByte, int port);

Function Description
This function will write 8 bit data to data register of port.

Function Parameters

Name	Description
port	Port number (0- PortA, 1- PortB)
OutputByte	One byte data will store to data register. (Range- 0 to 255)

Return Value
On error 0 or 1.

Usage Example
It will write a Byte to data register of port.

```
Port = 1;
OutputByte = 50;
HELIXDX3GPIOOutputByte (OutputByte, port);
```

4.53 HELIXSetSerialPortMode

Function Definition
Int HELIXSetSerialPortMode (int ModeValue);

Function Description
This function will set mode of protocol based on mode values. There are total 10 valid mode values you can configure using this function. Mode value is nothing but a first four bits [0-3] of a Byte. So, in decimal (2 to 11) are valid mode values to configure protocol on serial port. Serial Port 1 to 4 all are connected to DX3 processor.

Function Parameters

Name	Description
ModeValue	This is ten different mode values which can configure serial port as different protocol (ex. RS232, RS422 and RS485).

UART3	UART4	Slew Rate Limited	Mode Port2 [3:0]
RS232	RS232	Yes	0010
RS232	RS232	No	0011
RS232	RS485	Yes	0100
RS232	RS485	No	0101
RS232	RS422	Yes	0110
RS232	RS422	No	0111
RS485	RS485	Yes	1000
RS485	RS485	No	1001
RS422	RS422	Yes	1010
RS422	RS422	No	1011

Return Value
On error 0 or 1.

Usage Example
There are three ways to get input from user.

```
ModeValue = 2; //COM3=RS232, COM4=RS232, Slew Rate=Yes
HELIXSetSerialPortMode (ModeValue);
```

4.54 HELIXSerialDisplayMode

Function Definition
Int HELIXSerialDisplayMode (void);

Function Description
This function will print all valid ranges of mode. It will print information about Comport number, slew rate and mode value. These mode values will configure serial ports as RS232 or RS422 or RS485 protocol. Serial Port 1 to 4 all are connected to DX3 processor.

Function Parameters

Name	Description
void	void

Return Value
On error 0 or 1.

Usage Example
It will print all possible ranges to set serial port as a different protocol.

```
HELIXSerialDisplayMode (void);
```

4.55 HELIXDX3WDTConfiguration

Function Definition
Int HELIXDX3WDTConfiguration (int TimerValue);

Function Description
This function will configure timeout value of timer and it will start the timer. The computer will restart when timer will reach zero. DX3 Processor's internal Watchdog timer is used in Helix models without FPGA.

Function Parameters

Name	Description
TimerValue	timer value (Range 2 to 512seconds)

Return Value
On error 0 or 1.

Usage Example
It will configure timeout value of timer and it will start the timer.

```
TimerValue = 10;
HELIXDX3WDTConfiguration (TimerValue);
```

4.56 HELIXDX3WDTDisable

Function Definition
Int HELIXDX3WDTDisable (void);

Function Description
This function will disable the watchdog timer. The watchdog timer will stop after successfully call this function.

Function Parameters

Name	Description
void	void

Return Value
On error 0 or 1.

Usage Example
This function will disable the watchdog timer.

```
HELIXDX3WDTDisable (void);
```

4.57 HELIXDX3WDTRetrigger

Function Definition
Int HELIXDX3WDTRetrigger (void);

Function Description
This function will retrigger the Watchdog timer from initial value which is given by user. It will start again to count down to zero.

Function Parameters

Name	Description
void	void

Return Value
On error 0 or 1.

Usage Example
This function will retrigger the Watchdog timer.

```
HELIXDX3WDTRetrigger (void);
```

4.58 HELIXDX3ReadByte

Function Definition
Int HELIXDX3ReadByte (unsigned char address, unsigned char *ReadValue);

Function Description
This function will read 8 bit value from given address. User can read a byte from valid register.

Function Parameters

Name	Description
address	8 bit address value in hexadecimal
*ReadValue	Byte value of register will store in this pointer variable.

Return Value
On error 0 or 1.

Usage Example
This function will read a Byte from given address.

```
address = 0x98;
unsigned char ReadValue =  0;
HELIXDX3ReadByte (address, &ReadValue);
ReadValue = value of 0x98 register;
```

4.59 HELIXDX3WriteByte

Function Definition
Int HELIXDX3WriteByte (unsigned char address, unsigned char Writevalue);

Function Description
This function will write 8 bit value to particular register which is given from user.

Function Parameters

Name	Description
address	8 bit register value in hexadecimal format
Writevalue	Byte value Range (0-255).

Return Value
On error 0 or 1.

Usage Example
This function will write a Byte value to given address.

```
address =  0x99;
Writevalue = 150;
HELIXDX3WriteByte (address, Writevalue);
```

5. UNIVERSAL DRIVER DEMO APPLICATION DESCRIPTION

The Universal Driver supports the following applications on the HELIX 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

It 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. 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. The programmer is responsible for knowing the format of the waveform buffer, which determines the address at which to store the data.

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

This function can be used to program any desired function into a counter, except reading which is done by HelixCounterRead (). 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.

5.5 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. The counter is started immediately. This function is a simplified version of CounterConfig () optimized for the most popular application of rate generator.

5.6 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.

5.7 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.

5.8 AD Trigger

This function executes one A/D conversion or scan using the current board settings. It does not perform any configuration of the board but rather uses the current settings. If the board is configured for sample mode (SCANEN = 0), then one A/D conversion will be performed and stored in the Sample buffer. If the board is configured for scan mode (SCANEN = 1), then one scan of all channels between chlow and chhigh will be performed and all samples will be stored in the Sample buffer.

5.9 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 HelixADIntCancel () 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.

5.10 LED

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

5.11 HelixGpioControl

This function supports two types of direct digital I/O operations: input byte and output byte. User can configure GPIO port either as an input or output using HELIXDX3GPIOPortConfiguration (unsigned char *direction) function. User can read value of GPIO address using HELIXDX3GPIOInputByte (int, unsigned char *) function and can write 8 bit range value using HELIXDX3GPIOOutputByte (unsigned char, int) function.

5.12 HelixWatchDogTimer

The computer regularly restarts the watchdog timer to prevent it from timing out. We can set timer value using HELIXDX3WDTConfiguration (int) function. The timer will elapsed and generate timeout signal. The timeout signal is used to initiate corrective actions. User can retrigger this timer using HELIXDX3WDTRetrigger () function.

5.13 HelixSerialPort

This application will configure protocol and slew rate for COM3 and COM4. COM1 and COM2 are fixed as RS232 protocol. There are ten valid modes to configure COM3 and COM4 as RS232 or RS422 or RS485 protocol with high or low slew rate.

6. UNIVERSAL DRIVER DEMO APPLICATION USAGE INSTRUCTIONS

The following section illustrates how to enter the HELIX 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 Channel Number(0 – 3,Default 0):
- Enter D/A range value (0 = 0-2.5V, 1 = 0-5V) <default: 0):
- Enter output code (0-65535, default:32768):

To set channel 0 with 5V, run the DA Convert application and provide input as follows,

D/A range = 0
Channel Number=0
Output code=65535

Measure the voltage on channel 0 using a multi meter. It should show 5V, the application generated expected voltage.

6.2 D/A Scan Conversion

This application gets input from user as follows:

- Enter output range (0 = 0-2.5V, 1 = 0-5V) <default: 0> :)
- Enter Sim value (0 =Individual Update 1=Simultaneous update):
- Enter the channel enable flag for channel 0 (0 for FALSE, 1 for TRUE; default: 1):
- Enter the output code for channel 0 (0-65535; default: 32768):
- Enter the channel enable flag for channel 1 (0 for FALSE, 1 for TRUE; default: 1):
- Enter the output code for channel 1 (0-65535; default: 32768):
- Enter the channel enable flag for channel 2 (0 for FALSE, 1 for TRUE; default: 1):
- Enter the output code for channel 2 (0-65535; default: 32768):
- Enter the channel enable flag for channel 3 (0 for FALSE, 1 for TRUE; default: 1):
- Enter the output code for channel 3 (0-65535; default: 32768):

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

- Enter output range (0 = 0-2.5V, 1 = 0-5V) <default: 0> :) 1
- Enter Sim value (0 =Individual Update 1=Simultaneous update):1
- Enter the channel enable flag for channel 0 (0 for FALSE, 1 for TRUE; default: 1): 1
- Enter the output code for channel 0 (0-65535; default: 32768): 65535
- Enter the channel enable flag for channel 1 (0 for FALSE, 1 for TRUE; default: 1): 0
- Enter the output code for channel 1 (0-65535; default: 32768): 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; default: 32768): 65535
- Enter the channel enable flag for channel 3 (0 for FALSE, 1 for TRUE; default: 1): 0
- Enter the output code for channel 3 (0-65535; default: 32768): 0

Measure the voltage on channel 0 and 2 using a multi meter. It should show 5V, 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 Range value (0-1 0 = 0-2.5V, 1 = 0-5V) <Default =0>:
- 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 D1, 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:

- Enter D/A Channel Number (0-3):0
- Enter Range value (0-1 0 = 0-2.5V, 1 = 0-5V) <Default =0>:1
- Enter waveform size (1-2048) <Default =500>:500
- Enter Clock source (0-3 , 0 -Manual, 1-Counter 0 output, 2-Counter 1 output , 3-External trigger on DIO pin D1, Default =1 :1
- Select waveform type (0-Sine wave 1-Sawtooth Wave) <Default =0>: 0
- Enter waveform frequency <Default =100>:100
- Enter waveform mode(0-1, 0-One shot 1-Repeat mode, default = 1:)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 0-7 if port=3 else 0-255 or q to quit:

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

- Enter port number (0-3): 3
- Enter value 0-7 or q to quit: 7

The Byte value 7 is sent to port 3.

Measure voltage on Port 3 pins using a multi meter. It should show 3.3 on all the pins, 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:

- Enter port number (0-3):0

The application should show 0x01 on the screen.

6.5 Counter Function Application

This application gets input for counter 0 - 7from the user as follows:

- Configure counter number: 0 (1 = YES, 0 = NO, Default = 1)

If the user enters YES, the applications gets the following inputs, otherwise it will skip that particular counter

- Enter Counter Direction (0 = down counting, 1 = up counting; default = 0):

If the user enters direction as down counting, the application gets the following inputs,

- Select Clock source (0=External ,2=Internal clock 50MHz,3=Internal clock 1MHz; Default = 2):
- Enter Output Enable (0=disable, 1=Enable; default = 1):
- Enter Output Polarity (0=negative, 1= positive; default = 1):
- Enter output pulse width (0-3 ,0 = 1 clocks, 1 = 10 clocks, 2 = 100 clocks, 3 = 1000 clocks, default = 3) :
- Enter Counter Frequency (default = 100) :

If the user enters direction as up counting, the application gets the following input,

- Select Clock source (0=External ,2=Internal clock 50MHz,3=Internal clock 1MHz; Default = 0):

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

- Configure counter number 0: (1 = YES, 0 = NO, Default = 1):1
- Enter Counter Direction (0 = down counting, 1 = up counting; default = 0):0
- Select Clock source (0=External ,2=Internal clock 50MHz,3=Internal clock 1MHz; Default = 2):2
- Enter Output Enable (0=disable, 1=Enable; default = 1):1
- Enter Output Polarity (0=negative, 1= positive; default = 1):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 0th 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 for counter 0-7 from the user as follows:

- Configure counter number 0: (1 = YES, 0 = NO, Default = 1):

If the user enters YES, the applications gets the following inputs, otherwise it will skip that particular counter

- Enter Counter frequency rate (1-50MHz) (Default = 1000):
- Enter Output Enable (0=disable, 1=Enable):

If the Output is enabled, the application gets the following inputs from the user,

- 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) :
- Start counter immediately after configuration(1 = YES, 0 = NO(Later), Default = 1)

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

- Configure counter number 0 (1 = YES, 0 = NO, Default = 1):1
- Enter Counter frequency rate (1-50MHz) (Default = 1000):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) :3
- Start counter immediately after configuration(1 = YES, 0 = NO(Later), Default = 1):1

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, Default = 0):
- Select Output Frequency (1-50MHz ,Default = 100):
- Select Duty cycle value (1-100, Default = 50):
- Select Polarity (0 = pulse high, 1 = pulse low , Default = 0):
- 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, Default = 0):0
- Select Output Frequency (1-50MHz ,Default = 100): 100
- Select Duty cycle value (1-100, Default = 50): 50
- Select Polarity (0 = pulse high, 1 = pulse low, Default = 0): 0
- Press any key to start PWM

Place an oscilloscope probe on PWM channel 0 output pin (Port C 4th 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 (2-3; 2 = counter 2 output, 3 = counter 3 output, default =2) :
- 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 (2-3; 2 = counter 2 output, 3 = counter 3 output, default =2) :2
- 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, default: 0):
- Enter A/D polarity (0 for bipolar, 1 for unipolar, default: 0):
- Enter A/D full scale value (0 = 5V, 1 = 10V; default 0):
- Enter input mode (0 = single-ended, 1 = differential; default 0):
- Enter Clock source (0 = Manual, 1 = falling edge of external trigger (DIO bit D2), default 0):

The application gets the following inputs from the user if the clock source is not manual,

- Enter number of A/D conversions (must be multiple of FIFO threshold value default 1000):
- 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: 1000):

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, default: 0):0
- Enter A/D polarity (0 for bipolar, 1 for unipolar):1
- Enter A/D full scale value (0 = 5V, 1 = 10V; default 0):0
- Enter input mode (0 = single-ended, 1 = differential; default 0): 0
- Enter Clock source (0 = Manual, 1 = falling edge of external trigger (DIO bit D2), default 0):0

It will perform the sampling operation and display the AD sample value.

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 full scale value (0 = 5V, 1 = 10V; default 0):
- Enter input mode (0 = single-ended, 1 = differential; default 0):
- Enter Scan Interval (0 = 10us, 1 = 12.5us, 2 = 20us, 3 = programmable, default 0):

The application gets the following input from user if Scan interval was chose as programmable,

- Enter Programmable Scan Interval value (interval is this value times 100ns)(100-255, default 100):
- 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 (0-15, default: 0):0
- Enter the High channel (0-15, default:7):4
- Enter A/D full scale value (0 = 5V, 1 = 10V; default 0):0
- 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, default: 0):
- Enter A/D polarity (0 for bipolar, 1 for unipolar , default: 0):
- Enter A/D full scale value (0 = 5V, 1 = 10V; default 0):
- Enter input mode (0 = single-ended, 1 = differential; default 0):
- Enter Clock source (0 = Manual, 1 = falling edge of external trigger (DIO bit D2), default 0):

The application gets the following inputs from the user if the clock source is not manual,

- Enter number of A/D conversions (must be multiple of FIFO threshold value default 1000):
- 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: 1000):

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, default: 0):0
- Enter A/D polarity (0 for bipolar, 1 for unipolar):1
- Enter A/D full scale value (0 = 5V, 1 = 10V; default 0):0
- Enter input mode (0 = single-ended, 1 = differential; default 0): 0
- Enter Clock source (0 = Manual, 1 = falling edge of external trigger (DIO bit D2), default 0):0

It will perform the sampling operation and display the AD sample values.

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:15):
- Enter A/D polarity (0 for bipolar, 1 for unipolar, default: 0):
- Enter A/D full scale value (0 = 5V, 1 = 10V; default 0):
- Enter input mode (0 = single-ended, 1 = differential; default 0):
- Enter Scan Interval (0 = 10us, 1 = 12.5us, 2 = 20us, 3 = programmable, default 0):
- Enter Programmable Scan Interval value (interval is this value times 100ns)(100-255, default 100):
- Enter A/D clock source (1-3, 1= External trigger 2=Counter 0, 3=Counter 1 default=2) :
- Enter A/D sampling Rate (Default 1600) :
- 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 20000Hz, 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 full scale value (0 = 5V, 1 = 10V; 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):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) :20000
- Enter number of A/D conversions (must be multiple of FIFO threshold value default 1600):1000
- 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) :1000

The application scans AD channel from 0 to 4 at the 100Hz 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.

6.14 HelixGpioControl Application

This application gets input from the user as follows:

- Enter direction for PortA and PortB (0 – Input, 1 – Output)

To perform write operation user has to select write a byte from menu.

- Enter Port number for write value to port (0 – PortA, 1- PortB)
- Enter 8 bit wide value (Range 0- 255)

To perform Read operation user has to select read a byte from menu.

- Enter port number to read value from that port (0 –PortA, 1- PortB)

The application will update a byte value to particular port. So User can read value from another port using loopback connection between two ports.

6.15 HelixWatchDogTimer Application

This application gets input from the user as follows:

- Disable the watchdog timer.
- Enter Timeout value from user in seconds (2-512 sec).

Now, timer will start to count down to zero. To Retrigger this timer user has to follow below steps.

- User can press any key from keyboard to retrigger timer while timer is running.
- Retrigger the timer.

This application will use to watch on particular event. So user can do appropriate action on that event.

6.16 HelixSerialPort

This application gets input from the user three ways as follows:

1. File Input:
    - File contains Following value:0010

UART3	UART4	Slew Rate Limited	Mode Port2 [3:0]
RS232	RS232	Yes	0010
RS232	RS232	No	0011
RS232	RS485	Yes	0100
RS232	RS485	No	0101
RS232	RS422	Yes	0110
RS232	RS422	No	0111
RS485	RS485	Yes	1000
RS485	RS485	No	1001
RS422	RS422	Yes	1010
RS422	RS422	No	1011

    - Run application with file name on command line.
    - HelixSerialPort.exe HelixSerialconfig.ini
    - Convert string value 0010 to decimal value.

2. Command line Input:
    - HelixSerialPort.exe 0010
    - Convert string value 0010 to decimal value

3. User input:
    - Enter protocol for port 3: 0 = RS-232, 1 = RS-422, 2 = RS-485
    - Enter protocol for port 4: 0 = RS-232, 1 = RS-422, 2 = RS-485
    - Enter slew rate configuration (0 = full, 1 = limited)
    - Convert these input value to correct mode value.

This application will configure COM3 and COM4 as one among three protocol with slew rate option.

7. COMMON TASK REFERENCE

7.1 Data Acquisition Feature Overview

I/O Connectors

HELIX SBC provides two I/O connectors for the attachment of all user I/O signals. Diamond's I/O cable number 6980515 (2 per board) may be used to connect the user's signals to these connectors. This cable comes as a part of the HELIX Cable Kit. This cable has a common 2mm pitch IDC connector at the opposite end which may be plugged into a custom board or may be cut off and the wiring then used as needed. Unused signals do not need to be terminated. However, if the cable wiring is cut, care should be taken to avoid shorting any unused wires to any other voltages in the system, in order to prevent possible damage to the board or incorrect analog I/O readings.

Analog inputs

There are 16 analog input channels, numbered 0-15, located on connector J18 pins 7-22. In single-ended mode, each pin acts as an individual channel. The voltage on each channel is measured relative to the reference analog ground. 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 4 analog output channels, numbered 0-3, located on connector J18 pins 1-4. Analog outputs operate in single-ended mode only and are always referenced to the analog ground.

Waveform generator

The D/A waveform generator includes a 2048 x 18 bit waveform buffer, which is organized as 16 bits of D/A data and a 2 bit channel tag. Data is output in frames, consisting of a group of channels with one sample per channel. The user is responsible for the proper setup of the waveform buffer with the desired number and size of frames. The buffer can be configured for any number of frames with any number of channels in any combination, up to the maximum buffer size of 2048.

The waveform generator can be clocked in multiple ways, including software command, external digital signal on I/O connector J18 pin 24 (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	1800Hz
2	900Hz
3	600Hz
4	450Hz

Digital I/O signals

There are 27 digital I/O signals, divided into four groups as DIOA0-DIOA7, DIOB0-DIOB7, DIOC0-DIOC7 and DIOD0-DIOD2. Port A-C are on I/O connector J17 pins 1-24 and port D (DIOD0-DIOD2) is on I/O connector J18 pins 23-25.

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.

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 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 J18Pin	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

Pulse Width Modulators (PWMs)

This board includes 4 24-bit pulse width modulator circuits (PWMs). These circuits can be programmed to produce an output square wave up to 25MHz with a duty cycle anywhere from 0% to 100% (these limits are of course DC signals). The output polarity is programmable. PWM outputs are available on digital I/O port C pins. PWM operation is as follows: Each PWM contains a period counter and a duty cycle counter which are programmed for the desired values. When the PWM starts, both counters start to count down simultaneously, and the output pulse is set to the desired active polarity. When the duty cycle counter reaches 0, it stops, and the output changes to the inactive polarity. When the period counter reaches zero, both counters reload, the output is made active again, and the cycle repeats. The PWM duty cycle can be updated in real time without having to stop the circuit from running.

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 J18 Pin	DIO C Bit	PWM
21	4	0
22	5	1
23	6	2
24	7	3

7.2 Data Acquisition Software Task Reference

This section describes the various data acquisition tasks that may be performed with Helix 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 HelixInitBoard () this function must be called prior to any other function involving the HELIX SBC.
2. At the termination of the program the programmer may use HelixFreeBoard (), 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 falling edge of DIO bit D2, 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 J18 pin 25.

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. HelixADSetSettings () is used to select the input type, input voltage range, and input channel range.
2. HelixADConvert () is used to generate A/D conversions for A/D sample mode and HelixADTrigger () can used to generate A/D conversions for A/D scan 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 HELIX hardware user manual. The programmer may also convert this number to whatever engineering units are appropriate for the application.
3. 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, falling edge of DIO bit D2 (digital I/O port D2), 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 1 KHz 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. HelixADSetSettings () is used to select the input type, input voltage range, and input channel range.
2. HelixADInt () is used to install the interrupt handler.
3. If a counter/timer is being used to control A/D timing, then HelixCounterSetRate () is used to program the counter/timer for the desired sample rate.
4. To monitor A/D operations, use HelixADIntStatus ().

The interrupt operation can be monitored with the HelixADIntStatus () 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. HelixDASetSettings () to select the output range and simultaneous update mode if desired
2. HelixDAConvert () to update a single channel with the given value

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

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

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

1. HelixDASetSettings () to select the output range; set Sim = 1
2. HelixDAConvertScan () to load the given data into the selected channels
3. HelixDAUpdate () 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 _HelixWaveformBufferLoad () 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.

_HelixWaveformDataLoad () 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 _HelixWaveformConfig () 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 _HelixWaveformStart (). 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 _HelixWaveformPause (). The waveform may be restarted in its current position by using _HelixWaveformStart () again.

To reset the waveform generator use _HelixWaveformReset (). 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 HelixWaveformConfig () should be called followed by _HelixWaveformStart ().

If software increment is selected, the waveform is incremented with the function _HelixWaveformInc (). 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 HelixDIOConfig () configures all 3 or 8 bits of a selected port
```

Then execute whichever I/O function is desired. Byte read/write enables 3 or 8 bits of digital I/O to be updated at once. Bit operation enables a single bit to be updated.

```
BYTE HelixDIOOutputByte () outputs 8 bits of data
BYTE HelixDIOInputByte (BoardInfo* bi, int Port, byte* Data);
BYTE HelixDIOOutputBit (BoardInfo* bi, int Port, int Bit, int Value);
BYTE HelixDIOInputBit (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 HelixCounterSetRate (). 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 HelixCounterConfig () 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 HELIXCounterLatch () to latch all enabled/configured Counters at an instance.
3. Use HelixCounterRead () 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.
4. When the counting function is no longer needed, use HelixCounterReset () 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 HelixCounterFunction(). 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 or the counter's designated I/O pin on port C.
7	Enable auto-reload.

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

8	Enable counter output pulse on designated I/O pin on port C
9	Configure counter output pulse width; options include 1, 10, 100, or 1000 clock periods

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 one more counters. When a counter is reset its configuration and contents are lost.

Alternatively, the function HelixCounterReset() 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 latch by using HELIXCounterLatch () and the data can be read by using the HelixCounterRead () function.
When the counting function is no longer needed, the function HelixCounterReset () 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. HelixPWMConfig () configures the selected PWM for output frequency, duty cycle, and polarity. The PWM may optionally be started as well.
2. HelixPWMStart () can be used to start the PWM running if the config function did not start it.

To stop a PWM:

HelixPWMStop () 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 HelixPWMStart ().
To reset a PWM and return its assigned digital I/O output pin to normal operation use HelixPWMReset ().

To implement special functions, such as changing the duty cycle or frequency of a PWM while it is running, use HelixPWMCommand (). 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
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:

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

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

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

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 _HelixLED ().

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 HelixDAConvert ()

Step-By-Step Instructions

Call HelixDAConvert () 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, range=1, dasim=0;
unsigned int DACode = 65535;

// To D/A settings for 0-5V
if (HelixDASetSettings (bi, range, dasim)!= DE_NONE )
{
      dscGetLastError (&errorParams);
      printf ("HelixDASetSettings error: %s %s\n", dscGetErrorString
      (errorParams.ErrCode), errorParams.errstring);
      return 0;
}
/* Step 2 */
if (HelixDAConvert (bi, channel, DACode)! =DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ("HelixDAConvert 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 HelixDAConvertScan ().

Step-By-Step Instructions

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

Call HelixDAConvertScan () and pass it to a pointer to the scanned array values - this will generate 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 3 with DA code 65535 and 32768 respectively and the rest of the channels not to be changed from existing voltage level:

//memory allocation

```
ChannelSelect = (int*) malloc (sizeof (int) * 4);
DACodes = (unsigned int*) malloc (sizeof (unsigned int) * 4);
ChannelSelect [0] = 1;
DACodes [0] = 65535;
ChannelSelect [3] = 1;
DACodes [3] = 32768;
if ((HelixDAConvertScan( bi,ChannelSelect, DACodes ) ) != DE_NONE )
{
                  dscGetLastError (&errorParams);
                  fprintf (stderr, "HelixDAConvertScan error: %s %s\n",
                  dscGetErrorString (errorParams.ErrCode), errorParams.errstring);
                  free (DACodes); // remember to deallocate malloc () memory
                  free(ChannelSelect);
                  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 _HelixDIOConfig () ,HelixDIOOutputByte (), HelixDIOInputByte (), HelixDIOOutputBit (), HelixDIOInputBit ().

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 _HelixDIOConfig () to set the direction.

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;

/* 1. Configure Port 0 in output mode */
config =1;
port=0;
if(HelixDIOConfig(bi,port,config)!= DE_NONE)
{
            dscGetLastError (&errorParams);
            printf ("HelixDIOConfig error: %s %s\n", dscGetErrorString
            (errorParams.ErrCode), errorParams.errstring );
            return;
}
/* 2.  input bit - read bit 6 (port 0 is in input mode) */

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

```
bit = 6;
if ( (HelixDIOInputBit ( bi, port, bit, &digital_value ) != DE_NONE) )
{
      dscGetLastError (&errorParams);
      printf ("HelixDIOInputBit 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) */
config =0;
port = 0;
if(HelixDIOConfig(bi,port,config)!= DE_NONE)
{
            dscGetLastError(&errorParams);
            printf("HelixDIOConfig error: %s %s\n",
            dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
            return ;
}
if ( (HelixDIOInputByte ( bi, port, &input_byte ) != DE_NONE) )
{
      dscGetLastError (&errorParams);
      printf ("HelixDIOInputByte 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;
if(HelixDIOConfig(bi,port,config)!= DE_NONE)
{
            dscGetLastError (&errorParams);
            printf ("HelixDIOConfig error: %s %s\n",
            dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
            return;
}
if ( (HelixDIOOutputBit (bi, port, bit, digital_val) != DE_NONE) )
{
      dscGetLastError (&errorParams);
      printf ("HelixDIOOutputBit 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;
config = 1;
if(HelixDIOConfig(bi,port,config)!= DE_NONE)
{
            dscGetLastError (&errorParams);
            printf("HelixDIOConfig error: %s %s\n", dscGetErrorString
            (errorParams.ErrCode), errorParams.errstring );
            return;
}
```

```
output_byte = 0xFF;
if( (HelixDIOOutputByte ( bi, port, output_byte ) != DE_NONE) )
{
      dscGetLastError (&errorParams);
      printf ("HelixDIOOutputByte 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: _HelixPWMConfig () , _HelixPWMStart () and HelixPWMStop ().

Step-By-Step Instructions

Create a HelixPWM structure variable to hold the PWM settings and initialize the PWM structure variables, then call HelixPWMConfig () to configure the PWM. Call _HelixPWMStart () to start the PWM, and finally call HelixPWMStop () to stop the running PWM signal.

Example of Usage for PWM Operations

```
HelixPWM 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 (HelixPWMConfig (bi, &pwm)! =DE_NONE)
{
      dscGetLastError(&errorParams);
      printf ("HelixPWMConfigerror:%s%s\n", dscGetErrorString(errorParams.ErrCode),
      errorParams.errstring );
      return 0;
}

//The following function start the PWM
if (HelixPWMStart (bi,pwm.Num) !=DE_NONE)
 {
      dscGetLastError (&errorParams);
      printf("HelixPWMStarterror: %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 (HelixPWMStop (bi,pwm.Num) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf("HelixPWMStop 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: _HelixCounterConfig () and HelixCounterFunction ().

Step-By-Step Instructions

Create a HelixCOUNTER structure variable to hold the counter settings and initialize the counter structure variables, then call _HelixCounterConfig_() to configure the counter, call HELIXCounterStart() to start the configured counters, call HELIXCounterLatch() to latches a counter, call HelixCounterRead() to read back the latched value of a counter and finally the _HelixCounterReset () to stop the running counter.

Example of Usage for Counter Operations

```
int LatchCounter = 0, StartCounter=0;
HelixCOUNTER Ctr;
float frequency = 0.0;

LatchCounter |= (1<<counterNo);
StartCounter |=  (1 << counterNo ) ;
counter.Ctrs = 1 << counterNo;
counter.CtrCountDir =0;
counter.CtrClk = 2; //50MHz
counter.CtrData = 50000000/ frequency;
counter.CtrOutEn = 1;
counter.CtrOutPol = 1;
counter.ctrOutWidth = 3;
counter.CtrReload = 1;
counter.Start = 0;
//The following function configures the counter with desired rate
If (HelixCounterConfig (bi, & counter)! =DE_NONE)
{
      dscGetLastError (&errorParams);
      printf("HelixCounterConfigerror:%s%s\n",
      dscGetErrorString(errorParams.ErrCode),errorParams.errstring);
      return 0;
}
/*To start the configured counters*/
if(HELIXCounterStart(bi,StartCounter) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf("HELIXCounterStart                                          error:%s%s\n”,
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}
Printf ("Press any key to stop counting \n");
while ( !kbhit())
{
      dscSleep (1000);
      if(HELIXCounterLatch(bi,LatchCounter) !=DE_NONE)
      {
            dscGetLastError(&errorParams);
            printf("HELIXCounterLatcherror:%s%s\n",
            dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
            return 0;
```

```
      for (counterNo = 0; counterNo < 8; counterNo++)
      {
            if ((LatchCounter>>counterNo) & 0x01)
            {
                  if(HelixCounterRead(bi,(1<<counterNo),&counter.CtrData)!=
DE_NONE)
                  {
                        dscGetLastError (&errorParams);
                        printf("HelixCounterRead error:%s%s\n",
                        dscGetErrorStringerrorParams.ErrCode),
                        errorParams.errstring );
                        return 0;
                  }
            printf ("Counter %d Data %ld \r", counterNo, counter.CtrData);
            }
      }
}
/* to reset the running counters*/
for (counterNo = 0; counterNo < 8; counterNo++)
{
      if ((LatchCounter>>counterNo) & 0x01 == 1)
      {
            if (HelixCounterReset(bi,(1<<counterNo)) !=DE_NONE)
            {
                  dscGetLastError (&errorParams);
                  printf("HelixCounterReset        error:        %s        %s\n",
                  dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
                  return 0;
            }
      }
}
```

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: HelixCounterSetRate (), HELIXCounterStart (), HELIXCounterLatch (), HelixCounterRead (), and HelixCounterReset () .

Step-By-Step Instructions

Create a HelixCOUNTER structure variable to hold the counter settings and initialize the counter structure variables. Call HelixCounterSetRate () to program a counter for timer mode with down counting and continuous operation (reload enabled), call HELIXCounterStart () to start the configured counters, call HELIXCounterLatch () to latches a counter, then call HelixCounterRead () to read the counter value. Finally call HelixCounterReset () to stop the running counter.

Example of Usage for Counter Set Rate Operations

```
int LatchCounter = 0, StartCounter=0;
HelixCOUNTER counter;
LatchCounter |= (1<<counterNo);
counter.Ctrs = 1 << counterNo;
counter.Rate = 1000;   // Counter frequency
counter.CtrOutEn = 1;
counter.CtrOutPol = 1;
counter.ctrOutWidth = 3;
counter.Start = 0;

if(HelixCounterSetRate(bi,&counter) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf("HelixCounterSetRate error:%s%s\n",
      dscGetErrorString (errorParams.ErrCode), errorParams.errstring);
      return 0;
}
/*To start the configured counters*/
if(HELIXCounterStart(bi,StartCounter) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf("HELIXCounterStart error:%s%s\n",
      dscGetErrorString (errorParams.ErrCode), errorParams.errstring);
      return 0;
}

printf ("Press ENTER key to stop counting \n");

while ( !kbhit())
{
      dscSleep (1000);
      if(HELIXCounterLatch(bi,LatchCounter) !=DE_NONE)
      {
            dscGetLastError (&errorParams);
            printf("HELIXCounterLatch error:%s%s\n",
            dscGetErrorString (errorParams.ErrCode), errorParams.errstring );
            return 0;
      }
```

```
      for (counterNo = 0; counterNo < 8; counterNo++)
      {
            if ((LatchCounter>>counterNo) &0x01)
            {
                  if(HelixCounterRead(bi,(1<<counterNo),&counter.CtrData)!=
DE_NONE)
                  {
                        dscGetLastError (&errorParams);
                        printf ("HelixCounterRead error: %s %s\n",
                        dscGetErrorString (errorParams.ErrCode),
                        errorParams.errstring);
                        return 0;
                  }
                  printf ("Counter %d Data %ld \r", counterNo, counter.CtrData);
            }
                  }
                  fflush (stdout);
            }



      /* to reset the running counters*/
      for (counterNo = 0; counterNo < 8; counterNo++)
      {
            if ( (LatchCounter>>counterNo) &0x01)
            {
                  if (HelixCounterReset(bi,(1<<counterNo)) !=DE_NONE)
                  {
                        dscGetLastError (&errorParams);
                        printf
                        ("HelixCounterReseterror:%s%s\n",dscGetErrorString(errorPar
                        ams.ErrCode), errorParams.errstring );
                        return 0;
                  }
            }
      }
```

7.9 Performing User Interrupt Operations

Description:

The User Interrupt application configures counter 2 and counter 3 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: HelixUserInterruptConfig (),HelixUserInterruptRun (),

HelixCounterSetRate (), HelixUserInterruptCancel () and HelixCounterReset ().

Step-By-Step Instructions

Create a HelixUSERINT structure variable to hold the interrupt settings and initialize the interrupt structure variables, then call HelixUserInterruptConfig () to configure the user interrupt, HelixUserInterruptRun () to start user interrupts, call HelixCounterSetRate () to set the counter, and finally call HelixUserInterruptCancel () to stop the user interrupt.

Example of Usage for User Interrupt Operations

```
int count =0;
HelixUSERINT inter;
HelixCOUNTER Ctr;

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

if (HelixUserInterruptConfig (bi,&inter) !=DE_NONE)
{
      dscGetLastError (&errorParams );
      printf ("HelixUserInterruptConfig error: %s %s\n",
      dscGetErrorString (errorParams.ErrCode),errorParams.errstring );
      return 0;
}
if (HelixUserInterruptRun(bi,inter.Source,Bit,Edge) !=DE_NONE)
{
      dscGetLastError (&errorParams );
      printf ("HelixUserInterruptRun 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 (HelixCounterSetRate (bi,&counter) !=DE_NONE)
{
      dscGetLastError (&errorParams );
      printf ( " HelixCounterSetRate 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);
}
inter.Enable= 0;
if (HelixUserInterruptCancel(bi,&inter) !=DE_NONE)
{
      dscGetLastError (&errorParams );
      printf (" HelixUserInterruptCancel error: %s %s\n",
      dscGetErrorString (errorParams.ErrCode ), errorParams.errstring );
      return 0;
}

if (HelixCounterReset(bi,(1 << inter.Source)) !=DE_NONE)
{
      dscGetLastError (&errorParams );
      printf ("HelixCounterReset error: %s %s\n",
      dscGetErrorString (errorParams.ErrCode ), errorParams.errstring );
      return 0;
}

}
```

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: _HelixWaveformReset (), HelixWaveformConfig (),HelixDASetSettings(),HelixWaveformBufferLoad (),HelixWaveformStart (), HELIXWaveformPause(), and HELIXWaveformReset ().

Step-By-Step Instructions

Create a HelixWAVEFORMstructure variable to hold waveform settings and initialize the waveform structure variables, then call HELIXWaveformReset () to reset the D/A waveform buffer. Call HelixWaveformConfig () to configure the operating parameters of the waveform generator, call HelixWaveformBufferLoad () to load D/A conversion value into the D/A waveform buffer, call HelixDASetSettings () to set the D/A range, call HelixWaveformStart () to start the waveform generator, and finally call HelixWaveformPause () and HelixWaveformReset () to stop the waveform generator.

Example of Usage for D/A Waveform Generator:

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

```
if (sawtooth)
{
      intBuff= 0xFFFF/waveform.Frames;
      waveform.Waveform [0]=intial_value;
      waveform.Waveform [waveform.Frames-1] =intial_value;
      for (index=1; index <waveform.Frames-1; index++)
      {
            intial_value+=intBuff;
            waveform.Waveform [index] =intial_value;
      }
}
else
{
      for (index=0;index <waveform.Frames; index++)
      {
            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;
}
if (HelixDASetSettings (bi, range,0) != DE_NONE )
{
      dscGetLastError (&errorParams);
      printf ("HelixDASetSettings 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 (HelixWaveformBufferLoad (bi,&waveform) != DE_NONE )
{
      dscGetLastError (&errorParams);
      printf ("HelixWaveformBufferLoad error: %s %s\n",
      dscGetErrorString (errorParams.ErrCode), errorParams.errstring);
      return 0;
}

//start D/A conversions
Printf ("Starting DA wave form generator \n");
if (HelixWaveformStart (bi) != DE_NONE )
{
      dscGetLastError (&errorParams);
      printf ("HelixWaveformStart 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);
HelixWaveformPause (bi);
HelixWaveformReset (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: HelixADConvert ().

Step-By-Step Instructions

Create a HelixADSETTINGS structure variable to hold the A/D settings and initialize the structure variables. Call HelixADConvert () 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:

```
HelixADSETTINGS dscadsettings;
unsigned int sample;       // sample reading
int channel;
channel=0;
dscadsettings.Polarity = 0;
dscadsettings.Range = 0;
dscadsettings.Sedi = 0;
dscadsettings.Highch = channel;
dscadsettings.Lowch = channel;
dscadsettings.ADClock = 0;
if (HelixADConvert ( bi,& dscadsettings ,&sample)  != DE_NONE )
{
      dscGetLastError (&errorParams);
      printf ( " HelixADConvert 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 _HelixADSetSettings (), and HelixADSetTiming(), _HelixADTrigger().

Step-By-Step Instructions

Create a HelixADSETTINGS structure variable to hold A/D settings and initialize the structure variables. Call HelixADSetSettings () to configure the A/D settings as requested by the user. Finally call _HelixADTrigger () to execute the A/D conversion scan using the current board settings.

Example of Usage for A/D Scan:

```
HelixADSETTINGS dscadsettings;
unsigned int sample [16];       // sample reading
dscadsettings.Lowch=0;
dscadsettings.Highch=15;
dscadsettings.Polarity = 0;
dscadsettings.Range = 0;
dscadsettings.Sedi = 0;
dscadsettings.ScanInterval = 0;
dscadsettings.ADClock = 0;
dscadsettings.ScanEnable = 1;
if ( (HelixADSetSettings ( bi, &dscadsettings ) ) != DE_NONE )
{
      dscGetLastError (&errorParams);
      printf ("HelixADSetSettings error: %s %s\n",
      dscGetErrorString (errorParams.ErrCode), errorParams.errstring );
      return 0;
}
if ( (HelixADSetTiming (bi,&dscadsettings) ) != DE_NONE )
{
      dscGetLastError (&errorParams);
      printf ( " HelixADSetTiming error: %s %s\n",
      dscGetErrorString (errorParams.ErrCode), errorParams.errstring );
      return 0;
}
while (!kbhit() )
{
      if ( (HelixADTrigger ( bi, sample ) ) != DE_NONE )
      {
            dscGetLastError (&errorParams);
            printf ( " HelixADTrigger 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: HelixADSetSettings (), HelixADSetTiming (), HelixADInt(), HelixADSetTiming(), HelixADIntResume(), HelixADIntPause(), HelixADIntStatus(),and HelixADIntCancel ().

Step-By-Step Instructions:

Create a HelixADSETTINGS structure variable to hold A/D settings, create a HelixADINTstructure variable to hold A/D conversion interrupt settings, and create a HelixADINTSTATUS structure variable to hold A/D conversion interrupt status. Initialize the A/D settings structure variables, then use HelixADSetSettings () to configure the A/D settings as requested by the user. Call HelixADSetTiming () to configure A/D clock source, and scan settings. Initialize the A/D interrupt status structure variable, then call HelixADInt () to enable A/D interrupt operation using the current analog input settings. Call HelixADIntStatus () to get the interrupt routine status including, running / not running, number of conversions completed, cycle mode, FIFO status, and FIFO flags. Call HelixADIntResume () to resume the interrupt. Call HelixADIntPause () to pause the interrupt, and finally call HelixADIntCancel () to stop the A/D interrupt.

Example of Usage for A/D interrupt:

```
HelixADSETTINGS dscadsettings; // structure containing A/D conversion settings
HelixADINT   dscIntSettings;  //  structure  containing  A/D  conversion  interrupt
settings
HelixADINTSTATUS intstatus; // structure containing A/D conversion interrupt status
HelixCOUNTER Ctr;
int key = 0;
int Pause = 0;
/* Initializing A/D settings */
dscadsettings.Lowch = 0;
dscadsettings.Highch = 15;
dscadsettings.Polarity = 0; //bipolarity
dscadsettings.Range= 1; //5v
dscadsettings.Sedi = 0;
dscadsettings.ScanEnable = 0;
dscadsettings.ScanInterval = 0;
dscadsettings.ADClock = 2;
dscadsettings.ProgInt = 0;
if ( (HelixADSetSettings ( bi, &dscadsettings ) ) != DE_NONE )
{
      dscGetLastError(&errorParams);
      printf ("HelixADSetSettings error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode),errorParams.errstring);
      return 0;
}
if ( (HelixADSetTiming (bi,&dscadsettings) ) != DE_NONE )
{
      dscGetLastError (&errorParams);
      printf ( " HelixADSetTiming error: %s %s\n",
      dscGetErrorString (errorParams.ErrCode), errorParams.errstring );
      return 0;
}

/* initializing Interrupt settings */

dscIntSettings.NumConversions = 1600;
dscIntSettings.Cycle = 1;
```

```
dscIntSettings.FIFOEnable = 1;
dscIntSettings.FIFOThreshold = 1600;
dscIntSettings.ADBuffer        =        (SWORD*)        malloc        (sizeof(SWORD)*
dscIntSettings.NumConversions );
if (HelixADInt (bi,&dscIntSettings ) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ( " HelixADInt: %s %s\n", dscGetErrorString(errorParams.ErrCode),
      errorParams.errstring);
      return 0;
}
Ctr.Rate = rate;
Ctr.Ctrs = (1 << (dscadsettings.ADClock-2) );
Ctr.CtrOutEn      = 0;
Ctr.CtrOutPol     = 0;
Ctr.ctrOutWidth = 0 ;
Ctr.Start = 1;
if (HelixCounterSetRate (bi,&Ctr) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ( " HelixCounterSetRate: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring);
      return 0;
}
if ( (HelixADSetTiming (bi,&dscadsettings) ) != DE_NONE )
{
      dscGetLastError (&errorParams);
      printf ( " HelixADSetTiming error: %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 (HelixADIntStatus(bi,&intstatus) !=DE_NONE)
            {
                  dscGetLastError (&errorParams);
                  printf("HelixADIntStatuserror: %s%s\n",
                  dscGetErrorString (errorParams.ErrCode), errorParams.errstring);
                  return 0;
            }


            printf("No ofA/D conversions completed %d\n",intstatus.NumConversions);

            dscSleep (1000);
            if((dscIntSettings.Cycle==0)&&(intstatus.NumConversions
>=dscIntSettings.NumConversions) ) break;
      }
```

```
      else
      {
            key = getch ();
            if (key ==32 )
            {

                  if (Pause )
                  {

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

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

                        Pause = 1;
                  }
            }
            else
            {
                  break;
            }
      }


}
if (HelixADIntCancel (bi)!=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf("HelixADIntCancelerror:
      %s%s\n",dscGetErrorString(errorParams.ErrCode),
      errorParams.errstring);
      return 0;
}
if (HelixCounterReset (bi,(1 << (dscadsettings.ADClock-2) )) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf("HelixCounterReset 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 HelixLED () 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)
            {
                  if (HelixLED(bi,0) != DE_NONE)
                  {
                        dscGetLastError (&errorParams );
                        Printf ( "HelixLED error: %s %s\n",
                        dscGetErrorString (errorParams.ErrCode ),
                        errorParams.errstring);
                        return 0;
                  }
                  LEDOn = 0;
            }
            else
            {
                  if(HelixLED(bi,0) != DE_NONE)
                  {
                        dscGetLastError (&errorParams );
                        printf ("HelixLED error: %s %s\n",
                        dscGetErrorString (errorParams.ErrCode ),
                        errorParams.errstring);
                        return 0;
                  }

                  LEDOn = 1;
            }
      }
      else if(key == 'q')
      {
            break;
      }
}
```

8. INTERFACE CONNECTOR DETAILS

8.1 HELIX Digital GPIO Connector (J17)

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, C, and D). Port D pins 0-2 are present in J18

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
+5V/3.3V fused	25	26	Dground

Description: Standard 2mm dual row straight pin header.

8.2 HELIX Analog GPIO Connector (J18)

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

DAQ Analog GPIO Connector Pinout

Vout 0	1	2	Vout 1
Vout 2	3	4	Vout 3
Aground (Vout)	5	6	Aground (Vin)
Vin 0 / 0+	7	8	Vin 8 / 0-
Vin 1 / 1+	9	10	Vin 9 / 1-
Vin 2 / 2+	11	12	Vin 10 / 2-
Vin 3 / 3+	13	14	Vin 11 / 3-
Vin 4 / 4+	15	16	Vin 12 / 4-
Vin 5 / 5+	17	18	Vin 13 / 5-
Vin 6 / 6+	19	20	Vin 14 / 6-
Vin 7 / 7+	21	22	Vin 15 / 7-
DIO D0	23	24	DIO D1
DIO D2	25	26	Dground

Description Standard 2mm dual row straight pin header.

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.

Command	Function
0	Clear one or more counters. If a counter is running at the time it is cleared, it will continue
1	Load counter with user-defined 32-bit value
2	Select count direction, up or down
4	Enable / disable counting
5	Latch counter; a counter must be latched before its count value can be read back. The latch is a snapshot of the counter value at one moment in time. The counter continues to run after latching.
6	Select clock source for one or more counters. Options include internal 50MHz clock, internal 10MHz clock, or the counter's designated I/O pin on port C.
7	Enable / disable auto-reload
8	Enable counter output pulse on designated I/O pin on port C
9	Configure counter output pulse width; options include 1, 10, 100, or 1000 clock periods
10	Read counter value. A counter must be latched with command 5 before its contents can be read back.
15	Reset one more counters. When a counter is reset its configuration and contents are lost.

PWM Commands

The PWMs are configured and managed with a series of commands. These commands are implemented in the Diamond Systems Universal Driver software. Configuration commands operate on a single PWM at a time, however start, stop, and reset commands may operate on a single PWM or all PWMs at the same time.

Command	Function
0	Stop one or all PWMs. When a PWM is stopped, its output returns to its inactive state, and the counters 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. Stopping a PWM is not the same as resetting it. See command 4 below.
1	Load the period or duty cycle counter with user-defined value
2	Select output pulse polarity
3	Enable / disable PWM output. This must be done in conjunction with command 5 below.
4	Reset one or 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 output signal on designated Port C DIO pin.
6	Select clock source for period and duty cycle counters, either 50MHz or 1MHz. Both counters will use the same clock source.
7	Start one

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 J17 Pin	DIO C Bit	PWM
21	4	0
22	5	1
23	6	2
24	7	3