Win10 Driver Manual

PCI and PCIe models

ASCB Aircraft Standard Communication Bus

Manual Revision	01p1
Revision Date	7/18/2023

Dynamic Engineering
150 DuBois St. Suite B&C
Santa Cruz, CA 95060
(831) 457-8891
www.dyneng.com
sales@dyneng.com
Est. 1988

ASCB

Cautions and Warnings

The electronic equipment described herein generates, uses, and can radiate radio frequency energy. Operation of this equipment in a residential area is likely to cause radio interference, in which case the user, at their own expense, will be required to take whatever measures may be required to correct the interference.

Dynamic Engineering's products are not authorized for use as critical components in life support devices or systems without express written approval from the president of Dynamic Engineering.

Connection of incompatible hardware is likely to cause serious damage.

Table of Contents

Figures

Tables

Design Revision History

Table 1: Design Revision History

Revision	Date	Description

Manual Revision History

Table 2: Manual Revision History

Revision	Date	Description
1.1	7/18/23	Original

NOTE: Dynamic Engineering has made every effort to ensure that this manual is accurate and complete; that being said, the company reserves the right to make improvements or changes to the product described in this document at any time and without notice. Furthermore, Dynamic Engineering assumes no liability arising out of the application or use of the device described herein.

Product Description

ASCB is a 2-port device used to interact with ASCB using the D and E modes. [Manchester or 8B10B "enhanced"] Primary, and redundant interfaces within each port. This driver was developed as a Windows Kernel Driver using the KMDF.

Each port has memory configured as Dual Ported Ram. Both Target access and DMA are supported for memory interaction. The UserAp has references for both.

Software Description

The Dynamic Engineering ASCB driver uses the Win10 KMDV version 1.0 driver. This consists of two modules, a base driver and a channel driver module, that can be loaded using the .inf file.

This package comes with a User App which is used to test the hardware as well as provide an example of how to interface with the device through software:

Table 3: Header Files

File Name	Description
AscbBasePublic.h	Defines serves as the primary API for the device. Defines many of the data structures used by the IOCTL calls for the base device.
AscbChanPublic.h	Defines many of the data structures used by the IOCTL calls for the channel device.

Test Suite

The UserApp is used in-house to validate the hardware and provides a more extensive example of how to interact with the hardware. The UserApp was written in C/C++. Some of the test suite tests require a loop-back cable between the ports. The purpose of this is to test the HW within a single system and abstract away some of the complexity for the testing team.

Driver Installation

Driver Installation is simple, first right-click on the AscbBase.inf file and click "install", this will load the base driver automatically and enumerate the channels in the Window's Device Manager. Once this is installed, follow the same step by right clicking on the AscbChan.inf file and click "Install".

Once the driver is installed on a system, the driver files are automatically copied to the "Window's Store" (i.e., a special directory where windows stores driver files). The driver will automatically load every time the computer is booted.

Alternatively you can right click on the icon within Device Manager and navigate to the driver to install.

Uninstallation

Open the device manager and navigate to the "ASCB Device" element, expand the tree down by pressing the ">" arrow. There you will see the base and channel device. Right-click on the channel device and select "uninstall device" (IMPORTANT – once this uninstall is selected a window will pop up, check the box that says "Delete the driver software installed for this device" (this removes the driver from the windows store). For the second channel this pop-up will not offer the same box as the software is already removed from the store. Finally, do the same thing with the base driver – again remembering to check the box as the base driver is a separate piece of software that has been copied to the Window's Store.

API

The primary API for the device can be located in the Ioctl.c file, and there are some additional ioctl calls that can be found in the AscbBase/ChanPublic.h files.

Name(HANDLE, Structure or size, Length) Use the structure name to locate the controlling data. ULONG used when no structure defined. UserAp has examples of all of the following calls. Ioctl.c has the full definition of each of the macros. The top line of each along with a comment are included here for reference.

GetBaseInfo
(HANDLE hndl, PAscb_BASE_DRIVER_DEVICE_INFO info, PULONG length)
Returns information about the base driver and hardware. Driver revision, Flash Revision etc. Note PLL Device ID has no meaning for this design.

SetBaseConfig
(HANDLE hndl, PASCB_BASE_CONFIG config, PULONG length)
Pass structure to configure Base HW. Frame Counter, Port interrupts etc.

GetBaseConfig
(HANDLE hndl, PASCB_BASE_CONFIG config, PULONG length)
Returns structure with configuration of Base HW.

ReConfigBridge
(HANDLE hndl, PULONG length)
Call to cause Driver to reconfigure the Bridge. Occasionally the system is not ready to configure the bridge when the driver is loading. If the Bridge Configured status is not configured, use this call to update the settings in the bridge.

GetBaseReg
(HANDLE hndl, PULONG data, PULONG length)
Read the current value of the base register. Refer to the HW manual for the bit definitions.

GetBaseStatus
(HANDLE hndl, PULONG data, PULONG length)
Read the Base port status register. Refer to the HW manual for the bit definitions.

SetBaseStatus
(HANDLE hndl, PULONG data, PULONG length)
Write to the base port status register. Used to clear sticky bits.

GetBaseIsrStatus
(HANDLE hndl, PAscb_BASE_ISR_STAT isrstatus, PULONG length)
Returns the status from when an interrupt has occurred. Self-clearing.

RegisterBaseEvent

(HANDLE hndl, HANDLE* Event, PULONG length)
Used to set-up for an Interrupt event other than DMA [handled separately]. Also used to clear the Registered event. See the Force Interrupt routines for examples. The caller creates an event with CreateEvent() and supplies the handle returned from that call as the input to this IOCTL. The driver then obtains a system pointer to the event and signals the event when a user interrupt is serviced. The user interrupt service routine waits on this event, allowing it to respond to the interrupt.

SetTemp

(HANDLE hndl, PULONG Temperature, PULONG length)
Write the control word to the temperature interface.

GetTemp

(HANDLE hndl, PULONG Temperature, PULONG length)
Poll for completion and to return the temperature value. See the LM75 routines for conversion etc.

SetBaseEndCnt

(HANDLE hndl, PULONG EndCount, PULONG length)
Write to the Frame End Count Register. An 18-bit counter counts frame clocks. This count determines at what point the counter rolls over to zero again. See HW Manual for details of operation.

GetBaseEndCnt

(HANDLE hndl, PULONG EndCount, PULONG length)
Read from the Frame End Count Register. See HW Manual for details of operation.

SetBaseIntCnt

(HANDLE hndl, PULONG IntCount, PULONG length)
Write to the Frame Interrupt Count Register. Set the frame-clock counter value that causes the frame-count interrupt status bit to be asserted and can be configured to cause an interrupt if the frame count interrupt enable bit is set in the base control register. See HW Manual for details of operation.

GetBaseIntCnt

(HANDLE hndl, PULONG IntCount, PULONG length)
Read from the Frame Interrupt Count Register. See HW Manual for details of operation.

SetBasePktDly

(HANDLE hndl, PULONG DelayCount, PULONG length)
Write a new definition for the delay between packets. HW defaults to pre-programmed value. See HW manual for detail of operation. If the register contains zeros, as it will on power-up or after reset, the default value of 0x73 will be used. This results in the nominal inter-packet delay of 9.6 microseconds.

GetBasePktDly

(HANDLE hndl, PULONG DelayCount, PULONG length)
Returns current value of the Base Packet Delay register.

ReadBaseCnt
(HANDLE hndl, PULONG ReadCount, PULONG length)
Read the current counter value. Note the counter runs at the frame clock rate [not host rate]. Values may jump a bit when read on the fly.

GetChanInfo
(HANDLE hndl, PAscb_CHAN_DRIVER_DEVICE_INFO info, PULONG length)
Returns structure with Port information. See structure definition for members.

WriteMemData
(HANDLE hndl, PASCB_WRITE_WORD AscbWriteWord, PULONG length)
Pass structure to write to Dual Port RAM. See DPR tests for examples.

ReadMemData
(HANDLE hndl, PASCB_READ_WORD AscbReadWord, PULONG length)
Returns structure with data from DPR.

SetChanConfig
(HANDLE hndl, PASCB_CHAN_CONFIG config, PULONG length)
Pass structure to configure Port HW. Enables for Tx, Rx, Type of encoding etc.

GetChanConfig
(HANDLE hndl, PASCB_CHAN_CONFIG config, PULONG length)
Returns structurewith current configuration of HW.

GetChanStatus
(HANDLE hndl, PULONG status, PULONG length)
Returns current value of status register. Refer to HW manual for bit map.

RegisterChanEvent
(HANDLE hndl, HANDLE *Event, PULONG length)
Call to initialize event or clear initialized event. Use with non-DMA based interrupts. The caller creates an event with CreateEvent() and supplies the handle returned from that call as the input to this IOCTL. The driver then obtains a system pointer to the event and signals the event when a user interrupt is serviced. The user interrupt service routine waits on this event, allowing it to respond to the interrupt.

EnableChanInterrupt
(HANDLE hndl, PULONG length)
Enable the master Interrupt for the port. Must be enabled for non-DMA interrupts to pass through to Base level.

DisableChanInterrupt
(HANDLE hndl, PULONG length)
Clear Master Interrupt Enable for the port.

ForceChanInterrupt

(HANDLE hndl, PULONG length)
Call to enable the Force Interrupt bit and cause an interrupt request. ISR clears the enable.

GetChanIsrStatus

(HANDLE hndl, PAscb_CHAN_ISR_STAT isrstatus, PULONG length)
Returns status register value when interrupt was detected.   Self-clearing when read. Accumulates until read. [events setting status bits are accumulated until User has read, then all registered values cleared.]

SetDmaOffset

(HANDLE hndl, PASCB_CHAN_DMA_OFFSET AscbSetOffset, PULONG length)
Pass structure to for DMA operation. Since each port has Rx and Tx sections and the DPR is treated as a circular buffer it is necessary to initially point at the starting location.

GetDmaOffset

(HANDLE hndl, PASCB_CHAN_DMA_OFFSET AscbGetOffset, PULONG length)
Returns structure with current DMA pointer definitions.

InitChan

(HANDLE hndl, PULONG length)
Call to cause reset to HW within port.   Most settings are retained, all state-machines are reset and returned to Idle.

Additional Software Information

Write

ASCB DMA data is written to the referenced I/O port device using the write command. Writes are executed using the function WriteFile() and passing in the handle to the I/O device opened with CreateFile(), a pointer to a pre-allocated buffer containing the data to be written, an unsigned long integer that represents the size of that buffer in bytes, a pointer to an unsigned long integer to contain the number of bytes actually written, and a pointer to an optional Overlapped structure for performing asynchronous IO.

Read

ASCB DMA data is read from the referenced I/O channel device using the read command. Reads are executed using the function ReadFile() and passing in the handle to the I/O channel device opened with CreateFile(), a pointer to a pre-allocated buffer that will contain the data read, an unsigned long integer that represents the size of that buffer in bytes, a pointer to an unsigned long integer to contain the number of bytes actually read, and a pointer to an optional Overlapped structure for performing asynchronous IO.

Warranty and Repair

Please refer to the warranty page on our website for the warranty and options that are currently offered.

www.dyneng.com/warranty

Service Policy

Before returning a product for repair, verify to the best of your ability, that the suspected unit is as fault. Then call the Dynamic Engineering Customer Service Department for a Return Material Authorization (RMA) number. Carefully package the product, in the original packaging if possible, and ship prepaid and insured with the RMA number clearly written on the outside of the package. Include a return address and the telephone number of a technical contact. For out-of-warranty repairs, a purchase order for repair charges must accompany the return. Dynamic Engineering will not be responsible for damages due to improper packaging of returned items. For service on Dynamic Engineering products not purchased directly from Dynamic Engineering, contact your reseller. Products returned to Dynamic Engineering for repair by anyone other than the original customer will be treated as out-of-warranty.

Out-of-Warranty Repairs

Out-of-warranty repairs will be billed on a material and labor basis. Customer approval will be obtained before repairing any item if the repair charges will exceed one half of the list price for one of that kind of unit. Return transportation and insurance will be billed as part of the repair in addition to the minimum RMA charge.

Contact:

Customer Service Department
Dynamic Engineering
150 DuBois St. Suite B&C
Santa Cruz, CA 95060
(831) 457-8891
support@dyneng.com

Glossary

Baud	Used as the bit period when talking about UARTs; Not strictly correct, but is the common usage when talking about UARTs.
CardID	Unique number assigned to a design to distinguish between all designs of a particular vendor
CFM	Cubic feet per minute
FIFO	First In First Out memory
Flash	Non-volatile memory used on Dynamic Engineering boards to store FPGA configurations or BIOS
JTAG	Joint Test Action Group – a standard used to control serial data transfer for test and programming operations.
LFM	Linear feet per minute
LVDS	Low Voltage Differential Signaling
MUX	Multiplexor – multiple signals multiplexed to one with a selection mechanism to control which path is active.
Packed	When UART characters are always sent/received in groups of four, allowing full use of host bus/FIFO bandwidth.
Packet	Group of characters transferred. When the characteristics of the group of characters is known, the data can be stored in packets and transferred as such; the system is optimized as a result. Any number of characters can be transferred.
PCI	Peripheral Component Interconnect – parallel bus from host to this device
PIM	PMC Interface Module (PIM). Provides rear I/O in cPCI systems. Mounts to PIM Carrier
PIM Carrier	PIM Mounting Device. Mounts on rear of cPCI backplane.
PMC	PCI Mezzanine Card – establishes common connectors, connections, size and other mechanical features.
TAP	Test Access Port – basically a multi-state port that can be controlled with JTAG [TMS, TDI, TDO, TCK]. The TAP States are the states in the State Machine that are controlled by the commands received over the JTAG link.
TCK	Test Clock provides synchronization for the TDI, TDO, and TMS signals

TDI	Test Data in – this serial line provides the data input to the device controlled by the TMS commands. For example, the data to program the FLASH comes on the TDI line while the commands to the state machine to move through the necessary states comes over TMS. Rising edge of TCK valid.
TDO	Test Data Out is the shifted data out. Valid on the falling edge of the TCK. Not all states output data.
TMS	Test Mode State – this serial line provides the state switching controls. ‘1’ indicates to move to the next state, ‘0’ means stay put in cases where delays can happen; otherwise, 0,2 are used to choose which branch to take. Due to the complexity of state manipulation, the instructions are usually precompiled. Rising edge of TCK valid.
UART	Universal Asynchronous Receiver Transmitter. Common serialized data transfer with start bit, stop bit, optional parity, optional 7/8 bit data. Can be over any electrical interface. RS232 and RS422 are most common.
Unpacked	When UART characters are sent on an unknown basis requiring single character storage and transfer over the host bus
VendorID	Manufacturers number for PCI/PCIe boards. DCBA is Dynamic Engineering’s VendorID
VME	Versa Module European
VPX	Family of standards based on the VITA 46.0
XMC	Switched mezzanine card (PMC with PCIe)
ASCB	Avionics Standard Communications Bus