Universal Driver Software User Manual

Aries SBC

PC/104-Plus SBC with Bay Trail Processor and Data Acquisition

For Version 7.0.0 and later

Revision A.0 May 2015

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

FOR TECHNICAL SUPPORT PLEASE CONTACT:

support@diamondsystems.com

CONTENTS

1. INTRODUCTION

This user manual contains all essential information about the Universal Driver 7.0 Aries 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

Aries is an embedded single board computer (SBC) with a PC/104-*Plus* form factor. Aries integrates on-board memory, PC/104-*Plus* (ISA + PCI) expansion, one PCIe MiniCard socket, dual Gigabit Ethernet, and optional data acquisition circuit with analog and digital I/O.

The Aries SBC is based on Intel "Bay Trail" E3800 series processors. The form factor is similar to PC/104 with left and right side extensions that extend the full length of the two sides without providing the corners traditionally seen in PC/104 boards with "wings".

2.2 Specifications

- 1.91GHz Intel quad core E3845 or 1.46GHz dual core E3826 Bay Trail CPU
- 2GB or 4GB 64-bit DDR3 SDRAM soldered on board
- I/O Support:
  - 3 USB 2.0 ports, 1 USB 3.0 port
  - 4 RS-232/422/485 ports with programmable protocol and line termination
  - 2 10/100/1000Mbps Ethernet ports
  - 1 SATA port for disk-on-module or external drive
  - 24-bit LVDS LCD display and VGA CRT
  - DP and HDMI sharing on single channel
  - HD audio
  - Shared expansion socket auto-selects for either PCIe MiniCard or mSATA flash disk modules
- Data Acquisition:
  - 16 16-bit analog inputs, 250KHz max sample rate
  - 4 16-bit analog outputs with waveform generator
  - 22 digital I/O lines with programmable direction
  - 8 counter/timers, 4 PWMs
- PC/104-*Plus* and PCIe MiniCard expansion capability
- Windows 7, 8, Linux operating systems
- Driver packages available for each OS
- PC/104 form factor 4.5" x 4.0" (not including I/O connector overhang).
- -40°C to +85°C ambient operating temperature
- Power input: +5VDC +/- 5%

3. GENERAL PROGRAMMING GUIDELINES

3.1 Initialization and exit function calls

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

- dscInit ( ) : This function initializes the Universal Driver
- ARIESInitBoard (): This function initializes the Aries 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 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 "Aries.h"
ERRPARAMS errorParams; //structure for returning error code and error string
DSCCB dsccb; // structure containing board settings
BoardInfo *bi=NULL; //Structure containing board base address
ARIESINIT Init; // Structure containing board FPGA ID, Revision ID etc.

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

      if ( ARIESInitBoard ( &dsccb,&Init) != DE_NONE )
      {
            dscGetLastError (&errorParams);
            printf ("ARIESInitBoard 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_ARIES, 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, 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. User 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 ( "dscInit failed: %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 ARIESADSetSettings

Function Definition

BYTE ARIESADSetSettings (BoardInfo* bi, ARIESADSETTINGS* settings);

Function Description

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

Function Parameters

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

Return Value

Error code or 0.

Usage Example

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

```
ARIESADSETTINGS Ariesadsettings;
Ariesadsettings.Polarity = 1;
Ariesadsettings.Gain= 1;
Ariesadsettings.Sedi = 0;
Ariesadsettings.Highch = 0;
Ariesadsettings.Lowch = 0;
Ariesadsettings.ADClock = 0;
Ariesadsettings.ScanEnable = 0;
ARIESADSetSettings (bi, Ariesadsettings);
```

4.2 ARIESADSetRange

Function Definition

BYTE ARIESADSetRange (BoardInfo* bi, ARIESADSETTINGS* settings);

Function Description

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

Function Parameters

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

Return Value

Error code or 0.

Usage Example

To configure channel 0 in 0-5v with single ended and to leave other A/D settings untouched,

```
ARIESADSETTINGS Ariesadsettings;
Ariesadsettings.Gain = 1;
Ariesadsettings.Polarity = 1;
Ariesadsettings.Sedi = 0;
ARIESADSetSettings (bi, Ariesadsettings);
```

4.3 ARIESADSetChannel

Function Definition
BYTE ARIESADSetChannel (BoardInfo* bi, ARIESADSETTINGS* settings);

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

Function Parameters

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

Return Value
Error code or 0.

Usage Example
To configure low and high channel as 5 and 6 and with all other settings remain same,

```
ARIESADSETTINGS Ariesadsettings;
Ariesadsettings.Highch = 3;
Ariesadsettings.Lowch = 0;
ARIESADSetSettings (bi, Ariesadsettings);
```

4.4 ARIESADSetScan

Function Definition
BYTE ARIESADSetScan (BoardInfo* bi, ARIESADSETTINGS* settings);

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

Function Parameters

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

Return Value
Error code or 0.

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

```
ARIESADSETTINGS Ariesadsettings;
Ariesadsettings.ScanEnable = 1;
Ariesadsettings.ScanInterval= 1;
Ariesadsettings.ProgInt= 0;
ARIESADSetSettings (bi, Ariesadsettings);
```

4.5 ARIESADSetClock

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

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

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
ADclk	int 0-3 ; 0 = software command (ADSample or ADScan function), 1 = falling edge on DIO0, 2 = counter 0 output, 3 = counter 1 output.

Return Value
Error code or 0.

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

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

4.6 ARIESADEnableClock

Function Definition

BYTE ARIESADEnableClock (BoardInfo* bi);

Function Description

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

Return Value

Error code or 0.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Usage Example

To enable the clock which is configured using ARIESADSetClock () function,

```
ARIESADEnableClock (bi);
```

4.7 ARIESADStopClock

Function Definition

BYTE ARIESADStopClock (BoardInfo* bi);

Function Description

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

Return Value

Error code or 0.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Usage Example

To stop the clock which is configured using ARIESADSetClock () function,

```
ARIESADStopClock (bi);
```

4.8 ARIESADSample

Function Definition

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

Function Description

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

Function Parameters

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

Return Value

Error code or 0.

Usage Example

To do A/D conversion with preconfigured A/D settings using ARIESADSetSettings () function,

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

4.9 ARIESADScan

Function Definition

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

Function Description

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

Function Parameters

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

Return Value

Error code or 0.

Usage Example

To do A/D scan operation with preconfigured A/D settings using ARIESADSetSettings () function,

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

4.10 ARIESADRead

Function Definition

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

Function Description

This function reads one A/D sample out of the FIFO if the FIFO is not empty. If the FIFO is empty, it returns an error FIFO_EMPTY. The purpose of this function is to enable reading A/D data acquired with counter/timer or external triggering without having to use interrupts

Function Parameters

Name	**Description**
BoardInfo	The handle of the board to operate on
Sample	To hold A/D sample value

Return Value

Error code or 0.

Usage Example

To read A/D sample with A/D external clock source and without using the interrupts concept with preconfigured

A/D settings using ARIESADSetsettings ()

```
unsigned int sample = 0 ;
ARIESADSetClock (bi, 1);
While (1)
{
      if (ARIESADRead(bi,&sample) !=ARIES_FIFO_EMPTY)
      {
            printf (“A/D sample = 0x%x \n”, sample);
      }
}
```

4.11 ARIESADInt

Function Definition

BYTE ARIESADInt (BoardInfo* bi, ARIESADINT* Ariesadint);

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

Return Value

Error code or 0.

Usage Example

To do A/D sampling in interrupt mode,

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

4.12 ARIESADIntStatus

Function Definition
BYTE ARIESADIntStatus (BoardInfo* bi, ARIESADINTSTATUS* 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
ARIESADIntStatus	OpStatus int 0 = not running, 1 = running NumConversions int Number of conversions since interrupts started Cycle int 0 = one-shot operation, 1 = continuous operation FIFODepth int Current FIFO depth pointer UF, OF, FF, TF, EF int FIFO flags

Return Value
Error code or 0.

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

4.13 ARIESADIntPause

Function Definition
BYTE ARIESADIntPause (BoardInfo* bi);

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

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
To pause A/D interrupts,

```
ARIESADIntPause (bi);
```

4.14 ARIESADIntResume

Function Definition
BYTE ARIESADIntResume (BoardInfo* bi);

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

Function Parameters

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

Return Value
Error code or 0.

Usage Example
To resume A/D interrupts,

```
ARIESADIntResume (bi);
```

4.15 ARIESADIntCancel

Function Definition
BYTE ARIESADIntCancel (BoardInfo* bi);

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

Function Parameters

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

Return Value
Error code or 0.

Usage Example
To cancel A/D interrupts,

```
ARIESADIntCancel (bi);
```

4.16 ARIESDASetSettings

Function Definition

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

Function Description

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

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

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

Function Parameters

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

Return Value

Error code or 0.

Usage Example

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

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

4.17 ARIESDAConvert

Function Definition

BYTE ARIESDAConvert (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 ARIESDASetSettings (). If the board is not set for simultaneous update mode (DASIM = 0), the channel will be updated immediately after the DAC internal calibration circuit is finished. If the board is set for simultaneous update mode (DASIM = 1), the channel will not be updated, and a separate update command must be executed.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Channel	Channel number 0-4
DACode	Code to convert (0 – 65535)

Return Value

Error code or 0.

Usage Example

To set channel zero with 5V,

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

4.18 ARIESDAConvertScan

Function Definition

BYTE ARIESDAConvertScan (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 ARIESDASetSettings (). If the board is not set for simultaneous update mode (DASIM = 0), each channel will be updated immediately by the hardware after being written to. If the board is set for simultaneous update mode (DASIM = 1), the channel will not be updated, and a separate update command must be executed. This function simply calls ARIESDAConvert() once for each channel to be updated. It is mostly used for simultaneous mode operation where all channels will be updated at the same time after all data is written.

Function Parameters

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

Return Value

Error code or 0.

Usage Example

To update channel 0 and 4 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) *16);
DACodes = (int *) malloc (sizeof (int) *16);
ChannelSelect [0] = 1;
DACodes [0] = 65535;
ChannelSelect [4] = 1;
DACodes [4] = 32768;
ARIESDAConvertScan (bi, ChannelSelect, DACodes);
```

4.19 ARIESDASetSim

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

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

Function Parameters

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

Return Value
Error code or 0.

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

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

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

```
int Simup=1;
ARIESDASetSim (bi, Simup);
ARIESDAConvert (bi, 0, 65535);
ARIESDAConvert (bi, 1, 65535);
ARIESDAConvert (bi, 2, 65535);
ARIESDAConvert (bi, 3, 65535);
ARIESDAUpdate (bi);
```

4.20 ARIESDAUpdate

Function Definition

BYTE ARIESDAUpdate (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 ARIESDAConvertScan () function. In this case the programmer is using ARIESDAConvert () multiple times to write channel data individually. At the end of all these functions, the update command is needed to update all channels at once.

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

Return Value

Error code or 0.

Usage Example

```
ARIESDAUpdate (bi);
```

4.21 ARIESWaveformBufferLoad

Function Definition

BYTE ARIESWaveformBufferLoad(BoardInfo* bi, ARIESWAVEFORM* ARIESwaveform);

Function Description

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

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
ARIESwaveform	Structure with following member variables *Waveform - pointer to array of 16-bit unsigned data; If multiple channels are being set up at the same time, the data must be previously interleaved by the program, i.e. ch. 0, ch. 1, ch. 2, ch. 0, ch. 1, ch 2; the array size must be equal to Frames x FrameSize; the array size must be no greater than 2048. Frames - Total number of frames in the array FrameSize - no. of channels to be driven 1-4 Channels - List of channels to be driven by the waveform generator

Return Value

Error code 0.

Usage Example

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

```
ARIESWAVEFORM ARIESwaveform;
ARIESwaveform.Waveform = (int *) malloc (sizeof (int) *4);
ARIESwaveform.Waveform [0] = 0;
ARIESwaveform.Waveform [1] = 0;
ARIESwaveform.Waveform [2] = 65535;
ARIESwaveform.Waveform [3] = 65535;
ARIESwaveform.Frames         = 4;
ARIESwaveform.FrameSize     = 1;
ARIESwaveform.Channels [0] = 0;
Channel = 0;
Range = 0;
Overrange = 0;
ClearEnable = 0;
ARIESDASetSettings (bi, Channel, Range, overrange, ClearEnable);
ARIESWaveformBufferLoad (bi, &ARIESwaveform);
```

4.22 ARIESWaveformDataLoad

Function Definition
BYTE DSCUDAPICALL ARIESWaveformDataLoad (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;
ARIESWaveformDataLoad (bi, 258, 0, 65535);
```

4.23 ARIESWaveformConfig

Function Definition
BYTE ARIESWaveformConfig (BoardInfo* bi, ARIESWAVEFORM* ARIESwaveform);

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 ARIESWaveformBufferLoad () / ARIESWaveformDataLoad () before starting the waveform generator with ARIESWaveformStart ().

Function Parameters

Name	**Description**
BoardInfo	The handle of the board to operate on
ARIESwaveform	Structure with following member variables FrameSize - no. of channels to be driven 1-4 Clock - 0 = software increment; 1 = counter/timer 0 output; 2 = counter/timer 1 output; 3 = DIO pin D0 Rate - frame update rate, Hz (only used if Clock = 1 or 2) Cycle - 0 = one-shot operation; 1 = repetitive operation

Return Value
Error code or 0.

Usage Example
To configure waveform generator with 100 Hz frequency,

```
ARIESWAVEFORM ARIESwaveform;
ARIESwaveform.FrameSize = 1;
ARIESwaveform.Clock = 1;
ARIESwaveform.Rate = 100;
ARIESwaveform.Cycle = 1;
ARIESWaveformConfig (bi, & ARIESwaveform);
```

4.24 ARIESWaveformStart

Function Definition
BYTE ARIESWaveformStart (BoardInfo* bi);

Function Description
This function starts or restarts the waveform generator running based on its current configuration. Prerequisites: ARIESWaveformConfig (), ARIESWaveformBufferLoad () / ARIESWaveformDataLoad () have to have been run already in order to start predictable waveform.

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,

```
ARIESWaveformStart (bi);
```

4.25 ARIESWaveformPause

Function Definition
BYTE ARIESWaveformPause (BoardInfo* bi);

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

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,

```
ARIESWaveformPause (bi);
```

4.26 ARIESWaveformReset

Function Definition
BYTE ARIESWaveformReset (BoardInfo* bi);

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

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
To reset the waveform generator,

```
ARIESWaveformReset (bi);
```

4.27 ARIESWaveformInc

Function Definition
BYTE ARIESWaveformInc (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 to the each DAC output code in the buffer frame.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
To increment wave generator by one frame,

```
ARIESWaveformInc (bi);
```

4.28 ARIESDIOConfig

Function Definition
BYTE ARIESDIOConfig (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
ARIESDIOConfig	Input parameters: Port int 0-3 for port A, B, C, or D Config int 8-bit configuration values for selected port

Return Value
Error code or 0.

Usage Example
To set Port A as input mode,

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

4.29 ARIESDIOConfigAll

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

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

Function Parameters

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

Return Value
Error code or 0.

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

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

4.30 ARIESDIOOutputByte

Function Definition

BYTE ARIESDIOOutputByte (BoardInfo* bi, int Port, byte Data);

Function Description

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

Function Parameters

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

Return Value

Error code or 0.

Usage Example

To set Port 0 output as 0x77,

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

4.31 ARIESDIOInputByte

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

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

Function Parameters

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

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

4.32 ARIESDIOOutputBit

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

Function Description
This function configures a PWM for operation.

Function Parameters

Name	Description		
BoardInfo	The handle of the board to operate on		
ARIESDIOOutputBit	Input parameters:		
	Port	int	Port to write bit to: 0 = A, 1 = B, 2 = C, 3 = D
	Bit	int	0-7 indicates the bit position in the port (for port D the value should be 0-5)
	Value	int	0 or 1

Return Value
Error code or 0.

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

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

4.33 ARIESDIOInputBit

Function Definition
BYTE ARIESDIOInputBit (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 to data.

Function Parameters

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

Return Value
Error code or 0.

Usage Example
To read the value at port 0 bit 5

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

4.34 ARIESCounterSetRate

Function Definition

BYTE ARIESCounterSetRate (BoardInfo* bi, ARIESCOUNTER *Ctr);

Function Description

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

Function Parameters

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

Return Value

Error code or 0.

Usage Example

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

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

4.35 ARIESCounterConfig

Function Definition

BYTE ARIESCounterConfig (BoardInfo* bi, ARIESCTR *Ctr);

Function Description

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

Function Parameters

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

Return Value

Error code or 0.

Usage Example

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

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

4.36 ARIESCounterRead

Function Definition
BYTE ARIESCounterRead (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
ARIESCounterRead	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 current value of counter 0 when it is running and display on the screen,

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

4.37 ARIESCounterReset

Function Definition
BYTE ARIESCounterReset (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
Ctrno	int Counter number, 0-7

Return Value
Error code 0.

Usage Example
To reset counter 0.

CtrNum = 0;
ARIESCounterReset (bi, CtrNum);

4.38 ARIESCounterFunction

Function Definition
BYTE ARIESCounterFunction (BoardInfo* bi, ARIESCTR* Ctr);

Function Description
This function can be used to program any desired function into a counter, except reading which is done by ARIESCounterRead (). 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		
ARIESCounterFunction	Ctrno	int	Counter number, 0-7
	CtrData	uns. long	Initial load data, 32-bit straight binary
	CtrCmd	int	Counter command, 0-15 (see FPGA specification for available commands)
	CtrCmdData	int	Auxiliary data for counter command, 0-3 (see FPGA specification for usage)

Return Value
Error code 0.

Usage Example
To reset counter 0 only.

```
ARIESCOUNTER counter;
counter.Ctrno = 0;
counter.CtrCmd = 0;
counter.CtrCmdData = 0xF;
ARIESCounterFunction (bi, &counter);
```

4.39 ARIESPWMConfig

Function Definition

BYTE ARIESPWMConfig (BoardInfo* bi, ARIESPWM* Ariespwm);

Function Description

This function configures a PWM for operation. It can optionally start the PWM running.

Function Parameters

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

Return Value

Error code or 0.

Usage Example

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

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

4.40 ARIESPWMStart

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

Return Value
Error code or 0.

Usage Example
To start PWM 0,

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

4.41 ARIESPWMStop

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

Function Description
This function stops a PWM.

Function Parameters

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

Return Value
Error code or 0.

Usage Example
To stop PWM 0,

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

4.42 ARIESPWMReset

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

Function Description
This function resets a PWM. After a PWM is reset, it stops, and its settings are no longer valid. The config function must be used to configure it for further use. Resetting a PWM releases its DIO pin back to normal operation.

Function Parameters

Name	Description
BoardInfo	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 1,

```
Num =1;
BYTE ARIESPWMReset (bi, Num);
```

4.43 ARIESPWMCommand

Function Definition
BYTE ARIESPWMCommand (BoardInfo* bi, ARIESPWM* Ariespwm);

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

Function Parameters

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

Return Value
Error code or 0.

Usage Example
To reset all PWMs,

```
ARIESPWM pwm;
pwm.Command = 0x4; //Stop PWM
pwm.CmdData = 0x01; //Stop all PWM
ARIESPWMCommand (bi, &pwm);
```

4.44 ARIESUserInterruptConfig

Function Definition

BYTE ARIESUserInterruptConfig(BoardInfo* bi, ARIESUSERINT* ARIESuserint);

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
ARIESUSERINT	Structure with following member variables, IntFunc - pointer to user function to run when interrupts occur Mode - 0 = alone, 1 = before standard function, 2 = after standard function Source - Selects interrupt source: 0 = A/D, 1 = unused, 2 = counter 2 output, 4 = digital I/O Enable - 0 = disable interrupts, 1 = enable interrupts BitSelect - 0-20 selects which DIO line to use to trigger interrupts; only used if Source = 4 Edge - 1 = rising edge, 0 = falling edge; (only used if Source = 4)

Return Value

Error code or 0.

Usage Example

To install a function to be called whenever interrupt occurs,

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

4.45 ARIESUserInterruptRun

Function Definition

BYTE ARIESUserInterruptRun (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	Identifies interrupt source: 2 = counter 2 output, 3 = counter 3 output, 4 = digital input
Bit	0-20 selects which DIO bit will drive interrupts
Edge	0 = falling edge, 1 = rising edge

Return Value

Error code 0.

Usage Example

To start the user interrupt,

```
Source = 4;
Bit = 0;
Edge = 0;
ARIESUserInterruptRun (bi, Source, Bit, Edge);
```

4.46 ARIESUserInterruptCancel

Function Definition

BYTE ARIESUserInterruptCancel (BoardInfo* bi, int Source);

Function description

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

Function Parameters

Name	**Description**
BoardInfo	The handle of the board to operate on
Source	Identifies interrupt source: 2 = counter 2 output, 3 = counter 3 output, 4 = digital input

Return Value

Error code or 0.

Usage Example

To cancel the user interrupts,

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

4.47 ARIESWatchDogConfig

Function Definition

BYTE ARIESWatchDogConfig(bi, ARIESWATCHDOG *settings)

Function description

This function configures watchdog timer for desired operation and it does not enables the watch timer, the function ARIESWatchDogEnable () must be called to enable watch dog timer.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
ARIESWATCHDOG	Structure with following member variables, CtrAData - 1-65535 =16-bit timer value (runs at 100Hz - max 655.35 seconds) CtrBData - 1-255 = 8-bit timer value (runs at 100Hz - max 2.55 seconds) HardwareTriggerEnable - 0 = disable, 1 = enable EdgeSelection - if HardwareTriggerEnable = 1, then 0 = Raising edge= falling edge CountEarly - 0 = disable, 1 = enable i.e. DIO C4 goes high when counter A= 1 instead of 0 , This enables C4 tobeexternally wired to C5 to cause anautomaticrepetitive retrigger when WDIEN = 1 andWDEDGE = 0 InterruptEnable - 1= when counter A reaches 0 an interrupt willoccur,0=interrupt will not occur

Return Value

Error code or 0.

Usage Example

To configure Watch dog timer with raising edge Hardware trigger mode and to load 6 seconds and 2 seconds for counter A and Counter B respectively

```
ARIESWATCHDOG watch;
watch.InterruptEnable=1;
watch.CtrAData = 6 * 100;
watch.CtrBData = 2 * 100;
watch.HardwareTriggerEnable= 1;
watch.EdgeSelection=0;
if (ARIESWatchDogConfig(bi,&watch) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ("ARIESWatchDogConfig error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}
```

4.48 ARIESWatchDogEnable

Function Definition

BYTE ARIESWatchDogEnable(bi)

Function description

This function enables watchdog timer with preconfigured settings using ARIESWatchDogConfig() function.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value

Error code or 0.

Usage Example

To enable watch dog timer with preconfigured settings using ARIESWatchDogConfig()

```
ARIESWatchDogEnable (bi);
```

4.49 ARIESWatchDogDisable

Function Definition

BYTE ARIESWatchDogDisable (bi)

Function description

This function disables watchdog timer, also the configuration done to WDT is lost

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value

Error code or 0.

Usage Example

To disable watch dog timer

```
ARIESWatchDogDisable (bi);
```

4.50 ARIESWatchDogSoftwareReTrigger

Function Definition
BYTE ARIESWatchDogSoftwareReTrigger(bi)

Function description
This function reloads the Counter A and counter B with initial value i.e. to the original value which was loaded using ARIESWatchDogConfig() function and if this function is called prior to Counter A reaches 0 , it avoids the hardware reset.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
To do software retrigger

```
ARIESWatchDogSoftwareReTrigger (bi);
```

4.51 ARIESTemperatureSensorRead

Function Definition
BYTE ARIESTemperatureSensorRead(BoardInfo* bi, BYTE cmd, int *Data)

Function description
This function reads onboard temperature sensor value when the cmd value is provided as 0x01.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Cmd	0x01 to read temperature data 0x05 to read control register 0x07 to put device in normal mode 0x0F to put device in standby mode
*Data	To store on board read temperature value

Return Value
Error code or 0.

Usage Example
To read temperature
```
int data   = 0 ;

ARIESTemperatureSensorRead(bi,0x01,&data);
printf("Current temperature = %d \n", data);
```

4.52 ARIESInitBoard

Function Definition
BYTE ARIESInitBoard (DSCCB* dsccb);

Function Description
This function initializes the board.

Function Parameters

Name	Description
dsccb	The handle of the board to operate on

Return Value
Error code or 0.

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

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

4.53 ARIESFreeBoard

Function Definition
BYTE ARIESFreeBoard (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
```
ARIESFreeBoard (DSCB board);
```

4.54 ARIESFIFOStatus

Function Definition

BYTE ARIESFIFOStatus (BoardInfo* bi, ARIESFIFO* Ariesfifo);

Function Description

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

Function Parameters

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

Return Value

Error code or 0.

Usage Example

To read current FIFO status,

```
ARIESFIFO Ariesfifo;
ARIESFIFOStatus (bi, &Ariesfifo);
printf (“Depth = %d \n”, Ariesfifo. Depth);
printf (“Overflow flag = %d \n”, Ariesfifo.OF);
```

4.55 ARIESEEPROMRead

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

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

Function Parameters

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

Return Value
Error code or 0.

Usage Example
To read value at EEPROM address 32,

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

4.56 ARIESEEPROMWrite

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

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

Function Parameters

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

Return Value
Error code or 0.

Usage Example
To write 0x55 at EEPROM address 32,

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

4.57 ARIES Monitor

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

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

Function Parameters

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

Return Value
Error code or 0.

Usage Example
To read input voltage,

```
ARIESMonitor (bi, & Status);
```

4.58 ARIESLED

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

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

Function Parameters

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

Return Value
Error code or 0.

Usage Example
To turnoff blue LED on the board,

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

5. UNIVERSAL DRIVER DEMO APPLICATION DESCRIPTION

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

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

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. This output code can be translated to an output voltage using one of the formulas described in the hardware specification.

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 a multiple, specified output channels with each function call.

5.3 DA Waveform

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

5.4 DIO

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

5.5 Counter Function

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

5.6 Counter Set Rate

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

5.7 PWM

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

5.8 User Interrupt

The user interrupt feature of the Universal Driver enables to run a personal 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 user input and calls the user function periodically at the rate of interrupt frequency.

5.9 AD Sample

Performs a single A/D conversion on the currently selected channel and it will return a digital reading of an analog voltage signal applied to currently selected A/D board's analog input channel. The A/D board uses a device called an analog-to-digital (A/D) converter to convert the real-world analog signal (temperature, pressure, tank level, speed, etc.) into a digital value that the digital computer electronics can handle. This digital value can be translated back to the input voltage using the conversion formulas provided in the Aries user manual.

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

5.10 AD Sample Scan

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

5.11 AD Trigger

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

5.12 AD Interrupt Application

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 ARIESADIntCancel () is made.

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

5.13 LED Application

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

5.14 Watch Dog Timer Application

A watchdog timer is a simple countdown timer which is used to reset the board after a specific interval of time. The application will periodically restart the watchdog timer. After being restarted, the watchdog will begin timing another predetermined interval. When software or the device is not functioning correctly, software will not restart the watchdog timer before it times out. When the watchdog timer times out, it will cause a reset of the board. The watchdog timer can be retriggered in two ways i.e. software retrigger and hardware retrigger.

5.15 Temperature Sensor Application

The temperature sensor application reads and prints on board temperature value.

6. UNIVERSAL DRIVER DEMO APPLICATION USAGE INSTRUCTIONS

The following section illustrates how to enter into Aries 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 output range (0 = 0-5V, 1 = 0-10V, 2 = +/-5, 3 = +/-10) <default: 0>:
- Load calibration values (1-Enable,0-Disable)<default:1>:
- Enter output code (0-65535)<default:65535>:

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

Channel Number = 0
Output range = 3
Calibration values =1
Output code=0

Measure the voltage on D/A channel 0 (J17 Header 19th pin) using a multi meter. If it should show -10V, the application generated expected voltage.

6.2 D/A Scan Conversion

This application gets input from user as follows:

- Enter Range(0-4,0=0-5v, 1=0-10v, 2=+/-5, 3=+/-10):<default:1>:
- Enter the channel enable flag for channel 0
  (0 for FALSE, 1 for TRUE):
- Enter the output code for channel 0 (0-65535):
- Enter the channel enable flag for channel 1
  (0 for FALSE, 1 for TRUE ;):
- Enter the output code for channel 1 (0-65535):
- Enter the channel enable flag for channel 2
  (0 for FALSE, 1 for TRUE ;):
- Enter the output code for channel 2 (0-65535):
- Enter the channel enable flag for channel 3
  (0 for FALSE, 1 for TRUE ;):
- Enter the output code for channel 3 (0-65535):

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

- Enter Range(0-4,0=0-5v, 1=0-10v, 2=+/-5, 3=+/-10):<default:1>:2
- Enter the channel enable flag for channel 0
  (0 for FALSE, 1 for TRUE ;) 1
- Enter the output code for channel 0 (0-65535): 65535
- Enter the channel enable flag for channel 1
  (0 for FALSE, 1 for TRUE ;) 0
- Enter the channel enable flag for channel 2
  (0 for FALSE, 1 for TRUE; default: 1): 1
- Enter the output code for channel 2 (0-65535): 32768
- Enter the channel enable flag for channel 3
  (0 for FALSE, 1 for TRUE; default: 1): 0

Measure the voltage on D/A channel 0 (J17 Header 19th pin) and 2 (J17 Header 21st pin) using a multi meter. It should show 5V and 2.5V respectively, then the application generates the expected voltage.

6.3 DA Waveform

This application gets input from user as follows:

- Enter D/A Channel Number (0-3):
- Enter Range value (0-4 0 = 0-5V, 1 = 0-10V, 2 = +/5V, 3 = +/-10V,) <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 20, Default =1 :
- Select waveform type (0-Sine wave 1-Sawtooth Wave) <Default =0>:
- Enter waveform frequency <Default =100>:
- Enter waveform mode(0-1, 0-One shot 1-Repeat mode, default = 1:

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

Channel Number=0

Range =0

Waveform size = 500

Clock source = 1

Waveform type = 0; //sine wave

Waveform frequency = 100

Waveform mode = 1

Place an oscilloscope probe on D/A channel 0 (J17 Header 19th pin) 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

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

Output Byte:

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

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

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

The Byte value 119 is sent to port 0.

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

Input Byte:

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

To provide 3.3V to Port 0 pin 0 from VCC and it should read and display 0x01. To see the output, Run DIO application and provide input as follows

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

The application should show 0x01 on the screen.

6.5 Counter Function

This application gets input from the user as follows:

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

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

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

Place an oscilloscope probe on counter 0 output pin (J16 Header 10th pin) and set the voltage division to 1V range and the second division to 1ms. It should show a periodic pulse with 100Hz frequency, the application generated expected rate.

- Press any key and automatically the application stops counter output

6.6 Counter Set Rate

This application gets input from the user as follows:

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

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

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

Place an oscilloscope probe on counter 0 output pin (J16 Header 10th pin) and set the voltage division to 1V range and the second division to 1ms. It should show a periodic pulse with 100Hz frequency, the application generated expected rate.

- Press any key and automatically the application stops counter output

6.7 PWM

This application gets input from the user as follows:

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

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

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

Place an oscilloscope probe on PWM channel 0 output pin (J17 Header 24th pin) and set the voltage division to 1V range and the second division to 1ms. It should show a PWM wave form with a 50% duty cycle and 100Hz frequency, the application generated expected rate.

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

6.8 User Interrupt

This application gets input from the user as follows:

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

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

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

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

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

Press any key to cancel the interrupt.

6.9 A/D Sample

This application gets input from the user as follows:

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

To perform this operation for channel 0 with 5V range, run the AD Sample application, and provide an external voltage (5v) to channel 0 (J17 header 1st pin) and provide inputs as follows:

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

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

Sample readout: 32768, Actual voltage: 5.0V.

6.10 A/D Sample Scan

This application gets input from the user as follows:

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

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

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

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

6.11 A/D Trigger

This application gets input from the user as follows:

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

To perform this operation for channel 0 with 5V range, run the ADSample application, and provide an external voltage (5v) to channel 0 (J17 header 1st pin) and provide inputs as follows:

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

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

```
Sample readout: 32768, Actual voltage: 5.0V.
```

6.12 A/D Interrupt Application

This application gets input from the user as follows:

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

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

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

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

6.13 LED Application

This application gets input from the user as follows:

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

To toggle the on board LED, press the space bar key. To Disable the LED, press the same key used to enable the LED.

6.14 Watch Dog Timer Application

This application gets input from the user as follows:

- Enter duration for Counter A in seconds(Default=1 seconds):
- Enter duration for Counter B in Seconds (Default=1 seconds):
- Enter trigger mode 0-Software trigger 1-Hardware trigger(Default=Software trigger):
- Enter Edge Selection 0 = Raising edge 1 = falling edge(Default=Raising edge):
- Enter Count Early selection 0 = disable, 1 = enable(Default=Enable):
- Enable watchdog timer interrupt support 0=disable, 1=enable (Default = disable):

To perform watch dog timer operation with software trigger mode

- Enter duration for Counter A in seconds(Default=1 seconds): 1
- Enter duration for Counter B in seconds (Default= 1 seconds):1
- Enter trigger mode 0-Software trigger 1-Hardware trigger (Default=Software trigger):0

6.15 Temperature sensor application

This application reads and prints on board temperature value every second and user has to press any key to stop reading the temperature sensor value.

7. COMMON TASK REFERENCE

7.1 Data Acquisition Feature Overview

I/O Connectors

The Aries SBC provides 2 I/O connectors for the attachment of all data acquisition signals. Diamond's cable numbers 6980504 for analog I/O and 6980501 for digital I/O may be used to connect the user's signals to these connectors. These cables comes as part of the Aries cable kit, part number CK-ARS-01.

Analog inputs

There are 16 analog input channels, numbered 0-15, located on connector J17 pins 1-16. In single-ended mode, each pin acts as an individual channel. The voltage on each channel is measured relative to the reference analog ground, which must be connected to any of pins 17, 18. In differential mode, each consecutive pair of pins form a high-low pair; for example pins 1-2, or channels 0-1, 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 J17 pins 19-22. Analog outputs operate in single-ended mode only and are always referenced to the analog ground, which must be connected to any of pins 17 or 18.

Waveform generator

The board contains a 4-channel analog waveform generator using the analog outputs. One to four channels may operate simultaneously. The waveforms are stored in an on-board data buffer that holds 2048 samples. Any number of samples can be used up to the 2048 limit. If more than one channel is being used, then each channel's waveform must have the same number of samples, and the maximum length of each waveform is 2048 divided by the number of channels in use.

The waveform generator can be clocked in multiple ways, including: software command, external digital signal, 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	
2	
3	
4	

Digital I/O signals

There are 22 digital I/O signals, divided into three groups as DIOA0-DIOA7, DIOB0-DIOB7, and DIOC0-DIOC5. Ports A-B are on I/O connector J16 pins 2-17 and port C is on I/O connector J17 pins 24-28 and connector J16 pin 18.

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

All digital I/O signals have programmable pull-up/down. All signals within each group have the same pull direction. The default configuration for Aries is for all DIO signals to be pulled low.

Counter/timers

The board provides 8 32-bit counter/timers. Counter mode means the circuit will count external events that are connected via one of the digital I/O lines. Timer mode means the circuit will generate output pulses at a user-specified rate. The pulse width is programmable for both polarity (high or low pulse) and width (1, 10, 100, or 1000 clock pulses). The clock for timer mode is provided internally from an on-board 50MHz oscillator. This oscillator is divided by 50 to provide a 1MHz clock for very low pulse rates. The available range of output rates is 50MHz / 20ns (50MHz / 1) to .0002328Hz / 4295 sec (1MHz / 2^32).

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

Connector J16 Pin	**PortA Bit**	**Counter/Timer**
10	0	0
11	1	1
12	2	2
13	3	3
14	4	4
15	5	5
16	6	6
17	7	7

Pulse Width Modulators (PWMs)

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

The PWMs use the Port C C0-C3 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.

7.2 Data Acquisition Software Task Reference

This section describes the various data acquisition tasks that may be performed with Aries 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 ARIESInitBoard (). This function must be called prior to any other function involving the ARIES.
2. At the termination of the program the programmer may use ARIESFreeBoard (), 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, 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.

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. ARIESADSetSettings () is used to select the input type, input voltage range, and input channel range.
2. ARIESADSetClock () is used to select the A/D clock source and the scan mode if desired.
3. If software A/D clocking is selected, then ARIESADEnableClock () is used to enable the selected clock. ARIESADSample () is used to generate A/D conversions for A/D sample mode and ARIESADScan () is used to generate A/D conversions for A/D sample mode. The function must be called once for each A/D

sample. The board will auto-increment 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 ARIES hardware user manual. The programmer may also convert this number to whatever engineering units are appropriate for the application.

4. If external clocking or counter/timer clocking is selected, the A/D conversions will start as soon as the selected clock source becomes active. In this case, the A/D FIFO will start to fill up with samples, and the program should use the A/D interrupt functions described below to acquire the data from the FIFO.

A/D Interrupt Operations

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

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

The program must select the FIFO threshold based on two opposing parameters: The waiting time until the first data is available (threshold divided by sample rate) and the desired interrupt rate (sample rate divided by threshold). Generally the FIFO threshold should be selected to avoid exceeding a 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. ARIESADSetSettings () is used to select the input type, input voltage range, and input channel range.
2. ARIESADSetClock () is used to select the clock source.
3. ARIESADInt () is used to install the interrupt handler.
4. If a counter/timer is being used to control A/D timing, then ARIESCounterSetRate () is used to program the counter/timer for the desired sample rate.
5. ARIESADEnableClock () is used to enable selected clock source and data will automatically be transferred to the user-specified data buffer based on the selected FIFO threshold.
6. To monitor A/D operations, use ARIESADIntStatus ().

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

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

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

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

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

<u>Waveform Generator</u>

To use 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 <no. of channels> is also referred to as the frame size.

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

ARIESWaveformDataLoad () 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 ARIESWaveformConfig () 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 ARIESWaveformStart (). 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 ARIESWaveformPause (). The waveform may be restarted in its current position by using ARIESWaveformStart () again.

To reset the waveform generator use ARIESWaveformReset (). 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 ARIESWaveformConfig () should be called followed by ARIESWaveformStart ().

If software increment is selected, the waveform is incremented with the function ARIESWaveformInc (). 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 operation

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

BYTE ARIESDIOConfig () configures all 8 bits of a selected port
BYTE ARIESDIOConfigAll () configures all 30 bits at once.

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

BYTE ARIESDIOOutputByte () outputs 8 bits of data
BYTE ARIESDIOInputByte (BoardInfo* bi, int Port, byte* Data);
BYTE ARIESDIOOutputBit (BoardInfo* bi, int Port, int Bit, int Value);
BYTE ARIESDIOInputBit (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 ARIESCounterSetRate (). 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 ARIESCounterConfig () 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 ARIESCounterRead () to read the current contents of the counter. This function can be used repeatedly to monitor the operation. This is normally used with event counting.

3. When the counting function is no longer needed, use ARIESCounterReset () 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 ARIESCounterFunction (). This function must be used multiple times to execute each command needed to configure the counter. All commands take effect immediately upon execution. The typical command sequences for the most common operations are provided below. See the full list of counter/timer commands in the appendix.

For a rate generator:

Command	Function
15	Reset the counter/timer. This function should be used first to reset any previous configurations to ensure the counter/timer operates exactly as desired.
1	Load counter with desired divisor to select the desired output pulse rate. The output rate is the selected clock frequency divided by the divisor.
2	Select the count direction. For a rate generator the direction should be down.
6	Select clock source. Normally an internal clock (50MHz or 1MHz) will be selected.
7	Enable auto-reload. This means that the counter will operate continuously.

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

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

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

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

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

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

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

For an event counter:

15	Reset the counter/timer. This function should be used first to reset any previous configurations to ensure the counter/timer operates exactly as desired. This will reset the counter data register to 0.
2	Select the count direction. For an event counter the direction should be up.
6	Select clock source. Normally the associated digital I/O pin will be selected to enable counting external pulse.
7	Enable or disable auto-reload. If auto-reload is enabled, the counter will operate continuously, meaning that when it reaches 2^32-1 it will roll over to zero. In most cases auto-reload will be disabled for event counting.
4	Start the counter/timer running. (This function is also used to stop the counter/timer.)

While the counter is operating, its current count can be read by using the ARIESCounterRead () function.
When the counting function is no longer needed, the function ARIESCounterReset () 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. ARIESPWMConfig () configures the selected PWM for output frequency, duty cycle, and polarity. The PWM may optionally be started as well.
2. ARIESPWMStart () can be used to start the PWM running if the config function did not start it.

To stop a PWM:

ARIESPWMStop () 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 ARIESPWMStart ().
To reset a PWM and return its assigned digital I/O output pin to normal operation, use ARIESCounterReset ().

To implement special functions, such as changing the duty cycle or frequency of a PWM while it is running, use ARIESPWMCommand (). 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 (standalone or alone mode) or it can run before or after the driver's built-in interrupt function (Before and after modes). The available modes depend on the source of the interrupt:

Source	Source no.	Modes supported
A/D interrupts	0	1 before, 2 after
D/A interrupts	1	Not supported on Aries
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:

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

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

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

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

LED control

The ARIES 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 ARIESLED ().

7.3 Performing D/A Conversion

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 using one of the equations described in the hardware manual.

The Universal Driver function for performing a D/A conversion is ARIESDAConvert()

Step-By-Step Instructions

Call ARIESDAConvert() 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.

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

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

7.4 Performing D/A Scan Conversion

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

The Universal Driver function for performing a D/A conversion scan is:

ARIESDAConvertScan ()

Step-By-Step Instructions

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

Call ARIESDAConvertScan() 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 4 with DA code 65535 and 32768 respectively and the rest of the channels not to be changed from existing voltage level:

```
ChannelSelect = (int *) malloc (sizeof (int) *16);
DACodes = (int *) malloc (sizeof (int) *16);
ChannelSelect [0] = 1;
DACodes [0] = 65535;
ChannelSelect [4] = 1;
DACodes [4] = 32768;
if ((result = ARIESDAConvertScan ( bi, ChannelSelect, DACodes) ) != DE_NONE)
     return result;
```

7.5 Performing D/A Simultaneous Update

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

The Universal Driver functions described here are:

ARIESDASetSim (), ARIESDAConvert(), ARIESDAUpdate()

Step-By-Step Instructions

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

Example Usage for DA Simultaneous Update

```
ARIESDASetSim (bi, 1);
ARIESDAConvert (bi, 0, 65535);
ARIESDAConvert (bi, 1, 65535);
ARIESDAConvert (bi, 2, 65535);
ARIESDAConvert (bi, 3, 65535);
ARIESDAUpdate (bi);
ARIESDAConvert (bi, 0, 0);
ARIESDAConvert (bi, 1, 0);
ARIESDAConvert (bi, 2, 0);
ARIESDAConvert (bi, 3, 0);
ARIESDAUpdate (bi);
```

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

7.6 Performing Digital IO Operations

The driver supports four types of direct digital I/O operations: input bit, input byte, output bit, and output byte. Digital I/O is fairly straightforward - to perform digital input, 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 described here are:

ARIESDIOConfig (), ARIESDIOConfigAll (), ARIESDIOOutputByte (), ARIESDIOInputByte (), ARIESDIOOutputBit (), ARIESDIOInputBit ()

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 ARIESDIOConfigAll () to set the direction if necessary or use function ARIESDIOConfig () to set the direction for a single bit.

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

Example of Usage for Digital I/O Operations

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

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

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

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

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

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

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

7.7 Performing PWM Operations

The PWM operation generates a PWM signal with desired frequency and duty cycle value. The following functions are used for PWM operation: ARIESPWMConfig () , ARIESPWMStart() and ARIESPWMStop().

Step-By-Step Instructions

Create ARIESPWM structure variable to hold PWM settings and initialize PWM structure variables, then call ARIESPWMConfig () to configure the PWM, call the ARIESPWMStart () to start the PWM and finally call ARIESPWMStop ()to stop the running PWM signal.

Example of Usage for PWM Operations

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

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

7.8 Performing Counter Function Operations

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

The following functions are used for counter function operator

ARIESCounterConfig () and ARIESCounterFunction ()

Step-By-Step Instructions

Create ARIESCOUNTER structure variable to hold Counter settings and initialize counter structure variables then call ARIESCounterConfig () to configure the counter and finally ARIESCounterFunction () to stop the running counter.

Example of Usage for Counter Operations

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

7.9 Performing Counter Set Rate Operation

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

The following functions are used for counter set rate operation:

ARIESCounterSetRate (), ARIESCounterRead () and ARIESCounterFunction()

Step-By-Step Instructions

Create ARIESCOUNTER structure variable to hold Counter settings and initialize counter structure variables then call ARIESCounterSetRate () to program a counter for timer mode with down counting and continuous operation (reload enabled), then call ARIESCounterRead () to read the counter value and finally call ARIESCounterFunction() to stop the running counter.

Example of Usage for Counter Set Rate Operations

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

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

7.10 Performing User Interrupt Operations

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

To perform User Interrupt application following Universal driver function are used:

ARIESUserInterruptConfig (), ARIESCounterSetRate() and   ARIESUserInterruptStop ()

Step-By-Step Instructions

Create ARIESUSERINT structure variable to hold Interrupt settings and initialize Interrupt structure variables, then call ARIESUserInterruptConfig() to run the user interrupt , ARIESCounterSetRate() to set the counter and finally call ARIESUserInterruptCancel () to stop the user interrupt.

Example of Usage for User Interrupt Operations

```
int count =0;
ARIESUSERINT inter;
ARIESCOUNTER counter;

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

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

      While ( !kbhit())
      {
            dscSleep (1000);
            printf ("Count value %d \r",count);
            fflush (stdout);
      }
      inter.Enable= 0;
      if (ARIESUserInterruptStop(bi,&inter) !=DE_NONE)
      {
            dscGetLastError ( &errorParams );
            printf ( "ARIESUserInterruptStop error: %s %s\n", dscGetErrorString
            ( errorParams.ErrCode ),
            errorParams.errstring);
            return 0;
      }
      printf ("Press any key to terminate the application \n");
      while ( !kbhit())
      {
            dscSleep (1000);
            printf ("Count value %d \r",count);
            fflush (stdout);
      }

}
```

7.11 Generating D/A Waveform

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

The Universal Driver functions described here are:

ARIESWaveformBufferLoad(), ARIESWaveformDataLoad(), ARIESWaveformConfig(), ARIESWaveformStart(), ARIESWaveformPause(), ARIESWaveformReset(), ARIESWaveformInc().

Step-By-Step Instructions

First, configure the waveform generator. This can be done with ARIESWaveformConfig(). If a counter/timer is selected as the clock, ARIESWaveformConfig() automatically calls ARIESCounterSetRate(). Then, load the buffer full of DA packets that contain a channel and DA value it should have, this can be done with ARIESWaveformDataLoad() loading one packet at a time. After the buffer is loaded, one may wish to set up a counter/timer or input signal to be inputted to the board, so that the waveform can increment automatically. Otherwise, ARIESWaveformInc() should be used to single-increment the waveform manually via software. After loading the buffer, and selecting input clock, one must start the Waveform Generator, with ARIESWaveformStart(). If for some reason it is desirable to pause the waveform, use ARIESWaveformPause(), followed by ARIESWaveformStart() to resume. To start from the beginning of the buffer, run ARIESWaveformReset(), followed by ARIESWaveformStart().

Example of Usage for D/A Waveform Generator

```
ARIESDASetSettings (bi, 0, 0, 0, 0);            // Set up D/A with desired
configuration
ARIESWAVEFORM ARIESwaveform;
ARIESwaveform.Frames        = 4;          // Four frames of size:
ARIESwaveform.FrameSize    = 1;          // 1 value per frame
ARIESwaveform.Clock          = 1;             // Input clock to use:
//Counter/Timer 0
ARIESwaveform.Rate = 100;                 // 100Hz, uses ARIESCounterSetRate
ARIESwaveform.Cycle = 1;
ARIESWaveformConfig (bi, & ARIESwaveform);
ARIESwaveform.Waveform = (int*) malloc (sizeof (int) *4);
ARIESwaveform.Waveform [0] = 0;           // Position 0 data
ARIESwaveform.Waveform [1] = 16384;       // Position 1 data
ARIESwaveform.Waveform [2] = 32768;       // Position 2 data
ARIESwaveform.Waveform [3] = 0;           // Position 3 data
ARIESwaveform.Channels [0] = 0;                 // Position 0 channel
ARIESwaveform.Channels [1] = 0;           // Position 1 channel
ARIESwaveform.Channels [2] = 0;           // Position 2 channel
ARIESwaveform.Channels [3] = 0;           // Position 3 channel
ARIESWaveformBufferLoad (bi, &ARIESwaveform);   // Loads into D/A buffer
//In real-time change position 3 will have 65535 instead (or any other reason
//someone might want to change a D/A value on-the-fly):
ARIESWaveformDataLoad (bi, 3, 0, 65535);
ARIESWaveformStart (bi);                        // Waveform Generator triggers
//using Counter0
```

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

7.12 Performing A/D Sample

When an A/D conversion is being performed, you are getting a digital reading of an analog voltage signal applied to one of the A/D board's analog input channels.

The following Universal Driver function is used for A/D conversion:

ARIESADSample ()

Step-By-Step Instructions

Create ARIESADSETTINGS structure variable to hold A/D settings and initialize structure variables then call ARIESADSample () to configure the A/D settings as requested by the user, and It then takes a single A/D sample value of a single channel.

Example of Usage for A/D Sample

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

7.13 Performing A/D Scan

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

The Universal Driver functions for performing a A/D scan are ARIESADSetSettings (),ARIESADEnableClock (),ARIESADScan ().

Step-By-Step Instructions

Create ARIESADSETTINGS structure variable to hold A/D settings and initialize structure variables then call ARIESADSetSettings () configures the A/D settings as requested by the user, call ARIESADEnableClock () to configure the turbo mode, A/D clock source, and scan settings and finally call ARIESADScan () to execute A/D conversion scan using the current board settings.

Example of Usage for A/D Scan

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

7.14 Performing A/D interrupts

The AD interrupt application Performs A/D scans using interrupt-based I/O with one scan per A/D clock tick. The following Universal driver functions are used for A/D interrupt: ARIESADSetSettings(), ARIESADSetClock(), ARIESADInt(), ARIESADIntResume(), ARIESADIntStatus(), and ARIESADIntCancel().

Step-By-Step Instructions

Create ARIESADSETTINGS structure variable to hold A/D settings, create ARIESADINT structure variable to hold A/D conversion interrupt settings, create ARIESADINTSTATUS structure variable to hold A/D conversion interrupt status. Initialize A/D settings structure variables then call ARIESADSetSettings () configures the A/D settings as requested by the user, call ARIESADSetClock () to configure the turbo mode, A/D clock source, and scan settings. Initialize A/D interrupt status structure variable then call ARIESADInt() to enable A/D interrupt operation using the current analog input settings and call ARIESADIntStatus() to get the interrupt routine status including, running / not running, number of conversions completed, cycle mode, FIFO status, and FIFO flags, call ARIESADIntResume() to resume the interrupt, call ARIESADIntPause () to pause the interrupt and finally call ARIESADIntCancel() to stop the A/D interrupt.

Example of Usage for A/D interrupt

```
ARIESADSETTINGS dscadsettings; // structure containing A/D conversion settings
ARIESADINT   dscIntSettings; // structure containing A/D conversion interrupt
settings
ARIESADINTSTATUS intstatus; // structure containing A/D conversion interrupt status
int sample[8];       // sample reading
int key = 0;
int Pause = 0;
/* Initializing A/D settings */
dscadsettings.Lowch = 0;
dscadsettings.Highch = 7;
dscadsettings.Polarity = 0; //bipolarity
dscadsettings. Gain= 0; //5v
dscadsettings.Sedi = 0;
dscadsettings.ScanEnable = 0;
dscadsettings.ScanInterval = 0;
dscadsettings.ADClock = 2;
if ( (ARIESADSetSettings ( bi, &dscadsettings ) ) != DE_NONE )
{
      dscGetLastError (&errorParams);
      printf ("ARIESADSetSettings error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode),
      errorParams.errstring);
      return 0;
}
if ( (ARIESADSetClock (bi,&dscadsettings.ADClk) ) != DE_NONE )
{
      dscGetLastError (&errorParams);
      printf ( " ARIESADSetClock  error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode),
      errorParams.errstring);
      return 0;
}
/* initializing Interrupt settings */
dscIntSettings.NumConversions = 1000;
dscIntSettings.Cycle = 1;
dscIntSettings.FIFOEnable = 1;
dscIntSettings.FIFOThreshold = 1600;
dscIntSettings.ADBuffer = (SWORD*) malloc (sizeof (SWORD)*
dscIntSettings.NumConversions);
if (ARIESADInt (bi,&dscIntSettings ) !=DE_NONE)
```

```
{
      dscGetLastError (&errorParams);
      printf ( "ARIESADInterror: %s %s\n", dscGetErrorString(errorParams.ErrCode),
      errorParams.errstring);
      return 0;
}
if (ARIESCounterSetRate (bi,dscadsettings.ADClock-2,rate,0,0) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ( "ARIESCounterSetRateerror: %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 (ARIESADIntStatus (bi, &intstatus)! =DE_NONE)
            {
                  dscGetLastError (&errorParams);
                  printf ( "ARIESADIntStatuserror: %s %s\n",
                  dscGetErrorString(errorParams.ErrCode),
                  errorParams.errstring);
                  return 0;
            }
            printf ("No of A/D conversions
            completed %d\n",intstatus.NumConversions);
            dscSleep (1000);
            if ( (dscIntSettings.Cycle == 0) &&
            (intstatus.NumConversions  >=dscIntSettings.NumConversions) )
            {
                  break;
            }
            else
            {
                  key = getchar();
                  if (key ==32 )
                  {
                        If (Pause)
                        {
                              If (ARIESADIntResume (bi)! =DE_NONE)
                              {
                                    dscGetLastError (&errorParams);
                                    printf ( "ARIESADIntResume error: %s %s\n",
                                    dscGetErrorString(errorParams.ErrCode),
                                    errorParams.errstring );
                                    return 0;
                              }
                              Pause = 0;
                        }
                        else
                        {
                              if(ARIESADIntPause (bi) !=DE_NONE)
                              {
                                    dscGetLastError (&errorParams);
                                    printf ( "ARIESADIntPause error: %s %s\n",
                                    dscGetErrorString (errorParams.ErrCode),
                                    errorParams.errstring);
                                    return 0;
                              }
                              Pause = 1;
                        }
                  }
                  else
                  {
                        break;
                  }
            }
      }
}
If (ARIESADIntCancel (bi)! =DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ( "ARIESADIntCancel error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode),
      errorParams.errstring);
      return 0;
}
```

7.15 Performing LED operations

The application toggles on board LED whenever space bar key pressed. The ARIESLED () function is used to toggle 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)
            {
                  ARIESLED (bi, 0);
                  LEDOn = 0;
            }
            else
            {
                  ARIESLED (bi, 1);
                  LEDOn = 1;
            }
      }
      else if(key == 'q')
      {
            break;
      }
}
```

7.16 Performing Watch Dog Software Retrigger

The application starts watch dog timer with user loaded data on Counter A and counter B, also the application does software retrigger prior to Counter A times out, it avoids the board reset.

This application uses following universal driver functions

ARIESWatchDogConfig(),ARIESWatchDogEnable(),ARIESWatchDogSoftwareReTrigger() and ARIESWatchDogDisable()

Step by step instructions

Create ARIESWATCHDOG structure variable to hold Watch Dog timer configuration settings , then initialize the structure with user requirement and call ARIESWatchDogConfig() function which configures the Watch dog timer , to enable the watch dog timer call ARIESWatchDogEnable() fuction . The watch dog timer must be retriggered prior to Counter A times out or at least prior to Counter B times out with ARIESWatchDogSoftwareReTrigger() function.

Example usage of watch dog timer with software retrigger

```
ARIESWATCHDOG WDT;
memset ( & WDT, 0, sizeof(ARIESWATCHDOG) );
double CouterA_Time=1.0; // In seconds , maximum 655.35 second possible
double CouterB_Time=1.0; // In seconds , maximum 2.55 second possible

WDT.CtrAData= (int) (CouterA_Time *100);
WDT.CtrBData= (int) (CouterB_Time *100);
WDT.InterruptEnable= 0;
WDT.HardwareTriggerEnable=0;
WDT.EdgeSelection=0;
WDT.CountEarly=0;
if (ARIESWatchDogConfig(bi,&watch) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ("ARIESWatchDogConfig error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}

if(ARIESWatchDogEnable(bi) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ("ARIESWatchDogEnable error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}
printf ("Press any key to stop the watch dog timer:\n");
while (!kbhit())
{
      dscSleep(1000);
      if (ARIESWatchDogSoftwareReTrigger(bi) !=DE_NONE)
      {
            dscGetLastError (&errorParams);
            printf ("ARIESWatchDogSoftwareReTrigger error: %s %s\n",
            dscGetErrorString(errorParams.ErrCode), errorParams.errstring);
            return;
      }

}
```

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

7.17 Performing Watch Dog Hardware Retrigger

The application configures the Watch dog timer and enables hardware retrigger, the user has to specify that the watch dog retrigger must happen during falling edge or raising edge through EdgeSelection parameter of ARIESWATCHDOG structure. It is user responsibility to provide raising or falling edge on DIO C5 pin on Header J16 prior to Counter A times out or at least prior to Counter B times out, also WDT Output pin DIO C4 goes high to provide an indicator to an external circuit of the timeout of Counter A .

This application uses following universal driver functions

ARIESWatchDogConfig() ,ARIESWatchDogEnable() and ARIESWatchDogDisable()

Step by step instructions

Create ARIESWATCHDOG structure variable to hold Watch Dog timer configuration settings , then initialize the structure with user requirement and call ARIESWatchDogConfig() function which configures the Watch dog timer , to enable the watch dog timer call ARIESWatchDogEnable() fuction . The watch dog timer must be retriggered prior to CounterA times out or at least prior to Counter B times out.

Example usage of watch dog timer with Hardware Retrigger

```
ARIESWATCHDOG WDT;
memset ( & WDT, 0, sizeof(ARIESWATCHDOG) );
double CouterA_Time=1.0; // In seconds , maximum 655.35 second possible
double CouterB_Time=1.0; // In seconds , maximum 2.55 -second possible

WDT.CtrAData = (int) (CouterA_Time * 100);
WDT.CtrBData = (int) (CouterB_Time * 100);
WDT.InterruptEnable= 0;
WDT.HardwareTriggerEnable=1;
WDT.EdgeSelection=0;
WDT.CountEarly=1;
if (ARIESWatchDogConfig(bi,&watch) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ("ARIESWatchDogConfig error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}
if (ARIESWatchDogEnable(bi) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ("ARIESWatchDogEnable error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}
printf ("Press any key to stop the watch dog timer:\n");
while (!kbhit())
{

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

7.18 Performing Watch Dog operation with interrupt support

The application installs a user function, the function gets called when Counter A times out. So user can decide that what to do in the user function, then user can either do software retrigger using ARIESWatchDogSoftwareReTrigger() function or complexly disable the watch dog timer using ARIESWatchDogDisable()function along with user requirement.

This application uses following universal driver functions

ARIESWatchDogConfig(),ARIESWatchDogEnable(),ARIESWatchDogSoftwareReTrigger() and ARIESWatchDogDisable()

Step by step instructions

Create ARIESWATCHDOG structure variable to hold Watch Dog timer configuration settings , then initialize the structure with as user requirement , also a function pointer and call ARIESWatchDogConfig() function which configures the Watch dog timer , to enable the watch dog timer call ARIESWatchDogEnable() fuction . Once the counter A reaches 0, an interrupt gets triggered which in turn calls user registered function.

Example usage of watch dog timer with interrupt support

```
int count = 0 ;
void userfun(void *param)
{
if(ARIESWatchDogSoftwareReTrigger(bi) !=DE_NONE)
{
      dscGetLastError(&errorParams);
      printf("ARIESWatchDogSoftwareReTrigger error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring);
      return;
}
count++;
}
ARIESWATCHDOG WDT;
memset ( & WDT, 0, sizeof(ARIESWATCHDOG) );
double CouterA_Time=1.0; // In seconds , maximum 655.35 second possible
double CouterB_Time=1.0; // In seconds , maximum 2.55 -second possible

WDT.CtrAData = (int) (CouterA_Time * 100);
WDT.CtrBData = (int) (CouterB_Time * 100);
WDT.InterruptEnable= 1;
WDT.IntFunc = userfun;
WDT.HardwareTriggerEnable=0;
WDT.EdgeSelection=0;
WDT.CountEarly=0;
if(ARIESWatchDogConfig(bi,&watch) !=DE_NONE)
{
      dscGetLastError(&errorParams);
      printf("ARIESWatchDogConfig error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}
if(ARIESWatchDogEnable(bi) !=DE_NONE)
{
      dscGetLastError(&errorParams);
      printf("ARIESWatchDogEnable error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}
printf ("Press any key to stop the watch dog timer:\n");
```

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

7.19 Performing Temperature sensor read operation

This application reads and prints temperature sensor value every second

This application uses following universal driver functions

ARIESTemperatureSensorRead ()

Step by step instructions

Call ARIESTemperatureSensorRead() function with command as 0x01 for reading temperature , pass a integer pointer variable to hold temperature value. On success of the function, the variable will hold actual temperature of the board.

Example usage of Temperature sensor function

```
int temperature = 0 ;
printf("Press any key to stop reading the sensor\n");
while(!kbhit())
{

      if ( (ARIESTemperatureSensorRead(bi,0x01,&temperature)!= DE_NONE) )
      {
            dscGetLastError ( &errorParams );
            printf (  "ARIESInitBoard error: %s %s\n", dscGetErrorString
            ( errorParams.ErrCode ), errorParams.errstring );
            return 0;
      }
      printf("Temperature = %d%c%s\n",temperature,248,"C");
      dscSleep(1000);
}
```

8. INTERFACE CONNECTOR DETAILS

8.1 ARIES Digital GPIO Connector – J16

Aries provides three digital I/O ports with 8 lines on port A and B, and 6 lines on port C. Port A and B are provided on connector J16. Port C is on connector J17 and pin 18 of connector J16.

ARIES Digital GPIO Connector Pinout

VIO (fused)	**1**	**2**	DIO A0
DIO A1	**3**	**4**	DIO A2
DIO A3	**5**	**6**	DIO A4
DIO A5	**7**	**8**	DIO A6
DIO A7	**9**	**10**	DIO B0
DIO B1	**11**	**12**	DIO B2
DIO B3	**13**	**14**	DIO B4
DIO B5	**15**	**16**	DIO B6
DIO B7	**17**	**18**	DIO C5
GND	**19**	**20**	GND

Description: 20 position 1.25mm pitch right angle latching

8.2 ARIES Analog GPIO Connector – J17

The VIO pins on the analog and digital I/O connectors are tied together on the board and provide access to jumper-selectable 3.3V / 5V system voltage rail through a poly-switch resettable fuse. The fuse is rated for ~100mA maximum sustained current.

ARIES Analog GPIO Connector Pinout

AIN0	**1**	**2**	AIN8
AIN1	**3**	**4**	AIN9
AIN2	**5**	**6**	AIN10
AIN3	**7**	**8**	AIN11
AIN4	**9**	**10**	AIN12
AIN5	**11**	**12**	AIN13
AIN6	**13**	**14**	AIN14
AIN7	**15**	**16**	AIN15
AGND	**17**	**18**	AGND
AOUT0	**19**	**20**	AOUT1
AOUT2	**21**	**22**	AOUT3
AGND	**23**	**24**	AGND
DIO C1	**25**	**26**	DIO C2
DIO C3	**27**	**28**	DIO C4
VIO (fused)	**29**	**30**	DGND

Description: 2x15 position 1.5mm pitch right angle latching

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 the manual for more information.

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

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

2 Select count direction, up or down

3 Enable / disable external gate. This command is not implemented on the ARIES.

4 Enable / disable counting

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

6 Select counter clock source according to the table below:

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

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

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

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

8 Enable counter output and select the output pulse polarity. The initial logic level of the output pin will be the inactive state (the opposite state of the selected polarity). The output pulse always comes at the end of each clock period. The counter outputs are enabled on DIO pins according to the following table. Enabling a counter output automatically sets the corresponding DIO pin's direction to output. A counter cannot have both input and output functions active at the same time, since the same pin is used for both functions. If both are selected, the input function will prevail, and the requested output function will be ignored.

Connector J16 Pin	PortA Bit	Counter/Timer
10	0	0
11	1	1
12	2	2
13	3	3
14	4	4
15	5	5
16	6	6
17	7	7

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

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

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

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

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

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

PWM commands

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

0 Stop all PWMs / selected PWM
 0 = stop all PWMs (opposite polarity for "all" compared to other PWM commands)
 1 = stop single PWM

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

1 Load counter C0 (period counter) or C1 (duty cycle counter)
 0 = load C0 / period counter
 1 = load C1 = duty cycle counter

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

0 = pulse high
1 = pulse low

3 Enable/disable pulse output
0 = disable pulse output; output = opposite of polarity setting from command 0010
1 = enable pulse output

4 Reset all PWMs / selected PWM
0 = reset PWM selected with PWM2-0
1 = reset all PWMs

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

5 Enable/disable PWM outputs on DIO port F
0 = disable output
1 = enable output on DIO pin; this forces the DIO pin to output mode

6 Select clock source for a PWM
0 = 50MHz
1 = 1MHz

7 Start all PWMs / selected PWM
0 = start PWM selected with PWM2-0
1 = start all PWMs

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

PWM outputs may be made available on I/O pins 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).