ARINC 429 Multi-I/O Cards Software Development Kit (SDK)

Software Manual

Model: DD-42992SX

Providing the framework for developing applications for DDC's ARINC 429 Multi-IO series of devices with minimal development time, the DD-42992 Software Development Kit is written so all low-level access to the Multi-I/O device is performed through a set of driver modules. The library allows users to develop software for the card without detailed knowledge of the card's architecture.

Operating Systems

- Windows XP/Vista/7
- VxWorks 6.x for Intel x86 and PPC
- Linux Kernel 2.6.x and 3.x
- Integrity 5.07/Multi 4.23

Custom Design Capability - DDC can customize designs for all cards, ranging from simple modifications of standard products to fully customized solutions for commercial, military, aerospace, and industrial applications.

For more information: www.ddc-web.com/DD-42992SX

DDC's Data Networking Solutions

MIL-STD-1553 | ARINC 429 | Fibre Channel | AFDX®/ARINC 664

As the leading global supplier of data bus components, cards, and software solutions for the military, commercial, and aerospace markets, DDC's data bus networking solutions encompass the full range of data interface protocols to support the real-time processing demands of field-critical data networking between military vehicles, systems, and subsystems. These "products, along with our traditional MIL-STD-1553 solutions, represent a wide and flexible array of performance and cost requirements, enabling DDC to support multi-generational programs.

DDC has developed its line of high-speed Fibre Channel and Extended 1553 products to support the real-time processing of field-critical data networking netween sensors, compute notes, data storage displays, and weapons for air, sea, and ground military vehicles.

Whether employed in increased bandwidth, high-speed serial communications, or traditional avionics and ground support applications, DDC's data solutions fulfill the expanse of military requirements including reliability, determinism, low CPU utilization, real-time performance, and ruggedness within harsh environments. Our use of in-house intellectual property ensures superior mutli-generational support, independent of the life cycles of commercial devices. Moreover, we maintain software compatibility between product generations to protect our customers' investments in software development, system testing, and end-product qualification.

MIL-STD-1553

DDC provides an assortment of quality MIL-STD-1553 commercial, military, and COTS grade cards and components to meet your data conversion and data interface needs. DDC supplies MIL-STD-1553 board level products in a variety of form factors including AMC, USB, PCI, cPCI, PCI-104, PCI-Express PCMCIA, PMC, XMC, PC/104, PC/104-Plus, VME/VXI, and ISAbus cards. Our 1553 data bus board solutions are integral elements of military, aerospace, and industrial applications. Our extensive line of military and space grade components provide MIL-STD-1553 interface solutions for microprocessors, PCI buses, and simple systems. Our 1553 data bus solutions are designed into a global network of aircraft, helicopter, and missle programs.

ARINC 429

DDC also has a wide assortment of quality ARINC-429 commercial, military, and COTS grade cards and components, which will meet your data conversion and data interface needs. DDC supplies ARINC-429 board level products in a variety of form factors including AMC, USB, PCI, PMC, PCI-104, PCI-Express, PC/104 Plus, and PCMCIA boards. DDC's ARINC 429 components ensure the accurate and reliable transfer of flight-critical data. Our 429 interfaces support data bus development, validation, and the transfer of flight-critical data aboard commercial aerospace platforms.

Fibre Channel

DDC has developed its line of high-speed Fibre Channel network access controllers and switches to support the real-time processing demands of field-critical data networking between sensors, computer nodes, data storage, displays, and weapons, for air, sea, and ground military vehicles. Fibre Channel's architecture is optimized to meet the performance,reliability, and demanding environmental requirements of embedded, real time, military applications, and designed to endure the multi-decade life cycle demands of military/aerospace programs.

AFDX®/ARINC 664

DDC provides powerful, field-proved AFDX®/ARINC 664 solutions for test, simulation, and system integration. These cards support both Airbus and Boeing AFDX protocol.

SOFTWARE MANUAL FOR ARINC 429 MULTI-I/O CARDS

For use with Windows, Linux, VxWorks, and Integrity

MN-42992SX-001

105 Wilbur Place
Bohemia, New York 11716-2426
Tel: (631) 567-5600, Fax: (631) 567-7358
World Wide Web - http://www.ddc-web.com

For Technical Support - 1-800-DDC-5757 ext. 7771
United Kingdom - Tel: +44-(0)1635-811140, Fax: +44-(0)1635-32264
France - Tel: +33-(0)1-41-16-3424, Fax: +33-(0)1-41-16-3425
Germany Tel: +49-(0)89-15 00 12-11, Fax: +49-(0)89-15 00 12-22
Japan - Tel: +81-(0)3-3814-7688, Fax: +81-(0)3-3814-7689
Asia - Tel: +65-6489-4801
India - Tel: +91 080 301 10 200

Revision	Date	Pages	Description
A	09/07	All	New Release
B	12/07	143, 64	Removal of bookmark, change in function name.
C	3/08	All	Added new Functions, Updated style
D	5/08	46	Removed IRIG_A, IRIG_C and IRIG_D from format description
E	6/08	All	New Function added
F	11/08	27, 48	Special note added.
G	12/08	Various	Replaced figure 2. Added 91U references. Added AceXtreme references. Added new functions:GetChannelLoopBack, SetChannelLoopBack, GetLoopBackMapping, SetLoopBackMapping, and GetTimeStampConfig.
H	09/09	Various	Added additional supported hardware. Updated software installation for Windows Section 4.1. Added note to EnableTimeStamp, GetMailbox, GetMailboxStatus, InitCard, & ReadMailboxIrig functions. General typographical edits.
J	05/10	All	New Template applied. InstallHandler function edited: changed: void (*Handler)(short), to: short (*Handler)(short). Changed: void fifo_rx_ISR(S16BIT Card); to: short fifo_rx_ISR(S16BIT Card); New AceXtreme™ boards added. New Section 8, enumeration types, added. Enumeration names changed throughout document, refer to Section 8 for details.
K	08/11		Added Support for BU-67211U USB Device
L	08/12		Added Support for DD-40x00F/I/T/K. Added new Xtreme-429 function calls.
M	6/14	37, 38	Updated Words per Second information
N	7/16	various	Removed note in ReadMailboxIrig referring to legacy cards. Added supported software in sections 2.1 and 2.2. Added supported hardware in section 2.3. Added sections 3.2.3 , 3.4.6. Specified file directory for Integreity 10 and 11. Updated section 4.2.2.2 to include integration between software and DDC's new line of Avionics Boards. General typographical edits.
P	9/18	8 – 11, 17, 110, 125, 144	Added Windows 10, Integrity 5.11, Linux 4.x, Vx Works v7. Updated Flash Utility to Firmware, Added functions to Table 6: dd429x_ConfigRepeater, dd429x_GetRepeaterMode, and dd429x_SetRepeaterMode

1 PREFACE

This manual uses typographical conventions to assist the reader in understanding the content. This section will define the text formatting used in the rest of the manual

1.1 Text Usage

- **BOLD**–indicates important information and table, figure, and chapter references.
- ***BOLD ITALIC***–designates DDC Part Numbers.
- `Courier New`–indicates code examples.
- <…> - indicates user-entered text or commands.

1.2 Special Handling and Cautions

DDC's Multi-IO cards use state-of-the-art components. Exercise care to prevent damage to the device by Electrical Static Discharge (ESD), physical shock, or improper power surges. Be careful to avoid electrocution.

Warnings: Adhere closely to standard ESD precautions. At a minimum, one hand should be grounded to the power supply in order to equalize the static potential. Do not store disks in environments exposed to excessive heat, magnetic fields, or radiation.

1.3 Trademarks

All trademarks are the property of their respective owners.

1.4 What is included in this manual?

This User's Manual contains instructions for installing and using the ARINC 429 Multi-IO SDK for DDC's Multi-I/O cards. The SDK provides a level of abstraction that does not require understanding of the operation of the on-board chip set.

1.5 Technical Support

In the event that problems arise beyond the scope of this manual, you can contact DDC by the following:

US Toll Free Technical Support:
1-800-DDC-5757, ext. 7771

Outside of the US Technical Support:
1-631-567-5600, ext. 7771

Fax:
1-631-567-5758 to the attention of DATA BUS Applications

DDC Website:
www.ddc-web.com/ContactUs/TechSupport.aspx

Please note that the latest revisions of Software and Documentation are available for download at DDC's Web Site, www.ddc-web.com.

2 OVERVIEW

The ***DD-42992 SDK*** provides the framework for developing applications for DDC's ARINC 429 Multi-IO series of devices with minimal development time.

This SDK is written so all low-level access to the Multi-IO device is performed through a set of driver modules. All Board Support Package-specific functions and processor-specific issues are abstracted in a way that allows easy portability of the SDK to a variety of hardware and/or software platforms by modifying these low-level routines. This software library of functions is used to configure the card for receiving and transmitting data on the bus. The library allows users to develop software for the card without detailed knowledge of the card's architecture.

2.1 Features

- Library of "C" Routines and configuration files tested on:
  - Windows XP,Vista, 7, 8, 10
  - VxWorks v6.x, v7 for Intel x86 and PPC
  - Integrity 5.11, 10, 11
  - Linux Kernels 2.6.15, 3.x, 4.x
- Documentation
- Modular, Portable, Readable Code to Reduce Software Development Time
- Sample Programs and Compiled Binary Driver Modules for Quick Startup

2.2 System Requirements

- Windows XP/Vista/7/8/10
- VxWorks v6.x, v7 for Intel x86 and PPC
- Integrity 5.11, 10, 11 / Multi 4.23, 6.1
- Linux Kernel version between 2.6.15, 3.x, 4.x

2.3 Supported Hardware

The ***DD-42992S0 SDK*** has been tested on the following:

- Intel PC Pentium 233 MHz or better running Windows
- ***DD-40100F*** – PMC
- ***DD-40100I*** – PCI
- ***DD-40100T*** – cPCI
- ***DD-40000K*** – PCIe
- ***DD-40001H*** – Mini PCIe

- ***BU-67211Ux*** – USB
- ***BU-65590/91Ux*** – USB
- ***BU-67102/103Ux*** – USB
- ***BU-65590F/Mx*** – PMC
- ***BU-67107F/Mx*** – PMC
- ***BU-67118M*** – PMC
- ***BU-67118Y/Z*** – XMC
- ***BU-67118K*** – PCI-e
- ***BU-65590ix*** – PCI
- ***BU-67107i*** – PCI
- ***BU-67107T*** – cPCI
- ***BU-65590C*** – PC/104-Plus
- ***BU-67109C*** – PC/104-Plus
- ***BU-65591C*** – PCI-104
- ***BU-67108C*** – PCI-104
- ***BU-65590A*** – AMC
- ***BU-67118A*** – AMC
- ***BU-6711xW*** – Ethernet
- ***BU-67125Wx*** – Ethernet

The ***DD-42992S1 SDK*** has been tested on the following:

- Generic Pentium / X86
- Fedora Kernels 2.6.15, 3.x, 4.x
- ***DD-40100F*** – PMC
- ***DD-40100I*** – PCI
- ***DD-40100T*** – cPCI
- ***DD-40000K*** – PCIe
- ***DD-40001H*** – Mini PCIe
- ***BU-65590F/Mx*** – PMC
- ***BU-67107F/Mx*** – PMC
- ***BU-67118M*** – PMC
- ***BU-67118Y/Z*** – XMC
- ***BU-65590/91Ux*** – USB
- ***BU-67102/3U*** - USB
- ***BU-67211Ux*** – USB
- ***BU-67118K*** – PCI-e

- ***BU-65590ix*** – PCI
- ***BU-67107i*** – PCI
- ***BU-67107T*** – cPCI
- ***BU-65590C*** – PC/104-Plus
- ***BU-67109C*** – PC/104-Plus
- ***BU-65591C*** – PCI-104
- ***BU-67108C*** – PCI-104
- ***BU-65590A*** – AMC
- ***BU-67118A*** – AMC
- ***BU-6711xW*** – Ethernet
- ***BU-67125Wx*** – Ethernet

The ***DD-42992S2 SDK*** has been tested on the following:

- Generic Pentium / X86 (pcPentium)
- PPC VxWorks 6.1 Curtiss-Wright SVME-183
- ***DD-40100F*** – PMC
- ***DD-40001H*** – Mini PCIe
- ***BU-65590C –*** PC/104-*Plus*
- ***BU-67109C*** – PC/104-Plus
- ***BU-65591C*** – PCI-104
- ***BU-67108C*** – PCI-104
- ***BU-65590F/Mx*** – PMC
- ***BU-67107F/Mx*** – PMC
- ***BU-67118M*** – PMC
- ***BU-67118Y/Z*** – XMC
- ***BU-67102/3U*** - USB
- ***BU-67211Ux*** – USB
- ***BU-6711xW*** – Ethernet
- ***BU-67125Wx*** – Ethernet

The ***DD-42992S5 SDK*** has been tested on the following:

- Green Hills Software INTEGRITY v5.0.11,v10, v11
- Curtiss-Wright (Dy4) SVME-181
- Curtiss-Wright (Dy4) SVME-183
- Freescale (NXP) P2041
- Freescale (NXP) P4080
- ***BU-67109C*** – PC/104-Plus

- ***BU-67108C*** – PCI-104
- ***BU-65590F/Mx*** – PMC
- ***BU-67107F/Mx*** – PMC
- ***BU-67118M*** – PMC
- ***BU-67118Y/Z*** – XMC
- ***DD-40100F*** – PMC
- ***DD-40001H*** – Mini PCIe
- ***BU-6711xW*** – Ethernet
- ***BU-67125Wx*** – Ethernet

Note: *The **BU-65590/91Ux**, **BU-67102/3U** and the **BU-67211U** Avionics Devices require a USB 2.0 Port which meets the USB 2.0 Specification. There will be limited performance when using the **BU-65590/91Ux** and the **BU-67102/3U** with older USB revisions.*

Note: *PCI-PMC connector supports 3.3V or 5V PCI signaling. PCI-PMC connector with +5V & ±12V supply voltage available*

3 SOFTWARE INSTALLATION

Installation and usage procedures for the ***DD-42992Sx SDK*** differ depending on the operating system on which it is installed. This manual covers installation for the Windows, Linux, VxWorks OS and Integrity.

The ***DD-42992Sx SDK*** is can be installed from the Data Bus Software CD or downloaded from the DDC website at www.ddc-web.com. It is recommended that the user check the DDC website for the latest version of software, and download the newer version if necessary.

Table 1. ARINC 429 Multi-I/O SDK Part Number Descriptions

Part Number	Description	Requirements
DD-42992S0	Software Development Kit and Device Drivers for Windows	Windows XP/Vista/7/8/10
DD-42992S1	Software Development Kit and Device Drivers for Linux	Linux Kernel version 3.x, 4.x
DD-42992S2	Software Development Kit and Device Drivers for VxWorks	WindRiver VxWorks versions 6.x, 7
DD-42992S5	Software Development Kit and Device Drivers for Integrity	Green Hills Integrity 5.x, 10.x, 11.x

3.1 DD-42992S0 ARINC 429 Multi-IO SDK – Windows Installation

1. Insert the Data Bus software CD if installing the ***DD-42992S0*** from the Data Bus Software CD. If you have downloaded the ***DD-42992S0*** from the website you can skip to step 5. Once the CD is inserted allow the CD to auto start.

Figure 1. Data Bus Software CD Menu Screen

2. Choose your product from the list on the left (i.e., ***BU-65590Ux***).
3. Choose the ***DD-42992S0*** from the Choose Software Box.
4. Click on the install software icon to start Installation Program.
5. Follow the instructions given by the installation program to proceed with the installation. See Figure 2 for a sample installation screen during the installation of your software.

Figure 2. Windows DD-42992S0 Installation Screen

Once the software is installed the **DDC Card Manager** will be launched. This tool is used to view the status of installed cards, check current software driver revisions, and map an inserted card to a logical device number. In order to access an installed card you **must** assign it a logical device number.

3.1.1 Logical Device Numbers - Windows

The ***DD-42992 ARINC Multi-IO*** SDK uses Logical Device Numbers to access DDC hardware. A Logical Device Number (LDN) is a unique identifier referring to a particular ARINC 429 board. Most SDK functions will require a target LDN as the first parameter. Depending on your Operating System, the Logical Device Numbers will be auto-assigned or will require some user interaction. See the Logical Device Numbers in Windows section for more information.

3.1.1.1 Logical Device Numbers in Windows

Logical Device Numbers in Windows are assigned using the DDC Card Manager Applet, found in the Windows Control Panel. The Card Manager will display all connected hardware and will allow a unique LDN to be assigned to each device. To assign an LDN, click the desired entry and select a LDN from the drop down list.

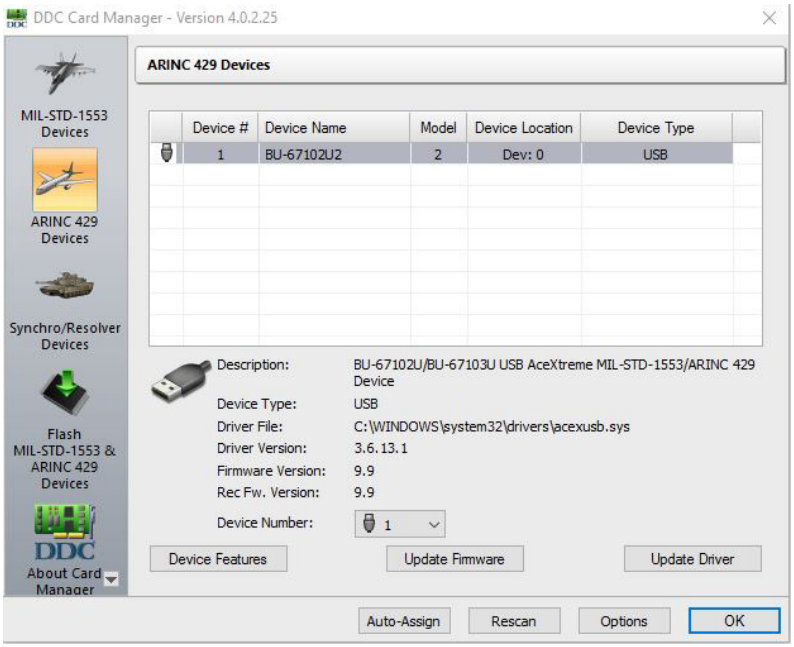


Figure 3. DDC Card Manager

Version 3.1.1 and later of the DDC Card Manager has the ability to flash all ***E²MA*** and ***AceXtreme***® devices. The Flash MIL-STD-1553 & ARINC 429 Devices button can be used to flash your device. For more information on flashing your device please review the flash instructions located in the FlashUtilty directory of your installed package.

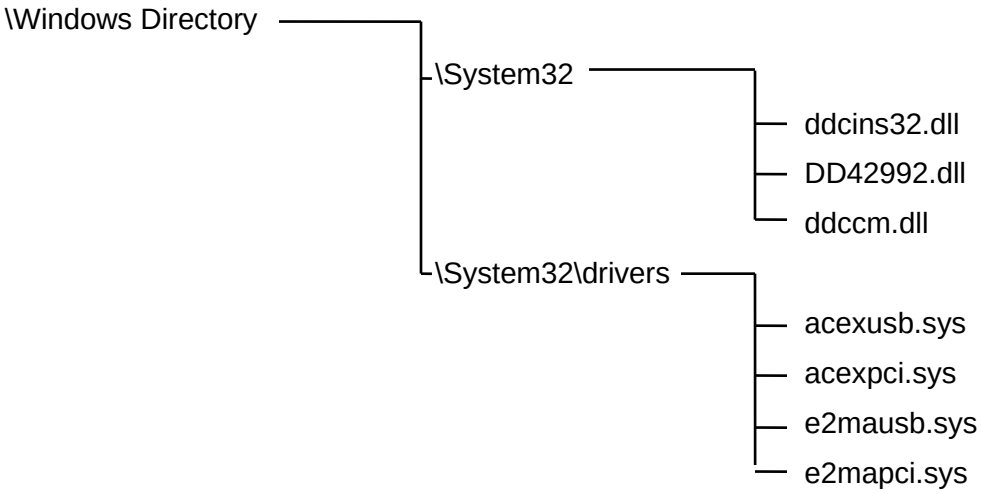
3.1.2 Files in DD-42992S0 ARINC 429 Multi-IO SDK

By default the toolbox will be installed to the "DDC\ DD42992SSKvXYZ". Included subdirectories are:

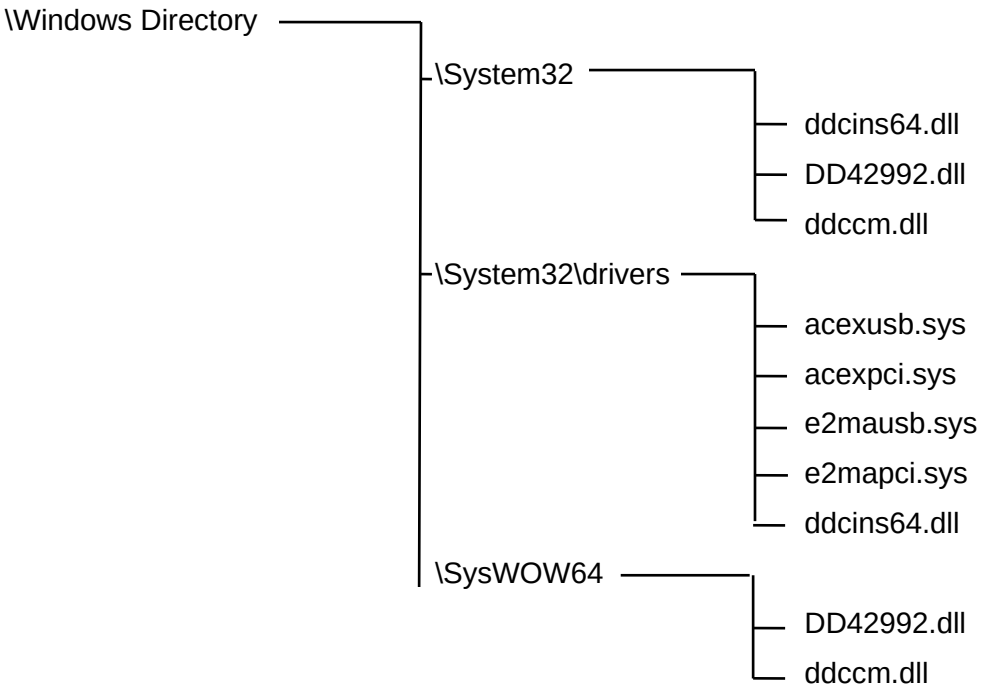
- \Documentation
- \Driver
- \Firmware
- \Include
- \Lib
- \Samples

Additionally the following files will be added to your Windows system directory as follows:

For 32-bit systems:



For 64-bit systems:



Directory of \Documentation contains the following files:

License.txt	License information for the ***DD-42992S0*** Multi-IO SDK.
ReleaseNotes.txt	Release information for each version of the ***DD-42992S0*** Multi-IO SDK.
Readme.txt	This is the readme file that contains some general information about the software.

Directory of \Drivers contains the following files:

acexusb.sys	Driver file for ***BU-67102/103Ux*** and ***BU-67211U***
acexpci.sys	Driver file for the ***BU-67107F/M/i/T***, ***BU-67108/9C, BU-67118K/M/Y/Z, DD-40001H, DD-40000K, and DD-40100F/i/T***
e2mausb.sys	Driver file for ***BU-65590U***

Directory of \Firmware contains the following files:

Flash Utility Manual\	Directory contains a .pdf with instructions on how to flash your ***E²MA*** or ***AceXtreme***® Device.

The firmware for all ***AceXtreme***® and ***E²MA*** devices.

Directory of \Include contains:

arinc_717.h	Contains prototypes for various ARINC 717 routines.
arinc717.h	
arinc717_tst.h	
arinc717r.h	
arinc717x.h	
can.h	Contains protoytpes for various for CAN routines.
can_tst.h	
canr.h	
canx.h	
cardinit.h	Contains the prototypes for card initialization, shutdown, and version information and interrupt facility configuration.
cardinit_tst.h	
cardinitr.h	
cardinitx.h	
control.h	Prototypes for various card feature control routines, including Tx/Rx group reset, loopback control, card level ARINC 429 Multi-IO SDK bit formatting, and discrete line control.
control_tst.h	
controlr.h	
controlx.h	
dd429.h	Contains common definitions used between the 429 library and driver.

dd429errorlist.h	Constants for error list.
ddccommon.h	Common definitions and types used in the 69092 and 42992 libraries.
ddcioctl.h	IOCTL definitions used between the 429 library and driver.
ddcos.h	OS specific definitions used between the 429 library and driver.
ddcsockmsg.h	DDC socket interface for remote devices. Contains socket errors as well.
device429.h	
device429_tst.h	
device429r.h	
device429x.h	
errors.h	Prototypes for the error display routine associated constants.
hardware.h	Prototypes for SDK level device register and memory access routines, and register, memory, and mask name associations.
hardware_tst.h	
hardwarex.h	
irig.h	Prototypes for card level IRIG configuration functions.
irig_tst.h	
irigr.h	
irigx.h	
receive.h	Prototypes for ARINC 429 Multi-IO SDK receive configuration, FIFO and mailbox reception routines.
receive_tst.h	
receiver.h	
receivex.h	

serial.h	Prototypes for card level Serial IO configuration, UART configuration control, and individual UART control.
serial_tst.h	
serialr.h	
serialx.h	
std429.h	Top Level Multi-IO ARINC 429 /Serial/Discrete header file to be included in any end user application program utilizing a DDC Multi-IO device.
tester.h	Prototypes for ARINC 429 tester functions.
tester_tst.h	
testerr.h	
testerx.h	
transmit.h	Prototypes for ARINC 429 Multi-IO SDK transmit configuration, FIFO and scheduled message transmissions.
transmit_tst.h	
transmitr.h	
transmitx.h	

Directory of \Lib compiled DLL's and link libraries:

\Win32	A directory that contains 32-bit Debug and Release versions of DD42992.lib & DD42992.dll
\x64	A directory that contains 64-bit Debug and Release versions of DD42992.lib & DD42992.dll

Directory of \samples contains the source code for the following samples:

arinc717Loopback	This example demonstrates how to configure and use the ARINC 717 channels.
avionic	This example demonstrates how to configure, set, and read the avionics lines.
canLoopTest	This example demonstrates how to configure, and use the CanBUS channels.
discrete	This example demonstrates how to configure, set, and read the discretes.
discrete_all	This example demonstrates how to configure, set, and read the discretes using the functions to access all the bits at once.
fifo_loopback	Demonstrates how to transmit and receive data on the RX channels with the FIFO mode on all channels.
fifo_rx	This example demonstrates how to configure and receive data on the RX channels.
fifo_rx_ch	This example demonstrates how to configure and receive data on the RX channels. This demo allows the user to configure different speeds and parity settings for individual channels.
fifo_rx_hbuf	This example demonstrates how to configure and receive data on the RX channels using the Host Buffer.
fifo_tx	Demonstrates how to configure and transmit low speed data on the TX channels.
mailbox_rx	This example demonstrates how to configure and receive data using mailbox mode with filters.
repeater	This example demonstrates how to configure and send data pollution.

rs232_lookback	This example demonstrates how to configure, transmit, and receive data on the RS232 port, as well as test RTS/CTS.
rs422_lookback	This example demonstrates how to configure, transmit, and receive data on the RS422 port.
sched_tx	Demonstrates how to configure and transmit low speed scheduled data on the TX channels.
sdlc_hdlc_rx	This example demonstrates high speed serial reception on the BU-67118x cards.
sdlc_hdlc_tx	This example demonstrates high speed serial transmission on the BU-67118x cards.
serial_rx	This example demonstrates how to configure and received data on the RS232 port.
serial_tx	This example demonstrates how to configure and transmit data on the RS232 port.
serial_tx_rx_loopback	This example combines sending and receiving on a serial port.
tx_frame	This example demonstrates how to transmit messages using minor/major frames.
voltage_mon	This example demonstrates how to monitor the voltage on the receive channels.

3.2 DD-42992S1 ARINC 429 MULTI-IO SDK – Linux Installation

The ***DD-42992S1*** software package is distributed in compressed form and is to be unzipped on the end user's host system. The package contains device drivers with full source for all supported devices, the run time library, a number of example programs demonstrating various SDK features, and a card FPGA reprogramming utility.

3.2.1 SDK Installation and Usage

Follow the instructions that come with the package in the Install.txt file in order to install the library. The instructions will guide you through extracting the compressed tar file and setting the correct permissions on the install directory.

Once the files have been installed the ReadMe.txt file will be placed in the newly created 42992 directory. This file will contain the instructions on building the device driver for your device and the steps required to loading the driver.

Also described in the Readme.txt is building the Card Manager and configuring a device number for your device. The Readme.txt will also describe how to build the flash utility and the samples that come with the ARINC 429 Multi-IO SDK.

3.2.2 Logical Device Numbers - Linux

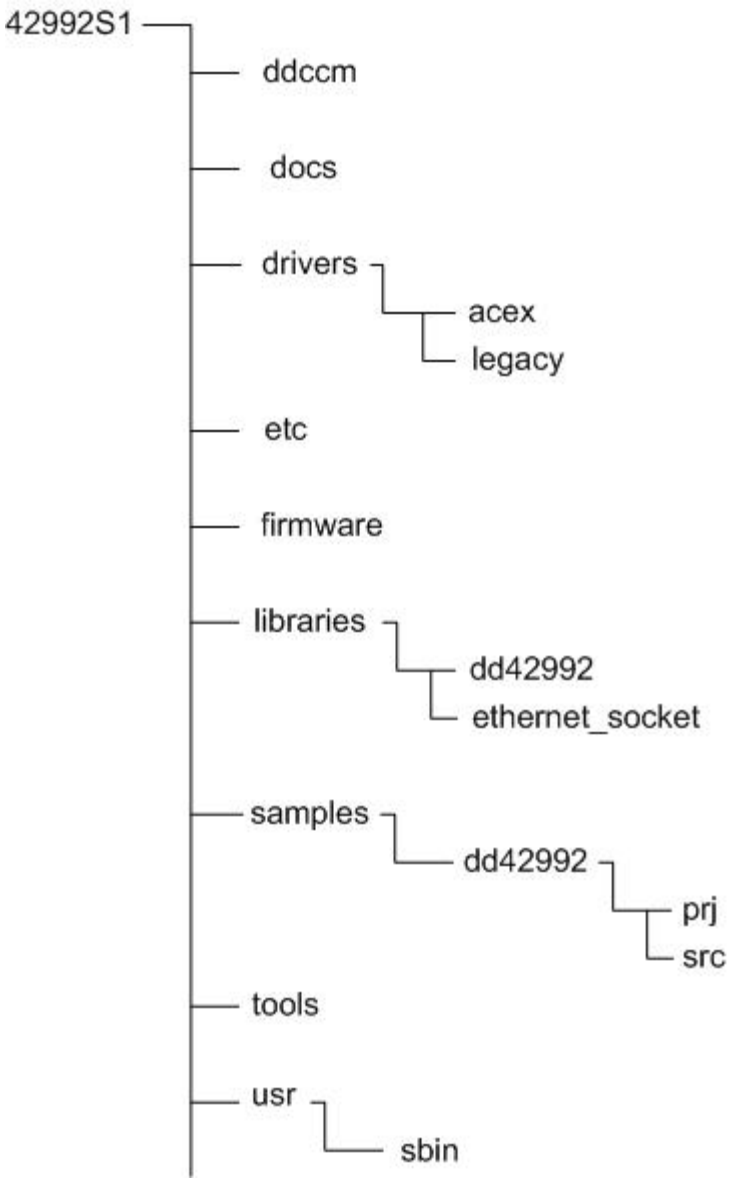
The ***DD-42992 ARINC Multi-IO*** SDK uses Logical Device Numbers to access DDC hardware. A Logical Device Number (LDN) is a unique identifier referring to a particular ARINC 429 board. Most SDK functions will require a target LDN as the first parameter. Depending on your Operating System, the Logical Device Numbers will be auto-assigned or will require some user interaction. See the Logical Device Numbers in Linux Section for more information.

3.2.2.1 Logical Device Numbers in Linux

Logical Device Numbers in Linux are assigned using the DDCCM Text-Based Card Manager (**./ddccm**), found in the **/usr/bin** folder. The Card Manager will display all connected hardware and will allow a unique LDN to be assigned to each channel. Use the menu-driven interface to assign LDN's to all detected channels.

3.2.2.2 Files in SDK for Linux

After installation is complete you will see the following directory structure.


42992S1
ddccm
docs
drivers
acex
legacy
etc
firmware
libraries
dd42992
ethernet_socket
samples
dd42992
prj
src
tools
usr
sbin

Directory of /ddccm contains:

A /prj and a /src folder.

/prj contains:

Makefile	The make file to build the sample applications.

/src contains:

ddccm.c

ddccm.h

ddccm_diag.c

ddccm_flash.c

ddccm_flash.h

ddccm_os.h

ddccm_os_flash.c

ddccm_os_flash.h

strtable.h

Directory of /docs contains:

Install .txt

License.txt

Release-notes-429-sdk.txt

Directory of /drivers contains:

An /acex and /legacy folder.

/acex contains:

/prj	Provides make file to build and/or load driver
/src	Contains necessary files to build driver

/tools Includes the load scripts to load or unload the driver.

/legacy contains:

/acexusb Provides necessary files to compile the aceXtreme USB drivers.

/e2mausb Provides necessary files to compile the e²ma USB driver.

dd429.h

dd429errorlist.h

ddc.rules

ddc1553.h

ddccommon.h

ddcdriver.h

ddclinux.c

ddclinux.h

ddcregmap.h

Directory of /etc contains:

A /udev folder which then contains a /rules.d directory.

/rules.d contains:

ddc.rules

Directory of /firmware contains:

FLASH_UTILITY.pdf Provides instructions to flashing firmware of a device.

Various .bin files Firmware files for DDC devices.

/dd42992 Contains all files for the various 429 modes and features.

\src

arinc717.h Contains prototypes for various ARINC 717 routines.

arinc717_tst.h

arinc717r.h

arinc717x.h

can.h Contains prototypes for various CAN routines.

can_tst.h

canr.h

canx.h

cardinit.h Contains the prototypes for card initialization, shutdown, and version information and interrupt facility configuration.

cardinit_tst.h

cardinitr.h

cardinitx.h

control.h Prototypes for various card feature control routines, including Tx/Rx group reset, loopback control, card level ARINC 429 Multi-IO SDK bit formatting, and discrete line control.

control_tst.h

controlr.h

controlx.h

device429.h Contains prototypes for operating system specific card initialization,

shutdown, and generalized system calls.

device429_tst.h

device429r.h

device429x.h

errors.h	Prototypes for the error display routine associated constants.
hardware.h	Prototypes for SDK level device register and memory access routines, and register, memory, and mask name associations.
hardware_tst.h	
hardwarex.h	
irig.h	Prototypes for card level IRIG configuration functions.
irig_tst.h	
irigr.h	
irigx.h	
receive.h	Prototypes for ARINC 429 Multi-IO SDK receive configuration, FIFO and mailbox reception routines.
receive_tst.h	
receiver.h	
receivex.h	
serial.h	Prototypes for card level Serial IO configuration, UART configuration control, and individual UART control.
serial_tst.h	
serialr.h	
serialx.h	
std429.h	Top Level Multi-IO ARINC 429 /Serial/Discrete header file to be included in any end user application program utilizing a DDC Multi-IO device.
tester.h	Contains prototypes for ARINC 429 tester functions.
tester_tst.h	
testerr.h	
testerx.h	

transmit.h Prototypes for ARINC 429 Multi-IO SDK transmit configuration, FIFO and scheduled message transmissions.

transmit_tst.h

transmitr.h

transmitx.h

/ethernet_socket Contains files for DDC's remote access mode.

Directory of /samples contains:

/prj and /src directories.

/prj contains:

Makefile Make file to build the sample applications.

/src contains the source code for the following sample applications:

arinc717Loopback This example demonstrates how to configure and use the ARINC 717 channels.

avionic This example demonstrates how to configure, set, and read the avionics lines.

canLoopTest This example demonstrates how to configure, and use the CanBUS channels.

discrete This example demonstrates how to configure, set, and read the discretes.

discrete_all This example demonstrates how to configure, set, and read the discretes using the functions to access all the bits at once.

fifo_loopback Demonstrates how to transmit and receive data on the RX channels
with the FIFO mode on all channels.

fifo_rx This example demonstrates how to configure and receive data on
the RX channels.

fifo_rx_ch	This example demonstrates how to configure and receive data on the RX channels. This demo allows the user to configure different speeds and parity settings for individual channels.
fifo_rx_hbuf	This example demonstrates how to configure and receive data on the RX channels using the Host Buffer.
fifo_tx	Demonstrates how to configure and transmit low speed data on the TX channels.
io_interrupt	This example demonstrates AIO interrupts.
mailbox_rx	This example demonstrates how to configure and receive data using mailbox mode with filters.
repeater	This example demonstrates how to configure and send data pollution.
rs232_lookback	This example demonstrates how to configure, transmit, and receive data on the RS232 port, as well as test RTS/CTS.
rs422_lookback	This example demonstrates how to configure, transmit, and receive data on the RS422 port.
sched_tx	Demonstrates how to configure and transmit low speed scheduled data on the TX channels.
sdlc_hdlc_rx	This example demonstrates high speed serial reception on the BU-67118x cards.
sdlc_hdlc_tx	This example demonstrates high speed serial transmission on the BU-67118x cards.
sdlc_hdlc_loopback	This exampls demonstrates the combination of reception/transmission on the BU-67118x cards.
serial_rx	This example demonstrates how to configure and received data on the RS232 port.
serial_tx	This example demonstrates how to configure and transmit data on the RS232 port.

serial_tx_rx_loopback	This example combines sending and receiving on a serial port.
tx_frame	This example demonstrates how to transmit messages using minor/major frames.
voltage_mon	This example demonstrates how to monitor the voltage on the receive channels.

Directory of /tools contains:

/tx_bc_init_gen directory

.setup-env.sh

drivers.ddc

flash.ddc

install-dd42992.sh

uninstall-dd42992.sh

3.3 DD-42992S2 ARINC 429 Multi-IO SDK – VxWorks Installation

The ***DD-42992S2*** software package is distributed in compressed form and is to be unzipped on the end user's host system. The package contains device drivers for all supported devices, the Software Development Kit (SDK) source code, a number of example programs demonstrating various SDK features, and a card FPGA reprogramming utility.

3.3.1 SDK Installation and Usage

Extract the ***DD-42992S2*** software package to the 42992S0 directory off your root drive. In order to utilize the SDK, a Workbench project needs to be generated. This involves creating a new downloadable kernel module for the desired board support package (BSP). The overall workspace file and the SDK project file should reside in the '42992S2' installation base directory.

Complete SDK source code is supplied in order to provide the greatest overall flexibility associated with building application programs utilizing the ARINC 429 Multi-IO SDK.

Once the project is created, add all of the SDK source files found in the 'source' directory to the project and build. The SDK should now be ready to download to the target system.

3.3.2 Logical Device Numbers - VxWorks

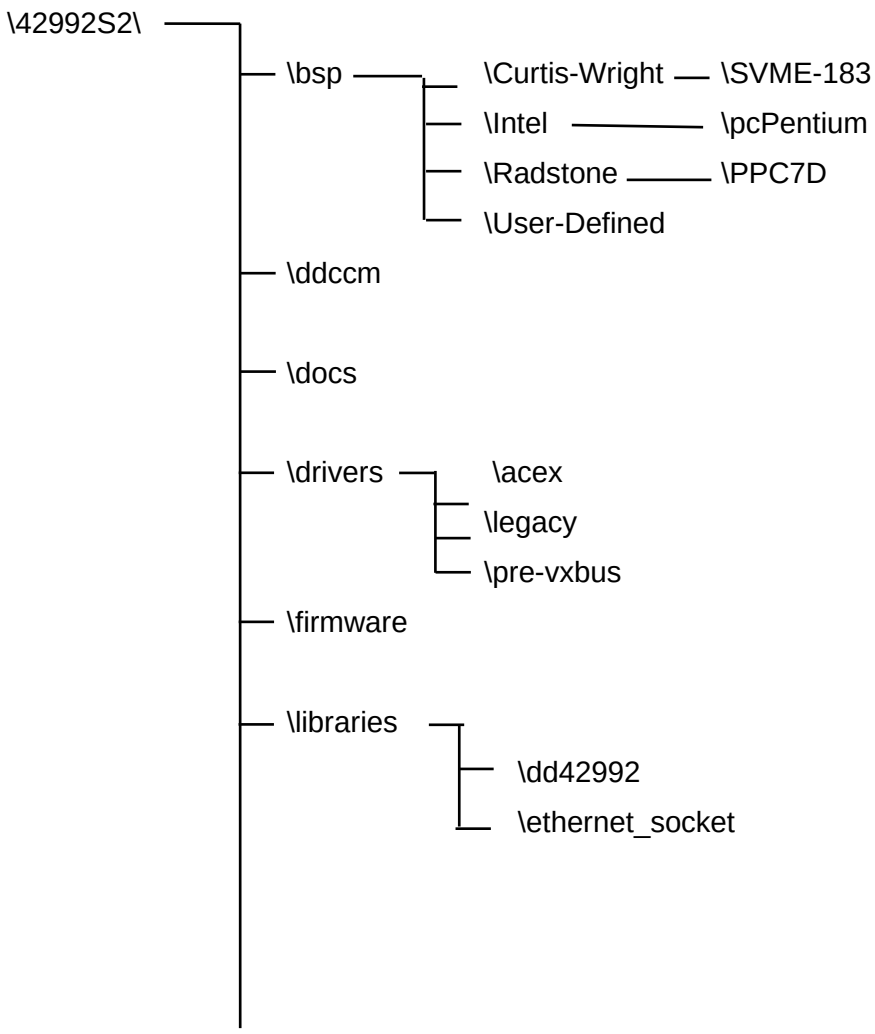
The ***DD-42992 ARINC Multi-IO*** SDK uses Logical Device Numbers to access DDC hardware. A Logical Device Number (LDN) is a unique identifier referring to a particular ARINC 429 board. Most SDK functions will require a target LDN as the first parameter. Depending on your Operating System, the Logical Device Numbers will be auto-assigned or will require some user interaction. See the Logical Device Numbers inVxWorks section for more information.

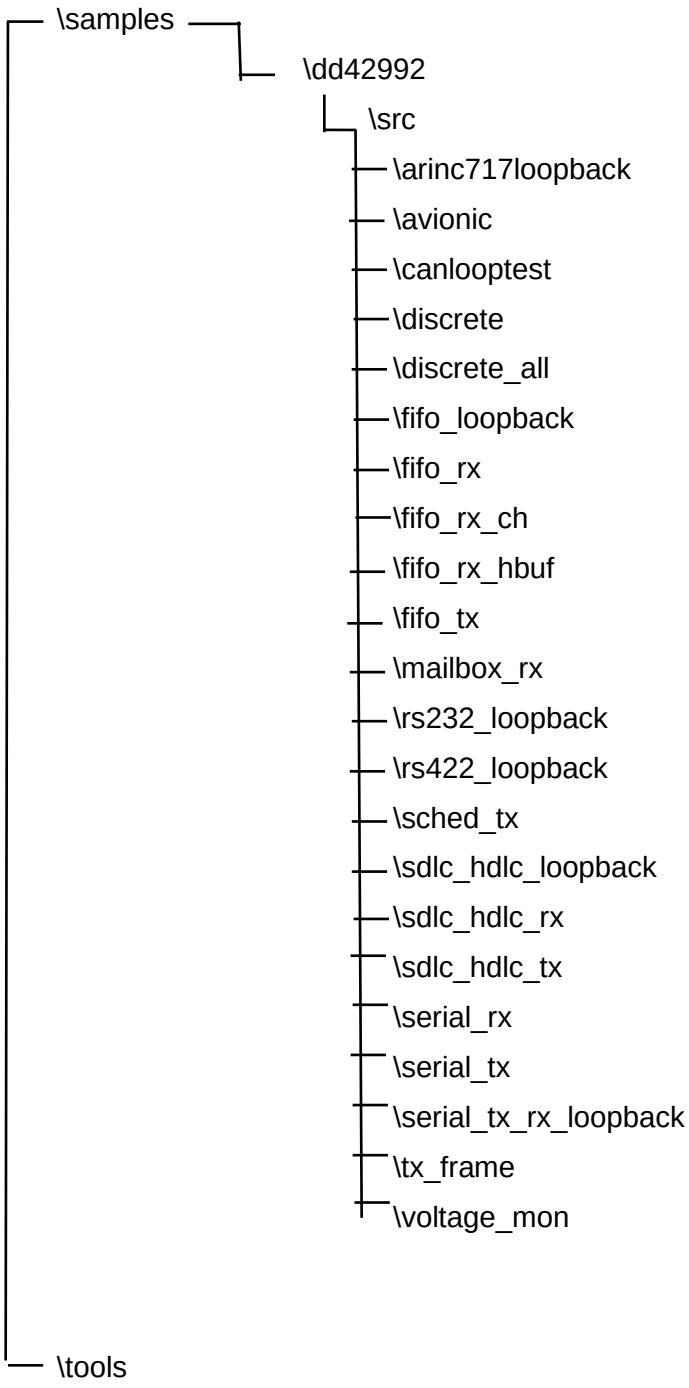
3.3.2.1 Logical Device Numbers inVxWorks

Logical Device Numbers in VxWorks are auto-assigned by calling the **ddcFind429Devices()** function. This function will call the necessary DDC VxWorks API calls to setup and detected hardware. After executing this function, a message will be displayed to the terminal informing the user of any found devices and their assigned LDN's.

3.3.3 Files in SDK for VxWorks

After extracting the zip file the following directory structure will be created:


\42992S2\
\bsp
\Curtis-Wright
\SVME-183
\Intel
\pcPentium
\Radstone
\PPC7D
\User-Defined
\ddccm
\docs
\drivers
\acex
\legacy
\pre-vxbus
\firmware
\libraries
\dd42992
\ethernet_socket


\samples
\dd42992
\src
\arinc717loopback
\avionic
\canlooptest
\discrete
\discrete_all
\fifo_loopback
\fifo_rx
\fifo_rx_ch
\fifo_rx_hbuf
\fifo_tx
\mailbox_rx
\rs232_loopback
\rs422_loopback
\sched_tx
\sdlc_hdlc_loopback
\sdlc_hdlc_rx
\sdlc_hdlc_tx
\serial_rx
\serial_tx
\serial_tx_rx_loopback
\tx_frame
\voltage_mon
\tools

Directory of \DD42992S2\bsp\ contains:

Curtiss-Wright	Ddc429BspConfig.c for SVME-183
Intel	Ddc429BspConfig.c for pcPentium
Radstone	Ddc429BspConfig.c for PPC7D
User Defined	User Defined Ddc429BspConfig.c

Directory of \DD42992S2\ddccm\src\ contains:

ddccm.c

ddccm.h

ddccm_diag.c

ddccm_flash.c

ddccm_flash.h

ddccm_os.h

ddccm_os_flash.c

ddccm_os_flash.h

strtable.h

Directory of \DD42992S2\docs contains:

license.txt

pre_vxbus_driver_readme.txt

release-notes-sdk.txt

vxbus_driver_readme.txt

Directory of \drivers contains:

An \acex, \legacy, \pre-vxbus folder.

\acex contains:

\prj	Provides make file to build and/or load driver
\src	Contains necessary files to build driver
\tools	Includes the load scripts to load or unload the driver.

\legacy contains:

\ddcacexusb Provides necessary files to compile the aceXtreme USB drivers.

\ddcemapci

dd429.h

dd429errorlist.h

ddcBspConfig.h

ddccommon.h

ddcPciConfig.h

ddcvxworks.h

\pre-vxbus contains:

ddcConfig.h

sysDdc.c

sysDdcEmaPci.c

Directory of \firmware contains:

FLASH_UTILITY.pdf Provides instructions to flashing firmware of a device.

Various .bin files Firmware files for DDC devices.

Directory of \libraries contains:

\dd42992 Contains all files for the various 429 modes and features.

\src

arinc717.h Contains prototypes for various ARINC 717 routines.

arinc717_tst.h

arinc717r.h

arinc717x.h	
can.h	Contains prototypes for various CAN routines.
can_tst.h	
canr.h	
canx.h	
cardinit.h	Contains the prototypes for card initialization, shutdown, and version information and interrupt facility configuration.
cardinit_tst.h	
cardinitr.h	
cardinitx.h	
control.h	Prototypes for various card feature control routines, including Tx/Rx group reset, loopback control, card level ARINC 429 Multi-IO SDK bit formatting, and discrete line control.
control_tst.h	
controlr.h	
controlx.h	
device429.h	Contains prototypes for operating system specific card initialization, shutdown, and generalized system calls.
device429_tst.h	
device429r.h	
device429x.h	
errors.h	Prototypes for the error display routine associated constants.
hardware.h	Prototypes for SDK level device register and memory access routines, and register, memory, and mask name associations.
hardware_tst.h	
hardwarex.h	

irig.h	Prototypes for card level IRIG configuration functions.
irig_tst.h	
irigr.h	
irigx.h	
receive.h	Prototypes for ARINC 429 Multi-IO SDK receive configuration, FIFO and mailbox reception routines.
receive_tst.h	
receiver.h	
receivex.h	
serial.h	Prototypes for card level Serial IO configuration, UART configuration control, and individual UART control.
serial_tst.h	
serialr.h	
serialx.h	
std429.h	Top Level Multi-IO ARINC 429 /Serial/Discrete header file to be included in any end user application program utilizing a DDC Multi-IO device.
tester.h	Contains prototypes for ARINC 429 tester functions.
tester_tst.h	
testerr.h	
testerx.h	
transmit.h	Prototypes for ARINC 429 Multi-IO SDK transmit configuration, FIFO and scheduled message transmissions.
transmit_tst.h	
transmitr.h	
transmitx.h	

\ethernet_socket Contains files for DDC's remote access mode.

Directory of \samples contains the source code for the following samples:

arinc717Loopback	This example demonstrates how to configure and use the ARINC 717 channels.
avionic	This example demonstrates how to configure, set, and read the avionics lines.
canLoopTest	This example demonstrates how to configure, and use the CanBUS channels.
discrete	This example demonstrates how to configure, set, and read the discretes.
discrete_all	This example demonstrates how to configure, set, and read the discretes using the functions to access all the bits at once.
fifo_loopback	Demonstrates how to transmit and receive data on the RX channels with the FIFO mode on all channels.
fifo_rx	This example demonstrates how to configure and receive data on the RX channels.
fifo_rx_ch	This example demonstrates how to configure and receive data on the RX channels. This demo allows the user to configure different speeds and parity settings for individual channels.
fifo_rx_hbuf	This example demonstrates how to configure and receive data on the RX channels using the Host Buffer.
fifo_tx	Demonstrates how to configure and transmit low speed data on the TX channels.
mailbox_rx	This example demonstrates how to configure and receive data using mailbox mode with filters.
rs232_lookback	This example demonstrates how to configure, transmit, and receive data on the RS232 port, as well as test RTS/CTS.

rs422_lookback This example demonstrates how to configure, transmit, and receive
data on the RS422 port.

sched_tx Demonstrates how to configure and transmit low speed scheduled
data on the TX channels.

sdlc_hdlc_rx This example demonstrates high speed serial reception on the BU-67118x cards.

sdlc_hdlc_tx This example demonstrates high speed serial transmission on the BU-67118x cards.

sdlc_hdlc_loopback This exampls demonstrates the combination of reception/transmission on the BU-67118x cards.

serial_rx This example demonstrates how to configure and received data on
the RS232 port.

serial_tx This example demonstrates how to configure and transmit data on
the RS232 port.

serial_tx_rx_loopback This example combines sending and receiving on a serial port.

tx_frame This example demonstrates how to transmit messages using minor/major frames.

voltage_mon This example demonstrates how to monitor the voltage on the receive channels.

Directory of \tools contains:

\tx_inhibit_bc_disable

flash.ddc

install_drivers.cmd

3.3.4 Building the Library

Complete Library source code is supplied in order to provide the greatest overall flexibility associated with building application programs utilizing the ARINC 429 Multi-IO SDK. View the readme.txt file on how to build a bootable image and the drivers for the Multi-IO cards.

3.4 DD-42992S5 ARINC 429 Multi-IO SDK – Integrity Installation

The following section contains information specific to the Multi-IO ARINC 429 Library release for the Green Hills Software Integrity Operating System (***DD-42992S5***). For additional information please view the “ReleaseNotes.txt” file included with the release.

3.4.1 Interrupt Processing

Interrupt servicing is accomplished by spawning a secondary task from the main user application task that will block until an interrupt has been issued. Upon blocking, a library level interrupt service routine will be called and an optional user specified routine may be called.

3.4.2 Package / Controller Adaptation

The distribution is organized such that a separate source module exists for each specific controller board type for the purpose of maintaining a generic card interface layer. All controller specific calls will exist in these modules. The appropriate driver module must be compiled into the kernel.

3.4.3 SDK Installation and Usage

To install the ***DD-42992S5*** on your system for INTEGRITY 5, perform the following steps:

1. Unzip the 42992s5.zip file to your system.
2. Building a kernel image which includes Multi-IO card support involves creating an INTEGRITY bootable kernel, adding the Multi-IO driver module to the project, and setting the build options accordingly.
3. From the GHS Integrated Development Environment (IDE), create a new Integrity bootable Kernel for your board type.
4. Select the 'INTEGRITY' 'Operating system', and choose the proper 'Board name'. Select the desired programming 'Language'. The 'Project type' is 'INTEGRITY Kernel'.

5. The next step is to specify the desired 'project directory' and 'project name'. Ensure the 'Dynamic Download' and 'ResourceManager' Kernel libraries are included in the 'Kernel Options' tab.
6. In the kernel build include the Multi-IO driver pcidrivr.c module from your installation directory. i.e:

   C:\42992S5\Drivers\V5\pcidrivr.c
7. Open the build settings dialog and add the 'kernel', 'intlib', and 'bsp' directories associated with the processor family to the 'Include Directories' option.
8. Rebuild your kernel and either flash it and ROM Boot or Network boot. (Reference Green Hills Integrity documentation on how to boot your new kernel).

For the Green Hills Software 'INTEGRITY' embedded operating system, no control panel utility exists for the purpose of assigning logical device numbers to specific installed devices. The device numbering scheme is based upon the order in which the Multi-IO devices are detected in the system during driver initialization. Logical device '1' would correspond to the first found device.

For INTEGRITY10 and 11 the SDK includes project files for each module.

In the /ghs/SDK/int1002_xxx (‘xxx’ is architecture) directory, there is a ‘default.gpj’

Which encompasses all samples, the driver, and the library to be deployed on the target.

3.4.4 Logical Device Numbers - Integrity

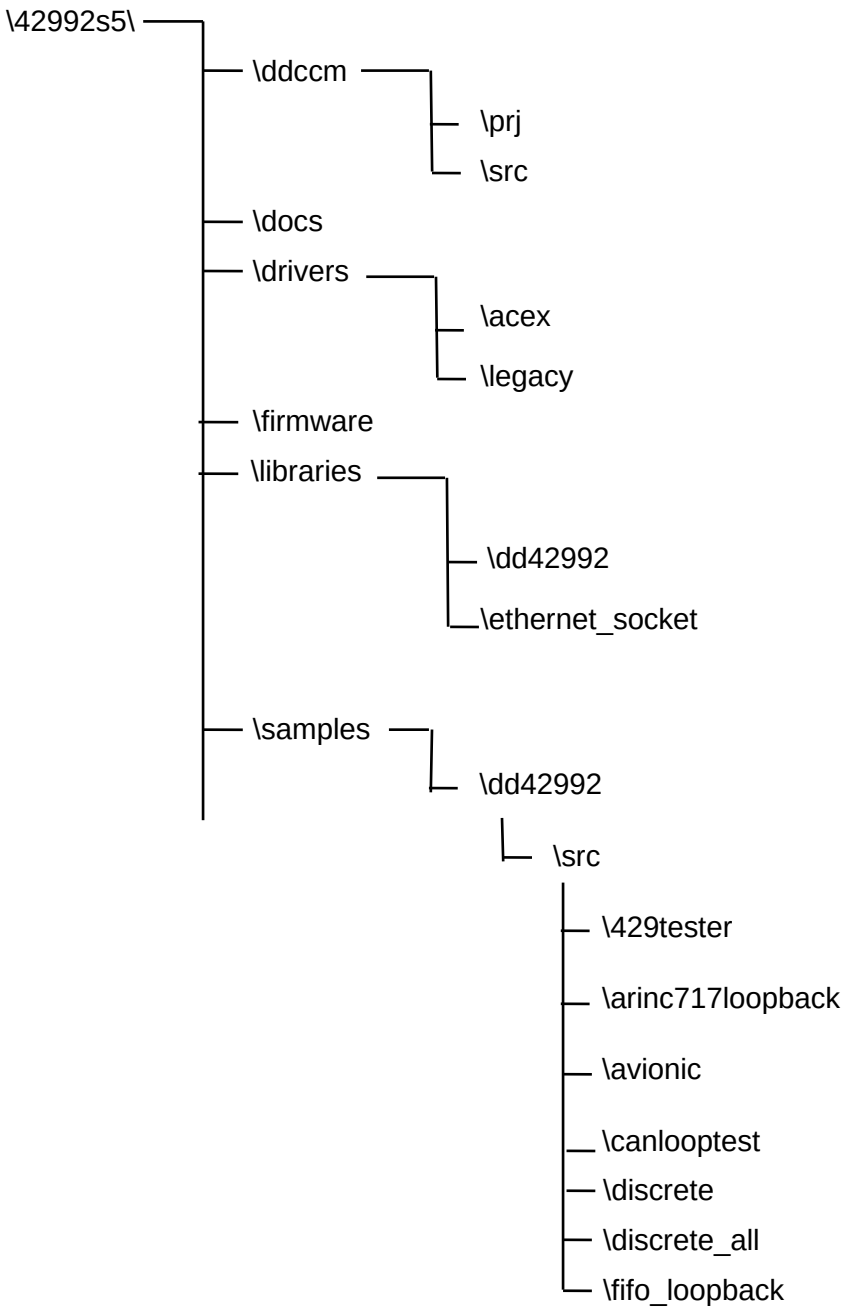
The ***DD-42992 ARINC Multi-IO*** SDK uses Logical Device Numbers to access DDC hardware. A Logical Device Number (LDN) is a unique identifier referring to a particular ARINC 429 board. Most SDK functions will require a target LDN as the first parameter. Depending on your Operating System, the Logical Device Numbers will be auto-assigned or will require some user interaction. See the Logical Device Numbers in Integrity section for more information.

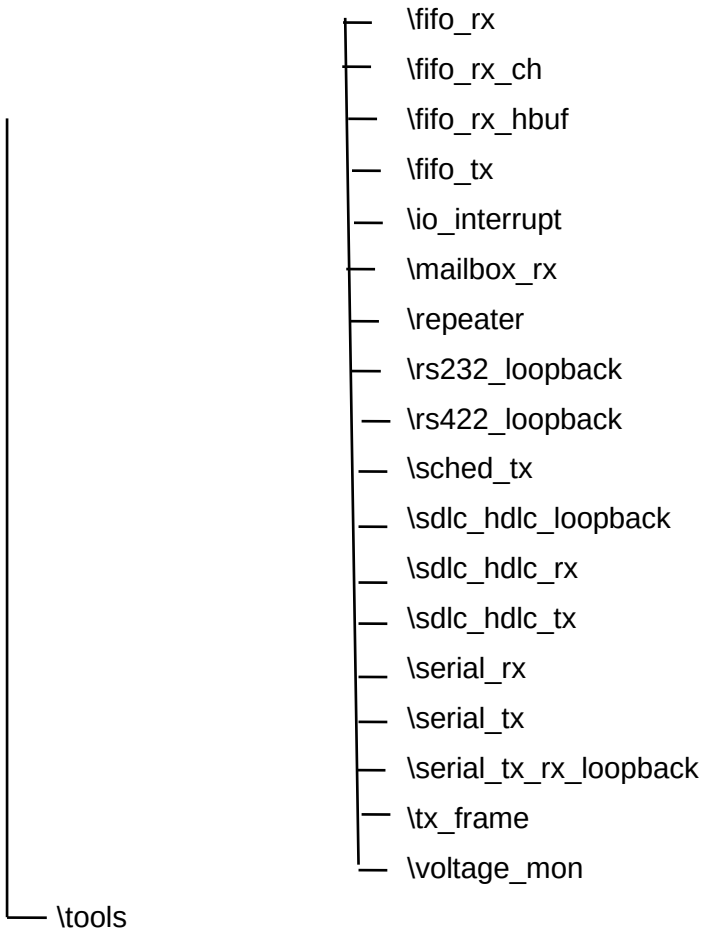
3.4.4.1 Logical Device Numbers in Integrity

Logical Device Numbers in Integrity are auto-assigned by the DDC device driver during kernel boot-up. After a successful power-on, a message will be displayed to the terminal informing the user of any found devices and their assigned LDN’s.

3.4.5 Files in SDK for Integrity

After extracting the zip file the following directory structure will be created:





Directory of \DD42992S2\ddccm\src\ contains:

ddccm.c

ddccm.h

ddccm_diag.c

ddccm_flash.c

ddccm_flash.h

ddccm_os.h

ddccm_os_flash.c

ddccm_os_flash.h

strtable.h

Directory of \docs contains:

license.txt

release-notes-sdk.txt

Directory of \drivers contains:

An \acex and \legacy folders.

\acex contains:

\prj Provides make file to build and/or load driver
\src Contains necessary files to build driver
\tools Includes the load scripts to load or unload the driver.

\legacy contains:

.\ddcemapci

Directory of \firmware contains:

FLASH_UTILITY.pdf Provides instructions to flashing firmware of a device.

Various .bin files Firmware files for DDC devices.

Directory of \libraries contains:

\dd42992 Contains all files for the various 429 modes and features.

\src

arinc717.h Contains prototypes for various ARINC 717 routines.

arinc717_tst.h

arinc717r.h

arinc717x.h

can.h	Contains prototypes for various CAN routines.
can_tst.h	
canr.h	
canx.h	
cardinit.h	Contains the prototypes for card initialization, shutdown, and version information and interrupt facility configuration.
cardinit_tst.h	
cardinitr.h	
cardinitx.h	
control.h	Prototypes for various card feature control routines, including Tx/Rx group reset, loopback control, card level ARINC 429 Multi-IO SDK bit formatting, and discrete line control.
control_tst.h	
controlr.h	
controlx.h	
device429.h	Contains prototypes for operating system specific card initialization, shutdown, and generalized system calls.
device429_tst.h	
device429r.h	
device429x.h	
errors.h	Prototypes for the error display routine associated constants.
hardware.h	Prototypes for SDK level device register and memory access routines, and register, memory, and mask name associations.
hardware_tst.h	
hardwarex.h	

irig.h	Prototypes for card level IRIG configuration functions.
irig_tst.h	
irigr.h	
irigx.h	
receive.h	Prototypes for ARINC 429 Multi-IO SDK receive configuration, FIFO and mailbox reception routines.
receive_tst.h	
receiver.h	
receivex.h	
serial.h	Prototypes for card level Serial IO configuration, UART configuration control, and individual UART control.
serial_tst.h	
serialr.h	
serialx.h	
std429.h	Top Level Multi-IO ARINC 429 /Serial/Discrete header file to be included in any end user application program utilizing a DDC Multi-IO device.
tester.h	Contains prototypes for ARINC 429 tester functions.
tester_tst.h	
testerr.h	
testerx.h	
transmit.h	Prototypes for ARINC 429 Multi-IO SDK transmit configuration, FIFO and scheduled message transmissions.
transmit_tst.h	
transmitr.h	
transmitx.h	
\ethernet_socket	Contains files for DDC's remote access mode.

Directory of \samples contains the source code for the following samples:

429tester	This application tests all hardware and software components to make sure the Registers, memory, and inturupts are in working order. Any problems that arise are reported back in descriptive errors.
arinc717Loopback	This example demonstrates how to configure and use the ARINC 717 channels.
avionic	This example demonstrates how to configure, set, and read the avionics lines.
canLoopTest	This example demonstrates how to configure, and use the CanBUS channels.
discrete	This example demonstrates how to configure, set, and read the discretes.
discrete_all	This example demonstrates how to configure, set, and read the discretes using the functions to access all the bits at once.
fifo_loopback	Demonstrates how to transmit and receive data on the RX channels with the FIFO mode on all channels.
fifo_rx	This example demonstrates how to configure and receive data on the RX channels.
fifo_rx_ch	This example demonstrates how to configure and receive data on the RX channels. This demo allows the user to configure different speeds and parity settings for individual channels.
fifo_rx_hbuf	This example demonstrates how to configure and receive data on the RX channels using the Host Buffer.
fifo_tx	Demonstrates how to configure and transmit low speed data on the TX channels.
io_interrupt	This example demonstrates AIO interrupts.
mailbox_rx	This example demonstrates how to configure and receive data using mailbox mode with filters.

repeater	This example demonstrates how to configure and send data pollution.
rs232_lookback	This example demonstrates how to configure, transmit, and receive data on the RS232 port, as well as test RTS/CTS.
rs422_lookback	This example demonstrates how to configure, transmit, and receive data on the RS422 port.
sched_tx	Demonstrates how to configure and transmit low speed scheduled data on the TX channels.
sdlc_hdlc_rx	This example demonstrates high speed serial reception on the BU-67118x cards.
sdlc_hdlc_tx	This example demonstrates high speed serial transmission on the BU-67118x cards.
sdlc_hdlc_loopback	This exampls demonstrates the combination of reception/transmission on the BU-67118x cards.
serial_rx	This example demonstrates how to configure and received data on the RS232 port.
serial_tx	This example demonstrates how to configure and transmit data on the RS232 port.
serial_tx_rx_loopback	This example combines sending and receiving on a serial port.
tx_frame	This example demonstrates how to transmit messages using minor/major frames.
voltage_mon	This example demonstrates how to monitor the voltage on the receive channels.

Directory of \tools contains:

\tx_inhibit_bc_disable

flash.ddc

3.4.6 Building the Library

Complete Library source code is supplied in order to provide the greatest overall flexibility associated with building application programs utilizing the ARINC 429 Multi-IO SDK. Due to the use of the INTEGRITY Resource Manager within the library, only monolithic or downloadable INTEGRITY application programs can be produced with this software suite. In the case of monolithic programs, simply include all library source files found in the 'source' directory in the overall project, along with the upper level source files that reference any of the ARINC429 Multi-IO SDK routines.

It is possible to create any type of INTEGRITY supported library binary by simply creating a project of the desired library type (.a, .so, etc.) and including all of the source files found in the 'source' directory and building the Library. Upper level application program projects must then be linked with the ARINC429 Multi-IO SDK as required by the development tools.

Note: *When building the Library source code, the 'bsp' directory associated with the particular Board Support Package in use must be listed in the 'Include Directories' field of the project 'Set Options...' dialog in order to provide access to the BSP's '<bus.h>'header file.*

3.4.7 Building Sample Applications

In order to exercise the Library via any of the included sample programs using the library method, the Library must first be generated as described above. For this example, assume the resultant library binary is named '42992s5.a'.

For the sample programs, an application project needs to be created using the IDE properly configured for the desired BSP. Typically, this involves creating an INTEGRITY application program for the appropriate system type containing the top level program source code, and linking it to the DDC Library.

1. From the IDE, create a new project and select the appropriate 'Processor family'. Select the 'INTEGRITY' 'Operating system' and choose the proper 'Board name'. Select the desired programming 'Language'. The project type is 'INTEGRITY Application (Dynamic Download)'.
2. Next specify the desired 'project directory' and 'project name'. Ensure that the 'Include custom .int file' option is selected in the 'INTEGRITY OS' options tab. No specific language or general options are required of the project.
3. Within the 'Project Builder', add the top level sample or end user source modules to the newly created project.

Note: *A newly generated default source module containing a 'main()' routine may be in conflict with another that could possibly be defined in the existing top*

level source.

4. In order to link the top level application code to the ARINC429 Multi-IO SDK, open the project 'Settings' dialog, select the 'Libraries' option, then traverse to and select the '42992s5.a' file previously generated.
5. Open the Integrate (.int) file and add the following to the bottom of the 'AddressSpace' section in order to increase the stack memory size to accommodate the SDK buffers:
    - Task Initial
    - StackLength 0x10000
    - EndTask
6. The application can now be built and executed.

Note: *The '#include' directives found in the sample programs consist of relative path names.*

Note: *In the event that an illegal access occurs while running programs that make calls to the library functions with large array parameters, the 'StackLength' configuration option shown above may need to be further increased in size.*

4 OPERATION

The following section provides the user with information about using the API library functions. A short description of common functions, as well as the order in which you must call the functions is discussed here. All of the functions are listed in alphabetical order in the last section of this manual called General/Initialization API Function List.

4.1 Initialization

In order for the card to function as desired, it needs to be initialized by calling the **InitCard()** command and passing it a logical device number (card number) that was previously set using the *DDC Card Manager.* Multiple cards may be used at the same time in one system. This is done by referencing the logical device number of the card to be accessed.

Next, the user must specify the channel as a transmitter or a receiver. To activate a receive channel, the **EnableRx()** command is used. To activate a transmit channel, the **EnableTx()** command is used.

4.2 ARINC 429/575 Operation

4.2.1 ARINC 429/575 Transmitting

Before trying to transmit on your DDC card, make sure that it is properly initialized by the **InitCard()** command. Then start the channel with the **EnableTx()** command.

After the channel has started, the user can configure the channel's transmission speed by calling the **SetTxSpeed()** function. The parity can be set by using the **SetTxParity()** command.

The user can view the parity and speed that was previously configured by using the **GetTxParity()** command and the **GetTxSpeed()** command respectively.

Beginning with DDC's new DD-40x00x ARINC cards, the user has the option to choose a custom transmission speed between 500-200,000 bps by calling the **dd429X_SetVariableSpeed()** function. A custom transmission speed can be set between 500-50,000 bps in increments of 100 bps, or between 50,000-200,000 bps in increments of 1,000 bps. The actual transmission speed set by the card can be verified by using the **dd429X_GetVariableSpeed()** function.

The function **dd429X_GetTxQueueFreeCount()** is designed to help the user manage the transmit FIFO. It will return with number of available slots in the message FIFO.

NOTE: *The legacy* ***dd429X***_LoadTxQueueOne***()*** *or the* ***LoadTxQueueMore()*** *may still be used to load messages on to the FIFO. The error injection settings will simply default to zero.*

4.2.2 Error Injection

DDC's new DD-40x00x ARINC 429 cards gives the user the ability to add error injection to each transmitted ARINC word. This is done by making use of the **DD429_TESTER_OPTIONS_TYPE** struct. The error injection options are Inter-Word Bit Gap, Word Size, Parity, and Bit-33.

```
typedef struct _DD429_TESTER_OPTIONS_TYPE
{
    U16BIT s16InterWordBitGapError;
    U8BIT u8WordSizeError;
    U8BIT u8ParityError;
    U8BIT u8Bit33;

} DD429_TESTER_OPTIONS_TYPE;
```

The Inter Word Bit Gap error allows the user to vary the inter word bit gap from 1 bit to 32,000 bits instead of the default 4 bits.

Setting the Parity error for a particular message will cause the parity bit to be flipped to the opposite of what was configured for the channel. If parity is not selected, then the parity bit is left alone.

The word size error will allow the user to vary the ARINC message from 2 to 32 bits.

The Extra bit error will add an extra bit to the ARINC message.

4.2.2.1 FIFO Transmission

Messages can be transmitted via the FIFO method by using the **LoadTxQueueOne()** or the **LoadTxQueueMore()** function. The **LoadTxQueueOne()** function loads one ARINC message to the FIFO at a time. The **LoadTxQueueMore()** gives the option to add an arrary of 32-bit ARINC words to the transmit queue. The data will be transmitted in the exact order in which it is placed into the FIFO queue.

DDC's new DD-40x00x ARINC cards allow the user to insert error injection options on a per message basis. The functions **dd429X_LoadTxQueueOne()** and the **dd429X_LoadTxQueueMore()** will add ARINC messages to the transmit FIFO along with the **DD429_TESTER_OPTIONS_TYPE** struct for error injection. Scheduled Transmission.

The function **dd429X_GetTxQueueFreeCount()**is designed to help the user manage the transmit FIFO. It will return with number of available slots in the message FIFO.

> ***NOTE:*** *The legacy* ***LoadTxQueueOne()*** *or the* ***LoadTxQueueMore()*** *may still be used to load messages on to the FIFO. The error injection settings will simply default to zero.*

4.2.2.2 Scheduled Transmission

The *DDC cards* can also transmit using a scheduled method of transmission by using the **AddRepeated()** command. This data will be transmitted at the frequency passed into the **AddRepeated()** command. The frequency field is in milliseconds.

DDC's new line of Avionics boards allows the user to schedule the transmission of an ARINC message via a message index. This allows the user flexibility in scheduling a repeated message. Multiple messages can be scheduled with the same Label or Label/SDI combination, as long as they each have a unique message index. Each message can be deleted or modified based on the index. It is left to the user to keep track of the index that's associated with a particular ARINC message. A total of 1024 messages can be scheduled on each transmit channel.

For DDC's new DD-40x00x ARINC cards, the user can add error injection to every ARINC word. This is done by using the **dd429X_AddRepeated()** or **dd429X_AddRepeatedItem()** and passing it the **DD429_TESTER_OPTIONS_TYPE** struct parameter. (see section 4.2.1.1 for more information on error injection).

The function **dd429X_AddRepeatedItem()** is used to add an item to the scheduler. The function will take an index, a 32-bit ARINC message, and error injection options as inputs.

If **dd429X_AddRepeatedItem()** is called with an index that already exists in the scheduler, then the new message will replace the existing indexed message. However, message timing will not be preserved for the replaced index. This new message will be added to the scheduler at the most current point in the scheduler cycle. To replace the ARINC message while preserving message timing, please use function **dd429X_AddRepeatedItem()**.

dd429X_DelRepeatedItem() can be used to delete the message from the scheduler. The message to be deleted should be specified by the message index. Once an indexed message has been deleted it, its timing will not be preserved when that index is added back to the scheduler. Instead, the new message will be added to the scheduler at the most current point in the scheduler cycle. To preserve message timing, please use function **dd429X_ModifyRepeatedDataItem()**.

dd429X_GetAllRepeatedItem() will return all active indexes via a user supplied buffer. The user can call this function to figure out which indexes are currently used by the scheduler. To get detailed information regarding the message, the use can then call **dd429X_GetRepeatedItem()**, which will return with the 32-bit ARINC word, the message's error injection options, the message rate, and the message offset.

dd429X_ModifyRepeatedDataItem() will replace a message in the scheduler based on the item index. The message timing will still be preserved for the modified message.

4.2.2.3 Major/Minor Frame Creation

Major and Minor frames can be created using DDC's new DD-40x00x ARINC cards.

A major frame is created by calling **dd429X_SetTxFrameControl()** function and passing it the DD429_TX_FRAME_INIT parameter.

Minor frames are added by using the **dd429X_AddTxFrame()** with an array of type **DD429_TX_MINOR_FRAME_PAYLOAD_TYPE** struct. The **DD429_TX_MINOR_FRAME_PAYLOAD_TYPE** struct contains the 32-bit ARINC dataword along with options for error injection via the **DD429_TESTER_OPTIONS_TYPE** struct. (see section 4.2.1.1 for more information on error injection).

```
typedef struct _DD429_TX_MINOR_FRAME_PAYLOAD_TYPE
{
    U32BIT u32Data;
    DD429_TESTER_OPTIONS_TYPE sTesterOptions;
} DD429_TX_MINOR_FRAME_PAYLOAD_TYPE;
```

The transmit FIFO can hold up to up to 1,000 ARINC messages. The **dd429X_GetTxFrameInfo()** can be called to check what the percentage of the transmit FIFO has been used.

A transmit resolution of 1 ms or 1 µs may be selected by calling **dd429X_SetTxFrameResolution()** function. A '0' is passed in for 1 ms resolution and a '1' is passed in for 1 µs resolution.

The **dd429X_SetTxMajorFrameRepeatCount()** specifies the number of times to run the major frame.

To start the execution of the frame, the user must call the **dd429X_SetTxFrameControl()** function again and passing in the

DD429_TX_FRAME_START parameter. To stop the frame, the DD429_TX_FRAME_STOP parameter is passed into the function.

4.2.2.4 Transmitting Asynchronous Messages

For DDC's new DD-40x00x ARINC cards, messages may be transmitted asynchronously, in high or low priority mode using **dd429X_SendTxFrameAsync().** Low priority messages will only be sent if there is enough time in between minor frames to do so. High priority messages go out immediately and can disrupt the timing of the transmit schedule.

4.2.2.5 Variable Voltage Output

With DDC's new DD-40x00x ARINC Cards, the user has the ability to vary the output voltage. The output voltage amplitude may be modified by calling the **dd429X_SetAmplitude()** function. An 8-bit parameter is passed into the function between 0x00 and 0xFF for a peak-to-peak amplitude between 0 and ±10 volts.

4.2.2.6 Data Repeater (Pollution)

The DDC DD-40x0x devices can be used as a data repeater, also known as data pollution.

One or more ARINC receive channels can be set up as data sources for a particular transmitter/repeater channel. Also, one or more transmit channels can be set up as repeaters for a single receiver source.

The function **dd429X_ConfigRepeater()** is used to map a transmitter to a receiver source. The function takes a single receiver and a single transmitter as inputs, along with an option to either map or unmap the two channels as a source/repeater pair. The function can be called multiple times to set up multiple source repeater pairs, map multiple sources to a single repeater, or map a single source to multiple repeaters. Once a transmitter/receiver has been mapped, all data will be repeated on the transmitters within 3ms.

The function **dd429X_SetRepeaterMode()** is used to set the data repeater and data manipulation options for the specified ARINC 429 repeater source channel. All labels will be repeated as on the transmitter/repeater until this function is called. To begin modifying a label, use the struct type DD429_REPEATER_MODE_TYPE *pRepeaterMode.

The function **dd429X_GetRepeaterMode()** is used to find out the current data pollution options for a given receiver source and return it via the DD429_REPEATER_MODE_TYPE *pRepeaterMode struct.

4.2.3 IRIG Output

DDC's new DD-40x00x ARINC cards are capable of outputting an IRIG digital signal. The IRIG transmit register is set with the **dd429X_SetIRIGTx()** function. The **dd429X_GetIRIGTx()** function can be used to get the current status of the IRIG transmitter.

4.2.4 ARINC 429/575 Receiving

Before trying to receive on your ***DDC Cards***, make sure that you have properly initialized your card using the **InitCard()** command. The next step is to set the receiver mode of the channel group by calling **SetRxMode()**. The two modes of operation are FIFO and mailbox. The groupings of each channel are as follows:

Group1 = Tx1, Rx1, Rx2, Rx9, and Rx10

Group2 = Tx2, Rx3, Rx4, Rx11, and Rx12

Group3 = Tx3, Rx5, Rx6, Rx13, and Rx14

Group4 = Tx4, Rx7, Rx8, Rx15, and Rx16

Note: *Channel groupings depend on the channel count of your DDC Multi-IO device.*

The current mode of a channel grouping may also be recalled from the card by calling **GetRxMode()**.

After setting the mode the user will be required to select the parity of the channel by calling **SetRxParity()**. The parity of the channel may also be read by calling **GetRxParity()**. Then you should start the channel with the **EnableRx()** command.

Specital Note: *DDC's new DD-40x00x ARINC cards no longer use channel grouping for the transmit and receive channels. Instead, the receivers must be set individually via the* ***SetRxChannelMode()*** *and the* ***SetRxChannelParity()*** *functions.*

After the channel has been enabled the user can configure the channel's transmission speed using the **SetRxSpeed()** command. The **GetRxSpeed()** command may be used to check the speed that was previously configured for the receive channel.

Special Note: *When operating the* ***BU-65590/91Ux, BU-65590F/Mx, BU-65590Cx*** *DDC cards, the ARINC TimeTag selection will override the Relative Time*

counter selection in the 1553 section. If IRIG is selected on the 1553, the ARINC then selects the global 48-bit counter, and the 1553 will be modified to use the global 48-bit counter as well.

DDC's new DD-40x00x ARINC cards gives the user the option to select a custom speed by using the **dd429X_SetVariableSpeed()** function. Between 500-50,000 bps, custom speeds may be set in increments of 100 bps. For 50,000-200,000 bps, the custom speed may be set in increments of 1,000 bps. After setting the transmit speed, the user may verify that the correct speed was set by calling the **dd429X_GetVariableSpeed()**.

Note: *The receive channel must be set to the same speed as the transmit channel or else the incoming message will be discarded.*

4.2.4.1 ARINC Time Tag of Receive channels

Time stamps can be enabled for the receive channels by calling the **EnableTimeStamp()** command. The user may configure the time stamp to use the internal relative time or an external, 1 Pulse-per-Second IRIG-B input by calling the **ConfigTimeStamp()** command. The IRIG-B 1 PPS signal will be used to complement the internal 100ns time stamp.

The **GetTimeStampStatus()** command tells the user if time stamping has been enabled for the channel.

4.2.4.2 FIFO Reception

The DDC Multi-IO Cards receive messages from the FIFO by using the **ReadRxQueueIrigOne()** and the **ReadRxQueueIrigMore()** commands. These commands will retrieve ARINC data words from the FIFO, along with the word's time stamps, in exactly the order in which it was placed in the receiver queue. **ReadRxQueueIrigOne()** will retrieve one ARINC message from the FIFO at a time while the **ReadRxQueueIrigMore()** command can retrieve an array of messages.

4.2.4.3 Scheduled Reception

Messages can be read using the mailbox method of reception by using the **ReadMailboxIrig()** command at the frequency that was defined by the user in the **AddRepeated()** , **dd429X_AddRepeated()**or **dd429X_AddRepeatedItem()** command.

The user can use the **GetMailboxStatus()** command to see if there are any new words received at a particular mailbox.

4.2.5 Voltage Monitoring on Receive Channels

DDC's new DD-40x00x ARINC cards give the user the ability to monitor the input voltage on the first eight receive channels.

Voltage monitoring is enabled by calling the **dd429X_VoltageMonitorEnable()** function. To start the voltage monitoring, the **dd429X_VoltageMonitorStart()** is called.

The **dd429X_VoltageMonitorGetData()** function is used to get the monitor data and stores it into a 16-bit user supplied buffer. Bits 15:12 give the channel number. Bits 11:10 padded '0's. Bits 9:0 give the 10-bit voltage value.

The **dd429X_VoltageMonitorGetStatus()** is used to see the voltage monitoring has completed.

4.2.6 ARINC 429/575 Channel Control Functions

These functions provide the user with control for a group. A group consists of one transmitter and two receivers. The **SetLoopBack()** command and the **GetLoopBack()** command can be used to set an internal loopback within a group and get the loopback status respectively.

The **SetBitFormat()** command can be used to set the card's bit format. The **GetBitFormat()** command will return the card's bit format.

The **ResetGroup()** command can be used to reset a group of transmitters and receivers on a card.

4.2.7 ARINC 429/575 Interrupts

The ARINC429 Multi-IO SDK has four functions for ARINC 429/575 interrupt support. These functions are **InstallHandler()**, **UninstallHandler()**, **SetIntCondition()**, and **GetIntStatus()**. The function **InstallHandler()** is used to configure the user call back routine. **UninstallHandler()** is the clean up routine for **InstallHandler(),** uninstalling the previously defined interrupt routine.

SetIntCondition() allows the user to define what condition on a specific interface will generate an interrupt. The interrupt conditions are:

- An ARINC Protocol word is received
- A Fail warning is received
- An ARINC Function test command is received

- An ARINC solo command word is received
- An ARINC transmit start command is issued
- No data Received
- End of transmission
- Normal operation mode received
- Data Pattern match
- Word type match
- Receiver buffer is full
- Received FIFO rollover

4.3 ARINC 717/573 Operation

4.3.1 ARINC 717/573 Transmitting

Before trying to transmit any data on your DDC device, make sure that it is properly initialized by the **InitCard()** command.

Once the card is initialized, you can set assign any ARINC 717/573 channel as a transmitter and set additional configuration options (speed, sub-frame count) using the **acexArinc717ProgConfig()** function. This function needs to be called additionally for any other ARINC 717/573 channels that are to be used.

Note: *Event Interrupts are required for proper sub-frame transmission synchronization. See Section 4.3.3 for addition information on using ARINC 717/573 interrupts.*

A complete ARINC 717/573 “Frame” repeats every four seconds and consists of four sub-frames (one sub-frame per second). By configuring different sub-frame word sizes (using **acexArinc717ProgConfig()**), the DDC device is able to achieve rates from 32 to 8192 Words-per-Second (WPS).

The hardware transmission buffer is designed to hold two sub-frames (of configured word size) for immediate transmission. Double Buffering allows the user to actively load data while transmitting the previous sub-frame buffer. Interrupt callbacks alert the user application when to load new data.

Sub-Frame data is loaded using the **acexArinc717ProgLoadTxData()** function. Using double-buffering methods, the function allows the user to specify to load the data into the Primary or Secondary buffer.

Note: *Per ARINC 717/573, the first word in each sub-frame needs to have a*

Sychronization pattern (12-bits), which should be manually configured in the user buffer on every transmitted buffer. Failure to do so will cause the previously loaded Sub-Frame pattern to be retransmitted.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0	0	0	0	Data Word 1												0	0	0	0	Synchronization Pattern											
0	0	0	0	Data Word 3												0	0	0	0	Data Word 2											
0	0	0	0	Data Word 5												0	0	0	0	Data Word 4											
0	0	0	0	Data Word 7												0	0	0	0	Data Word 6											
0	0	0	0	Data Word 9												0	0	0	0	Data Word 8											
0	0	0	0	Data Word 11												0	0	0	0	Data Word 10											
0	0	0	0	Data Word 13												0	0	0	0	Data Word 12											
0	0	0	0	Data Word 15												0	0	0	0	Data Word 14											

Figure 4. ARINC 717/573 Sub-Frame Format (e.g. 64 WPS)

Once the channel is configured and data is loaded into the transmission buffers, the channel can be independently started using the **acexArinc717ProgState()** function by setting the state to **ARINC_717_RUN**.

When data is sent, the Interrupt callback routine will be called, notifying the user application to load new transmit data into the desired buffer (Primary or Secondary).

4.3.2 ARINC 717/573 Receiving

Before trying to receive any data on your DDC device, make sure that it is properly initialized by the **InitCard()** command.

Once the card is initialized, you can set assign any ARINC 717/573 channel as a receiver and set additional configuration options (speed, Output) using the **acexArinc717ProgConfig()** function. This function needs to be called additionally for any other ARINC 717/573 channels that are to be used.

Note: *Event Interrupts are required for proper sub-frame receiver synchronization. See Section 4.3.3 for addition information on using ARINC 717/573 interrupts.*

A complete ARINC 717/573 “Frame” repeats every four seconds and consists of four sub-frames (one sub-frame per second). By configuring different sub-frame word sizes (using **acexArinc717ProgConfig()**), the DDC device is able to achieve rates from 32 to 8192 Words-per-Second (WPS).

The hardware reception buffer is designed to hold up to 2 sub-frames (of configured word size). Double Buffering allows the user to actively read data while new data is

being loaded into the alternate sub-frame buffer. Interrupt callbacks alert the user application when to read new data and from which buffer.

Once the channel is configured, the channel can be independently started using the **acexArinc717ProgState()** function and setting the state to **ARINC_717_RUN.**

Sub-Frame data is read using the **acexArinc717ProgRxData()** function. Since double-buffering is used, the function requires the caller to specify to read data from the Primary or Secondary buffer.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0	0	0	0	Data Word 1												0	0	0	0	Synchronization Pattern											
0	0	0	0	Data Word 3												0	0	0	0	Data Word 2											
0	0	0	0	Data Word 5												0	0	0	0	Data Word 4											
0	0	0	0	Data Word 7												0	0	0	0	Data Word 6											
0	0	0	0	Data Word 9												0	0	0	0	Data Word 8											
0	0	0	0	Data Word 11												0	0	0	0	Data Word 10											
0	0	0	0	Data Word 13												0	0	0	0	Data Word 12											
0	0	0	0	Data Word 15												0	0	0	0	Data Word 14											

Figure 5. ARINC 717/573 Sub-Frame Format (e.g. 64 WPS)

When new data is received, the Interrupt callback routine will be called, notifying the user application to read the new data and from which buffer (Primary or Secondary).

4.3.3 ARINC 717/573 Interrupts

The ARINC429 Multi-IO SDK has four functions for ARINC 717/573 interrupt support. These functions are **InstallHandler()**, **UninstallHandler()**, **acexArinc717Interrupts()**, and **GetIntStatus()**. The function **InstallHandler()** is used to configure the user call back routine. **UninstallHandler()** is the clean up routine for **InstallHandler(),** uninstalling the previously defined interrupt routine.

acexArinc717Interrupts() allows the user to define what condition(s) (on a specific channel) will generate an interrupt. The interrupt conditions are:

- Transmit Primary Sub-Frame Buffer has been sent.
- Transmit Secondary Sub-Frame Buffer has been sent.
- Receiver Primary Sub-Frame Buffer has been filled.
- Receiver Secondary Sub-Frame Buffer has been filled.
- Receiver channel Synchronize Success.

- Receiver channel Synchronize Error.
- Receiver BIT Error Detected.

See see the API usage of **acexArinc717Interrupts()** on how to enable specific interrupt conditions.

See see the API usage of **GetIntStatus()** on how to rectreive specific ARINC 717/573 interrupt events.

4.4 CanBUS Operation

4.4.1 CanBUS Transmitting

Before trying to transmit any data on your DDC device, make sure that it is properly initialized by the **InitCard()** command.

Once the card is initialized, you can set channel configuration options (baud rate, interrupt enable, loopback, filterring) using the **acexCanBusConfig()** function. This function needs to be called additionally for any other CanBUS channels that are to be used.

CanBUS data is transmitted “on-demand” from the User Application and loaded a hardware FIFO buffer. By configuring a different baud rate (using **acexCanBusConfig()**), the DDC device is able to achieve rates from 20 to 1000 Kilobits-per-Second (KBS). Each CanBUS Message consists of a 24 byte packet (6 double words) and can be transmitted using the **acexCanBusTxData()** function.

<table>
<tr><td colspan="16">31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>09</td><td>08</td><td colspan="8">07 06 05 04 03 02 01 00</td></tr>
<tr><td colspan="15">DDC Device ID Constant
(0x4DDC)</td><td>Reserved</td><td>TX Queue Overflow</td><td>Tx Msg Error</td><td>Reserved</td><td>Use Ext. Msg ID</td><td>Reserved</td><td>RTR</td><td colspan="8">DDC CanBUS Channel Number</td></tr>
<tr><td>Reserved</td><td colspan="31">CanBUS Message ID
(12 or 29 bits depending on “Use Ext. Msg ID”)</td></tr>
<tr><td colspan="28">Reserved</td><td colspan="4">Message Length (1-8 bytes)</td></tr>
<tr><td colspan="16">Reserved</td><td colspan="16">Message Timestamp</td></tr>
<tr><td colspan="8">Data Byte 4</td><td colspan="8">Data Byte 3</td><td colspan="8">Data Byte 2</td><td colspan="8">Data Byte 1</td></tr>
<tr><td colspan="8">Data Byte 8</td><td colspan="8">Data Byte 7</td><td colspan="8">Data Byte 6</td><td colspan="8">Data Byte 5</td></tr>
</table>

Figure 6. CanBus Message Packet Format

Once the channel is configured (and some data is loaded into the transmission buffers), the channel can be independently started using the **acexCanBusState()** function by setting the state to **CAN_BUS_RUN**.

Addiitional transmit data can be added to the transmit FIFO (while in the '**CAN_BUS_RUN'** state) by calling **acexCanBusTxData()** repeatedly.

4.4.2 CanBUS Receiving

Before trying to transmit any data on your DDC device, make sure that it is properly initialized by the **InitCard()** command.

Once the card is initialized, you can set channel configuration options (baud rate, interrupt enable, loopback, filterring) using the **acexCanBusConfig()** function. This function needs to be called additionally for any other CanBUS channels that are to be used.

CanBUS data is is stored in a hardware buffer and can be received by polling or using interrupt events. By configuring different a baud rate (using **acexCanBusConfig()**), the DDC device is able to achieve rates from 20 to 1000 Kilobits-per-Second (KBS). Each CanBUS Message consists of a 24 byte packet (6 double words).

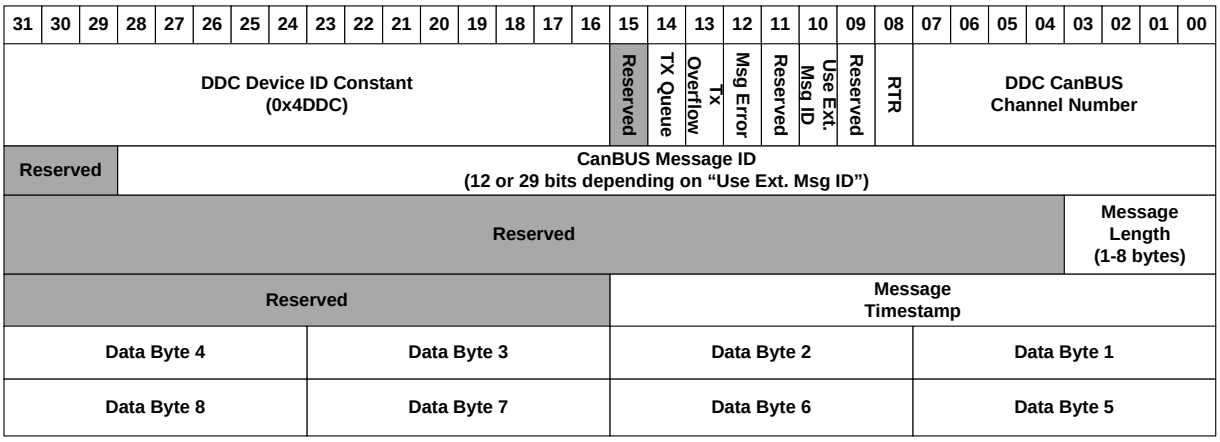


Figure 7. CanBus Message Packet Format

Once the channel is configured, the channel can be independently started using the **acexCanBusState()** function by setting the state to **CAN_BUS_RUN**.

Received data can be read at any time (while in the **CAN_BUS_RUN** state) by calling **acexCanBusRxData()** repeatedly.

Additionally, the user has the option to enable CanBUS Interrupts, which will alert the user of when (and how many) messages have been received. See Section 4.4.3 for more information.

4.4.3 CanBUS Interrupts

The ARINC429 Multi-IO SDK has four functions for CanBUS interrupt support. These functions are **InstallHandler()**, **UninstallHandler()**, **acexCanBusConfig()**, and **GetIntStatus()**. The function **InstallHandler()** is used to configure the user call back routine. **UninstallHandler()** is the clean up routine for **InstallHandler(),** uninstalling the previously defined interrupt routine.

acexCanBusConfig() ('bInterrupt' parameter) allows the user to enable/disable CanBus Receiver Interrupts per channel. If Interrupts are enabled, the user can determine how many messages are in the receiver buffer (per CanBUS channel) by calling **GetIntStatus()**.

See see the API usage of **GetIntStatus()** on how to rectreive specific CanBUS interrupt information.

4.5 Serial I/O (RS-232/422/485) Operation

Before using any Serial channel, it needs to be enabled and configured by using the **EnableUart()** function. This function also allows programmable devices to switch between RS-232/422/485 electrical operational modes.

There are two functions that are used to read or write from the device serial (COM) port. The **WriteUart()** function allows the user to Write access to any UART configuration / control / data register. The **ReadUart()** function similarly allows the user to read these registers.

4.6 Avionics/Digital Discrete I/O Operation

There are five basic commands for handling the card's discrete outputs. The first function **SetDiscDir()** is used set the direction of the discrete line. The next function **GetDiscDir()** is used to return the direction of the discrete line. **SetDiscOut()** command is used for setting the value of all discrete outputs on the card, and the **GetDiscOut()** routine is used for getting the current state of a discrete output. The last function **GetDiscIn()** is used to return the state of the selected discrete input line.

4.7 Error Handling

The **GetLibVersion()** command can be used to find out the library version that you are using. You can use the **GetErrorMsg()** command to get the message string that corresponds to the error number that is returned.

5 API FUNCTION DEFINITIONS

This section contains a detailed description of all the functions in the PC Card Library. Each function description contains information about it's functionality, parameters passed, and the possible error responses.

The PC Card Library will provide a hardware abstraction layer so that the programmer does not need to know the hardware architecture of the card. When writing code for this card the appropriate library file and appropriate header files must be included in your project.

The **"std429.h"** header file includes all of the other header files for the library. As a result, you only need to include this file in your program to access the API Library Functions. For more information about the header files please see the **Header Files** section in a previous section of this manual. The **"DD42992.lib"** file is the Microsoft explicit link library file that must be included in your project in order to access the functions in the dll. This library file will create an explicit link to our DLL from your Microsoft Visual C++ compiler.

5.1 Conventions

In each function, the *Card* parameter always starts from one. The *Card* parameter indicates the logical number that was assigned to the card in the **DDC 429 Multi-IO SDK Card Manager**.

The prototype for each function shows the header file that contains the function prototype. You do not need to include this header file in your code to use our library of functions. Please see the section 3.1.2 for a detailed description of the necessary header files that need to be included in your code in order to use the API library of functions.

"**Return**"	Any explicitly listed negative return integer indicates an error status. All error messages associated with the status numbers are listed at the end of this chapter and can be returned using the function **GetErrorMsg**.
"short"	A two-byte signed integer.
"long"	A four-byte signed integer.
U64BIT	unsigned 64-bit value (OS dependent)
S64BIT	signed 64-bit value (OS dependent)
U32BIT	unsigned long

S32BIT	long
U16BIT	unsigned short
S16BIT	short
U8BIT	unsigned char
S8BIT	char
BOOLEAN	unsigned char
DWORD	unsigned long

5.2 General/Initialization API Function List

Summarized below is a list of all API functions specific to Genera/Initialization operation. The following sections describe fully the operation of the functions. A list of Error Messages begins on page 311.

Table 2. Initialization Functions

Function Name	Description	Page
FreeCard	Close card and free up resources.	186
InitCard	Initialize or reset a card.	240
IOFree	Frees up any resources used by the discrete or avionic IO.	244
IOInitialize	Initializes the discrete and avionic IO.	245

Table 3. Information Functions

Function Name	Description	Page
GetCardType	Gets the type of card.	195
GetErrorMsg	Get the message string of an error number.	202
GetHwVersionInfo	Returns information regarding the card.	205
GetLibVersion	Get the version of this library.	211
GetSwVersionInfo	Returns information regarding the SDK.	231

Table 4. Channel Control Functions		
Function Name	**Description**	**Page**
dd429X_GetVariableSpeed*	Get the current speed configuration for the channel	133
dd429X_SetVariableSpeed*	Set the speed for the Transmit or Receive channel	160
GetBitFormat	Get a card's bit format.	194
GetLoopBack	Get a group's loopback status.	213
ResetGroup	Reset a group (1 transmitter and multiple receivers) in a card.	265
SetBitFormat	Set a card's bit format.	273
SetLoopBack	Set the loopback within a group.	280

* dd429x_ functions apply to the new ***DD-40x00F/H/I/T/K, DD-40002M/X, & BU-67118K/M/Y/Z*** boards

5.3 ARINC 429/575 API Function List

Summarized below is a list of all API functions specific to **ARINC 429/575** operation. The following sections describe fully the operation of the functions. A list of Error Messages begins on page 311.

Table 5. ARINC 429/575Transmitter Control Functions		
Function Name	**Description**	**Page**
dd429X_GetAmplitude*	Get the voltage amplitude setting for the transmitter	114
dd429X_GetIRIGTx*	This function gets the current status of the IRIG transmitter.	116
dd429X_GetTxFrameInfo*	Return the status of the transmit FIFO	129
dd429X_SetAmplitude*	Set the voltage amplitude for the transmitter	150
dd429X_SetIRIGTx*	This function sets the IRIG Transmitter registers.	152
dd429X_SetTxFrameControl*	Initialize, Start or Stop the transmission of data	154
EnableTx	Enable or disable a transmitter.	182
GetTxParity	Get a transmitter's parity setting.	236
GetTxSpeed	Get a transmitter's speed.	237
GetTxStatus	Get a transmitter's enabling status.	238
ResetTxChannel	Reset a transmitter's channel	269
SetTxParity	Turn on or off a transmitter's odd parity.	291
SetTxSpeed	Set a transmitter's speed as low or high.	293

* dd429x_ functions apply to the new ***DD-40x00F/H/I/T/K, DD-40002M/X, & BU-67118K/M/Y/Z*** boards

Table 6. ARINC 429/575 Transmit Functions		
Function Name	**Description**	**Page**
AddRepeated	Add a word for repeated transmission.	90
ClearRepeated	Delete all words in repeated transmission.	95
DelRepeated	Remove a word from repeated transmission.	171
dd429X_AddRepeated*	Add a word for repeated transmission.	101
dd429X_AddRepeatedItem*	Add a word for repeated transmission at a certain frequency.	104
dd429X_AddTxFrame*	Add a Minor Frame	107
dd429X_ConfigRepeater*	Map a transmitter to a receiver source	110
dd429X_DelRepeatedItem*	This function will delete a previously defined message from the scheduler	112
dd429X_GetRepeated*	Get the data and setting of a repeated label/SDI.	118
dd429X_GetRepeatedItem*	Returns information regarding a scheduled message	121
dd429X_GetAllRepeatedItem*	Returns all active indexes in the scheduler into a user array	123
dd429X_GetRepeaterMode*	To find out the current data pollution options for a given receiver source	125
dd429X_GetTxQueueFreeCount*	Designed to help the user manage the transmit FIFO	128
dd429X_GetTxFrameInfo*	Return the status of the transmit FIFO	129
dd429X_GetTxFrameResolution*	Get the transmit resolution setting for the channel	131
dd429X_LoadTxQueueMore*	Load multiple words into a transmitter queue.	135
dd429X_LoadTxQueueOne*	Load one word into a transmitter's queue.	137
dd429X_ModifyRepeatedData*	Modify data of a existing repeated transmission.	139
dd429X_ModifyRepeatedDataItem*	Replace a message in the scheduler based on the item index. The message timing will still be preserved for the modified message.	141
dd429X_ModifyTxFrameData*	Modify data based on Label or SDI	143
dd429X_SetRepeaterMode*	To set the data repeater and data manipulation options for the specified ARINC 429 repeater source channel	145
dd429X_SendTxFrameAsync*	Send asynchronous messages in high or low priority	148
dd429X_SetTxFrameControl*	Initialize, Start or Stop the transmission of data	154
dd429X_SetTxFrameResolution*	Set the transmit resolution for ARINC messages	156
dd429X_SetTxMajorFrameRepeatCount*	Number of times to run Major Frame	158
GetAllRepeated	Get all labels/SDIs in repeated transmission.	189
GetNumOfRepeated	Get total number of words in repeated transmission.	220
GetRepeated	Get the data and setting of a repeated label/SDI.	221
GetTxQueueStatus	Get number of words in a transmitter queue.	239
LoadTxQueueOne	Load one word into a transmitter's queue.	248
LoadTxQueueMore	Load multiple words into a transmitter queue.	246

Table 6. ARINC 429/575 Transmit Functions		
Function Name	**Description**	**Page**
ModifyRepeatedData	Modify data of a existing repeated transmission.	250

* dd429x_ functions apply to the new ***DD-40x00F/H/I/T/K, DD-40002M/X, & BU-67118K/M/Y/Z*** boards

Table 7. ARINC 429/575 Receiver Control Functions		
Function Name	**Description**	**Page**
ConfigTimeStamp	Configure the IRIG/48-Bit time tag.	99
DisableRxHostBuffer	Disable the Receive Host Buffer.	173
EnableRx	Enable or disable a receiver.	177
EnableRxHostBuffer	Enable the Receive Host Buffer.	178
EnableTimeStamp	Enable or disable a receiver's time stamp function.	180
dd429X_VoltageMonitorEnable*	Enable the voltage monitoring functionality for the channel	162
dd429X_VoltageMonitorGetData*	Copy the collected voltage samples to a user supplied buffer	164
dd429X_VoltageMonitorGetStatus*	Determine if voltage monitoring has completed	166
dd429X_VoltageMonitorStart*	Start the collecting of voltage monitor data	168
GetRxChannelMode	Return the Receiver's mode of operation.	223
GetRxChannelParity	Get the Receiver's parity setting	224
GetRxChannelSpeed	Get the Receiver's speed	225
GetRxParity	Return the Receiver's parity setting	228
GetRxSpeed	Get receiver speed.	229
GetRxStatus	Get a receiver's enabling status.	230
GetTimeStamp	Retrieve the current 48-bit Time Tag value.	232
GetTimeStampStatus	Get a receiver's time stamp status.	235
InstallFifoRxHostBuffer	Install the Receive Host Buffer for FIFO messages.	241
ResetRxChannel	Reset the receiver	267
ResetTimeStamp	Reset the timer for time stamps.	268
SetRxChannelMode	Configure the Receiver mode of operation	284
SetRxChannelParity	Configure the Receiver's parity setting	285
SetRxChannelSpeed	Set the Receiver's speed	286
SetRxParity	Set the Receiver parity.	287
SetRxSpeed	Set receiver speed as low or high.	289
UninstallFifoRxHostBuffer	Uninstall the Receive Host Buffer for FIFO messages	294

* dd429x_ functions apply to the new ***DD-40x00F/H/I/T/K, DD-40002M/X, & BU-67118K/M/Y/Z*** boards

Table 8. ARINC 429/575 Receiver Filter Functions		
Function Name	**Description**	**Page**
AddFilter	Add a filter to a receiver.	89
ClearFilter	Delete a receiver's all filters.	93
DelFilter	Delete a filter from a receiver.	169
EnableFilter	Enable or disable the use of a receiver's all filters.	175
GetAllFilter	Get a receiver's all existing filters.	187
GetFilter	Get the existence status of a filter.	203
GetFilterStatus	Get the usage status of a receiver's all filters.	204
GetNumOfFilter	Get a receiver's total number of existing filters.	219

Table 9. ARINC 429/575 Receive Functions		
Function Name	**Description**	**Page**
ClearMailbox	Mark all words in a mailbox as read already.	94
ClearRxQueue	Remove all words in a receiver queue.	96
GetMailbox	Get number and locations of new words in a receiver's mailbox.	215
GetMailboxStatus	Check if a new word is received at a mailbox location.	217
GetRxQueueStatus	Get total number of words in a receiver queue.	226
GetRxMode	Get the receiver mode of operation: FIFO or MAILBOX	227
ReadMailboxIrig	Read a word from a receiver's mailbox.	254
ReadRxHostBuffer	Read data from the Receive Host Buffer	257
ReadRxQueueIrigMore	Read multiple words from a receiver queue.	259
ReadRxQueueIrigOne	Read one word from a receiver queue.	261
SetRxMode	Set the receiver mode of operation: FIFO or MAILBOX	288

5.4 ARINC 717/573 API Function List

Summarized below is a list of all API functions specific to **ARINC 717** operation. The following sections describe fully the operation of the functions. A list of Error Messages is available in Section 8.

Table 10. ARINC 717/573 Functions

Function Name	Description	Page
acexArinc717ProgConfig	Programs ARINC 717/573 Configuration	82
acexArinc717ProgState	Programs ARINC 717/573 state of operation	88
acexArinc717ProgLoadTxData	Loads ARINC 717/573 Transmission Data	86
acexArinc717Interrupts	Enables/Disables ARINC 717/573 Interrupts	80
acexArinc717ProgRxData	Reads ARINC 717/573 Received data	84

5.5 CanBUS API Function List

Summarized below is a list of all API functions specific to **CanBUS** operation. The following sections describe fully the operation of the functions. A list of Error Messages is available in Section 8.

Table 11. CANBus Functions

Function Name	Description	Page
acexCanBusConfig	Programs CanBus Configuration	74
acexCanBusRxData	Reads CanBus Received data	75
acexCanBusState	Programs CanBus state of operation	77
acexCanBusTxData	Loads CanBus Transmission Data	78

5.6 Serial I/O (RS-232/422/485) API Function List

Summarized below is a list of all API functions specific to **Serial I/O (RS-232/422/485)** UART operation. The following sections describe fully the operation of the functions. A list of Error Messages is available in Section 8.

Table 12. Serial I/O (UART) Functions

Function Name	Description	Page
DisableUart	Disables a specific Serial I/O (UART) channel.	174
EnableUart	Enables a specific Serial I/O (UART) channel.	184
ReadUart	Reads data from a specific Serial I/O (UART) channel.	252
ReadUartConfig	Reads the current configuration of a Serial I/O (UART) channel.	263
WriteUart	Writes data to a specific Serial I/O (UART) channel.	296
WriteUartConfig	Writes the configuration of a Serial I/O (UART) channel.	298

5.7 Discrete Digital/Avionics I/O API Function List

Summarized below is a list of all API functions specific to **Digital Discrete/Avionics I/O** operation. The following sections describe fully the operation of the functions. A list of Error Messages is available in Section 8.

Table 13. Discrete I/O Functions

Function Name	Description	Page
GetAvionAll	Gets the enable states and levels for all avionics I/O channels.	190
GetAvionIn	Returns the current input level of an avionics channel.	191
GetAvionOut	Returns the current output level of an avionics channel.	192
GetAvionOutEnable	Returns the current output enable state of an avionics channel.	193
GetDiscALL	Gets the bit directions and levels for all Discrete I/O channels.	198
GetDiscDir	Returns the direction of the discrete line	199
GetDiscIn	Returns the state of the discrete input line.	200
GetDiscOut	Gets the current state of a discrete output.	201
SetAvionAll	Sets the enable states and levels for all avionics I/O channels.	270
SetAvionOut	Sets the output level of an avionics channel.	271
SetAvionOutEnable	Sets the enable state of an avionics channel.	272
SetDiscAll	Sets the bit directions and levels for all Discrete I/O channels.	275
SetDiscDir	Sets the direction of the discrete line.	276
SetDiscOut	Sets discrete outputs high or low.	277

acexCanBusConfig

PROTOTYPE

#include “control.h”
S16BIT **acexCanBusConfig** (S16BIT Card,
PCAN_BUS_CONFIG sConfig

HARDWARE

BU-67211Ux

PARAMETERS

Card	(input parameter) The card number given to the card in the DDC Card Manager located in the Control Panel.
sConfig	(input parameter) CanBus Configuration to load.

DESCRIPTION

This function configures a single CanBUS Channel.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_FEATURE_NOT_SUPPORTED	Feature not Supported on target hardware.
ERR_INVALID_CHANNEL_NO	Invalid/Unknown ARINC 717 channel referenced.
ERR_INVALID_CH_TYPE	Invalid/Unknown Channel Type
ERR_SUCCESS	The function returned successfully.

EXAMPLE

```
// This example configures CanBus channel ‘1’ on Card ‘1’

      S16BIT card = 1, Status = 0;
      CAN_BUS_CONFIG sConfig;

      sConfig.u8Speed = CAN_BUS_SPEED_1_MBS;
      sConfig.u8Channel = 1;
      sConfig.bInterrupt = TRUE;
      sConfig.u32ConfigOption = CAN_BUS_SPEED_OPT
      |CAN_BUS_RX_INTERRUPT_OPT;

      Status = acexCanBusConfig(card, sConfig);
```

SEE ALSO

acexCanBusRxData()
acexCanBusTxData()
acexCanBusState()
PCAN_BUS_CONFIG

acexCanBusRxData

PROTOTYPE

```
#include “control.h”
S16BIT acexCanBusRxData( S16BIT Card,
                         U8BIT u8Channel,
                         U32BIT *pRxData,
                         U32BIT u32MsgCount,
                         U32BIT *pu32MsgsRead);
```

HARDWARE

BU-67211Ux

PARAMETERS

Card	(input parameter) The card number given to the card in the DDC Card Manager located in the Control Panel.
u8Channel	(input parameter) The CanBUS channel number to Read from.
pRxData	(input/output parameter) U32BIT Pointer to Receiver Buffer.
u32MsgCount	(input parameter) Maximum Number of Messages to receive Note: 1 message = 24 bytes.
pu32MsgsRead	(output parameter) Number of Messages Read. Note: 1 message = 24 bytes.

DESCRIPTION

This function reads CanBus Messages from the hardware receive buffer.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_FEATURE_NOT_SUPPORTED	Feature not Supported on target hardware.
ERR_INVALID_CHANNEL_NO	Invalid/Unknown CanBus channel referenced.
ERR_INVALID_CH_TYPE	Invalid/Unknown Channel Type
ERR_SUCCESS	The function returned successfully.

acexCanBusRxData (continued)

EXAMPLE

```
// This example will read data from CanBus channel ‘1’ on Card ‘1’

     S16BIT card = 1, Status = 0;
     U32BIT u3w2MsgsToRead = 500, u32MsgsRead = 0;
     U32BIT u32RxData[5000];

     Status=
acexCanBusRxData(card,1,u32RxData,u32MsgsToRead,&u32MsgsRead);
```

SEE ALSO

acexCanBusConfig() **acexCanBusState()**
acexCanBusTxData()

acexCanBusState

PROTOTYPE

```
#include “control.h”
S16BIT acexCanBusState(S16BIT Card,
                       U8BIT u8Channel
                       CAN_BUS_RUN_STATE eState) ;
```

HARDWARE

BU-67211Ux

PARAMETERS

Card	(input parameter) The card number given to the card in the DDC Card Manager located in the Control Panel.
u8Channel	(input parameter) The CanBUS channel number to program.
eState	(input parameter) Desired States: CAN_BUS_RUN CAN_BUS_READY CAN_BUS_RESET CAN_BUS_PAUSE (Transmitter only)

DESCRIPTION

This function sets the state of a CanBUS channel on a compatible device.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_FEATURE_NOT_SUPPORTED	Feature not Supported on target hardware.
ERR_INVALID_CHANNEL_NO	Invalid/Unknown CanBus channel referenced.
ERR_INVALID_CH_TYPE	Invalid/Unknown Channel Type
ERR_SUCCESS	The function returned successfully.

EXAMPLE

```
// This example will set the State of CanBus channel ‘1’ on Card ‘1’
// to the “RUN” State

     S16BIT card = 1, Status = 0;
     Status = acexCanBusState(1,1,CAN_BUS_RUN);
```

SEE ALSO

acexCanBusConfig() **acexCanBusRxData()**
acexCanBusTxData()

acexCanBusTxData

PROTOTYPE

```
#include “control.h”
S16BIT acexCanBusTxData(   S16BIT Card,
                           U8BIT u8Channel,
                           U32BIT *pTxData,
                           U32BIT u32MsgCount
                           U32BIT u32Option);
```

HARDWARE

BU-67211Ux

PARAMETERS

Card	(input parameter) The card number given to the card in the DDC Card Manager located in the Control Panel.
u8Channel	(input parameter) The CanBUS channel number to transmit on.
pTxData	(input parameter) U32BIT Pointer to Transmit Buffer.
u32MsgCount	(input parameter) Number of Messages to transmit Note: 1 message = 24 bytes fixed.
u32Option	(output parameter) Future Option Expansion (Reserved). Note: No Options defined. Pass in value of 0.

DESCRIPTION

This function loads CanBus Messages into the hardware transmit buffer.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_FEATURE_NOT_SUPPORTED	Feature not Supported on target hardware.
ERR_INVALID_CHANNEL_NO	Invalid/Unknown CanBus channel referenced.
ERR_INVALID_CH_TYPE	Invalid/Unknown Channel Type
CAN_BUS_TX_BUSY	CanBus Transmitter is currently busy.
ERR_SUCCESS	The function returned successfully.

acexCanBusTxData (continued)

EXAMPLE

```
// This example will load CanBus data into CanBus channel ‘1’ on Card
‘1’

     S16BIT card = 1, Status = 0;
     U32BIT u3w2MsgsToWrite = 500
     U32BIT u32TxData[5000];

     Status= acexCanBusTxData(card,1,u32TxData,u32MsgsToWrite,0);
```

SEE ALSO

acexCanBusConfig() **acexCanBusRxData()**
acexCanBusState()

acexArinc717Interrupts

PROTOTYPE

```
#include “control.h”
S16BIT acexArinc717Interrupts (S16BIT Card,
                               U8BIT u8Channel,
                               U8BIT u8GetStatus,
                               U8BIT Enable
                               U32BIT u32Interrupt) ;
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the DDC Card Manager located in the Control Panel.
u8Channel	(input parameter) ARINC 717 Channel.
u8GetStatus	(input parameter) FALSE = Set Interrupt Status
Enable	(input parameter) TRUE = Enable Interrrupt options in ‘u32Interrupt’ FALSE = Disable Interrrupt options in ‘u32Interrupt’
u32Interrupt	(input parameter) Interrupt Mask (Can be logically ‘ored’ together). Valid values are:

ARINC_717_PROG_INT_TX_MARKER0_ENA	Primary Transmit Buffer has been sent. (Load 2nd Buffer)
ARINC_717_PROG_INT_TX_MARKER1_ENA	Secondary Transmit Buffer has been sent (Load 1st Buffer)
ARINC_717_PROG_INT_RX_50_PC_MEM_ENA	Primary Receiver Buffer filled (Read 1st Buffer)
ARINC_717_PROG_INT_RX_100_PC_MEM_ENA	Secondary Receiver Buffer filled (Read 2nd Buffer)
ARINC_717_PROG_INT_RX_REC_SYNCED_ENA	Valid sync word detected when not synced
ARINC_717_PROG_INT_RX_REC_SYNCED_ERR_ENA	Valid sync word not detected at start of frame
ARINC_717_PROG_INT_RX_AUTO_DETECT_LOCK_ENA	Receiver detected correct speed
ARINC_717_PROG_INT_RX_AUTO_DETECT_LOST_ENA	Receiver lost lock on correct speed

DESCRIPTION

This function enables/disables ARINC 717 Interrupt conditions for a given card/channel.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_FEATURE_NOT_SUPPORTED	Feature not Supported on target hardware.
ERR_INVALID_CHANNEL_NO	Invalid/Unknown ARINC 717 channel referenced.
ERR_SUCCESS	The function returned successfully.

acexArinc717Interrupts (continued)

EXAMPLE

```
// This example will configure ARINC 717 interrupts for channel ‘2’

     S16BIT card = 1, Status = 0;
     U32BIT u32Interrupt;

// Set all Rx interrupts for Channel 2
     u32Interrupt =  ARINC_717_PROG_INT_RX_50_PC_MEM_ENA        |
                     ARINC_717_PROG_INT_RX_100_PC_MEM_ENA       |
                      ARINC_717_PROG_INT_RX_REC_SYNCED_ENA      |
                      ARINC_717_PROG_INT_RX_REC_SYNCED_ERR_ENA  |
                      ARINC_717_PROG_INT_RX_BIT_ERR_DETECTED_ENA;

          Status = acexArinc717Interrupts(Card, 2, FALSE, TRUE,
                                          u32Interrupt);
```

SEE ALSO

acexArinc717ProgConfig() **acexArinc717ProgLoadTxData()**
acexArinc717ProgRxData() **acexArinc717ProgState()**
GetIntStatus(

acexArinc717ProgConfig

PROTOTYPE

#include “control.h”
S16BIT **acexArinc717ProgConfig** (S16BIT Card,
PARINC_717_PROGRMMABLE_CONFIGsConfig);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the DDC Card Manager located in the Control Panel.
sConfig	(input parameter) ARINC 717 Channel Configuration to load.

DESCRIPTION

This function configures a ARINC 717 channel on a compatible device.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_FEATURE_NOT_SUPPORTED	Feature not Supported on target hardware.
ERR_INVALID_CHANNEL_NO	Invalid/Unknown ARINC 717 channel referenced.
ERR_SUCCESS	The function returned successfully.

EXAMPLE

```
// This example will configure ARINC 717 channel ‘1’ on Card ‘1’
      S16BIT card = 1, Status = 0;
      ARINC_717_PROGRMMABLE_CONFIG sConfig;

// Configure Transmit channel parameters
      sConfig.u32ConfigOption = ARINC_717_PROGRMMABLE_TYPE_OPT |
      ARINC_717_PROGRMMABLE_FRAME_COUNT_OPT |
            ARINC_717_PROGRMMABLE_PROTOCOL_OPT                    |
      ARINC_717_PROGRMMABLE_SLOPE_OPT;
      sConfig.u8Channel = 1;
      sConfig.u8Type = ARINC717_TX_CHANNEL;
      sConfig.u8ProtocolType = ARINC_717_PROG_HBP_MASK;
      sConfig.u8SlopeControl = ARINC_717_PROG_TX_SLOPE_CONTROL_398PF;
      sConfig.u16FrameCount = 0;

      Status = acexArinc717ProgConfig(Card, sConfig);
```

acexArinc717ProgConfig (continued)

SEE ALSO

acexArinc717Interrupts()
PARINC_717_PROGRMMABLE_CONFIG
acexArinc717ProgState()
acexArinc717ProgLoadTxData()
acexArinc717ProgRxData()
GetIntStatus()

acexArinc717ProgRxData

PROTOTYPE

```
#include “control.h”
S16BIT acexArinc717ProgRxData (S16BIT Card,
                               U8BIT u8Channel
                               U32BIT *pFrameData,
                               U32BIT u32FrameSize,
                               U32BIT u32Option);
```

HARDWARE

Any

PARAMETERS

Card (input parameter)
The card number given to the card in the DDC Card Manager located in the Control Panel.

u8Channel (input parameter)
The ARINC 717 channel number to receive data from.

*pFrameData (input parameter)
Potiner to Array to store 717 Sub-Frame Received Data

u32FrameSize (input parameter)
Maximum Size of the Sub-Frame Frame Data Array (in bytes)

u32Option (input parameter)
Valid Options:

ARINC_717_TX_DATA_PRIMARY_BUFFER_OPT
Read Data from Primary Buffer.

ARINC_717_TX_DATA_SECONDARY_BUFFER_OPT
Read Data from Secondary Buffer.

DESCRIPTION

This function queries a receive buffer for updated Sub-Frame data.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_FEATURE_NOT_SUPPORTED	Feature not Supported on target hardware.
ERR_INVALID_CHANNEL_NO	Invalid/Unknown ARINC 717 channel referenced.
ERR_INVALID_CH_TYPE	Invalid/Unknown Channel Type
ERR_SUCCESS	The function returned successfully.

acexArinc717ProgRxData (continued)

EXAMPLE

```
// This example will receive Sub-Frame Data into the primary buffer

S16BIT card = 1, Status = 0;

U32BIT u32FrameData[32] =
{
 0x0DDC0000,    0x0DDC0111,    0x0DDC0111,    0x0DDC0111,    0x0DDC0111,
 0x0DDC0111,    0x0DDC0111,    0x0DDC0111,    0x00000000,    0x0DDC0111,
 0x0DDC0111,    0x0DDC0111,    0x0DDC0111,    0x0DDC0111,    0x0DDC0111,
 0x0DDC0111,    0x00000000,    0x0DDC0111,    0x0DDC0111,    0x0DDC0111,
 0x0DDC0111,    0x0DDC0111,    0x0DDC0111,    0x0DDC0111,    0x00000000,
 0x0DDC0111,    0x0DDC0111,    0x0DDC0111,    0x0DDC0111,    0x0DDC0111,
 0x0DDC0111, 0x00000000
};

     Status = acexArinc717ProgRxData(Card,1,&u32FrameData,
                                     sizeof(u32FrameData),
                                     ARINC_717_TX_DATA_PRIMARY_BUFFER_OPT);
```

SEE ALSO

acexArinc717Interrupts()
acexArinc717ProgConfig()
acexArinc717ProgLoadTxData()
acexArinc717ProgState()
GetIntStatus()

acexArinc717ProgLoadTxData

PROTOTYPE

```
#include “control.h”
S16BIT acexArinc717ProgLoadTxData (S16BIT Card,
                                   U8BIT u8Channel
                                   U32BIT *pFrameData,
                                   U32BIT u32FrameSize,
                                   U32BIT u32Option);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the DDC Card Manager located in the Control Panel.
u8Channel	(input parameter) The ARINC 717 channel number to program.
*pFrameData	(input parameter) Potiner to Array of 717 Sub-Frame Transmission Data
u32FrameSize	(input parameter) Maximum Size of the Sub-Frame Transmission Array (in bytes)
u32Option	(input parameter) Valid Options: ARINC_717_TX_DATA_PRIMARY_BUFFER_OPT Load Data into Primary Buffer. ARINC_717_TX_DATA_SECONDARY_BUFFER_OPT Load Data into Secondary Buffer.

DESCRIPTION

This function loads a transmit buffer with updated Sub-Frame data.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_FEATURE_NOT_SUPPORTED	Feature not Supported on target hardware.
ERR_INVALID_CHANNEL_NO	Invalid/Unknown ARINC 717 channel referenced.
ERR_INVALID_CH_TYPE	Invalid/Unknown Channel Type
ERR_SUCCESS	The function returned successfully.

acexArinc717ProgLoadTxData (continued)

EXAMPLE

```
// This example will load Sub-Frame Data into the primary buffer

     S16BIT card = 1, Status = 0;

U32BIT u32FrameData[32] =
{
 0x0DDC0000,    0x0DDC0111,    0x0DDC0111,    0x0DDC0111,    0x0DDC0111,
 0x0DDC0111,    0x0DDC0111,    0x0DDC0111,    0x00000000,    0x0DDC0111,
 0x0DDC0111,    0x0DDC0111,    0x0DDC0111,    0x0DDC0111,    0x0DDC0111,
 0x0DDC0111,    0x00000000,    0x0DDC0111,    0x0DDC0111,    0x0DDC0111,
 0x0DDC0111,    0x0DDC0111,    0x0DDC0111,    0x0DDC0111,    0x00000000,
 0x0DDC0111,    0x0DDC0111,    0x0DDC0111,    0x0DDC0111,    0x0DDC0111,
 0x0DDC0111, 0x00000000
};

     Status = acexArinc717ProgLoadTxData(Card,1,&u32FrameData,
                                  sizeof(u32FrameData),
                                  ARINC_717_TX_DATA_PRIMARY_BUFFER_OPT);
```

SEE ALSO

acexArinc717Interrupts() **acexArinc717ProgConfig()**
acexArinc717ProgRxData() **acexArinc717ProgState()**
GetIntStatus()

acexArinc717ProgState

PROTOTYPE

#include “control.h”
S16BIT **acexArinc717ProgState** (S16BIT Card,
U8BIT u8Channel
ARINC_717_STATE eState) ;

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the DDC Card Manager located in the Control Panel.
u8Channel	(input parameter) The ARINC 717 channel number to program.
eState	(input parameter) Desired States: ARINC_717_RUN ARINC_717_READY ARINC_7171_RESET ARINC_717_PAUSE (Transmitters only)

DESCRIPTION

This function sets the state of a ARINC 717 channel on a compatible device.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_FEATURE_NOT_SUPPORTED	Feature not Supported on target hardware.
ERR_INVALID_CHANNEL_NO	Invalid/Unknown ARINC 717 channel referenced.
ERR_INVALID_CH_TYPE	Invalid/Unknown Channel Type
ERR_SUCCESS	The function returned successfully.

EXAMPLE

```
// This example will set the State of ARINC 717 channel ‘1’ on Card
‘1’to
// the “RUN” State

     S16BIT card = 1, Status = 0;
     Status = acexArinc717ProgState(1,1,ARINC_717_RUN);
```

SEE ALSO

acexArinc717Interrupts()
acexArinc717ProgConfig()
acexArinc717ProgLoadTxData()
acexArinc717ProgRxData()
GetIntStatus()

AddFilter

PROTOTYPE

```
#include “Receive.h”
short AddFilter (short Card,
                 short Receiver,
                 short LabelSdi);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The desired receiver channel number.
LabelSdi	(input parameter) The label (bits 0..7), and SDI (bits 8..9)

DESCRIPTION

This function adds a filter to the specified receiver. Filtering will be limited to label value only if IRIG time tagging is enabled.

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_RX	Invalid receiver defined.
ERR_LABELSDI	Invalid label and SDI
NUMBER	Zero if the filter already exists. A one if the filter was added successfully

EXAMPLE

```
//This will add a filter for a label of 255 and an SDI of 3 to rx 1 of
//card 1.
      short card = 1;
      short receiver = 1;
      short labelsdi = 1023;
      addfilter_return = AddFilter (card, receiver, labelsdi);
```

SEE ALSO

DelFilter()
ClearFilter()
GetNumOfFilter()
GetAllFilter()
EnableFilter()
GetFilterStatus()
GetFilter()

AddRepeated

PROTOTYPE

```
#include “Transmit.h”
short AddRepeated (short Card,
                   short Transmitter,
                   unsigned long Data,
                   short Frequency,
                   short Offset);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Transmitter	(input parameter) The transmitter
Data	(input parameter) The ARINC word including the label and SDI.
Frequency	(input parameter) Defines the frequency of the transmission in number of milliseconds. The word will be transmitted every frequency milliseconds (1..32767).
Offset	(input parameter) The offset in milliseconds relative to the other scheduled words. This value must be less than the Frequency parameter (1..32767).

DESCRIPTION

This function schedules a repeated transmission of the word with the specified label and SDI (source-destination identifier) by putting the word into the transmitter’s schedule table. The system will automatically copy the word from the schedule table to the queue every *Frequency* milliseconds, so the word will be transmitted every *Frequency* milliseconds. If the transmitter schedule table contains another word with the same label and SDI, this new one will replace the old one.

For example, **AddRepeated(1, 1, 0x12345678, 3, 0)** transmits the word 0x12345678 every 3 milliseconds, and **AddRepeated(1, 1, 0x90ABCDEF, 4, 0)** transmits the word 0x90ABCDEF every 4 milliseconds. The transmission of 0x12345678 will be stopped and replaced by 0x53124278 if you run **AddRepeated(1, 1, 0x53124278, 4, 2)** since 0x12345678 and 0x53124278 have the same label (0x78) and same SDI (2, in bits 8 and 9 if in the original bit format). Although 0x90 and 0x53124278 will have the same frequency (once every 4 milliseconds = 250 Hz), the two words will not be transmitted in the same millisecond because of their different offsets: 0x90ABCDEF will be transmitted every first (offset 0) millisecond, while 0x53124278 will be transmitted every third (offset 2) millisecond.

AddRepeated (continued)

To cancel the scheduled transmission of a label and SDI you must call the **DelRepeated()** function. To cancel all scheduled transmissions you must call the **ClearRepeated()** function. This function does not enable the transmitter if it is disabled, and therefore if the transmitter becomes disabled, the scheduled transmissions will be paused and will be restored when the transmitter is enabled again. Both the queue size and the transmission speed limit the maximum number of scheduled labels in the system. If there are too many words in the tables or if the repeating is too fast, the software may become dead because the DLL cannot do anything except copying the schedule table to the queue (the DLL cannot escape from the scheduling interrupt handler). If too many words are already in the queue by the functions **LoadTxQueueOne()** and **LoadTxQueueMore()**, the repeated transmission may not be on time because the queue may become full so that the word from the schedule table cannot enter the queue.

Note: *There is a zero-based location index in the schedule table. If the word replaced an existing one, the location will be the same. If the word does not replace an existing one, it will be appended to the end of the table.*

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter defined
ERR_LABELSDI	Invalid label or SDI
ERR_FREQ	Invalid frequency
ERR_OFFSET	Invalid offset
ERR_ENABLE	Transmitter not enabled
ERR_TXQUEUESZ	Queue overflows
NUMBER	The total number of words in the schedule table that exist before the current entry.

EXAMPLE

```
//This example sets up the repeated transmission of the word
//0x12345678 on transmitter one of card one every three ms.

          unsigned long data = 0x12345678;
          short card = 1;
          short transmitter = 1;
          short frequency = 3;
          short offset = 0;
          short addrepeated_return = 0;

          addrepeated_return = AddRepeated (card,
                                            transmitter,
                                            data,
                                            frequency,
                                            offset);
```

AddRepeated (continued)

SEE ALSO

GetNumOfRepeated()
GetAllRepeated()
ClearRepeated()
GetRepeated()
DelFilter()
dd429X_AddRepeated()

ClearFilter

PROTOTYPE

```
#include “Receive.h”
short  ClearFilter (short Card,
                    short Receiver);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receive channel number.

DESCRIPTION

This function deletes a receiver’s filters. If the filter usage is on and all filters are deleted, no word will be received by the queue. You can do this if you want to use the mailbox only.

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_RX	Invalid receiver defined
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example will get rid of all of the label/sdi filters that are
//specified for receiver 1 of card 1.
      short card = 1;
      short receiver = 1;
      short clearfilter_return = 0;
      clearfilter_return = ClearFilter (card, receiver);
```

SEE ALSO

DelFilter()
AddFilter()
GetNumOfFilter()
GetAllFilter()
EnableFilter()
GetFilterStatus()
GetFilter()

ClearMailbox

PROTOTYPE

```
#include “Receive.h”
short ClearMailbox  (short Card,
                     short Receiver);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receive channel number).

DESCRIPTION

This function clears all slots in a receiver’s mailbox. It is equivalent to reading all slots once. The data in the mailbox slots still exist but will be marked as “old” data.

RETURN VALUE

ERR_CRDINIT	Card is not initialized
ERR_RX	Invalid receiver defined
ERR_MODE	Invalid operating mode
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example will mark all of the data in the mailbox slots of card
//1 on receiver 1 as old data
short card = 1;
short receiver = 1;
short clearmailbox_return = 0;
clearmailbox_return = ClearMailbox (card, receiver);
```

SEE ALSO

ReadMailboxIrig() **GetMailboxStatus()**
GetMailbox()

ClearRepeated

PROTOTYPE

```
#include “Transmit.h”
short ClearRepeated (short Card,
                     short Transmitter);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel
Transmitter	(input parameter) The transmit channel number.

DESCRIPTION

This function clears the transmitter’s schedule table and cancels the repeated transmissions of all words by this transmitter.

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter defined
ERR_SUCCESS (0)	The function returned successfully

EXAMPLE

```
//This example will clear the schedule table on transmitter 1 of card
1.
      short card = 1
      short transmitter = 1;
      short clearrepeated_return = 0;
      clearrepeated_return = ClearRepeated (card, transmitter);
```

SEE ALSO

GetNumOfRepeated()
GetRepeated()
GetAllRepeated()
DelRepeated()
AddRepeated()

ClearRxQueue

PROTOTYPE

#include “Receive.h”
short **ClearRxQueue** (short Card,
short Receiver);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receive channel number.

DESCRIPTION

This function clears the receiver’s queue.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid receiver defined.
ERR_MODE	Invalid operating mode
NUMBER	A zero means the function was successful.

EXAMPLE

```
//This example will clear the queue on receiver 1 of card 1.
      short card = 1;
      short receiver = 1;
      short clearrxqueue_return = 0;
      clearrxqueue_return = ClearRxQueue (card, receiver);
```

SEE ALSO

GetRxQueueStatus() **ReadRxQueueIrigOne()**
ReadRxQueueIrigMore()

ConfigFilter

PROTOTYPE

#include “Receive.h”
short **ConfigFilter** (short Card,
short Receiver,
short Mode);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receive channel number.
Mode	(input parameter) Bitwise OR combination of filter criteria: FILTER_LABEL_SDI FILTER_PARITY_ERR FILTER_STALE_MSG

DESCRIPTION

This function configures the filter of the receive channel.

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_RX	Invalid receiver defined
ERR_LABELSDI	Invalid LABEL or SDI
ERR_MODE	Invalid operating mode
ERR_FILTER	Not enough available filters.
NUMBER	A zero means the filter already exists. A one means the function configured the filter correctly without any issues.

EXAMPLE

```
//This example will configure the filter for parity errors on receiver
//4,card 1.
          short card = 1;
          wStatus = += ConfigFilter(Card, 4, FILTER_PARITY_ERR);
```

ConfigFilter (continued)

SEE ALSO

AddFilter()
ClearFilter()
GetNumOfFilter()
GetAllFilter()
EnableFilter()
GetFilterStatus()
GetFilter()
DelFilter()

ConfigTimeStamp

PROTOTYPE

```
#include “Receive.h”
short ConfigTimeStamp (short Card,
                       unsigned char Format,
                       unsigned char Rollover,
                       unsigned char Resolution);
```

HARDWARE

Any

PARAMETERS

Card (input parameter)
The card number given to the card in the Card Manager in the Control Panel.

Format (input parameter)
Time Tag Format.
Valid Values:

TT48	48-Bit Timer
IRIG_B	IRIG B Format

Rollover (input parameter)
48-Bit Time Tag rollover value:

TTRO_16	2^{16}
TTRO_17	2^{17}
TTRO_18	2^{18}
TTRO_19	2^{19}
TTRO_20	2^{20}
TTRO_21	2^{21}
TTRO_22	2^{22}
TTRO_48	2^{48}

Resolution (input parameter)
48-Bit LSB Resolution:

TTRES_1	1µS
TTRES_2	2µS
TTRES_4	4µS
TTRES_8	8µS
TTRES_16	16µS
TTRES_32	32µS
TTRES_64	64µS

ConfigTimeStamp (continued)

DESCRIPTION

This function configures a card's time stamp feature.

Special Note: *When operating the* ***BU-65590/91Ux***, ***BU-65590F/Mx***, ***BU-65590Cx*** *DDC cards, the ARINC TimeTag selection will override the Relative Time counter selection in the 1553 section. If IRIG is selected on the 1553 and the ARINC then selects the global 48-bit counter, the 1553 will be modified to use the global 48-bit counter as well.*

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TTFORMAT	Invalid time tag format
ERR_TTRO	Invalid time tag roll over.
ERR_TTRES	Invalid time tag resolution.
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example will configure the time stamp for a 48 bit time stamp
// with a rollover of 2^48 and a resolution of 1 µsecond.

     short card = 1;
     wStatus = += ConfigTimeStamp(Card,
                                  TT48,
                                  TTRO_48,
                                  TTRES_1);
```

SEE ALSO

None

dd429X_AddRepeated

PROTOTYPE

```
#include “Transmit.h”
S16BIT dd429X_AddRepeated(
                    S16BIT s16Card,
                    S16BIT s16Transmitter,
                    U32BIT u32Data,

DD429_TESTER_OPTIONS_TYPE *psTesterOptions,
                    S16BIT s16Frequency,
                    S16BIT s16Offset);
```

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager
s16Transmitter	(input parameter) The transmit channel to access
u32Data	(input parameter) 32-bit ARINC dataword to transmit
psTesterOptions	(input parameter) Pointer to **DD429_TESTER_OPTIONS_TYPE** tester options structure *{ Inter-Word Bit Gap Error Injection,* *Word Size Error,* *Parity Error,* *Set bit 33 Error }}*
s16Frequency	(input parameter) Defines the frequency of transmission of the ARINC word in number of milliseconds
s16Offset	(input parameter) The offset in milliseconds relative to the other scheduled words. This value must be less than the Frequency parameter.

dd429X_AddRepeated (continued)

DESCRIPTION

This function schedules a repeated transmission of the ARINC word using the specified label and SDI (source-destination identifier). The word is added into the transmitter's schedule table. The frequency of transmission of the ARINC word is determined by the *s16Frequency* parameter.

This function is the same as the legacy **AddRepeated()** function, with the with the addition of options for error injection as defined by the **DD429_TESTER_OPTIONS_TYPE** input structure.

***Note**: If the transmitter's schedule table contains another word with the same label and SDI, that word will be replaced by the one specified in the u32Data parameter.*

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter defined
ERR_ENABLE	Channel not enabled
ERR_FREQ	Invalid Frequency parameter defined
ERR_OFFSET	Invalid Offset parameter defined
ERR_NULL	NULL pointer defined
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

EXAMPLE

```
//This example sets up the repeated transmission of the word
//0x12345678 on transmitter one of card one every three ms.
     S16BIT s16Card = 1;
     S16BIT s16Transmitter = 1;
     U32BIT u32Data = 0x12345678;
     DD429_TESTER_OPTIONS_TYPE psTesterOptions = {0x0000, 0x00, 0x00,
     0x00};
     S16BIT s16Frequency = 3;
     S16BIT s16Offset = 0;

     s16Result = dd429X_AddRepeated(s16Card,
                                    s16Transmitter,
                                    u32Data,
                                    psTesterOptions,
                                    s16Frequency,
                                    s16Offset);

   if(s16Result != ERR_SUCCESS)
     {
        printf("Error in dd429X_AddRepeated() function \n");
        PrintErrorMsg(s16Result);
        return 1;
     }
```

dd429X_AddRepeated (continued)

SEE ALSO

GetNumOfRepeated()
DelFilter()
ClearRepeated()
dd429X_ModifyRepeatedData()
GetAllRepeated()
AddRepeated()
dd429X_GetRepeated()
DD429_TESTER_OPTIONS_TYPE

dd429X_AddRepeatedItem

PROTOTYPE

S16BIT **dd429X_AddRepeatedItem**(
S16BIT s16Card,
S16BIT s16Transmitter,
U32BIT u32Data,
DD429_TESTER_OPTIONS_TYPE *psTesterOptions,
S16BIT s16Frequency,
S16BIT s16Offset,
U16BIT u16ItemIndex);

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager.
s16Transmitter	(input parameter) The transmit channel to access.
u32Data	(input parameter) 32-bit ARINC dataword to transmit.
psTesterOptions	(input parameter) Pointer to tester options structure. *{ Inter-Word Bit Gap Error Injection,* *Word Size Error,* *Parity Error,* *Set bit 33 Error }}*
s16Frequency	(input parameter) Defines the frequency of transmission of the ARINC word in number of milliseconds.
s16Offset	(input parameter) The offset in milliseconds relative to the other scheduled words. This value must be less than the Frequency parameter.
u16ItemIndex	(input parameter) User index assigned to the scheduled data.

dd429X_AddRepeatedItem (continued)

DESCRIPTION

This function schedules a repeated transmission by putting the *u32Data* into the transmitter's schedule table. The system will automatically copy the *u32Data* from the schedule table to the queue every *s16Frequency* milliseconds, so the word will be transmitted every *s16Frequency* milliseconds. The item can subsequently be accessed by using the returned *16ItemIndex* as an identifier. It is possible to place multiple *u32Data* in to the schedule table that have the same Label and SDI defined.

NOTE: We leave it up to the user to keep track of the available indexes that are currently being used. **dd429X_GetAllRepeatedItem()** can be used to retrieve an array of all active indexes. If **dd429X_AddRepeatedItem()** is called with an index that already exists in the scheduler, then the new message will replace the existing indexed message. However, message timing will not be preserved for the replaced index. The new message will be added to the scheduler at the most current point in the scheduler cycle. To replace the ARINC message while preserving message timing, please use function **dd429X_ModifyRepeatedDataItem()**.

RETURN VALUE

A positive return code signifies that an existing message index has been overwritten. In this case, the function will return with the number of messages that are currently in the scheduler.

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid Transmitter
ERR_ENABLE	Channel is not in enabled state
ERR_FREQ	Invalid Frequency
ERR_OFFSET	Invalid Offset
ERR_NULL	NULL pointer defined
ERR_ TXQUEUESZ	Invalid transmitter buffer size
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

EXAMPLE

```
//This example sets up the repeated transmission of the word
//0x12345678 on transmitter one of card one every three ms.
     S16BIT s16Card = 1;
     S16BIT s16Transmitter = 1;
     U32BIT u32Data = 0x12345678; \\data 0x123456 Label170 SDI10
     DD429_TESTER_OPTIONS_TYPE psTesterOptions = {0, 0, 0, 0};
     S16BIT s16Frequency = 3;
     S16BIT s16Offset = 0;
     U16BIT u16ItemIndex = 1;
```

dd429X_AddRepeatedItem (continued)

```
        s16Result = dd429X_AddRepeatedItem(s16Card,
                                           s16Transmitter,
                                           u32Data,
                                           &psTesterOptions,
                                           s16Frequency,
                                           s16Offset,
                                           u16ItemIndex);

      if(s16Result != ERR_SUCCESS)
        {
           printf("Error in dd429X_AddRepeatedItem() function \n");
           PrintErrorMsg(s16Result);

           return 1;
        }
```

SEE ALSO

dd429X_DelRepeatedItem()
dd429X_GetAllRepeatedItem()
dd429X_GetRepeatedItem()
dd429X_ModifyRepeatedDataItem()

dd429X_AddTxFrame

PROTOTYPE

```
#include “Transmit.h”
S16BIT dd429X_AddTxFrame (
                S16BIT Card,
                S16BIT s16Transmitter,
                DD429_TX_MINOR_FRAME_PAYLOAD_TYPE *psMinorFramePayload[],
                U32BIT u32PayloadCount,
                U32BIT u32FrameInterval);
```

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

Card	(input parameter) Logical device number defined in Card Manager
s16Transmitter	(input parameter) The transmit channel to access
psMinorFramePayload[]	(input parameter) Pointer to array of payload data (data + error injection) *{ 32-bit ARINC dataword,* *{ Inter-Word Bit Gap Error Injection,* *Word Size Error,* *Parity Error,* *Set bit 33 Error }}*
u32PayloadCount	(input parameter) Number of psMinorFramePayload entries
u32FrameInterval	(input parameter) Minor frame interval

DESCRIPTION

This function is used for the creation of a minor frame. Messages can be added by creating an array of type **DD429_TX_MINOR_FRAME_PAYLOAD_TYPE** structure, which contains an array of ARINC 429 datawords + error injection settings.

A 1K message FIFO is available for the creation of transmit messages. Minor frames can be added as long as the output FIFO is not exceeded. The status of the output buffer can be determined by calling the **dd429X_GetTxFrameInfo()** function. Only one Major Frame can be created per channel.

Note: This function cannot be called when the channel is already transmitting.

dd429X_AddTxFrame (continued)

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter defined
ERR_ENABLE	Transmitter not enabled
ERR_INTER_WORD_GAP	Invalid Inter Word Bit Gap parameter
ERR_WORD_SIZE	Invalid Word Size parameter
ERR_PARITY	Invalid Parity parameter
ERR_BIT_33	Invalid Extra Bit (bit 33) parameter
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

EXAMPLE

```
//This example sets up the creation of a transmit minor frame
S16BIT Card = 1; /* Card Number 1 */
S16BIT s16Result = 0;
S16BIT s16Transmitter = 1; /* Channel Number 1 */
U32BIT u32PayloadCount = 3; /* no. of messages in minor frame */
U32BIT u32FrameInterval = 1000;

//setup messages for the minor frame
//{transmit_data, {error injection settings}}
     DD429_TX_MINOR_FRAME_PAYLOAD_TYPE User_Payload[3] = {
          /***data***/   /****error injection****/
          { 0x12345671, {0x0000, 0x00, 0x00, 0x00}},
          { 0x00101112, {0x0000, 0x00, 0x00, 0x00}},
          { 0x00101113, {0x0000, 0x00, 0x00, 0x00}}
     };

s16Result = dd429X_AddTxFrame(
     Card, /* User configured card number */
     s16Transmitter, /* Transmit Channel */
     User_Payload, /* Payload of 32-bit ARINC messages */
     u32PayloadCount, /* Number of Messages in Minor Frame*/
     u32FrameInterval /* Frame Interval */
     );

    if(s16Result != ERR_SUCCESS)
    {
        printf("Error in dd429X_AddTxFrame() function \n");
        PrintErrorMsg(s16Result);
        return 1;
    }
```

SEE ALSO

dd429X_GetTxFrameInfo() **DD429_TX_MINOR_FRAME_PAYLOAD_TYPE**

dd429X_ConfigRepeater

PROTOTYPE

S16BIT **dd429X_ConfigRepeater**(
S16BIT s16Card,
S16BIT s16Receiver,
S16BIT s16Transmitter,
U8BIT u8Option);

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager.
s16Receiver	(input parameter) The receive channel to access.
s16Transmitter	(input parameter) The transmit channel to access.
u8Option	(input parameter) Options to map or unmap the repeater. DD429_UNMAP or DD429_MAP

DESCRIPTION

This function configures the s16Transmitter channel to act as a data repeater. This function maps the s16Receiver to the s16Transmitter. The u8Option parameter is used to map or unmap the transmit channel as a transmitter. A transmitter can have multiple receivers as sources. A single receiver can be the source for multiple transmitters.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter specified
ERR_ENABLE	Channel is not in enabled state
ERR_RX	Invalid receiver specified
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card
ERR_REPEATER_OPTION	Invalid repeater option received

EXAMPLE

```
//This example sets up channel 2 as a repeater
//ARINC429 data will be mapped from receiver 1 to transmitter 2
     S16BIT s16Card = 1;
     S16BIT s16Receiver = 1;
     S16BIT s16Transmitter = 2;
```

dd429X_ConfigRepeater (continued)

```
            U8BIT u8Option = DD429_MAP;

            s16Result = dd429X_ConfigRepeater(s16Card,
                                              s16Receiver,
                                              s16Transmitter,
                                              u8Option);

          if(s16Result != ERR_SUCCESS)
            {
               printf("Error in dd429X_ConfigRepeater() function \n");
               PrintErrorMsg(s16Result);
               return 1;
            }
```

SEE ALSO

dd429X_GetRepeaterMode() **dd429X_SetRepeaterMode()**

dd429X_DelRepeatedItem

PROTOTYPE

S16BIT **dd429X_DelRepeatedItem**(
S16BIT s16Card,
S16BIT s16Transmitter,
U16BIT u16ItemIndex);

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager.
s16Transmitter	(input parameter) The transmit channel to access.
u16ItemIndex	(input parameter) User index assigned to the scheduled data.

DESCRIPTION

This function will delete a previously defined message from the scheduler based on the message's index. Note. Once an indexed message has been deleted it, its timing will not be preserved when the index is added back to the scheduler. Instead, the new message will be added to the scheduler at the most current point in the scheduler cycle. To preserve message timing, please use function **dd429X_ModifyRepeatedDataItem()**.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid Transmitter
ERR_ENABLE	Channel is not in enabled state
ERR_INDEX	Invalid index parameter

EXAMPLE

```
//This example will remove message with index 1
//from the scheduler
      S16BIT s16Card = 1;
      S16BIT s16Transmitter = 1;
      U16BIT u16ItemIndex = 1;
```

dd429X_DelRepeatedItem (continued)

```
s16Result = dd429X_DelRepeatedItem(s16Card,
                                   s16Transmitter,
                                   u16ItemIndex);

if(s16Result != ERR_SUCCESS)
  {
     printf("Error in dd429X_DelRepeatedItem() function \n");
     PrintErrorMsg(s16Result);

     return 1;
  }
```

SEE ALSO

dd429X_AddRepeatedItem()
dd429X_GetAllRepeatedItem()
dd429X_GetRepeatedItem()
dd429X_ModifyRepeatedDataItem()

dd429X_GetAmplitude

PROTOTYPE

```
#include “Transmit.h”
S16BIT dd429X_GetAmplitude (
                S16BIT s16Card,
                S16BIT s16Transmitter,
                U32BIT *u32Amplitude);
```

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager
s16Transmitter	(input parameter) The transmit channel to access
u32Amplitude	(output parameter) Pointer to storage for transmit amplitude (!=NULL) 0x00 – 0xFF

DESCRIPTION

This function gets the voltage amplitude that was set for a particular transmit channel.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter defined
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

EXAMPLE

```
/* This function gets the voltage amplitude that was set for a given
Tx Channel */

S16BIT s16Card = 1;
S16BIT s16Result = 0;
S16BIT s16Transmitter = 1;
U32BIT u32Amplitude;

s16Result = dd429X_GetAmplitude(s16Card, s16Transmitter,
&u32Amplitude);
```

dd429X_GetAmplitude (continued)

```
if(s16Result != ERR_SUCCESS)
{
    printf("Error in dd429X_GetAmplitude() function \n");
    PrintErrorMsg(s16Result);
    return 1;
}
```

SEE ALSO

dd429X_SetAmplitude()

dd429X_GetIRIGTx

PROTOTYPE

#include “Transmit.h”
S16BIT **dd429X_GetIRIGTx**(
S16BIT s16Card,
DDC_IRIG_TX_TYPE *sIRIG);

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager
sIRIG	(input parameter) Pointer to **PDDC_IRIG_TX_TYPE**structure to store IRIG-B information

DESCRIPTION

This function gets the current status of the IRIG transmitter.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_NULL	NULL pointer defined
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

EXAMPLE

```
/* This example determines if the device supports IRIG timing */

DDC_IRIG_TX_TYPE sIRIG;

s16Result = GetIRIGTx(DevNum, &sIRIG);

if(sIRIG.u16IRIGBTxSupported == 1))
{
   if(s16Result != ERR_SUCCESS)
   {
       printf("GetIRIGTx Failed!\n");
       PrintOutError(s16Result);
   }
}
else
{
       printf("IRIG Transmitter not supported.\n");
}
```

dd429X_GetIRIGTx (continued)

SEE ALSO

dd429X_SetIRIGTx() **PDDC_IRIG_TX_TYPE**

dd429X_GetRepeated

PROTOTYPE

```
#include “Transmit.h”
S16BIT dd429X_GetRepeated(
                    S16BIT s16Card,
                    S16BIT s16Transmitter,
                    S16BIT s16LabelSdi,
                    U32BIT *pu32Data,
                    DD429_TESTER_OPTIONS_TYPE *psTesterOptions,
                    S16BIT *s16Frequency,
                    S16BIT *s16Offset);
```

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager
s16Transmitter	(input parameter) The transmit channel to access
s16LabelSdi	(input parameter) Label/SDI of the ARINC message to get
pu32Data	(output parameter) Pointer to store the 32-bit ARINC dataword
psTesterOptions	(output parameter) Pointer to **DD429_TESTER_OPTIONS_TYPE** tester options structure *{ Inter-Word Bit Gap Error Injection,* *Word Size Error,* *Parity Error,* *Set bit 33 Error }}*
s16Frequency	(output parameter) Pointer to store the transmit frequency
s16Offset	(output parameter) Pointer to store the transmit offset

DESCRIPTION

This function determines whether or not a specific label and SDI are in the transmitter’s schedule table. If they are, the function returns the ARINC word, the tester options, the frequency in milliseconds, and the offset.

This function is the same as the legacy **GetRepeated()** function, with the with the addition of options to get the error injection information as defined by the **DD429_TESTER_OPTIONS_TYPE** output structure.

dd429X_GetRepeated (continued)

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter defined
ERR_ENABLE	Channel not enabled
ERR_FREQ	The Frequency could not be determined
ERR_OFFSET	The Offset could not be determined
ERR_NULL	NULL Pointer defined
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

EXAMPLE

```
//This example returns whether or not a label of 5 and an SDI of 0 are
//in the transmitter's schedule table for transmitter 1 of card 1. The
//function then gets the ARINC word, the frequency in milliseconds and
//the offset.
     S16BIT s16Card = 1;
     S16BIT s16Transmitter = 1;
     s16LabelSdi = 5;
     U32BIT pu32Data;
     DD429_TESTER_OPTIONS_TYPE psTesterOptions;
     S16BIT s16Frequency;
     S16BIT s16Offset;

     s16Result = dd429X_GetRepeated(s16Card,
                                    s16Transmitter,
                                    s16LabelSdi,
                                    &pu32Data,
                                    &psTesterOptions,
                                    &s16Frequency,
                                    &s16Offset);

     if(s16Result != ERR_SUCCESS)
     {
        printf("Error in dd429X_GetRepeated() function \n");
        PrintErrorMsg(s16Result);
        return 1;
     }
```

SEE ALSO

GetNumOfRepeated()
GetRepeated()
GetAllRepeated()
DelFilter()
ClearRepeated()
dd429X_AddRepeated()
dd429X_ModifyRepeatedData()
DD429_TESTER_OPTIONS_TYPE

dd429X_GetRepeatedItem

PROTOTYPE

S16BIT **dd429X_GetRepeatedItem**(
S16BIT s16Card,
S16BIT s16Transmitter,
U16BIT u16ItemIndex,
U32BIT *u32Data,
DD429_TESTER_OPTIONS_TYPE *psTesterOptions,
S16BIT *s16Frequency,
S16BIT *s16Offset,
);

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager.
s16Transmitter	(input parameter) The transmit channel to access.
u16ItemIndex	(input parameter) User index assigned to the scheduled data.
u32Data	(output parameter) 32-bit ARINC dataword to transmit.
psTesterOptions	(output parameter) Pointer to tester options structure. *{ Inter-Word Bit Gap Error Injection,* *Word Size Error,* *Parity Error,* *Set bit 33 Error }}*
s16Frequency	(output parameter) Defines the frequency of transmission of the ARINC word in number of milliseconds.
s16Offset	(output parameter) The offset in milliseconds relative to the other scheduled words.

dd429X_GetRepeatedItem (continued)

DESCRIPTION

This function returns information regarding a scheduled message by using the message index.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid Transmitter
ERR_ENABLE	Channel is not in enabled state
ERR_INDEX	Invalid index parameter

EXAMPLE

```
//This example will get detailed information for index 1
      S16BIT s16Card = 1;
      S16BIT s16Transmitter = 1;
      U16BIT u16ItemIndex = 1;
      U32BIT retData;
      DD429_TESTER_OPTIONS_TYPE retOptions;
      S16BIT retFrequency;
      S16BIT retOffset;

      s16Result = dd429X_GetRepeatedItem(s16Card,
                                         s16Transmitter,
                                         u16ItemIndex,
                                         &retData,
                                         &retOptions,
                                         &retFrequency,
                                         &retOffset);

    if(s16Result != ERR_SUCCESS)
      {
         printf("Error in dd429X_GetRepeatedItem() function \n");
         PrintErrorMsg(s16Result);
         return 1;
      }
```

SEE ALSO

dd429X_AddRepeatedItem()
dd429X_DelRepeatedItem()
dd429X_GetAllRepeatedItem()
dd429X_ModifyRepeatedDataItem()

dd429X_GetAllRepeatedItem

PROTOTYPE

S16BIT **dd429X_GetAllRepeatedItem**(
S16BIT s16Card,
S16BIT s16Transmitter,
U16BIT u16ItemIndexBufferSize,
U16BIT *pu16ItemIndex);

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card (input parameter)
Logical device number defined in Card Manager.

s16Transmitter (input parameter)
The transmit channel to access.

u16ItemIndexBufferSize
(input parameter)
Size of user buffer.

pu16ItemIndex (output parameter)
Pointer to user buffer in which to store returned indexes.

DESCRIPTION

Returns all active indexes in the scheduler into a user array specified by pu16ItemIndex.

RETURN VALUE

If successful, the function will return with the number of messages that are currently in the scheduler.

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid Transmitter
ERR_ENABLE	Channel is not in enabled state
ERR_NULL	NULL pointer defined

EXAMPLE

```
//This example will get all active indexes and put them in the
// user specified buffer defined by pu16ItemIndex[1024]
     S16BIT s16Card = 1;
     S16BIT s16Transmitter = 1;
     U16BIT u16ItemIndexBufferSize = 1024;
     U16BIT pu16ItemIndex[1024]={0};
     U32BIT counter = 0;
```

dd429X_GetAllRepeatedItem (continued)

```
//make sure to clear the array before filling it
memset(pu16ItemIndex, 0, sizeof(pu16ItemIndex));
s16Result = dd429X_GetAllRepeatedItem(s16Card,
                                      s16Transmitter,
                                      u16ItemIndexBufferSize,
                                      pu16ItemIndex);

    if(s16Result > 0)
      {
         for(counter=0; counter < s16Result; counter++)
         printf("index:   %d   is   currently   being   used   \n",
          pu16ItemIndex[counter]);
         return 1;
      }
```

SEE ALSO

dd429X_AddRepeatedItem()
dd429X_DelRepeatedItem()
dd429X_GetRepeatedItem()
dd429X_ModifyRepeatedDataItem()

dd429X_GetRepeaterMode

PROTOTYPE

```
S16BIT dd429X_GetRepeaterMode(
                S16BIT s16Card,
                S16BIT s16Receiver,
                U16BIT  u16LabelSdi,
                U32BIT *pu32Mode,
                U32BIT *pu32Value);
```

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager.
s16Receiver	(input parameter) The receive channel to access.
u16LabelSdi	(input parameter) The Label/SDI parameter for which to return repeater information.
pu32Mode	(output parameter) Pointer to store returned mode for the Label/SDI specified.
pu32Value	(output parameter) Pointer to store returned value for the Label/SDI specified.

DESCRIPTION

This function will get the current data pollution options for a particular repeater source channel. The message information can be obtained via the specified label/SDI. Please see function **dd429X_SetRepeaterMode()** for a list of modes and their hexadecimal values.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_ENABLE	Channel is not in enabled state
ERR_RX	Invalid receiver specified
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card
ERR_REPEATER_OPTION	Invalid repeater option received
ERR_NULL	NULL pointer defined
ERR_LABELSDI	Invalid Label/SDI specified

EXAMPLE

```
//This example will return the current data pollution settings
```

```
//for label 246 for source receiver 1
```

dd429X_GetRepeaterMode (continued)

```
S16BIT s16Card = 1;
S16BIT s16Receiver = 1;
U16BIT u16LabelSdi = 0xa6;
U32BIT pu32Mode, pu32Value;

s16Result = dd429X_GetRepeaterMode(S16BIT s16Card,
                                   s16Receiver,
                                   u16LabelSdi,
                                   &pu32Mode,
                                   &pu32Value);

if(s16Result != ERR_SUCCESS)
  {
     printf("Error in dd429X_GetRepeaterMode() function \n");
     PrintErrorMsg(s16Result);
     return 1;
  }
```

SEE ALSO

dd429X_ConfigRepeater() **dd429X_SetRepeaterMode()**

dd429X_GetTxQueueFreeCount

PROTOTYPE

S16BIT **dd429X_GetTxQueueFreeCount**(
S16BIT s16Card,
S16BIT s16Transmitter);

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager.
s16Transmitter	(input parameter) The transmit channel to access.

DESCRIPTION

This function is designed to help the user manage the transmit FIFO. It will return with number of available slots in the message FIFO.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid Transmitter
ERR_ENABLE	Channel is not in enabled state

EXAMPLE

```
//This example will return with the number of available slots in
//the message FIFO
      S16BIT s16Card = 1;
      S16BIT s16Transmitter = 1;
      S16BIT  LabelsAvailable ;
      LabelsAvailable = dd429X_GetTxQueueFreeCount(s16Card,
                                        s16Transmitter);

    if(LabelsAvailable < 1)
      {
         printf("Error in dd429X_GetTxQueueFreeCount() function \n");
         PrintErrorMsg(s16Result);
         return 1;
      }
```

SEE ALSO

dd429X_LoadTxQueueOne() **dd429X_LoadTxQueueMore()**

dd429X_GetTxFrameInfo

PROTOTYPE

```
#include “Transmit.h”
S16BIT dd429X_GetTxFrameInfo(S16BIT s16Card,
                             S16BIT s16Transmitter,
                             DD429TX_FRAME_INFO_TYPE *pTxFrameInfo);
```

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager
s16Transmitter	(input parameter) The transmit channel to access
pTxFrameInfo	(output parameter) Pointer to a **DD429TX_FRAME_INFO_TYPE** structure to store the current frame information

DESCRIPTION

This function is used to determine the status of the scheduled transmit FIFO in terms of percentage used.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter defined
ERR_ENABLE	Channel not enabled
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

dd429X_GetTxFrameInfo (continued)

EXAMPLE

```
//This example will print, to the screen, how much of the
//scheduled TX FIFO is being used in percentage
     S16BIT s16Card = 1;
     S16BIT s16Transmitter = 1;
     DD429TX_FRAME_INFO_TYPE pTxFrameInfo;
     S16BIT s16Result = 0;

     s16Result = dd429X_GetTxFrameInfo(s16Card,
                                       s16Transmitter,
                                       &pTxFrameInfo);

   if(s16Result == ERR_SUCCESS)
       printf("The tx fifo is %d pct full",
               pTxFrameInfo.u8PercentageFull);
   else
     {
       printf("Error in dd429X_GetTxFrameInfo() function \n");
       PrintErrorMsg(s16Result);
       return 1;
     }
```

SEE ALSO

dd429X_SetTxMajorFrameRepeatCount()
dd429X_SetTxFrameResolution()
dd429X_GetTxFrameResolution()
DD429TX_FRAME_INFO_TYPE
dd429X_AddTxFrame()
dd429X_SetTxFrameControl()
dd429X_ModifyTxFrameData()

dd429X_GetTxFrameResolution

PROTOTYPE

#include “Transmit.h”
S16BIT **dd429X_GetTxFrameResolution**(S16BIT s16Card
S16BIT s16Transmitter,
U32BIT *u32Resolution);

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager
s16Transmitter	(input parameter) The transmit channel to access
u32Resolution	(output parameter) Pointer to storage for transmitter resolution (!=NULL) 0=1ms 1=1µs

DESCRIPTION

This function gets the transmit resolution that was previously set for the selected *s16Transmitter* transmit channel.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter defined
ERR_ENABLE	Channel not enabled
ERR_RESOLUTION	Invalid Resolution parameter specified
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

dd429X_GetTxFrameResolution (continued)

EXAMPLE

```
//This example will get the current transmit resolution for //channel
1 of card 1
     S16BIT s16Card = 1;
     S16BIT s16Transmitter = 1;
//set tx channel for 1us resolution
     U32BIT u32Resolution;
     S16BIT s16Result = 0;

     s16Result = dd429X_GetTxFrameResolution(s16Card,
                                             s16Transmitter,
                                             &u32Resolution);
   if(s16Result != ERR_SUCCESS)
   {
       printf("Error in dd429X_GetTxFrameResolution() function \n");
       PrintErrorMsg(s16Result);
       return 1;
   }
```

SEE ALSO

dd429X_SetTxMajorFrameRepeatCount()
dd429X_SetTxFrameResolution()
dd429X_GetTxFrameInfo()
dd429X_AddTxFrame()
dd429X_SetTxFrameControl()
dd429X_ModifyTxFrameData()

dd429X_GetVariableSpeed

PROTOTYPE

```
#include “Transmit.h”
S16BIT dd429X_GetVariableSpeed(
                S16BIT s16Card,
                S16BIT s16Channel,
                U32BIT *pu32Speed,
                U8BIT u8ChannelType);
```

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager
s16Channel	(input parameter) The transmit/receive channel to access
pu32Speed	(output parameter) Pointer to store the currently set variable speeds in bps (!=NULL)
u8ChannelType	(input parameter) Specifies if this is a receive or a transmit channel CHAN_TYPE_429_RX or CHAN_TYPE_429_TX

DESCRIPTION

This function gets the current speed setting for a particular ARINC 429 channel and stores it in the *pu32speed* pointer in bps.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_NULL	NULL pointer defined
ERR_INVALID_CHANNEL_NO	Channel not initialized
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

dd429X_GetVariableSpeed (continued)

EXAMPLE

```
/* This function gets variable speed, in bps, that was set for a given
Tx Channel */

S16BIT s16Card = 1;
S16BIT s16Result = 0;
S16BIT s16Channel = 1;
U32BIT pu32Speed = 1000;
U8BIT u8ChannelType = CHAN_TYPE_429_RX;

s16Result = dd429X_GetVariableSpeed(s16Card, s16Channel, &pu32Speed,
u8ChannelType);

    if(s16Result != ERR_SUCCESS)
    {
        printf("Error in dd429X_GetVariableSpeed() function \n");
        PrintErrorMsg(s16Result);
        return 1;
    }
```

SEE ALSO

dd429X_SetVariableSpeed()

dd429X_LoadTxQueueMore

PROTOTYPE

```
#include “Transmit.h”
S16BIT dd429X_LoadTxQueueMore (S16BIT s16Card,
                               S16BIT s16Transmitter,
                               S16BIT s16N,
                               U32BIT *pu32Data,
                               DD429_TESTER_OPTIONS_TYPE *psTesterOptions);
```

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager
s16Transmitter	(input parameter) The transmit channel to access
s16N	(input parameter) The number of ARINC words in the array. Max size is 256 ARINC messages.
pu32Data	(input parameter) An array of 32-bit words to load.
psTesterOptions	(input parameter) Pointer to **DD429_TESTER_OPTIONS_TYPE** tester options structure *{ Inter-Word Bit Gap Error Injection,* *Word Size Error,* *Parity Error,* *Set bit 33 Error }}*

DESCRIPTION

This function loads multiple 32-bit words into the transmitter’s queue, until either the *s16N* words are all loaded or the queue becomes full. Make sure the transmitter is enabled to transmit these words. This is the same as the legacy **dd429X_LoadTxQueueMore()** function with the addition of options for error injection as defined by the **DD429_TESTER_OPTIONS_TYPE** input structure.

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter defined
ERR_ENABLE	Channel not enabled
ERR_NULL	NULL pointer defined
ERR_INTER_WORD_GAP	Invalid Inter Word Bit Gap parameter
ERR_WORD_SIZE	Invalid Word Size parameter

dd429X_LoadTxQueueMore (continued)

ERR_PARITY	Invalid Parity parameter
ERR_BIT_33	Invalid Extra Bit (bit 33) parameter
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

EXAMPLE

```
//This example will load the 3 data words into the queue of
//transmitter 1 of card 1. The function returns the actual number
//of ARINC words that were loaded.
     S16BIT s16Card = 1;
     S16BIT s16Transmitter = 1;
     S16BIT s16N = 3;
     U32BIT pu32Data[3];
     data[0] = 0x12345678;
     data[1] = 0x87654321;
     data[2] = 0x12344321;
     DD429_TESTER_OPTIONS_TYPE psTesterOptions[3];
     psTesterOptions[0] = {0x0000, 0x00, 0x00, 0x00};
     psTesterOptions[1] = {0x0000, 0x00, 0x00, 0x00};
     psTesterOptions[2] = {0x0000, 0x00, 0x00, 0x00};
     S16BIT s16Result = 0;

     s16Result = dd429X_LoadTxQueueMore (s16Card,
                                         s16Transmitter,
                                         s16N,
                                         pu32Data
                                         psTesterOptions);
   if(s16Result != 3)
   {
       printf("Error in dd429X_LoadTxQueueMore() function \n");
       PrintErrorMsg(s16Result);
       return 1;
   }
```

SEE ALSO

dd429X_LoadTxQueueOne() **LoadTxQueueOne()**
LoadTxQueueMore() **DD429_TESTER_OPTIONS_TYPE**

dd429X_LoadTxQueueOne

PROTOTYPE

```
#include “Transmit.h”
S16BIT dd429X_LoadTxQueueOne(
                S16BIT s16Card,
                S16BIT s16Transmitter,
                U32BIT u32Data,
                DD429_TESTER_OPTIONS_TYPE *psTesterOptions);
```

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager
s16Transmitter	(input parameter) The transmit channel to access
u32Data	(input parameter) 32-bit ARINC dataword to transmit
psTesterOptions	(input parameter) Pointer to **DD429_TESTER_OPTIONS_TYPE** tester options structure *{ Inter-Word Bit Gap Error Injection,* *Word Size Error,* *Parity Error,* *Set bit 33 Error }}*

DESCRIPTION

This function loads the 32-bit word directly into the transmitter’s queue. Make sure the transmitter is enabled to transmit the word. This is the same as the legacy **LoadTxQueueOne()** function with the addition of options for error injection as defined by the **DD429_TESTER_OPTIONS_TYPE** input structure.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter defined
ERR_ENABLE	Channel not enabled
ERR_NULL	NULL pointer defined

dd429X_LoadTxQueueOne (continued)

EXAMPLE

```
//This example loads the 32-bit data 0x12345678 directly into the
//queue of transmitter 1 of card 1.
     S16BIT s16Result = 0;
     S16BIT s16Card = 1;
     S16BIT s16Transmitter = 1;
     U32BIT u32Data = 0x12345678;
     DD429_TESTER_OPTIONS_TYPE psTesterOptions
          = {0x0000, 0x00, 0x00, 0x00};

     s16Result = dd429X_LoadTxQueueOne(
                                s16Card,
                                s16Transmitter,
                                u32Data
                                psTesterOptions);

    if(s16Result != ERR_SUCCESS)
    {
        printf("Error in dd429X_LoadTxQueueOne() function \n");
        PrintErrorMsg(s16Result);
        return 1;
    }
```

SEE ALSO

dd429X_LoadTxQueueMore()
LoadTxQueueMore()
LoadTxQueueOne()
DD429_TESTER_OPTIONS_TYPE

dd429X_ModifyRepeatedData

PROTOTYPE

```
#include “Transmit.h”
S16BIT dd429X_ModifyRepeatedData(
                    S16BIT s16Card,
                    S16BIT s16Transmitter,
                    U32BIT u32Data,
                    DD429_TESTER_OPTIONS_TYPE *psTesterOptions,
                    U32BIT u32Option);
```

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager
s16Transmitter	(input parameter) The transmit channel to access
u32Data	(input parameter) The 32-bit ARINC word (with Label and SDI info) that will replace the current ARINC word with the same Label and SDI
psTesterOptions	(input parameter) Pointer to **DD429_TESTER_OPTIONS_TYPE** tester options structure *{ Inter-Word Bit Gap Error Injection,* *Word Size Error,* *Parity Error,* *Set bit 33 Error }}*
u32Option	(input parameter) Modify via Label only or Label/SDI

DESCRIPTION

This function updates the data portion of a previously scheduled ARINC 429 message (using **dd429X_AddRepeated()**). This function has no effect on any message scheduling and it’s sole purpose is to update the data portion (bits 11-29) of a currently scheduled ARINC word.

This function is the same as the legacy **ModifyRepeatedData()** function, with the with the addition of options for error injection as defined by the **DD429_TESTER_OPTIONS_TYPE** output structure.

Note: *SDI/Label must exist in scheduled transmission list for data to be updated.*

dd429X_ModifyRepeatedData (continued)

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter defined
ERR_ENABLE	Channel not enabled
ERR_INTER_WORD_GAP	Invalid Inter Word Bit Gap parameter defined
ERR_ WORD_SIZE	Invalid Word Size parameter defined
ERR_PARITY	Invalid Parity parameter defined
ERR_BIT_33	Invalid Extra Bit (bit 33) parameter
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

EXAMPLE

```
/* This example modifies the data for SDI/label 0x1F on transmitter one
of card one.  Note, that an ARINC Message for SDI/label 0x1F must be
previously scheduled via AddReapeated() before using this function. */

 S16BIT s16Card = 1;
 S16BIT s16Transmitter = 1;
 s16LabelSdi = ;
 U32BIT u32Data = 0x1234561F;
 DD429_TESTER_OPTIONS_TYPE psTesterOptions =
                                        {0x0000, 0x00, 0x00, 0x00};

 s16Result = dd429X_ModifyRepeatedData(s16Card,
                                  s16Transmitter,
                                  u32Data,
                                  psTesterOptions,
                                  DD429_MODIFY_VIA_SDI_LABEL);

 if(s16Result != ERR_SUCCESS)
 {
    printf("Error in dd429X_ModifyRepeatedData() function \n");
    PrintErrorMsg(s16Result);
    return 1;
 }
```

SEE ALSO

GetNumOfRepeated()	**GetAllRepeated()**
DelFilter()	**ClearRepeated()**
dd429X_AddRepeated()	**dd429X_GetRepeated()**
ModifyRepeatedData()	**DD429_TESTER_OPTIONS_TYPE**

dd429X_ModifyRepeatedDataItem

PROTOTYPE

S16BIT **dd429X_ModifyRepeatedDataItem**(
S16BIT s16Card,
S16BIT s16Transmitter,
U16BIT u16ItemIndex,
U32BIT u32Data,
DD429_TESTER_OPTIONS_TYPE *psTesterOptions);

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager.
s16Transmitter	(input parameter) The transmit channel to access.
u16ItemIndex	(input parameter) User index assigned to the scheduled data.
u32Data	(input parameter) 32-bit ARINC dataword to transmit.
psTesterOptions	(input parameter) Pointer to tester options structure. *{ Inter-Word Bit Gap Error Injection,* *Word Size Error,* *Parity Error,* *Set bit 33 Error }}*

DESCRIPTION

This function will replace a message in the scheduler based on the item index. The message timing will still be preserved for the modified message.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid Transmitter
ERR_ENABLE	Channel is not in enabled state
ERR_INDEX	Invalid index parameter

dd429X_ModifyRepeatedDataItem (continued)

EXAMPLE

```
//This example will replace the message specified by index 1
     S16BIT s16Card = 1;
     S16BIT s16Transmitter = 1;
     U16BIT u16ItemIndex = 1;
     U32BIT u32Data = 0x12345678; \\data 0x123456 Label170 SDI10
     DD429_TESTER_OPTIONS_TYPE psTesterOptions = {0, 0, 0, 0};


     s16Result = dd429X_ModifyRepeatedDataItem(s16Card,
                                        s16Transmitter,
                                        u16ItemIndex,
                                        u32Data,
                                        &psTesterOptions);

    if(s16Result != ERR_SUCCESS)
     {
        printf("Error in dd429X_ModifyRepeatedDataItem() function
\n");
        PrintErrorMsg(s16Result);
        return 1;
     }
```

SEE ALSO

dd429X_AddRepeatedItem()
dd429X_DelRepeatedItem()
dd429X_GetRepeatedItem()
dd429X_GetAllRepeatedItem()

dd429X_ModifyTxFrameData

PROTOTYPE

```
#include “Transmit.h”
S16BIT dd429X_ModifyTxFrameData(
                    S16BIT s16Card,
                    S16BIT s16Transmitter,
                    DD429_TX_MINOR_FRAME_PAYLOAD_TYPE *psMinorFramePayload,
                    U32BIT u32Option);
```

HARDWARE

DD-40x00F/i/T/K, DD-40001H, BU-67118K/M/Y/Z

PARAMETERS

s16Card — (input parameter)
Logical device number defined in Card Manager

s16Transmitter — (input parameter)
The transmit channel to access.

psMinorFramePayload
(input parameter)
Pointer to an instance of payload data (data + error injection)
{ 32-bit ARINC dataword,
{ Inter-Word Bit Gap Error Injection,
Word Size Error,
Parity Error,
Set bit 33 Error }}

u32Option — (input parameter)
Update Options: Valid choices include:

DD429_MODIFY_VIA_LABEL,
Update Data if only the Label matches.

DD429_MODIFY_VIA_SDI_LABEL
Update Data if the SDI and Label both match.

dd429X_ModifyTxFrameData (continued)

DESCRIPTION

This function updates the data portion of a previously scheduled ARINC 429 message. This function has no effect on any message scheduling and it's sole purpose is to update the data portion (bits 11-29) of a currently scheduled ARINC word.

Note: *SDI/Label must exist in scheduled transmission list for data to be updated.*

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter defined
ERR_ENABLE	Channel not enabled
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

EXAMPLE

```
/* This example modifies the data for SDI/label 0x1F on transmitter
one of card one.  Note, that an ARINC Message for SDI/label 0x1F must
be previously scheduled via AddTxFrame() before using this function.
*/
      S16BIT s16Card = 1;
      S16BIT s16Transmitter = 1;
      DD429_TX_MINOR_FRAME_PAYLOAD_TYPE *psMinorFramePayload =
      {u32Data, {0x0000, 0x00, 0x00, 0x00}};

      s16Result = dd429X_ModifyTxFrameData(s16Card,
                                  s16Transmitter,
                                  psMinorFramePayload,
                                  DD429_MODIFY_VIA_SDI_LABEL);


    if(s16Result != ERR_SUCCESS)
      {
         printf("Error in dd429X_ModifyTxFrameData() function \n");
         PrintErrorMsg(s16Result);
         return 1;
      }
```

SEE ALSO

dd429X_SetTxMajorFrameRepeatCount()
dd429X_SetTxFrameResolution()
dd429X_GetTxFrameResolution()
DD429_TX_MINOR_FRAME_PAYLOAD_TYPE
dd429X_AddTxFrame()
dd429X_SetTxFrameControl()
dd429X_GetTxFrameInfo()

dd429X_SetRepeaterMode

PROTOTYPE

S16BIT **dd429X_SetRepeaterMode**(
S16BIT s16Card,
S16BIT s16Receiver,
U16BIT u16LabelSdi,
U32BIT u32Mode,
U32BIT u32Data);

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager.
s16Receiver	(input parameter) The receive channel to access.
u16LabelSdi	(input parameter) User specified Label/SDI parameter
u32Mode	(input parameter) Specifies how the repeated data will be processed. A valid value consists of a REPEATER OPTION parameter OR'd with a REAPTER MODE parameter
u32Data	(input parameter) The new data to be used for REPLACE operations Bitwise operator for SET, FLIP, or CLEAR operations

DESCRIPTION

This function is used to set the data repeater and data manipulation options for the specified ARINC 429 repeater source channel. All labels will be inactive on the transmitter/repeater until this function is called. To begin repeating a label, use REPEATER_OPTION__LABEL_ONLY OR'd with REPEATER_MODE__REPEAT, or any other REPEATER MODE PARAMETER. To repeat or manipulate all incoming labels, this function must be called individually for every label.

The REPEATER OPTION PARAMETER selects the message to repeat or manipulate via the message label or labe/SDI combination. The REPEATER MODE PARAMETER chooses the mode of operation to be applied. If a label or SDI combination has already been defined, then the previous definition will be written by the latest one. For REPLACE operations, the u32Data parameter is used to specify the value to be replaced. For SET, FLIP, or CLEAR, operations, the u32Data parameter is used as a bitwise operator to specify which bits to set, flip, or clear.

dd429X_SetRepeaterMode (continued)

The table below describes the available repeater modes and options.

REPEATER OPTION PARAMETER	REPEATER OPTION DESCRIPTION	HEX Value
REPEATER_OPTION__LABEL_ONLY	Choose message to modify based on the message label.	0x10000000
REPEATER_OPTION__LABEL_SDI	Choose message to modify based on the message label/SDI.	0x20000000

REPEATER MODE PARAMETER	REPEATER MODE DESCRIPTION	HEX Value
REPEATER_MODE__DISABLED	Don't repeat the selected data.	0x00000000
REPEATER_MODE__REPEAT	Repeat the exact data received.	0x00010000
REPEATER_MODE__DATA_REPLACE	Replace the data portion of the message with the value specified by u32Data.	0x00020000
REPEATER_MODE__LABEL_REPLACE	Replace the label portion of the message with the value specified by u32Data.	0x00030000
REPEATER_MODE__MESSAGE_REPLACE	Replace any portion of the message with the value specified by u32Data.	0x00040000
REPEATER_MODE__DATA_FLIP	Flip the value of the specified data bit. u32Data will be used as a bitwise operator to specify which bit to flip.	0x00050000
REPEATER_MODE__LABEL_FLIP	Flip the value of the specified label bit. u32Data will be used as a bitwise operator to specify which bit to flip.	0x00060000
REPEATER_MODE__MESSAGE_FLIP	Flip the value of the specified message bit. u32Data will be used as a bitwise operator to specify which bit to flip.	0x00070000
REPEATER_MODE__DATA_SET	Set the specified data bit to a '1'. u32Data will be used as a bitwise operator to specify which bit to set	0x00080000
REPEATER_MODE__LABEL_SET	Set the specified label bit to a '1'. u32Data will be used as a bitwise operator to specify which bit to set	0x00090000
REPEATER_MODE__MESSAGE_SET	Set the specified message bit to a '1'. u32Data will be used as a bitwise operator to specify which bit to set	0x000A0000
REPEATER_MODE__DATA_CLEAR	Clear the specified data bit to zero. u32Data will be used as a bitwise operator to specify which bit to set	0x000B0000
REPEATER_MODE__LABEL_CLEAR	Clear the specified label bit to zero. u32Data will be used as a bitwise operator to specify which bit to set	0x000C0000
REPEATER_MODE__MESSAGE_CLEAR	Clear the specified message bit to zero. u32Data will be used as a bitwise operator to specify which bit to set	0x000D0000

dd429X_SetRepeaterMode (continued)

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_ENABLE	Channel is not in enabled state
ERR_RX	Invalid receiver specified
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card
ERR_LABELSDI	Invalid Label/SDI specified
ERR_REPEATER_OPTION	Invalid repeater option received

EXAMPLE

```
//This example sets up data pollution mode for receiver 1
//Data portion of label 246 will be replaced with
//Data portion specified by u32Data
      S16BIT s16Card = 1;
      S16BIT s16Receiver = 1;
      U16BIT u16LabelSdi = 0xa6;

      U32BIT    u32Mode    =    REPEATER_OPTION__LABEL_ONLY          |
           REPEATER_MODE__DATA_SET;
      U32BIT u32Data = 0x012340a6;

      s16Result = dd429X_SetRepeaterMode(s16Card,
                                         s16Receiver,
                                         u16LabelSdi,
                                         u32Mode,
                                         u32Data);

     if(s16Result != ERR_SUCCESS)
       {
          printf("Error in dd429X_SetRepeaterMode() function \n");
          PrintErrorMsg(s16Result);
          return 1;
       }
```

SEE ALSO

dd429X_ConfigRepeater() **dd429X_GetRepeaterMode()**

dd429X_SendTxFrameAsync

PROTOTYPE

```
#include “Transmit.h”
S16BIT dd429X_SendTxFrameAsync(
                    S16BIT s16Card,
                    S16BIT s16Channel,
                    U8BIT u8Priority,
                    U32BIT u32Data,
                    DD429_TESTER_OPTIONS_TYPE *psTesterOptions);
```

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager
s16Channel	(input parameter) The transmit channel to access.
u32Data	(input parameter) 32-bit ARINC dataword to transmit
u8Priority	(input parameter) DD429_ASYNC_PRIORITY_LOW, DD429_ASYNC_PRIORITY_HIGH
psTesterOptions	(input parameter) Pointer to **DD429_TESTER_OPTIONS_TYPE** tester options structure *{ Inter-Word Bit Gap Error Injection,* *Word Size Error,* *Parity Error,* *Set bit 33 Error }}*

DESCRIPTION

This function is used to send an ARINC message asynchronously in Low or High priority mode.

Note: *Low priority messages will only transmit if there is enough free time between messages to do so. High priority messages will go out immediately after the current message completes and can, in effect, disrupt the scheduled timing. The low priority or high priority messages are placed in separate low priority or high priority FIFOs. These are limited to 8 messages each.*

dd429X_SendTxFrameAsync (continued)

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter defined
ERR_ENABLE	Channel not enabled
ERR_INVALID_CHANNEL_NO	Channel not initialized
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

EXAMPLE

```
/* This example will send one message in low priority mode*/
      S16BIT s16Card = 1;
      S16BIT s16Channel = 1
      U32BIT u32Data = 0x1234561F;
      DD429_TESTER_OPTIONS_TYPE psTesterOptions = {0x0000, 0x00, 0x00,
      0x00};

      s16Result = dd429X_SendTxFrameAsync(s16Card,
                                          s16Channel,
                                          DD429_ASYNC_PRIORITY_LOW,
                                          u32Data,
                                          psTesterOptions);

    if(s16Result != ERR_SUCCESS)
      {
         printf("Error in dd429X_SendTxFrameAsync() function \n");
         PrintErrorMsg(s16Result);
         return 1;
      }
```

SEE ALSO

DD429_TESTER_OPTIONS_TYPEdd429X_SetAmplitude

PROTOTYPE

#include “Transmit.h”
S16BIT **dd429X_SetAmplitude** (
S16BIT s16Card,
S16BIT s16Transmitter,
U32BIT u32Amplitude);

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager
s16Transmitter	(input parameter) The transmit channel to access
u32Amplitude	(input parameter) 8-bit Voltage Amplitude Parameter between 0x00 and 0xFF

DESCRIPTION

This function sets the output voltage amplitude for a particular channel.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter defined
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

EXAMPLE

```
//This function sets the voltage amplitude for a given Tx Channel
S16BIT s16Card = 1;
S16BIT s16Result = 0;
S16BIT s16Transmitter = 1;
U32BIT u32Amplitude = 0xFF;  //set max output amplitude

s16Result = dd429X_SetAmplitude(s16Card, s16Transmitter,
u32Amplitude);

    if(s16Result != ERR_SUCCESS)
    {
        printf(“Error in dd429X_SetAmplitude() function \n”);
        PrintErrorMsg(s16Result);
        return 1;  }
```

dd429X_SetAmplitude (continued)

SEE ALSO

dd429X_GetAmplitude()

dd429X_SetIRIGTx

PROTOTYPE

#include “Transmit.h”
S16BIT **dd429X_SetIRIGTx**(
S16BIT s16Card,
DDC_IRIG_TX_TYPE sIRIG);

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager
sIRIG	(input parameter) **PDDC_IRIG_TX_TYPE**structure that contains the settings for the IRIG transmitter registers

DESCRIPTION

This function sets the IRIG Transmitter registers.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

EXAMPLE

```
/* This example turns on the IRIG transmitter and initializes it */

S16BIT s16Result = 0;
DDC_IRIG_TX_TYPE sIRIG;
sIRIG.u16Enable = TRUE;
sIRIG.u16Days = 156;
sIRIG.u16Hours = 4;
sIRIG.u16Minutes = 40;
sIRIG.u16Seconds = 0;
sIRIG.u16Year = 9;
sIRIG.u32Control = 0;

s16Result = SetIRIGTx(DevNum, strIRIGTxSet);
if(s16Result != ERR_SUCCESS)
{
      printf("SetIRIGTx Failed!\n");
      PrintOutError(s16Result);
}
```

dd429X_SetIRIGTx (continued)

SEE ALSO

dd429X_GetIRIGTx() **PDDC_IRIG_TX_TYPE**

dd429X_SetTxFrameControl

PROTOTYPE

```
#include “Transmit.h”
S16BIT dd429X_SetTxFrameControl(S16BIT s16Card,
                                S16BIT s16Transmitter,
                                U8BIT u8Control);
```

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager
s16Transmitter	(input parameter) The transmit channel to access
u8Control	(input parameter) Parameter to initialize, start, stop, or clear the major frame DD429_TX_FRAME_INIT , DD429_TX_FRAME_START, DD429_TX_FRAME_STOP,

DESCRIPTION

This function is used to manipulate the major frame of a transmit channel. The function is first called with the DD429_TX_FRAME_INIT parameter to initialize a major frame. Then **dd429X_AddTxFrame()** is called to create minor frames within the major frame. **dd429X_SetTxMajorFrameRepeatCount()** is called to specified the number of times to run the major frame. Then **dd429X_SetTxFrameControl()** is called again with DD429_TX_FRAME_START or DD429_TX_FRAME_STOP, to start or stop the execution of the major frame.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter defined
ERR_ENABLE	Channel not enabled
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

dd429X_SetTxFrameControl (continued)

EXAMPLE

```
//This example will initialize the major frame for card 1 ch 1
     S16BIT s16Card = 1;
     S16BIT s16Transmitter = 1;
     U8BIT u8Control = DD429_TX_FRAME_INIT;
     S16BIT s16Result = 0;

     s16Result = dd429X_SetTxFrameControl(s16Card,
                                          s16Transmitter,
                                          u8Control);

    if(s16Result != ERR_SUCCESS)
    {
        printf("Error in dd429X_SetTxFrameControl() function \n");
        PrintErrorMsg(s16Result);
        return 1;
    }
```

SEE ALSO

dd429X_SetTxMajorFrameRepeatCount()
dd429X_SetTxFrameResolution()
dd429X_GetTxFrameResolution()
dd429X_AddTxFrame()
dd429X_GetTxFrameInfo()
dd429X_ModifyTxFrameData()

dd429X_SetTxFrameResolution

PROTOTYPE

```
#include “Transmit.h”
S16BIT dd429X_SetTxFrameResolution(S16BIT s16Card
                                   S16BIT s16Transmitter,
                                   U32BIT u32Resolution);
```

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager
s16Transmitter	(input parameter) The transmit channel to access
u32Resolution	(input parameter) Input parameter for transmit/receive resolution 0=1ms 1=1µs

DESCRIPTION

This function sets the transmit resolution of the selected *s16Transmitter* transmit channel.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter defined
ERR_ENABLE	Channel not enabled
ERR_RESOLUTION	Invalid Resolution parameter specified
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

dd429X_SetTxFrameResolution (continued)

EXAMPLE

```
//This example will set the transmit resolution for card 1
//channel 1 to 1 us
     S16BIT s16Card = 1;
     S16BIT s16Transmitter = 1;
//set tx channel for 1us resolution
     U32BIT u32Resolution = 1;
     S16BIT s16Result = 0;

     s16Result = dd429X_SetTxFrameResolution(s16Card,
                                             s16Transmitter,
                                             u32Resolution);
    if(s16Result != ERR_SUCCESS)
    {
          printf("Error in dd429X_SetTxFrameResolution() function
          \n");
          PrintErrorMsg(s16Result);
          return 1;
    }
```

SEE ALSO

dd429X_SetTxMajorFrameRepeatCount()
dd429X_GetTxFrameResolution()
dd429X_GetTxFrameInfo()
dd429X_AddTxFrame()
dd429X_SetTxFrameControl()
dd429X_ModifyTxFrameData()

dd429X_SetTxMajorFrameRepeatCount

PROTOTYPE

#include “Transmit.h”
S16BIT **dd429X_SetTxMajorFrameRepeatCount**(S16BIT s16Card,
S16BIT s16Transmitter,
U32BIT u32FrameRepeatCount);

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card (input parameter)
Logical device number defined in Card Manager

s16Transmitter (input parameter)
The transmit channel to access

u32FrameRepeatCount
(input parameter)
Number of times to repeat major frame
0 = run forever

DESCRIPTION

This function determines how many times to repeat the major frame of the transmit channel. The maximum size for the *u32FrameRepeatCount* parameter is 0x100000.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter defined
ERR_ENABLE	Channel not enabled
ERR_ TX_FRAME_REPEAT_COUNT	Invalid Frame Repeat Count parameter specified

dd429X_SetTxMajorFrameRepeatCount (continued)

EXAMPLE

```
//This example will repeat the major frame 5 times
      S16BIT s16Card = 1;
      S16BIT s16Transmitter = 1;
      U32BIT u32FrameRepeatCount = 5;
      S16BIT s16Result = 0;

      s16Result = dd429X_SetTxMajorFrameRepeatCount(s16Card,
                                          s16Transmitter,
                                          u32FrameRepeatCount);

    if(s16Result != ERR_SUCCESS)
    {
        printf("Error in dd429X_SetTxMajorFrameRepeatCount() function
\n");
        PrintErrorMsg(s16Result);
        return 1;
    }
```

SEE ALSO

dd429X_SetTxFrameResolution()
dd429X_GetTxFrameResolution()
dd429X_SetTxFrameControl()
dd429X_AddTxFrame()
dd429X_GetTxFrameInfo()
dd429X_ModifyTxFrameData()

dd429X_SetVariableSpeed

PROTOTYPE

```
#include “Transmit.h”
S16BIT dd429X_SetVariableSpeed(
                S16BIT s16Card,
                S16BIT s16Channel,
                U32BIT *pu32Speed,
                U8BIT u8ChannelType);
```

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager
s16Channel	(input parameter) The transmit/receive channel to access
pu32Speed	(input parameter) Speed in bps Range 500-50,000 bps use 100 bps increments Range 50,000-200,000 bps use 1,000 bps increments Actual speed set by the device will be returned back to pu32speed
pu32Speed	(output parameter) Reusing the same pointer to store the actual speed set by the device (!=NULL)
u8ChannelType	(input parameter) Specifies if this is a receive or a transmit channel CHAN_TYPE_429_RX or CHAN_TYPE_429_TX

DESCRIPTION

This function sets the speed to be used for a particular ARINC 429 channel. *pu32speed* is used as both an input to specify the speed in bps and as an output to store the actual speed as determined by the device. The user may check the value of *pu32speed*, again, after the function call to verify that the speed set by the device is the one desired.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_SPEED	Invalid speed input
ERR_INVALID_CHANNEL_NO	Channel not initialized
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

dd429X_SetVariableSpeed (continued)

EXAMPLE

```
/* This function sets the variable speed in bps for a given Tx/Rx
Channel */

S16BIT s16Card = 1;
S16BIT s16Result = 0;
S16BIT s16Channel = 1;
U32BIT pu32Speed = 1000;
U8BIT u8ChannelType = CHAN_TYPE_429_RX;

s16Result = dd429X_SetVariableSpeed(s16Card, s16Channel, &pu32Speed,
u8ChannelType);

    if(s16Result != ERR_SUCCESS)
    {
        printf("Error in dd429X_SetVariableSpeed() function \n");
        PrintErrorMsg(s16Result);
        return 1;
    }
```

SEE ALSO

dd429X_GetVariableSpeed()

dd429X_VoltageMonitorEnable

PROTOTYPE

```
#include “Transmit.h”
S16BIT dd429X_VoltageMonitorEnable (
                S16BIT s16Card,
                S16BIT s16NumChannels,
                S16BIT *ps16Channels);
```

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager
s16NumChannels	(input parameter) Number of channels to monitor
ps16Channels	(input parameter) List of channel numbers to monitor

DESCRIPTION

This function enables voltage monitoring on the receive channels as defined in the *ps16Channels* parameter.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_NULL	NULL pointer defined
ERR_TX	Invalid transmitter defined
ERR_RX	Invalid receiver defined
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

dd429X_VoltageMonitorEnable (continued)

EXAMPLE

```
/* This function enables voltage monitoring on the given receive
Channels */
S16BIT s16Card = 1;
/* enable voltage monitoring on 4 channels */
S16BIT s16NumChannels = 4;
S16BIT ps16Channels[4] = {1, 2, 3, 4};
S16BIT s16Result = 0;

s16Result = dd429X_VoltageMonitorEnable(s16Card, s16NumChannels,
&ps16Channels);

   dd429X_VoltageMonitorEnable (continued)
 if(s16Result != ERR_SUCCESS)
    {
        printf("Error in dd429X_VoltageMonitorEnable() function \n");
        PrintErrorMsg(s16Result);
        return 1;
    }
```

SEE ALSO

dd429X_VoltageMonitorStart() **dd429X_VoltageMonitorGetStatus()**
dd429X_VoltageMonitorGetData()

dd429X_VoltageMonitorGetData

PROTOTYPE

```
#include “Transmit.h”
S16BIT dd429X_VoltageMonitorGetData(
                S16BIT s16Card,
                U16BIT *pu16Buffer,
                U32BIT *pu32Bytes);
```

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager
pu16Buffer	(output parameter) Pointer to store data values
pu32Bytes	(output parameter) Pointer to store the number of bytes returned into pu16Buffer

DESCRIPTION

This function stores the 10-bit voltage monitoring samples along with the 4-bit channel number into a 16-bit wide storage buffer.
Bits 15:12 give the channel number.
Bits 11:10 padded ‘0’s.
Bits 9:0 give the 10-bit voltage value.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_NULL	NULL pointer defined
ERR_TX	Invalid transmitter defined
ERR_RX	Invalid receiver defined
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

dd429X_VoltageMonitorGetData (continued)

EXAMPLE

```
//This function returns voltage values into pu16Buffer
S16BIT s16Card = 0;
S16BIT s16Result = 0;
U32BIT pu32Status;
U16BIT pu16Buffer[2048]={0x0000}; //4k bytes, max monitor fifo
U32BIT pu32Bytes;

s16Result = dd429X_VoltageMonitorGetData(s16Card, &pu16Buffer,
&pu32Bytes);

    if(s16Result != ERR_SUCCESS)
    {
        printf("Error in dd429X_VoltageMonitorGetData() function \n");
        PrintErrorMsg(s16Result);
        return 1;
    }
```

SEE ALSO

dd429X_VoltageMonitorEnable()
dd429X_VoltageMonitorStart()
dd429X_VoltageMonitorGetStatus()

dd429X_VoltageMonitorGetStatus

PROTOTYPE

#include “Transmit.h”
S16BIT **dd429X_VoltageMonitorGetStatus**(
S16BIT s16Card
U32BIT *pu32Status);

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager
pu32Status	(output parameter) Pointer to storage for status of the voltage monitor SAMPLING_IN_PROGRESS 0x40000000 SAMPLING_COMPLETE 0x80000000

DESCRIPTION

This function checks to see if voltage sampling has completed.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

EXAMPLE

```
//This function sets the voltage amplitude for a given Tx Channel
S16BIT s16Card = 0;
S16BIT s16Result = 0;
U32BIT pu32Status;

s16Result = dd429X_VoltageMonitorGetStatus(s16Card, &pu32Status);

    if(s16Result != ERR_SUCCESS)
    {
           printf(“Error in dd429X_VoltageMonitorGetStatus() function
           \n”);

           PrintErrorMsg(s16Result);
           return 1;
    }
```

dd429X_VoltageMonitorGetStatus (continued)

SEE ALSO

dd429X_VoltageMonitorEnable()
dd429X_VoltageMonitorStart()
dd429X_VoltageMonitorGetData()

dd429X_VoltageMonitorStart

PROTOTYPE

#include “Transmit.h”
S16BIT **dd429X_VoltageMonitorStart** (
S16BIT s16Card);

HARDWARE

DD-40x00F/i/T/K

PARAMETERS

s16Card	(input parameter) Logical device number defined in Card Manager

DESCRIPTION

This function starts voltage monitoring on the receive channels as defined by the **dd429X_VoltageMonitorEnable()** function.

RETURN VALUE

ERR_SUCCESS	The function completed successfully
ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_FEATURE_NOT_SUPPORTED	The function is not supported by the card

EXAMPLE

```
/* This function starts the voltage monitoring on the receive channels
*/
S16BIT s16Card = 0;
S16BIT s16Result = 0;

s16Result = dd429X_VoltageMonitorStart(s16Card);

    if(s16Result != ERR_SUCCESS)
    {
        printf(“Error in dd429X_VoltageMonitorStart() function \n”);
        PrintErrorMsg(s16Result);
        return 1;
    }
```

SEE ALSO

dd429X_VoltageMonitorEnable()
dd429X_VoltageMonitorGetStatus()
dd429X_VoltageMonitorGetData()

DelFilter

PROTOTYPE

```
#include “Receive.h”
short DelFilter (short Card,
                 short Receiver,
                 short LabelSdi);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receive channel number.
LabelSdi	(input parameter) The label (bits 0..7) and SDI (bits 8..9).

DESCRIPTION

This function deletes one filter for the specified receiver.

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_RX	Invalid receiver defined
ERR_LABELSDI	Invalid LABEL or SDI
NUMBER	A zero means the function did not find the specified filter. A one means the function found the filter and deleted it

EXAMPLE

```
//This example will delete the filter for a label of 255 and an SDI of
//3 that is set up on receiver 1 of card 1.
      short card = 1;
      short receiver =1;
      short labelsdi = 1023;
      short delfilter_return = 0;
      delfilter_return = DelFilter (card,
                                    receiver,
                                    labelsdi);
```

DelFilter (continued)

SEE ALSO

AddFilter()
ClearFilter()
GetNumOfFilter()
GetAllFilter()
EnableFilter()
GetFilterStatus()
GetFilter()
ConfigFilter()

DelRepeated

PROTOTYPE

```
# include “Transmit.h”
short DelRepeated   (short Card,
                     short Transmitter,
                     short LabelSdi);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in theControl Panel.
Transmitter	(input parameter) The transmit channel number.
LabelSdi	(input parameter) The label (bits 0..7) and SDI (bits 8..9).

DESCRIPTION

This function deletes the label/sdi from the transmitter’s schedule table and therefore cancels the frequent transmission of the word.

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter defined
ERR_LABELSDI	Invalid Label or SDI value
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example will delete the entry from the transmitter’s schedule
// table with a label of 241 and an SDI of 3 that was set up on
// transmitter 1 of card 2.
      short card = 2;
      short transmitter =1;
      short labelsdi = 1009;
      short delrepeated_return = 0;
      delrepeated_return = DelRepeated (card,
                                        transmitter,
                                        labelsdi);
```

DelRepeated (continued)

SEE ALSO

GetNumOfRepeated()
GetAllRepeated()
ClearRepeated()
GetRepeated()
AddRepeated()

DisableRxHostBuffer

PROTOTYPE

#include “receive.h”
S16BIT **DisableRxHostBuffer** (S16BIT Card
S16BIT receiver);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receiver channel number to be disabled.

DESCRIPTION

This function will disable data for a specific FIFO receiver to be directed into the Receive FIFO Host Buffer. The Host Buffer will improve performance on receiving data and can be used instead of the ReadRxQueueIrig() functions.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_SUCCESS	The function returned successfully.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Channel is not configured as a receiver.
ERR_MODE	Receiver is not configured in FIFO mode.
ERR_RXQUEUESZ	Requested number of messages to Read is 0.
ERR_NULL	One or more pointer inputs are NULL.
ERR_RX_HBUF_INSTALL	Receive Host Buffer is notinstalled.
ERR_NORES	Hardware is not responding.

EXAMPLE

```
// This example shows how to enable a receiver to use the Host
//Buffer.
     S16BIT wStatus = 0;
     S16BIT card = 1;
     S16BIT receiver = 1;
     wStatus = DisableRxHostBuffer(card,
                                   receiver);
```

SEE ALSO

InstallFifoRxHostBuffer()
UninstallFifoRxHostBuffer()
EnableRxHostBuffer()
DisableRxHostBuffer()
ReadRxHostBuffer()

DisableUart

PROTOTYPE

```
#include “serial.h”
S16BIT DisableUart (S16BIT Card,
                    U8BIT Chan);
```

HARDWARE

BU-67211Ux, BU-67107F/M/i/T

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Chan	(input parameter) The specified Serial I/O channel to enable.

DESCRIPTION

This function will Disable a UART (Serial I/O) channel and internally disconntect (disable) the physical external connections.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_SUCCESS	The function returned successfully.
ERR_CRDINIT	Card is not initialized.
ERR_SERCHNL	Chanel in not a valid serial I/O channel.

EXAMPLE

```
// This example shows how to disable a UART channel.
     S16BIT wStatus = 0;
     S16BIT card = 1;
     S16BIT serial_chan = 1;
     wStatus = DisableUart(card,
                           serial_chan);
```

SEE ALSO

DisableUart()	**EnableUart()**
ReadUartConfig ()	**WriteUartConfig()**
ReadUart()	**WriteUart()**

EnableFilter

PROTOTYPE

#include “Receive.h”
short ***EnableFilter*** *(short* Card*,*
short Receiver*,*
short Enable*);*

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receiver channel number.
Enable	(input parameter) A value of DD429_DISABLE (0) will turn off the receiver’s filters and allow all labels to be received. A value of DD429_ENABLE (1) will turn on any existing filters.

DESCRIPTION

This function turns on or turns off all existing filters for the receiver’s queue. The receiver’s mailbox does not use filters. Therefore, to use mailbox only, you can turn on the usage of the filters without any filter defined.

Filtering will be limited to Label only when using IRIG time tagging.

Note: *When the filters are off, the filters still exist****.***

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_RX	Invalid receiver define
ERR_ENABLE	Invalid enabled state
ERR_SUCCESS	The function returned successfully

EnableFilter (continued)

EXAMPLE

```
//This example will turn off all of the defined filters that are on
// receiver 1 of card 1. The filters still exist, they just aren't
// active.
      short card = 1;
      short receiver = 1;
      short enable = DD429_DISABLE;
      short enablefilter_return = 0;
      enablefilter_return = EnableFilter (card,
                                          receiver,
                                          enable);
```

SEE ALSO

AddFilter()
ClearFilter()
GetNumOfFilter()
GetAllFilter()
DelFilter()
GetFilterStatus()
GetFilter()

EnableRx

PROTOTYPE

```
#include “Receive.h”
short  EnableRx (short Card,
                 short Receiver,
                 short Enable);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receiver channel number (1..4).
Enable	(input parameter) A value of DD429_DISABLE (0) disables the receiver. A value of DD429_ENABLE (1) enables the receiver.

DESCRIPTION

This function enables or disables the receiver. All receivers are disabled as default when the card is initialized.

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_RX	Invalid receiver defined
ERR_ENABLE	Invalid enabling
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example will enable receiver 1 of card 1.
      Short card = 1;
      Short receiver = 1;
      Short enable = DD429_ENABLE;
      Short enablerx_return = 0;
      Enablerx_return = EnableRx (card,
                                  receiver,
                                  enable);
```

SEE ALSO

SetRxSpeed() **GetRxSpeed()**
GetRxStatus() **SetRxParity()**
GetRxParity()

EnableRxHostBuffer

PROTOTYPE

```
#include “receive.h”
S16BIT EnableRxHostBuffer (S16BIT Card
                           S16BIT receiver,
                           U32BIT reserved);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receiver channel number to read.
Reserved	(input parameter) Reserved for future use. Set to zero.

DESCRIPTION

This function will enable data for a specific FIFO receiver to be directed into the Receive FIFO Host Buffer. The Host Buffer will improve performance on receiving data and can be used instead of the ReadRxQueueIrig() functions.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_SUCCESS	The function returned successfully.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Channel is not configured as a receiver.
ERR_MODE	Receiver is not configured in FIFO mode.
ERR_RXQUEUESZ	Requested number of messages to Read is 0.
ERR_NULL	One or more pointer inputs are NULL.
ERR_RX_HBUF_INSTALL	Receive Host Buffer is notinstalled.
ERR_NORES	Hardware is not responding.

EnableRxHostBuffer (continued)

EXAMPLE

```
// This example shows how to enable a receiver to use the Host
// Buffer.
     S16BIT wStatus = 0;
     S16BIT card = 1;
     S16BIT receiver = 1;
     wStatus = EnableRxHostBuffer(card,
                                  receiver,
                                  0);
```

SEE ALSO

InstallFifoRxHostBuffer()
UninstallFifoRxHostBuffer()
EnableRxHostBuffer()
DisableRxHostBuffer()
ReadRxHostBuffer()

EnableTimeStamp

PROTOTYPE

```
#include “Receive.h”
short EnableTimeStamp (short Card,
                       short Receiver,
                       short Enable);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receiver channel number.
Enable	(input parameter) A value of DD429_DISABLE (0) disables the receiver’s time stamp function. A value of DD429_ENABLE (1) enables the receiver’s time stamp function.

DESCRIPTION

This function enables or disables the receiver’s time stamp function for both the queue and the mailbox. All time stamp functions are disabled as default when the card is initialized.

> ***Note:*** *When you use this function to enable the time stamp function, the time stamps are undefined for those words that are already received in the mailbox and queue. When you use this function to disable the time stamp function, the time stamps will also be deleted for the received words in the mailbox and queue.*

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_RX	Invalid receiver defined
ERR_ENABLE	Invalid enabling
ERR_SUCCESS	The function returned successfully

EnableTimeStamp (continued)

EXAMPLE

```
//This example enables the time stamp function for receiver 1 of card
//3.
      short card = 3;
      short receiver =1;
      short enable = DD429_ENABLE;
      short enabletimestamp_return = 0;
      enabletimestamp_return = EnableTimeStamp (card,
                                                receiver,
                                                enable);
```

SEE ALSO

ResetTimeStamp() **GetTimeStampStatus()**
ConfigTimeStamp()

EnableTx

PROTOTYPE

#include “Transmit.h”
short ***EnableTx*** *(short* Card*,*
short Transmitter*,*
short Enable*);*

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Transmitter	(input parameter) The transmit channel number.
Enable	(input parameter) A value of DD429_DISABLE (0) disables the transmitter and a value of DD429_ENABLE (1) one enables it.

DESCRIPTION

This function enables or disables the transmitter. All transmitters are disabled by default when the card is initialized. After a transmitter is enabled, the data loaded to its queue will be transmitted immediately. After a transmitter is disabled, it is still legal to load data into its queue, but the data loaded will not be transmitted until the transmitter is enabled again.

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter defined
ERR_ENABLE	Invalid enabling
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example will enable transmitter 1 of card 1.
      short card =1;
      short transmitter =1;
      short enable = DD429_ENABLE;
      short enabletx_return = 0;
      enabletx_return = EnableTx (card,
                                  transmitter,
                                  enable);
```

EnableTx (continued)

SEE ALSO

SetTxSpeed()
GetTxSpeed()
GetTxStatus()
SetTxParity()
GetTxParity()

EnableUart

PROTOTYPE

```
#include “serial.h”
S16BIT EnableUart (S16BIT Card,
                   U8BIT Chan,
                   U16BIT Options);
```

HARDWARE

BU-67211Ux, BU-67107F/M/i/T

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Chan	(input parameter) The specified Serial I/O channel to enable.
Options	(input parameter) Configuration Options. The following options are valid: UART_MODE_RS232 Set channel to RS-232 mode. UART_MODE_RS485 Set channel to RS-485 mode. UART_MODE_RS422 Set channel to RS-422 mode. UART_SLEW_RATE_HI Enable High Slew-Rate.

Note: *Only one Serial Protocol can be selected at once..*

DESCRIPTION

This function will Enable and configure a UART (Serial I/O) channel for a specific Serial-based protocol and makes the physical connections active.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_SUCCESS	The function returned successfully.
ERR_CRDINIT	Card is not initialized.
ERR_SERCHNL	Channel in not a valid serial I/O channel.
ERR_ENABLE	Options input is not valid.
ERR_MODE	Selected channel doesn’t support requested mode/options.

EnableUart (continued)

EXAMPLE

```
// This example shows how to set a UART channel to RS-232.
     S16BIT wStatus = 0;
     S16BIT card = 1;
     S16BIT serial_chan = 1;
     wStatus = EnableUart(card,
                          serial_chan
                          UART_MODE_RS232);
```

SEE ALSO

DisableUart() **EnableUart()**
ReadUartConfig() **WriteUartConfig()**
ReadUart() **WriteUart()**

FreeCard

PROTOTYPE

#include “CardInit.h”
short **FreeCard** (short Card);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.

DESCRIPTION

This function closes the card and frees up any resources that the card was holding. This function must be used to ensure that the next time you initialize the card it doesn’t fail.

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example will free card 1 and of the resources that card 1 was
//using.
      short card = 1;
      short freecard_return = 0;
      freecard_return = FreeCard (card);
```

SEE ALSO

InitCard()

GetAllFilter

PROTOTYPE

```
#include “Receive.h”
short GetAllFilter (short Card,
                    short Receiver,
                    short *LabelSdi);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receiver channel number.
LabelSdi	(output parameter) An array of All existing filters where bits 0..7 are the label and bits 8..9 are the SDI.

DESCRIPTION

This function returns an array of all of the filters defined for a receiver.

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_RX	Invalid receiver defined
ERR_NULL	Null pointer
NUMBER	The number of existing filters defined

EXAMPLE

```
//This example will get an array of all the filters set up on receiver
//1 of card 1.
      short card = 1;
      short receiver = 1;
      short labelsdi[1024];
      short getallfilter_return = 0;
      getallfilter_return = GetAllFilter (card,
                                          receiver,
                                          *labelsdi);
```

GetAllFilter (continued)

SEE ALSO

DelFilter()
ClearFilter()
GetNumOfFilter()
AddFilter()
EnableFilter()
GetFilterStatus()
GetFilter()
ConfigFilter()

GetAllRepeated

PROTOTYPE

#include “Transmit.h”
short **GetAllRepeated** (short Card,
short Transmitter,
short *LabelSdi)

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Transmitter	(input parameter) The transmit channel number (1..2).
LabelSdi	(output parameter) An array of all existing filters where bits 0..7 are the label and bits 8..9 are the SDI.

DESCRIPTION

This function returns all labels and SDIs in the transmitter’s schedule table.

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter defined
ERR_NULL	Null pointer
NUMBER	The number of labels in the scheduled transmission table.

EXAMPLE

```
//This example will get all the label and SDI combinations in the
//schedule table set up for transmitter 1 of card 1.
      short card = 1;
      short transmitter = 1;
      short labelsdi[1024];
      short getallrepeated_return = 0;
      getallrepeated_return = GetAllRepeated (card,
                                              transmitter,
                                              *labelsdi);
```

SEE ALSO

GetNumOfRepeated()
AddRepeated()
ClearRepeated()
GetRepeated()
DelRepeated()

GetAvionAll

PROTOTYPE

```
#include “Control.h”
short GetAvionAll (S16BIT Card,
                   U16BIT *OutEnables,
                   U16BIT *Levels);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
OutEnable	(output parameter) Bitwise representation of levels of avionic (bit 15 = MSB) 0 = Avionic line disabled 1 = Avionic line enabled
Levels	(output parameter) Bitwise representation of levels of avionic (bit 15 = MSB) 0 = Avionic line is set to Low 1 = Avionic line is set to High

DESCRIPTION

This function will return the current state of all Avionic IO bits (ENABLE/DISABLE) and their current levels (0 for Low and 1 for High).

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_NULL	Null Pointer
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//Returns the current status of the Avionic bits and their level.

     S16BIT Card = 1;
     U16BIT OutEnables = 0;
     U16BIT Levels =0;
     Return_value = GetAvionAll(Card, &OutEnables, &Levels);
```

SEE ALSO

GetAvionOut()
SetAvionOut()
SetAvionOutEnable()
GetAvionOutEnable()
GetAvionIn()
SetAvionAll()

GetAvionIn

PROTOTYPE

#include “Control.h”
short **GetAvionIn** (S16BIT Card,
S16BIT Avionic);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Avionic	(input parameter) Avionic Line.

DESCRIPTION

This function returns either a zero to signify the Avionic line level is low or a 1 which represents the level is high.

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_AVIONIC	Not a valid Avionic Line
NUMBER	A zero means the Avionic level is low and a one means the Avionic level is high.

EXAMPLE

```
//This example will get the current Avionic line level.

     S16BIT Card = 1;
     S16BIT Avionic = 1
     Return_value = GetAvionIn(Card, Avionic);
```

SEE ALSO

GetAvionOut()
SetAvionOutEnable()
GetAvionAll()
SetAvionOut()
GetAvionOutEnable()
SetAvionAll()

GetAvionOut

PROTOTYPE

#include “Control.h”
short **GetAvionOut** (S16BIT Card,
S16BIT Avionic);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Avionic	(input parameter) Avionic Line.

DESCRIPTION

This function returns the current level (Low (0) or High (1)) of the avionic line passed into GetAvionOut.

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_AVIONIC	Not a valid Avionic Line
NUMBER	A zero means the Avionic level is low and a one means the Avionic level is high.

EXAMPLE

```
//This example will get the current level of the Avionic line.

     S16BIT Card = 1;
     S16BIT Avionic = 1
     Return_value = GetAvionOut(Card, Avionic);
```

SEE ALSO

SetAvionOut()
SetAvionOutEnable()
GetAvionOutEnable()
GetAvionIn()
GetAvionAll()
SetAvionAll()

GetAvionOutEnable

PROTOTYPE

#include “Control.h”
short **GetAvionOutEnable** (S16BIT Card,
S16BIT Avionic);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Avionic	(input parameter) Avionic Line.

DESCRIPTION

This function returns the current state (enable/disable) of the Avionic output line passed into GetAvionOutEnable.

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_AVIONIC	Not a valid Avionic Line
NUMBER	A zero means the Avionic level is low and a one means the Avionic level is high.

EXAMPLE

```
//This example will get the current state of Avionic Line 1.

     S16BIT Card = 1;
     S16BIT Avionic = 1
     Return_value = GetAvionOutEnable(Card, Avionic);
```

SEE ALSO

GetAvionOut() **SetAvionOut()**
SetAvionOutEnable() **GetAvionIn()**
GetAvionAll() **SetAvionAll()**

GetBitFormat

PROTOTYPE

#include “Control.h”
short **GetBitFormat** (short Card);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.

DESCRIPTION

This function returns the current transmit and receive bit format for all ARINC 429 Multi-IO SDK channels found on the card. The bit format refers to “original” or “alternate” ARINC word formatting.

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
NUMBER	DD429_BITFORMAT_ORIG means it is the original bit format and DD429_BITFORMAT_ALT means it is the alternate bit format

EXAMPLE

```
//This example retrieves the bit format (original or alternate)
//for card 3;
      short card = 3;
      short getbitformat_return = 0;
      getbitformat_return = GetBitFormat (card);
```

SEE ALSO

SetBitFormat()

GetCardType

PROTOTYPE

#include “CardInit.h”
short **GetCardType** (short Card);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.

DESCRIPTION

This function returns the type of card given a logical card number.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
NUMBER	0x1300 = BU-65590F/M 0x5590 = BU-65590UX 0x1700 = BU-65590CX 0x7102 = BU-67102U 0x7103 = BU-67103U

EXAMPLE

```
//This example will return the type of that card 1 is.
      short card = 1;
      short getcardtype_return = 0;
      getcardtype_return = GetCardType (card);
```

SEE ALSO

GetLibVersion() **GetErrorMsg()**

GetChannelCount

PROTOTYPE

#include “CardInit.h”
short **GetChannelCount** (short Card,
CHANCOUNT_p pChanCount);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the DDC Card Manager located in the Control Panel.
pChanCount	(output parameter) Updated Channel count structure.

DESCRIPTION

This function returns the channel count structure with the number of channels and groups of each type available on the device.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_NULL	The structure pChanCount is not valid.
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example will return the number of channels and groups of the on
//card 1.
      short card = 1;
      CHANCOUNT_t strChanCount;
      wStatus = GetChannelCount(Card, &strChanCount);
```

SEE ALSO

None

GetChannelLoopBack

PROTOTYPE

#include “Control.h”
S16BIT **GetChannelLoopBack** (S16BIT Card,
S16BIT Receiver);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receive channel number

DESCRIPTION

This function is used to determine the status of a receiver. The receiver can be either configured for internal or external mode. This function returns if the receiver is internally connected to a transmitter on the board, or connected to an external transmitter.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid receiver defined.

EXAMPLE

```
// This example returns whether or not the loopback function is
// enabled for receiver 1.

     S16BIT Card = 1;
     S16BIT Receiver = 1;
     Return_value = GetChannelLoopBack(Card, Receiver);
```

SEE ALSO

SetChannelLoopBack() **GetLoopBackMapping()**
SetLoopBackMapping()

GetDiscALL

PROTOTYPE

#include “Control.h”
short **GetDiscAll** (S16BIT Card,
U16BIT *Directions,
U16BIT *Levels);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the DDC Card Manager located in the Control Panel.
Directions	(output parameter) Bit wise representation of direction of discrete (bit 15 = MSB) 0 = Discrete input 1 = Discrete output
Levels	(output parameter) Bit wise representation of level of discrete (bit 15 = MSB) 0 = Low 1 = High

DESCRIPTION

This function returns all discrete IO bits directions (input (0) or output (1)) and their levels (0 for Low or a 1 for High).

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_NULL	Null pointer.
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//Will return the current direction of all discrete bits and their
levels.

     S16BIT Card = 1;
     U16BIT Directions, Levels =0;
     Return_value = GetDiscAll(Card, &Directions, &Levels);
```

SEE ALSO

GetDiscOut()
GetDiscDir()
SetDiscDir()
ResetGroup()
GetDiscIn()
SetDiscAll()

GetDiscDir

PROTOTYPE

#include “Control.h”
short ***GetDiscDir*** *(short* Card*,*
short Discrete*);*

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Discrete	(input parameter) The Discrete line to set.

DESCRIPTION

This function retrieves the direction of a discrete line.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_DISCRETE	Invalid discrete output number.
NUMBER	INPUT(0) or OUTPUT(1).

EXAMPLE

```
//This example gets the direction of discrete 1 of card 1.
      short card = 1;
      short discrete = 1;
      short getdiscdir_return = 0;
      getdisdir_return = GetDiscDir (card, discrete);
```

SEE ALSO

GetDiscOut() **ResetGroup()**
SetDiscDir() **GetDiscIn()**

GetDiscIn

PROTOTYPE

```
#include “Control.h”
short GetDiscIn (short Card,
                 short Discrete);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the DDC Card Manager located in the Control Panel.
Discrete	(input parameter) Selects which discrete line to set.

DESCRIPTION

This function returns the current state of the discrete output

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_DISCRETE	Invalid discrete output number.
NUMBER	A zero means the discrete output level is Low and a one means the discrete output level is High.

EXAMPLE

```
//This example will return the level of the discrete output for IRQ 3
// on card 1.
      short card = 1;
      short discrete = 2;
      short getdiscin_return = 0;
      getdiscin_return = GetDiscIn (card, discrete);
```

SEE ALSO

SetDiscOut()
SetDiscDir()
GetDiscDir()
GetDiscOut()

GetDiscOut

PROTOTYPE

```
#include “Control.h”
short GetDiscOut (short Card,
                  short Discrete);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the DDC Card Manager Located in the Control Panel.
Discrete	(input parameter) Selects which discrete line to set.

DESCRIPTION

This function returns the current state of the discrete output.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_DISCRETE	Invalid discrete output number.
NUMBER	A zero means the discrete output level is low and a one means the discrete output level is high.

EXAMPLE

```
//This example will return the level of the discrete output for IRQ 3
// on card 1.
      short card = 1;
      short discrete = 2;
      short getdiscout_return = 0;
      getdiscout_return = GetDiscOut (card, discrete);
```

SEE ALSO

SetDiscOut()
SetDiscDir()
GetDiscDir()
GetDiscIn()

GetErrorMsg

PROTOTYPE

```
#include “Errors.h”
short GetErrorMsg  (short ErrorNumber,
                    char ErrorMessage[]);
```

HARDWARE

Any

PARAMETERS

ErrorNumber	(input parameter) The error status as a negative integer returned by any function in this library.
ErrorMessage	(output parameter) The error message represented by the error number. This array must be at least 80 characters long.

DESCRIPTION

These routines were designed to return information about errors occurring while running the software/hardware. This function reads the error message value and returns the error message associated with the error number.

RETURN VALUE

ERR_FUNCTION	The function failed.
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example will get the error message that is associated with
// error number –33.
      short errornumber = -33;
      char errormessage[120];
      short geterrormsg_return = 0;
      geterrormsg_return = GetErrorMsg (errornumber,
                                        *errormessage);
```

SEE ALSO

GetLibVersion() **GetCardType()**

GetFilter

PROTOTYPE

```
#include “Receive.h”
short GetFilter   (short Card,
                   short Receiver,
                   short LabelSdi);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The Receiver Channel.
LabelSdi	(input parameter) The label (bits 0..7) and SDI (bits 8..9).

DESCRIPTION

This function will tell the user whether or not filters exist.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid receiver defined.
ERR_LABELSDI	Invalid label and SDI.
NUMBER	Returns a zero if the filter does not exist or one if the filter does exist.

EXAMPLE

```
//This example will tell the user whether or not a filter with a label
//of 67 and an SDI of 1 exists on receiver 2 of card 1.
      short card = 1;
      short receiver = 2;
      short labelsdi = 323;
      short getfilter_return = 0;
      getfilter_return = GetFilter (card,
                                    receiver,
                                    labelsdi);
```

SEE ALSO

GetAllFilter()	**GetNumOfFilter()**
DelFilter()	**AddFilter()**
ClearFilter()	**GetFilterStatus()**
EnableFilter()	**ConfigFilter()**

GetFilterStatus

PROTOTYPE

#include “Receive.h”
short **GetFilterStatus** (short Card,
short Receiver);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receiver.

DESCRIPTION

This function will indicate whether the filtering feature is enabled or disabled.

Filtering will be limited to Label only when using IRIG time tagging.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid receiver defined.
NUMBER	Returns a zero if all the filters are off, or a one if all filtering is enabled.

EXAMPLE

```
//This example will return whether or not the filters are on for
// receiver 1 of card 1.
      short card = 1;
      short receiver = 1;
      short getfilterstatus = 0;
      getfilterstatus = GetFilterStatus (card,
                                         receiver);
```

SEE ALSO

EnableFilter() **GetFilterStatus()**
ClearFilter() **ConfigFilter()**

GetHwVersionInfo

PROTOTYPE

#include “CardInit.h”
short **GetHwVersionInfo** (S16BIT Card,
HWVERSIONINFO *pHwVersionInfo);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
pSwVersionInfo	(output parameter) Updated SWVERSIONINFO structure.

DESCRIPTION

This function returns the firmware version and the driver version being used by the ARINC 429 Multi-IO card.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_NULL	Null pointer.
ERR_FUNCTION	The function failed.
ERR_SUCCESS	The function returned successfully.

EXAMPLE

```
//This example will get the current version of the firmaware and
// device driver.

     S16BIT Card = 1;
     HWVERSIONINFO pHwVersionInfo;
     Return_value = GetHwVersionInfo(Card, &pSwVersionInfo);
```

SEE ALSO

GetSwVersionInfo()

GetIntStatus

PROTOTYPE

```
#include “CardInit.h”
short GetIntStatus (short Card,
                    unsigned char Type,
                    unsigned char Channel,
                    unsigned long *Status);
```

HARDWARE

Any

PARAMETERS

Card (input parameter)
The card number given to the card in the Card Manager in the Control Panel.

Type (input parameter)
The interface type. Valid values:
- CHAN_TYPE_429 (Arinc 429/575)
- CHAN_TYPE_UART (RS-232/422/485)
- CHAN_TYPE_717_PROG (Arinc 717/573)
- CHAN_TYPE_CAN – (CanBUS)

Channel (input parameter)
Depends on Type value input above:
- CHAN_TYPE_ARINC_429 – Arinc 429/575 channel number
- CHAN_TYPE_UART – Serial (Uart) channel number
- CHAN_TYPE_717_PROG – Don’t Care (Use ‘1’)
- CHAN_TYPE_CAN – Don’t Care (Use ‘1’)

Status (input/output parameter)
Pointer to Interrupt Status Packet.

***Note**: Definition and Size Differs based on ‘Type’ value above:*

***Note**: A BU-67118K/M/Y/Z board uses the Cast IO H16550S IP, refer to the BU-67118K/M/Y/Z table below*

GetIntStatus (continued)

ARINC 429/575 Interrupt Status Packet Type = CHAN_TYPE_429 Packet Size = 4 bytes (32 bits)		
Bit Value	**Description**	**Constant**
32-14	Reserved.	Reserved.
13	Protocol Word	INT_COND_429_PROTO
12	Fail Warning	INT_COND_429_FAIL
11	Function Test	INT_COND_429_FUNCTEST
10	Solo Word	INT_COND_429_SOLO
09	Start of Transmisiion	INT_COND_429_TXSTART
08	No Data	INT_COND_429_NODATA
07	End of Transmission	INT_COND_429_TXEND
06	Normal Operation	INT_COND_429_NORMAL
05	Data Match	INT_COND_429_DATAMATCH
04	Word Type	INT_COND_429_WORDTYPE
03	FIFO Full	INT_COND_429_RXFULL
02	Parity Error	INT_COND_429_RXPARITY
01	FIFO Overflow	INT_COND_429_RXOVRFLO

BU-67118K/M/Y/Z UART(RS-232/422/485) Interrupt Status Packet Type = CHAN_TYPE_UART Packet Size = 2 DWORDS (64 bits)		
Status DWORD	**Description**	**Information**
Status DWord 1	Interrupted channel number.	Bit wise data example: Value of 0x00000001 = CH 1 Value of 0x00000002 = CH 2. Value of 0x00000004 = CH 3.
Status Dword 2	Number of bytes added to the host buffer.	This value represents the number of bytes the current interrupt added to the buffer.

GetIntStatus (continued)

UART (RS-232/422/485) Interrupt Status Packet Type = CHAN_TYPE_UART Packet Size = 1 byte (8 bits)		
8 Bit Value (Hex)	**Description**	**Constant**
0x20	CTS/RTS Change of State	INT_STAT_SER_STATE
0x10	Received Xon//Xoff	INT_STAT_SER_XONOFF
0x0C	Receive Data Timeout	INT_STAT_SER_RXTO
0x06	Receiver Line Status	INT_STAT_SER_LSR
0x04	Received Data Ready	INT_STAT_SER_RXRDY
0x02	Transmitter Ready	INT_STAT_SER_TXRDY

ARINC 717/573 Interrupt Status Packet Type = CHAN_TYPE_717_PROG Packet Size = 8 bytes + ('channel count' * 4) Example: 2x 717 channels = 8 bytes + (2*4) = 16 bytes (4 Dwords)		
Dword	**Byte**	**Description**
17	68	Channel #16 Interrupts
…	…	…
03	12-15	Channel #2 Interrupt Status Word (See Below for Bits)
02	08-11	Channel #1 Interrupt Status Word (See Below for Bits)
01	04-07	Global Channel Event Status Word (See Below for Bits)
00	00-03	Master Status Word (Reserved)

ARINC 717/573 Global Channel Event Status Word		
Bit Value	**Description**	**Constant**
32	Channel #32 Interrupted	ARINC_717_CH32_INT
…	…	…
02	Channel #02 Interrupted	ARINC_717_CH2_INT
01	Channel #01 Interrupted	ARINC_717_CH1_INT

GetIntStatus (continued)

ARINC 717/573 Channel #x Interrrupt Status Word		
Bit Value	**Description**	**Constant**
32-25	Reserved	Reserved
24	Receiver lost lock on correct speed (Bit Errors).	ARINC_717_PROG_INT_RX_AUTO_DETECT_LOST_ENA
23	Receiver detected correct speed.	ARINC_717_PROG_INT_RX_AUTO_DETECT_LOCK_ENA
22	Valid Sync Word not detected at start of frame	ARINC_717_PROG_INT_RX_REC_SYNCED_ERR_ENA
21	Valid Sync Word detected when not synced.	ARINC_717_PROG_INT_RX_REC_SYNCED_ENA
20	Secondary Receiver Buffer filled.	ARINC_717_PROG_INT_RX_100_PC_MEM_ENA
19	Primary Receiver Buffer filled.	ARINC_717_PROG_INT_RX_50_PC_MEM_ENA
18-03	Reserved	Reserved
02	Secondary Transmit Buffer Sent.	ARINC_717_PROG_INT_TX_MARKER1_ENA
01	Primary Transmit Buffer Sent.	ARINC_717_PROG_INT_TX_MARKER0_ENA

CanBUS Interrupt Status Packet Type = CHAN_TYPE_CAN Packet Size = 4 bytes + ('channel count' * 4) Example: 2x CanBUS channels = 4 bytes + (2*4) = 12 bytes (3 Dwords)		
Dword	**Byte**	**Description**
08	32-35	Number of Messages received on Channel #8 (32-bits)
…	…	…
02	08-11	Number of Messages received on Channel #2 (32-bits)
01	04-07	Number of Messages received on Channel #1 (32-bits)
00	00-03	Global Channel Event Status Word (See Below for Bits)

CanBUS Global Channel Event Status Word		
Bit Value	**Description**	**Constant**
32	Channel #21 Interrupted	MIO_INT_STATUS_MASK_CAN_21
…	…	…
13	Channel #02 Interrupted	MIO_INT_STATUS_MASK_CAN_2
12	Channel #01 Interrupted	MIO_INT_STATUS_MASK_CAN_1
11-01	Reserved	Reserved

GetIntStatus (continued)

DESCRIPTION

This function will get the current status of interrupt condition for various protocol interfaces.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_SUCCESS	The function returned successfully.
ERR_CRDINIT	Card is not initialized.
ERR_IRQ	IRQ(s) in use.
ERR_RX	Invalid receiver defined.
ERR_NULL	Null pointer.

EXAMPLE

```
//This gets the current status of the interrupt for ARINC429 Multi-IO
// SDK protocol on channel 1.

     Short card = 1;
     Short wStatus = 0;
     Unsigned char bChan = 1
     Unsigned long Status;
     wStatus += GetIntStatus(Card, CHAN_TYPE_429, bChan, &Status);
```

SEE ALSO

InstallHandler()
UninstallHandler()
SetIntCondition()
acexArinc717Interrupts()

GetLibVersion

PROTOTYPE

#include “CardInit.h”
unsigned short **GetLibVersion** (void);

HARDWARE

Any

PARAMETERS

None

DESCRIPTION

This function returns the version of this library. The high byte contains the major version and the low byte contains the minor version. For example, this function returns 0x102 for “Version 1.02”, 0x22C for “Version 2.44”. New software packages should use **GetLibVersionEx**.

RETURN VALUE

Number	The version of this DLL/Toolbox.

EXAMPLE

```
//This example returns the library version.
//unsigned short getlibversion_return = 0;

      getlibversion_return = GetLibVersion ();
```

SEE ALSO

GetCardType() **GetErrorMsg()**
GetLibVersionEx()

GetLibVersionEx

PROTOTYPE

```
#include “CardInit.h”
short GetLibVersionEx  (unsigned short *major,
                        unsigned short *minor,
                        unsigned short *engRelease);
```

HARDWARE

Any

PARAMETERS

major	(output parameter) The major version number of the library.
minor	(output parameter) The minor version number of the library.
engRelease	(output parameter) The engineering version number of the library.

DESCRIPTION

This function will get the library version. The last digit of an engineering version number that is not equal to zero means that this version of the library has not been officially released and validated. For example, version 4.4.0 is an official release whereas version 4.3.1 is an intermediate engineering release.

RETURN VALUE

ERR_NULL	Null pointer.
ERR_SUCCESS	Function returned successfully.

EXAMPLE

```
//This example will get the library version.
      unsigned short major = 0;
      unsigned short minor = 0;
      unsigned short engRelease = 0;

      GetLibVersionEx (major, minor, engRelease);
      printf (“The library version is %d”, major);
      printf (“.%d”, minor);
      printf (“.%d”, engRelease);
```

SEE ALSO

GetCardType()
GetErrorMsg()
GetLibVersion()

GetLoopBack

PROTOTYPE

```
#include “Control.h”
short GetLoopBack  (short Card,
                    short Group);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Group	(input parameter) The group number

DESCRIPTION

This function tells the status of the loopback from the transmitter to the receivers in the same group. This function is not recommended for use. The function GetChannelLoopBack is recommended for use over GetLoopBack.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_CHNLGROUP	Invalid channel group.
NUMBER	A one if the loopback from the transmitter to the receivers is enabled, or a zero if the loopback is disabled.

EXAMPLE

```
//This example returns whether or not the loopback function is enabled
//for group 2 (transmitter 2 and receivers 3 and 4) of card 1.
      short card = 1;
      short group = 2;
      short getloopback_return = 0;
      getloopback_return = GetLoopBack (card,
                                        group);
```

SEE ALSO

SetLoopBack()

GetLoopBackMapping

PROTOTYPE

#include “Control.h”
S16BIT **GetLoopBackMapping** (S16BIT Card,
S16BIT Receiver,
S16BIT *TxMapping);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receive channel number
TxMapping	(output parameter) The transmitter connected to the Receiver

DESCRIPTION

This function will return which transmitter is connected to the specified Receiver. This function is only valid for internal loopback mode.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_Rx	Invalid receiver defined.
ERR_NORES	Card not responding.
ERR_NULL	A null value.
ERR_SUCCESS	The function returned successfully.

EXAMPLE

```
//This example which transmitter is connected to Receiver.

     S16BIT Card = 1;
     S16BIT Receiver = 1;
     S16BIT TxMapping = 0;
     Return_value = GetLoopBackMapping(Card, Receiver, &TxMapping);
```

SEE ALSO

GetChannelLoopBack() **SetChannelLoopBack()**
SetLoopBackMapping()

GetMailbox

PROTOTYPE

```
#include “Receive.h”
short GetMailbox (short Card,
                  short Receiver,
                  short N,
                  short *LabelSdi);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receiver.
N	(input parameter) The maximum number of locations to get and return in the array.
LabelSdi	(output parameter) An array of label/SDI locations where a new ARINC word is received. The label (bits 0..7) and SDI (bits 8..9).

DESCRIPTION

This function tells the locations in the mailbox where new words are received. If **N**=0 or **LabelSdi**=NULL, this function returns the total number of new words that have been received by the receiver’s mailbox.

> ***Note:*** *This function does not read the ARINC words and thus does not change their status from “new” to “old”.*

GetMailbox (continued)

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid receiver.
ERR_MODE	Invalid mode of operation
ERR_NULL	Null pointer.
NUMBER	The number of new word locations returned in the array. If N is zero or LabelSdi is NULL, then the total number of new words in all mailbox slots is returned.
ERR_RXQUEUESZ	Invalid queue size.

EXAMPLE

```
//This example returns the number of new word locations returned in
//the labelsdi array for receiver 1 of card 1.
      short card = 1;
      short receiver = 1;
      short n = 5;
      short labelsdi[1024];
      short getmailbox_return = 0;
      getmailbox_return = GetMailbox (card,
                                      receiver,
                                      n,
                                      *labelsdi);
```

SEE ALSO

ReadMailboxIrig() **GetMailbox()**
ClearMailbox()

GetMailboxStatus

PROTOTYPE

```
#include “Receive.h”
short GetMailboxStatus (short Card,
                        short Receiver,
                        short LabelSdi);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receiver (1..4).
LabelSdi	(input parameter) The label/SDI for a particular mailbox slot. Bits 0..7 are the label and bits 8..9 are the SDI.

DESCRIPTION

This function tells whether the mailbox slot has received a new word for a specific LabelSDI location.

Note: *When the time stamp is enabled the SDI portion of the SDI/Label parameter passed into the function* ***ReadMailboxIrig()*** *must be ‘0’. Any value other than ‘0’ will result in an error being returned from the function* ***ReadMailboxIrig()****. The enabled time stamp makes the SDI a “Don’t Care” which allows the function to return the last specified label.*

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid receiver.
ERR_LABELSDI	Invalid label and SDI.
ERR_MODE	Invalid mode of operation
NUMBER	A one if a new word is in this mailbox slot, or zero if the word is old.

GetMailboxStatus (continued)

EXAMPLE

```
//This example returns a 1 if a new word is in the mailbox slot for a
//label of 5 and an sdi of 0 for receiver 1 of card 1.
      short card = 1;
      short receiver = 1;
      short labelsdi = 5;
      short getmailboxstatus_return = 0;
      getmailboxstatus_return = GetMailboxStatus (card,
                                                  receiver,
                                                  labelsdi);
```

SEE ALSO

ReadMailboxIrig() **GetMailbox()**
ClearMailbox()

GetNumOfFilter

PROTOTYPE

#include “Receive.h”
short **GetNumOfFilter** (short Card,
short Receiver);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receiver.

DESCRIPTION

This function returns the number of existing filters for one receiver.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid receiver.
NUMBER	The number of receiver’s existing filters.

EXAMPLE

```
//This example returns the number of existing filters for receiver 1
// of card 1.
      Short card = 1;
      Short receiver = 1;
      Short getnumoffilter_return = 0;
      Getnumoffilter_return = GetNumOfFilter (card,
                                              receiver);
```

SEE ALSO

GetFilter()	**GetAllFilter()**
DelFilter()	**AddFilter()**
ClearFilter()	**GetFilterStatus()**
EnableFilter()	**ConfigFilter()**

GetNumOfRepeated

PROTOTYPE

#include “Transmit.h”
short **GetNumOfRepeated** (short Card,
short Transmitter);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Transmitter	(input parameter) The transmitter (1..2).

DESCRIPTION

This function returns the total number of words that are in the transmitter’s schedule table.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_TX	Invalid transmitter defined.
NUMBER	The total number of words in the scheduled transmission table.

EXAMPLE

```
//This example returns the total number of words that are in the
//schedule table for transmitter 1 of card 1.
      Short card = 1;
      Short transmitter = 1;
      Short getnumofrepeated_return = 0;
      Getnumofrepeated_return = GetNumOfRepeated (card,
                                                  transmitter);
```

SEE ALSO

GetRepeated()
GetAllRepeated()
AddRepeated()
DelRepeated()
ClearRepeated()

GetRepeated

PROTOTYPE

```
#include “Transmit.h”
short GetRepeated   (short Card,
                     short Transmitter,
                     short LabelSdi,
                     unsigned long *Data,
                     short *Frequency,
                     short *Offset);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Transmitter	(input parameter) The transmitter.
LabelSdi	(input parameter) The mailbox slot of the specified label/SDI. Bits 0..7 are the label and bits 8..9 are the SDI.
Data	(output parameter) The entire 32-bit ARINC word.
Frequency	(output parameter) The frequency in milliseconds.
Offset	(output parameter) Offset in milliseconds.

DESCRIPTION

This function determines whether a specific label and SDI are in the transmitter’s schedule table or not. If they are, the function returns the ARINC word, the frequency in milliseconds and the offset.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_TX	Invalid transmitter.
ERR_LABLESDI	Invalid Label/SDI value
ERR_NULL	Null pointer.
NUMBER	A one if it is in the table. A zero if it is not in the table.

GetRepeated (continued)

EXAMPLE

```
//This example returns whether or not a label of 5 and an SDI of 0 are
//in the transmitter's schedule table for transmitter 1 of card 1. The
//function then gets the ARINC word, the frequency in milliseconds and
//the offset.
      short card = 1;
      short transmitter = 1;
      short labelsdi = 5;
      unsigned long data[1];
      short frequency[1];
      short offset[1];
      short getrepeated_return = 0;
      getrepeated_return = GetRepeated (card,
                                        transmitter,
                                        labelsdi,
                                        *data,
                                        *frequency,
                                        *offset);
```

SEE ALSO

GetNumOfRepeated()
GetAllRepeated()
AddRepeated()
DelRepeated()
ClearRepeated()

GetRxChannelMode

PROTOTYPE

#include “Receive.h”
short **GetRxChannelMode** (short Card,
short Receiver);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receiver.

DESCRIPTION

This function returns the receiver’s mode of operation, either FIFO or Mailbox.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid receiver.
NUMBER	FIFO_MODE (1) for when channel is using a FIFO. A value of MAILBOX_MODE (0) for when the channel is in mailbox mode.

EXAMPLE

```
//This example returns which mode receiver 1 is in.
      short card = 1;
      short Receiver = 1;
      short getrxmode_return = 0;
      getrxmode_return = GetRxChannelMode(card, Receiver);
```

SEE ALSO

SetRxChannelMode()

GetRxChannelParity

PROTOTYPE

#include “Receive.h”
short **GetTxChannelParity** (short Card,
short Receiver);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The Receiver

DESCRIPTION

This function returns the receiver’s parity setting.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid Receiver defined.
NUMBER	Returns either DD429_NO_PARITY (0), DD429_ODD_PARITY (1), or DD429_EVEN_PARITY (2).

EXAMPLE

```
//This example returns the parity setting for receiver 2 of card 1.
      short card = 1;
      short receiver = 2;
      short getrxparity_return = 0;
      getrxparity_return = GetRxChannelParity (card, receiver);
```

SEE ALSO

SetRxChannelParity()

GetRxChannelSpeed

PROTOTYPE

#include “Receive.h”
short **GetRxChannelSpeed** (short Card,
short Receiver);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receiver.

DESCRIPTION

This function returns the speed of the receiver.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid receiver.
ERR_ARINC575	ARINC 575 bit rate
NUMBER	A value of DD429_HIGH_SPEED (1) for high speed. A value of DD429_LOW_SPEED (0) for low speed. A value of 2 represents DD429_ARINC575 speed.

EXAMPLE

```
//This example returns the speed of receivers 1 of card 1
      short card = 1;
      short Receiver = 1;
      short getrxspeed_return = 0;
      getrxspeed_return = GetRxChannelSpeed (card, Receiver);
```

SEE ALSO

SetRxChannelSpeed() **SetRxChannelParity()**
GetRxChannelParity()

GetRxQueueStatus

PROTOTYPE

#include “Receive.h”
short **GetRxQueueStatus** (short Card,
short Receiver);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receiver.

DESCRIPTION

This function tells the status of the receiver queue.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid receiver.
ERR_MODE	Invalid mode of operation
NUMBER	Returns the number of words left in the queue.

EXAMPLE

```
//This example returns the number of words left in the queue for
//receiver 1 of card 1.
      short card = 1;
      short receiver = 1;
      short getrxqueuestatus_return = 0;
      getrxqueuestatus_return = GetRxQueueStatus (card,
                                                  receiver);
```

SEE ALSO

ClearRxQueue()
ReadRxQueueIrigMore()
ReadRxQueueIrigOne()

GetRxMode

PROTOTYPE

#include “Receive.h”
short **GetRxMode** (short Card,
short Group);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Group	(input parameter) The receiver group.

DESCRIPTION

This function returns the receiver’s mode of operation, either FIFO or Mailbox.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RXGROUP	Invalid receiver.
ERR_MODE	Invalid operating mode
NUMBER	FIFO_MODE (1) for when the group is using a FIFO. A value of MAILBOX_MODE (0) for when the group is in mailbox mode.

EXAMPLE

```
//This example returns which mode the group 1 is in.
      short card = 1;
      short group = 1;
      short getrxmode_return = 0;
      getrxmode_return = GetRxMode(card, group);
```

SEE ALSO

SetRxMode() **EnableRx()**
GetRxStatus()

GetRxParity

PROTOTYPE

#include “Receive.h”
short **GetRxParity** (short Card,
short Group);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Group	(input parameter) The Receiver Group

DESCRIPTION

This function returns the receiver’s parity setting.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid Receiver defined.
ERR_PARITY	Invalid parity.
NUMBER	Returns either DD429_NO_PARITY (0), DD429_ODD_PARITY (1), or DD429_EVEN_PARITY (2).

EXAMPLE

```
//This example returns the parity setting for receiver 2 of card 1.
      short card = 1;
      short receiver = 2;
      short getrxparity_return = 0;
      getrxparity_return = GetRxParity (card, receiver);
```

SEE ALSO

SetRxParity() **EnableRx()**
GetRxStatus()

GetRxSpeed

PROTOTYPE

```
#include “Receive.h”
short GetRxSpeed(short Card,
                 short Group);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Group	(input parameter) The receiver group.

DESCRIPTION

This function returns the speed of the receivers in a group.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RXGROUP	Invalid receiver.
ERR_ARINC575	ARINC 575 bit rate
NUMBER	A value of DD429_HIGH_SPEED (1) for high speed. A value of DD429_LOW_SPEED (0) for low speed. A value of 2 represents DD429_ARINC575 speed.

EXAMPLE

```
//This example returns the speed of the receivers in group 1
//(receivers 1 and 2) of card 1;
      short card = 1;
      short group = 1;
      short getrxspeed_return = 0;
      getrxspeed_return = GetRxSpeed (card, group);
```

SEE ALSO

SetRxSpeed()
GetRxStatus()
GetRxParity()
EnableRx()
SetRxParity()

GetRxStatus

PROTOTYPE

#include “Receive.h”
short **GetRxStatus** (short Card,
short Receiver);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receiver.

DESCRIPTION

This function returns the receiver’s status.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid receiver.
NUMBER	DD429_ENABLED (1 is returned if enabled. DD429_DISABLE (0) is returned if disabled).

EXAMPLE

```
//This example returns whether or not receiver 1 is enabled .
      short card = 1;
      short receiver = 1;
      short getrxstatus_return = 0;
      getrxstatus_return = GetRxStatus (card, receiver);
```

SEE ALSO

EnableRx()
GetRxSpeed()
SetRxSpeed()
SetRxParity()
GetRxParity()

GetSwVersionInfo

PROTOTYPE

#include “CardInit.h”
short **GetSwVersionInfo** (SWVERSIONINFO *pSwVersionInfo);

HARDWARE

Any

PARAMETERS

pSwVersionInfo	(output parameter) Updated SWVERSIONINFO structure.

DESCRIPTION

This function gets the version of the Multi-IO SDK and stores it into the SWVERSIONINFO structure.

RETURN VALUE

ERR_NULL	Null pointer.
ERR_SUCCESS	The function returned successfully.

EXAMPLE

```
//This example will get the current version of the SDK.

      SWVERSIONINFO pSwVersionInfo;
      Return_value = GetSwVersionInfo(&pSwVersionInfo);
```

SEE ALSO

GetHwVersionInfo()

GetTimeStamp

PROTOTYPE

#include “Receive.h”
short **GetTimeStamp** (S16BIT Card,
U64BIT *ullTTValue);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
ullTTValue	(output parameter) Current Hardware Time Tag Value

DESCRIPTION

This function returns the 48-Bit Time Tag value.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_NULL	Null Pointer
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example will get the current 48-Bit Time Tag.

     S16BIT Card = 1;
     U64BIT ullTTValue =0;
     Return_value = GetTimeStamp(Card, &ullTTValue);
```

SEE ALSO

EnableTimeStamp() **ConfigTimeStamp()**
GetTimeStampStatus() **ResetTimeStamp()**

GetTimeStampConfig

PROTOTYPE

```
#include “Receive.h”
S16BIT GetTimeStampconfig(S16BIT Card,
                          U8BIT *Format,
                          U8BIT *Rollover,
                          U8BIT *Resolution);
```

HARDWARE

Any

PARAMETERS

Card (input parameter)
The card number given to the card in the Card Manager in the Control Panel.

Format (output parameter)
The format of the time tag
- TT48
- IRIG_B

Rollover (output parameter)
The rollover value

TTRO_16	2^{16}
TTRO_17	2^{17}
TTRO_18	2^{18}
TTRO_19	2^{19}
TTRO_20	2^{20}
TTRO_21	2^{21}
TTRO_22	2^{22}
TTRO_48	2^{48}

Resolution (output parameter)
48-Bit LSB Resolution

TTRES_1	1µS
TTRES_2	2µS
TTRES_4	4µS
TTRES_8	8µS
TTRES_16	16µS
TTRES_32	32µS
TTRES_64	64µS

DESCRIPTION

This function will return the time tag configuration of the device.

GetTimeStampConfig (continued)

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_NORES	Card not responding.
ERR_SUCCESS	The function returned successfully.

EXAMPLE

```
//This example configures transmitter 1 to receiver 1 in loopback
mode.

     S16BIT Card = 1;
     S16BIT Receiver = 1;
     U8BIT Format, Rollover, Resolution =0;
     Return_value = GetTimeStampConfig(Card,
                                       *Format,
                                       *Rollover,
                                       *Resolution);
```

SEE ALSO

GetTimeStampStatus() **GetTimeStamp()**
ConfigTimeStamp()

GetTimeStampStatus

PROTOTYPE

#include “Receive.h”
short **GetTimeStampStatus** (short Card,
short Receiver);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receiver.

DESCRIPTION

This function returns the receiver’s time stamp status.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid receiver.
NUMBER	The current receiver’s time stamp status. 0 = Disabled 1 = Enabled

EXAMPLE

```
//This example returns the current time stamp status for receiver 2 of
//card 2.
      short card = 2;
      short receiver = 2;
      short gettimestampstatus_return = 0;
      gettimestampstatus_return = GetTimeStampStatus (card,  receiver);
```

SEE ALSO

EnableTimeStamp() **ResetTimeStamp()**

GetTxParity

PROTOTYPE

#include “Transmit.h”
short **GetTxParity** (short Card,
short Transmitter);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Transmitter	(input parameter) The transmitter (1..2).

DESCRIPTION

This function returns the transmitter parity setting.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_TX	Invalid transmitter defined.
NUMBER	Returns either DD429_NO_PARITY (0), DD429_ODD_PARITY (1), or DD429_EVEN_PARITY (2).

EXAMPLE

```
//This example returns the parity setting for transmitter 2 of card 1.
      short card = 1;
      short transmitter = 2;
      short gettxparity_return = 0;
      gettxparity_return = GetTxParity (card,
                                        transmitter);
```

SEE ALSO

SetTxParity() **EnableTx()**
GetTxStatus()

GetTxSpeed

PROTOTYPE

```
#include “Transmit.h”
short GetTxSpeed   (short Card,
                    short Transmitter);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Transmitter	(input parameter) The transmitter (1..2).

DESCRIPTION

This function returns the transmitter’s speed.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_TX	Invalid transmitter defined.
ERR_ARINC575	ARINC 575 bit rate
NUMBER	DD429_LOW_SPEED (0) means low speed. DD429_HIGH_SPEED (1) means high speed. ARINC575 (2) represents DD429_ARINC575.

EXAMPLE

```
//This example returns the speed setting for transmitter 1 of card 1.
      short card = 1;
      short transmitter = 1;
      short gettxspeed_return = 0;
      gettxspeed_return = GetTxSpeed (card, transmitter);
```

SEE ALSO

SetTxSpeed()
GetTxStatus()
GetTxParity()
EnableTx()
SetTxParity()

GetTxStatus

PROTOTYPE

#include “Transmit.h”
short **GetTxStatus** (short Card,
short Transmitter);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Transmitter	(input parameter) The transmitter (1..2).

DESCRIPTION

This function returns the transmitter’s status.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_TX	Invalid transmitter defined.
NUMBER	DD429_DISABLE (0) means transmitter is disabled. DD429_ENABLE (1) means transmitter is enabled.

EXAMPLE

```
//This example returns whether or not transmitter 1 of card 1 is
enabled.
      short card = 1;
      short transmitter = 1;
      short gettxstatus_return = 0;
      gettxstatus_return = GetTxStatus (card,
                                        transmitter);
```

SEE ALSO

EnableTx()

GetTxQueueStatus

PROTOTYPE

```
#include “Transmit.h”
short GetTxQueueStatus (short Card,
                        short Transmitter);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Transmitter	(input parameter) The transmitter.

DESCRIPTION

This function determines a transmitter’s queue status.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_TX	Invalid transmitter defined.
NUMBER	A Zero if queue is empty, MIO_SCH_FIFO_LEN otherwise.

EXAMPLE

```
//This example returns the number of words left in the queue of
//transmitter 1 of card 1.
      short card = 1;
      short transmitter = 1;
      short gettxqueuestatus_return = 0;
      gettxqueuestatus_return = GetTxQueueStatus (card,
                                                  transmitter);
```

SEE ALSO

LoadTxQueueMore() **LoadTxQueueOne()**

InitCard

PROTOTYPE

#include “CardInit.h”
short **InitCard** (short Card);

HARDWARE

Any

PARAMETERS

Card	(input parameter) Logical device number assigned to device via the DDC Card Manager.

DESCRIPTION

Initializes the selected card to the following state:
Bit format of the ARINC words: original.
No loopback from the transmitter to the receivers in each group.
Odd transmitter parity setting. ***See note below**
Low speed for all transmitters.
All transmitters disabled.
Transmitter queues cleared.
No repeated transmission scheduled.
Low speed for all receivers.
All receivers disabled.
No existing receiver filter.
Receiver filters disabled (not used).
No new data in receiver’s mailbox.
Receiver queues cleared.
No time stamps for received words.

***Note**: The parity setting does not apply to **DD-40x00x** series cards.
A user must configure the parity via the set parity functions.*

RETURN VALUE

ERR_INITFAIL	Low level initialization failure
ERR_FLASH	Card firmware not supported
ERR_INUSE	Card in use.
ERR_NOCRD	No card is present.
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example initializes card 1.
     Short card = 1;
     InitCard (card);
```

SEE ALSO

FreeCard()

InstallFifoRxHostBuffer

PROTOTYPE

#include “receive.h”
S16BIT **InstallFifoRxHostBuffer** (short Card,
U32BIT reserved);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
reserved	(input parameter) Reserved for future use. Set to 0.

DESCRIPTION

This function will install a Host Buffer for ARINC 429 recevied FIFO traffic. The Host Buffer will improve performance on receiving data and can be used instead of the ReadRxQueue functions.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_SUCCESS	The function returned successfully.
ERR_CRDINIT	Card is not initialized.
ERR_CRDINIT	Card is not initialized.
ERR_RX_HBUF_INSTALL	Receive Host Buffer is already installed.
ERR_NORES	Hardware is not responding.

EXAMPLE

```
// This example installs an receive Host Buffer for card 1.
     Short card = 1;
     Short wStatus = 0;
     wStatus += InstallFifoRxHostBuffer(Card, 0);
```

SEE ALSO

UninstallFifoRxHostBuffer()
ReadRxHostBuffer()
EnableRxHostBuffer()
DisableRxHostBuffer()

InstallHandler

PROTOTYPE

```
#include “CardInit.h”
short InstallHandler    (short Card,
                         unsigned char Type,
                         short (*Handler)(short),
                         short param);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Type	(input parameter) The interface type. Valid values: CHAN_TYPE_429 CHAN_TYPE_UART CHAN_TYPE_717_PROG CHAN_TYPE_CAN
Handler	(input parameter) User defined interrupt handler.
Param	The parameter to be passed into the interrupt handler. Typically the logical device number of the card.

DESCRIPTION

This function will install a user defined interrupt handler for a specified type of interface.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_SUCCESS	The function returned successfully.
ERR_NULL	Null function pointer.
ERR_CRDINIT	Card is not initialized.
ERR_CHAN_TYPE	Invalid channel type.
ERR_IRQ	Interrupts not enabled.
ERR_CHNLGROUP	Invalid channel group.
ERR_SERIAL	Serial error.
ERR_CHAN_TYPE	Invalid channel type.

InstallHandler (continued)

EXAMPLE

```
//This example installs an interrupt handler for card 1.
      Short card = 1;
      Short wStatus = 0;
      short fifo_rx_ISR(S16BIT Card);
      wStatus += InstallHandler(Card, CHAN_TYPE_429, fifo_rx_ISR,Card);
```

SEE ALSO

UninstallHandler()
SetIntCondition()
GetIntStatus()
acexArinc717Interrupts()

IOFree

PROTOTYPE

#include “Control.h”
short **IOFree** (S16BIT Card);

HARDWARE

Any

PARAMETERS

Card	(input parameter) Logical device number assigned to device via the DDC Card Manager.

DESCRIPTION

This function frees up resources used by the discrete or avionic IO. This function must be used to ensure the next call to **IOInitialize** will run properly.

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example will get the current 48-Bit Time Tag.

     S16BIT Card = 1;
     Return_value = IOFree(Card);
```

SEE ALSO

IOInitialize()

IOInitialize

PROTOTYPE

#include “Control.h”
short **IOInitialize** (S16BIT Card, U16BIT Options);

HARDWARE

Any

PARAMETERS

Card	(input parameter) Logical device number assigned to device via the DDC Card Manager.
Options	(input parameter) Future Use

DESCRIPTION

This function initializes the discrete and avionic IO separately from the InitCard() function.

RETURN VALUE

ERR_NOCRD	No card is present
ERR_INUSE	Avionics and Discrete IO are already in use.
ERR_CRDSERV	Failed attempt to access device via the driver.
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example will get the current 48-Bit Time Tag.

     S16BIT Card = 1;
     Return_value = IOInitalize(Card);
```

SEE ALSO

IOFree()

LoadTxQueueMore

PROTOTYPE

```
#include “Transmit.h”
short LoadTxQueueMore (short Card,
                       short Transmitter,
                       short N,
                       unsigned long *Data);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Transmitter	(input parameter) The transmitter (1..2).
N	(input parameter) The number of ARINC words in the array. Max size is 256 of ARINC message.
Data	(input parameter) An array of 32-bit words to load.

DESCRIPTION

This function loads multiple 32-bit words into the transmitter’s queue, until either the **N** words are all loaded or the queue becomes full. Make sure the transmitter is enabled to transmit these words.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_TX	Invalid transmitter defined.
ERR_LABELSDI	Invalid label or SDI.
ERR_ENABLE	Channel not enabled
ERR_TIMEOUT	A timeout occurred when writing message to the hardware queue.
NUMBER	Returns the actual number of ARINC words that are loaded.

LoadTxQueueMore (continued)

EXAMPLE

```
//This example will load the 3 data words into the queue of
//transmitter 1 of card 1. The function returns the actual number of
//ARINC words that were loaded.
      short card = 1;
      short transmitter = 1;
      short n = 3;
      unsigned long data[3];
      data[0] = 0x12345678;
      data[1] = 0x87654321;
      data[2] = 0x12344321;
      short loadtxqueuemore_return = 0;
      loadtxqueuemore_return = LoadTxQueueMore (card,
                                                transmitter,
                                                n,
                                                *data);
```

SEE ALSO

LoadTxQueueOne() **GetTxQueueStatus()**

LoadTxQueueOne

PROTOTYPE

#include “Transmit.h”
short **LoadTxQueueOne** (short Card,
short Transmitter,
unsigned long Data);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Transmitter	(input parameter) The transmitter (1..2).
Data	(input parameter) The 32-bit data word to load.

DESCRIPTION

This function loads the 32-bit word directly into the transmitter’s queue. Make sure the transmitter is enabled to transmit the word.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_TX	Invalid transmitter defined.
ERR_LABELSDI	Invalid label or SDI.
ERR_ENABLE	Channel not enabled
ERR_TIMEOUT	A timeout occurred when writing message to the hardware queue.
NUMBER	Returns a one if function was successful. Returns a zero if the queue is full.

EXAMPLE

```
//This example loads the 32-bit data 0x12345678 directly into the
//queue of transmitter 1 of card 1.
      short card = 1;
      short transmitter = 1;
      unsigned long data = 0x12345678;
      short loadtxqueueone_return = 0;
      loadtxqueueone_return = LoadTxQueueOne (card,
                                              transmitter,
                                              data);
```

LoadTxQueueOne (continued)

SEE ALSO

LoadTxQueueMore() **GetTxQueueStatus()**

ModifyRepeatedData

PROTOTYPE

```
#include “Transmit.h”
short ModifyRepeatedData (short Card,
                          short Transmitter,
                          unsigned long u32Data,
                          unsigned long u32Option);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Transmitter	(input parameter) The desired transmitter.
u32Data	(input parameter) The updated ARINC word including the label and SDI.
u32Option	(input parameter) Update Options: Valid choices include: ARINC_429_MODIFY_REPEATED_VIA_LABEL Update Data if only the Label matches. ARINC_429_MODIFY_REPEATED_VIA_SDI_LABEL Update Data if the SDI and Label both match. This value must be less than the Frequency parameter (1..32767).

DESCRIPTION

This function updates the data portion of a previously scheduled ARINC 429 message (using **AddRepeated**). This function has no effect on any message scheduling and it’s sole purpose is to update the data portion (bits 11-29) of a currently scheduled ARINC word.

Note: *SDI/Label must exist in scheduled transmission list for data to be updated.*

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_TX	Invalid transmitter defined
ERR_LABELSDI	Invalid label or SDI
ERR_ENABLE	Transmitter not enabled
ERR_FUNCTION	SDI/Label not in scheduled transmission list.

ModifyRepeatedData (continued)

EXAMPLE

```
/* This example modifies the data for SDI/label 0x1F on transmitter one of
card one.  Note, that an ARINC Message for SDI/label 0x1F must be previously
scheduled via AddReapeated() before using this function. */

            unsigned long data = 0x1234561F;
            short card = 1;
            short transmitter = 1;
            short modifyrepeated_return = 0;
            modifyepeated_return = ModifyRepeated(card,
                                                  transmitter,
                                                  data,

            ARINC_429_MODIFY_REPEATED_VIA_SDI_LABEL);
```

SEE ALSO

GetNumOfRepeated()
GetAllRepeated()
AddRepeated()
GetRepeated()
DelFilter()
ClearRepeated()

ReadUart

PROTOTYPE

```
#include “serial.h”
short ReadUart   (short Card,
                  unsigned char Chan,
                  unsigned char Reg,
                  unsigned char *Data);
```

HARDWARE

BU-67211Ux, BU-67107F/M/i/T, BU-67118K/M/Y/Z

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Chan	(input parameter) Serial I/O Channel. Valid Values: 1 – 4 (depending on hardware)
Reg	(input parameter) UART configuration Register
Data	(output parameter) 8-Bit UART Data

DESCRIPTION

This function provides the user read access to the UART configuration / control /data registers. Refer to the Exar XR16L784 datasheet for configuration and usage details.

For customers using the ***BU-67118K/M/Y/Z*** series cards, you can refer to the H16650S CAST IO datasheet for configuration and usage details.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_SUCCESS	The function returned successfully.
ERR_CRDINIT	Card is not initialized.
ERR_SERCHNL	Invalid serial channel.
ERR_SERREG	Invalid serial register.
ERR_NO_RX_HBUF	No remaining data in host buffer.
ERR_NULL	Null pointer.

ReadUart (continued)

EXAMPLE

```
//This example writes to the MIO_LCR register on ch1 for card 1.
      Short card = 1;
      Short wStatus = 0;
      Unsigned char Chan = 1;
      Unsigned char Data;
      wStatus += ReadUart(Card, Chan, MIO_LCR, &Data);
```

SEE ALSO

DisableUart() **EnableUart()**
ReadUartConfig() **WriteUartConfig()**
ReadUart() **WriteUart()**

ReadMailboxIrig

PROTOTYPE

```
#include “Receive.h”
short ReadMailboxIrig  (short Card,
                        short Receiver,
                        short LabelSdi,
                        unsigned long *Data,
                        long *StampHi ,
                        long *StampLo);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receiver.
LabelSdi	(input parameter) The mailbox slot of the specified label (bits 0...7) and SDI (bits 8..9).
Data	(output parameter) The 32-bit ARINC word read from the mailbox.
StampHi	(output parameter) The time stamp in milliseconds or IRIG time of the receiving time for this word. This parameter can be null if the time stamp is not needed. If this word has no time stamp then a –1 will be assigned to the Stamp variable.
StampLo	(output parameter) The time stamp in milliseconds of the receiving time for this word. This parameter can be null if the time stamp is not needed. If this word has no time stamp then a –1 will be assigned to the Stamp variable.

ReadMailboxIrig (continued)

DESCRIPTION

This function reads the latest ARINC word received of a particular label and SDI. This function also gets the ARINC word's time stamp if this feature is enabled.

Note: *If the original bit format is used, the word's 32nd bit is an error indicator, not the actual bit received. If the bit is 1, the word had even parity when it was received. If the bit is 0, the word had odd parity. If the alternate bit format is used, the parity error indicator is the 9th bit and the SDI is the 12th and 13th bits.*

Note: This note applies to Legacy cards only. *When the time stamp is enabled, the SDI portion of the SDI/Label parameter passed into the function ReadMailboxIrig() must be '0'. Any value other than '0' will result in an error being returned from the function ReadMailboxIrig(). The enabled time stamp makes the SDI a "Don't Care" which allows the function to return the last specified label.*

This note does not apply to cards such as: DD-40x00F/i/T/K, DD-40002M, DD-40001H, BU-67118

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid receiver.
ERR_LabelSdi	Invalid label and SDI.
ERR_NULL	Null pointer
ERR_MODE	Invalid operating mode
NUMBER	Returns a one if a new word is in the mailbox. Returns a zero if an old word is in the mailbox.

EXAMPLE

```
//This example will read the mailbox slot that has a label of 3 and an
//SDI of 0 for receiver 1 of card 1.
      short card = 1;
      short receiver = 1;
      short labelsdi = 3;
      short  readmailbox_return = 0;
      unsigned long data[1];
      long stampHi[1], stampLo[1];
      readmailbox_return = ReadMailbox (card,
                                        receiver,
                                        labelsdi,
                                        *data,
                                        stampHi,
                                        stampLo);
```

ReadMailboxIrig (continued)

SEE ALSO

GetMailboxStatus()
ClearMailbox()
EnableTimeStamp()
GetMailbox()
ConfigTimeStamp()

ReadRxHostBuffer

PROTOTYPE

```
#include “receive.h”
S16BIT ReadRxHostBuffer (S16BIT Card
                         S16BIT Receiver,
                         U16BIT numMsgsToRead,
                         U16BIT *numMsgsRead,
                         PRX_HBUF_MESSAGE messages)
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receiver channel number to read.
numMsgsToRead	(input parameter) Maximum number of messages to read.
numMsgsRead	(output parameter) Number of messages returned in “*messages*”..
Messages	(input/output parameter) Pointer to array of PRX_HBUF_MESSAGE to receive the message data.

DESCRIPTION

This function will read the requested number of message (for the specified channel) from the Receive FIFO Host Buffer. The Host Buffer will improve performance on receiving data and can be used instead of the ReadRxQueueIrig() functions.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_SUCCESS	The function returned successfully.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Channel is not configured as a receiver.
ERR_MODE	Receiver is not configured in FIFO mode.
ERR_RXQUEUESZ	Requested number of messages to Read is 0.
ERR_NULL	One or more pointer inputs are NULL.
ERR_RX_HBUF_INSTALL	Receive Host Buffer is notinstalled.
ERR_NORES	Hardware is not responding.

ReadRxHostBuffer (continued)

EXAMPLE

```
// This example shows how to read from the Receive FIFO Host Buffer.
      S16BIT wStatus = 0;
      S16BIT card = 1;
      S16BIT receiver = 1;
      U16BIT numMessagesRead = 0;
      U16BIT MAX_MSGS = 10;
      RX_HBUF_MESSAGE messages[MAX_MSGS];
      wStatus = ReadRxHostBuffer(card,
                                 receiver,
                                 MAX_MSGS,
                                 &numMessagesRead,
                                 messages);
```

SEE ALSO

InstallFifoRxHostBuffer()
UninstallFifoRxHostBuffer()
EnableRxHostBuffer()
DisableRxHostBuffer()
ReadRxHostBuffer()

ReadRxQueueIrigMore

PROTOTYPE

```
#include “Receive.h”
short ReadRxQueueIrigMore  ( short Card,
                             short Receiver,
                             short N,
                             unsigned long *Data,
                             long *StampHi,
                             long *StampLo);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receiver (1..4).
N	(input parameter) The maximum number of words to read.
Data	(output parameter) The array of the 32-bit ARINC words read from the queue.
StampHi	(output parameter) The array of time stamps in milliseconds of the words. This parameter can be NULL if the time stamps are not needed. Otherwise, -1 will be assigned to any word in the array that has no time stamp.
StampLo	(output parameter) The array of time stamps in milliseconds of the words. This parameter can be NULL if the time stamps are not needed. Otherwise, -1 will be assigned to any word in the array that has no time stamp.

ReadRxQueueIrigMore (continued)

DESCRIPTION

This function reads multiple 32-bit ARINC words from a receiver's queue. If the word time stamps are not needed or if the receiver's time stamp function is disabled, set Stamp=NULL.

Note: *If the original bit format is used, the word's 32nd bit is an error indicator, not the actual bit received. If the bit is 1, the word had even parity when it was received. If the bit is 0, the word had odd parity. If the alternate bit format is used, the parity error indicator is the 9th bit and the SDI is the 12th and 13th bits.*

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid receiver.
ERR_NULL	Null pointer.
ERR_MODE	Invalid operating mode.
ERR_TIMEOUT	Timeout from a message read operation from the hardware.
ERR_RXQUEUESZ	Invalid size of array.
ERR_BORES	Device Resources are unavailable
ERR_OVERFLOW	Device receiver FIFO buffer has overflowed with a potential loss of data
NUMBER	The actual number of ARINC words read from the queue.

EXAMPLE

```
//This example will read 5 32-bit words from the queue of receiver 1
//of card 1.
      short card = 1;
      short receiver = 1;
      short n = 5;
      unsigned long data[5];
      long stampHi[5], stampLo[5];
      short ReadRxQueueIrigMore_return = 0;
      ReadRxQueueIrigMore_return = ReadRxQueueIrigMore (card,
                                                        receiver,
                                                        n,
                                                        *data,
                                                        *stampHi,
                                                        *stampLo);
```

SEE ALSO

ReadRxQueueIrigOne()
ClearRxQueue()
GetRxQueueStatus()
ConfigTimeStamp()
EnableTimeStamp()

ReadRxQueueIrigOne

PROTOTYPE

```
#include “Receive.h”
short ReadRxQueueIrigOne (short Card,
                          short Receiver,
                          unsigned long *Data,
                          long *StampHi,
                          long *StampLo);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receiver.
Data	(output parameter) The 32-bit ARINC word read from the queue.
StampHi	(output parameter) The time stamp of this word in milliseconds. This parameter can be NULL if the time stamps are not needed. Otherwise, -1 will be assigned to the Stamp variable of any word that has no time stamp
.	
StampLo	(output parameter) The time stamp of this word in milliseconds. This parameter can be NULL if the time stamps are not needed. Otherwise, -1 will be assigned to the Stamp variable of any word that has no time stamp.

DESCRIPTION

This function reads one 32-bit ARINC word from a receiver’s queue. If the word’s time stamp is not needed or if the receiver’s time stamp function is disabled, set Stamp=NULL.

> ***Note:*** *If the original bit format is used, the word’s 32nd bit is an error indicator, not the actual bit received. If the bit is 1, the word had even parity when it was received. If the bit is 0, the word had odd parity. If the alternate bit format is used, the parity error indicator is the 9th bit and the SDI is the 12th and 13th bits.*

ReadRxQueueIrigOne (continued)

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid receiver.
ERR_NULL	Null pointer
ERR_MODE	Invalid operating mode
ERR_NORES	Device Resources are unavailable
NUMBER	A zero if no word is in the queue. A one if the word is in the queue.

EXAMPLE

```
//This example will read one 32-bit word from the queue of receiver 1
//of card 1.
      short card = 1;
      short receiver = 1;
      unsigned long data[1];
      long stampHi[1], stampLo[1];
      short ReadRxQueueIrigOne_return = 0;
      readrxqueueIrigone_return = ReadRxQueueIrigOne (card,
                                                      receiver,
                                                      *data,
                                                      stampHi,
                                                      stampLo);
```

SEE ALSO

ReadRxQueueIrigMore()
ClearRxQueue()
GetRxQueueStatus()
ConfigTimeStamp()
EnableTimeStamp()

ReadUartConfig

PROTOTYPE

```
#include “serial.h”
short ReadUartConfig (short Card,
                      unsigned char Reg,
                      unsigned char *Data);
```

HARDWARE

BU-67211Ux, BU-67107F/M/i/T

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Reg	(input parameter) UART configuration Register: MIO_INT0 MIO_INT1 MIO_INT2 MIO_INT3 MIO_TIMERCTL MIO_TIMER MIO_TIMERLSB MIO_TIMERMSB MIO_8XMODE MIO_REG1 MIO_RESET MIO_SLEEP MIO_DREV MIO_DVID MIO_REG2
Data	(output parameter) 8 Bit UART configuration Data.

DESCRIPTION

This function provides the user read access to the quad UART configuration registers. Refer to the XL16L784 datasheet for configuration details.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_SERREG	Invalid serial register.
ERR_SUCCESS	The function returned successfully.
ERR_NULL	Null pointer.
ERR_CRDINIT	Card is not initialized.
ERR_SERIAL	Invalid serial channel.

ReadUartConfig (continued)

EXAMPLE

```
//This example reads from the MIO_INT register for card 1.
      Short card = 1;
      Short wStatus = 0;
      Unsigned char Data;
      wStatus += ReadUartConfig(Card, MIO_INT, &Data);
```

SEE ALSO

DisableUart()
EnableUart()
ReadUartConfig ()
WriteUartConfig()
ReadUart()
WriteUart()

ResetGroup

PROTOTYPE

```
#include “Control.h”
short ResetGroup (short Card,
                  short Group);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Group	(input parameter) The group number.

DESCRIPTION

This function resets a channel group consisting of one transmitter and multiple receivers.

The number of transmitters per receiver are as follows:

BU-65590F0/F1	1 Transmitter for every 2 Receivers
BU-65590F2	1 Transmitter for every 4 Receivers
BU-65590M0/M1	1 Transmitter for every 2 Receivers
BU-65590M2	1 Transmitter for every 4 Receivers
BU-65590/91U0/U2/U3	1 Transmitter for every 2 Receivers
BU-65590/91C	1 Transmitter for every 4 Receivers
BU-67103U	1 Transmitter for every 2 Receivers

The grouping of channels will depend on how many channels are on your card. Below is how each group is configured:

Group1 = Transmitter1, Receivers 1, 2, 9 and 10.
Group2 = Transmitter2, Receivers 3, 4, 11 and 12.
Group3 = Transmitter3, Receivers 5, 6, 13 and 14.
Group4 = Transmitter4, Receivers 7, 8, 15 and 16.

The group’s status after this function is as follows:

No loopback from the transmitter to the receivers in this group.
Odd transmitter parity setting.
Low speed for the transmitter.
The transmitter disabled.
Transmitter queue cleared.
No repeated transmission scheduled.
Low speed for both receivers.

ResetGroup (continued)

Both receivers disabled.
No existing receiver filters.
Receiver filters disabled (not used).
No new data in receiver's mailbox.
Receiver queues cleared.
No time stamps for received words.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_CHNLGROUP	Invalid channel group.
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example will reset group 1.
      short card = 1;
      short group = 1;
      short resetgroup_return = 0;
      resetgroup_return = ResetGroup (card,
                                      group);
```

SEE ALSO

GetChannelCount()

ResetRxChannel

PROTOTYPE

#include “Control.h”
short **ResetRxChannel** (short Card,
short Receiver);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The Receiver.

DESCRIPTION

This function resets a receiver channel.

The Receiver’s status after this function is as follows:

No loopback from the transmitter to the receivers in this group.
Low speed for the receiver.
The receiver is disabled.
No existing receiver filters.
Receiver filters disabled (not used).
No new data in receiver’s mailbox.
Receiver queues cleared.
No time stamps for received words.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example will reset Receiver 1.
      short card = 1;
      short Receiver = 1;
      short resetgroup_return = 0;
      resetgroup_return = ResetRxChannel(card, Receiver);
```

SEE ALSO

ResetTxChannel()

ResetTimeStamp

PROTOTYPE

#include “Receive.h”
short **ResetTimeStamp** (short Card);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.

DESCRIPTION

This function resets the timer used for all of the receiver’s time stamps to zero. The time stamps of the words already received will not change.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example resets the time stamp on receiver 1 of card 1.
      short card = 1;
      short resettimestamp_return = 0;
      resettimestamp_return = ResetTimeStamp (card);
```

SEE ALSO

EnableTimeStamp()
ConfigTimeStamp()
GetTimeStampStatus()

ResetTxChannel

PROTOTYPE

```
#include “Control.h”
short ResetTxChannel  (short Card,
                       short Transmitter);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Transmitter	(input parameter) The Transmitter.

DESCRIPTION

This function resets a transmitter.

The Transmitter’s status after this function is as follows:

No loopback from the transmitter to the receivers in this group.
Odd transmitter parity setting.
Low speed for the transmitter.
The transmitter disabled.
Transmitter queue cleared.
No repeated transmission scheduled.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example will reset transmitter 1.
      short card = 1;
      short Transmitter = 1;
      short resetgroup_return = 0;
      resetgroup_return = ResetTxChannel(card, Transmitter);
```

SEE ALSO

ResetRxChannel()

SetAvionAll

PROTOTYPE

#include “Control.h”
short **SetAvionAll** (S16BIT Card,
U16BIT OutEnables,
U16BIT Levels);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
OutEnable	(input parameter) Bitwise representation of output enable of avionic (bit 15 = MSB) 0 = Avionic line disabled 1 = Avionic line enabled
Levels	Bitwise representation of levels of avionic (bit 15 = MSB) 0 = Avionic line is Low 1 = Avionic line is High

DESCRIPTION

This function enables/disables all Avionic IO bits and configures their output level to either Low (0) or High (1).

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example configures the Avionic IO bits.

     S16BIT Card = 1;
     U16BIT OutEnables = 0xFFFF
     U16BIT Levels =0x10A1;
     Return_value = GetAvionAll(Card, &OutEnables, &Levels);
```

SEE ALSO

GetAvionOut()
SetAvionOut()
SetAvionOutEnable()
GetAvionOutEnable()
GetAvionIn()
GetAvionAll()

SetAvionOut

PROTOTYPE

#include “Control.h”
short **SetAvionOut** (S16BIT Card,
S16BIT Avionic,
S16BIT Level);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Avionic	(input parameter) Avionic Line.
Level	(input parameter) Level of Avionic Line 0 = Low 1 = High

DESCRIPTION

This function configures an avionic output level to either High (1) or Low (0).

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_AVIONIC	Not a valid Avionic Line
ERR_DLEVEL	Invalid discrete output level
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example will set the level of the Avionic line.

     S16BIT Card = 1;
     S16BIT Avionic = 1
     S16BIT Level = 1;
     Return_value = SetAvionOut(Card, Avionic, Level);
```

SEE ALSO

GetAvionOut()
GetAvionOutEnable()
GetAvionAll()
SetAvionOutEnable()
GetAvionIn()
SetAvionAll()

SetAvionOutEnable

PROTOTYPE

#include “Control.h”
short **SetAvionOutEnable** (S16BIT Card,
S16BIT Avionic,
S16BIT State);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Avionic	(input parameter) Avionic Line.
State	(input parameter) State of Avionic Line DD429_DISABLE DD429_ENABLE

DESCRIPTION

This function enables/disables the Avionic output line.

RETURN VALUE

ERR_NOCRD	No card is present
ERR_CRDINIT	Card is not initialized
ERR_AVIONIC	Not a valid Avionic Line
ERR_DLEVEL	Invalid discrete output level
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example enable the Avionic line.

     S16BIT Card = 1;
     S16BIT Avionic = 1
     S16BIT State = DD429_ENABLE;
     Return_value = SetAvionOutEnable(Card, Avionic, State);
```

SEE ALSO

GetAvionOut()
GetAvionOutEnable()
GetAvionAll()
SetAvionOut()
GetAvionIn()
SetAvionAll()

SetBitFormat

PROTOTYPE

#include “Control.h”
short **SetBitFormat** (short Card,
short BitFormat);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
BitFormat	(input parameter) The bit format where DD429_BITFORMAT_ORIG (0) represents the original format and DD429_BITFORMAT_ALT (1) represents the alternate format. In the original format bits 0..7 make up the label, bits 8..9 make up the SDI ,bits 10..28 make up the data, bits 29..30 make up the SSM, and bit 31 represents the parity. The SSM and parity bits can be included as data. In the alternate format bits 0..7 make up the label, bit 8 represents the parity, bits 9..10 make up the SSM, bits 11..12 make up the SDI, and bits 13..31 make up the data.

DESCRIPTION

This function sets the user’s transmit and receive bit format of the ARINC words. It is applicable to all transmitters and receivers on a card. It is recommended to call this function only once after the card initialization, before any transmission or receiving starts.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_BITFORMAT	Invalid bit format.
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example will set card 1 to the original bit format.
     short card = 1;
     short bitformat = DD429_BITFORMAT_ORIG;
     short setbitformat_return = 0;
     setbitformat_return = SetBitFormat (card,
                                         bitformat);
```

SEE ALSO

GetBitFormat() **ResetGroup()**

SetChannelLoopBack

PROTOTYPE

#include “Control.h”
S16BIT **SetChannelLoopBack** (S16BIT Card,
S16BIT Receiver,
S16BIT LoopBack);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receive channel number.
Loopback	(input parameter) DD429_DISABLE (0), DD429_ENABLE (1), DD429_ENABLE_ALT (2).

DESCRIPTION

This function is used to enable or disable loopback mode for a given receiver. The default pairing of receivers to transmitters will be RX1 and RX2 connected to Tx1, and Rx3 and Rx4 connected to Tx2 if SetLoopBackMapping() is not called to configure which transmitters the receivers pairing.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid receiver defined.
ERR_LOOPBACK	Invalid loopback.
ERR_SUCCESS	The function returned successfully.

EXAMPLE

```
// This example set the Receiver to loopback mode.

     S16BIT Card = 1;
     S16BIT Receiver = 1;
     S16BIT LoopBack = DD429_ENABLE;
     Return_value = SetChannelLoopBack(Card, Receiver, LoopBack);
```

SEE ALSO

GetChannelLoopBack() **GetLoopBackMapping()**
SetLoopBackMapping()

SetDiscAll

PROTOTYPE

#include “Control.h”
short **SetDiscAll** (S16BIT Card,
U16BIT Directions,
U16BIT Levels);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Directions	(input parameter) Bit wise representation of direction of discrete (bit 15 = MSB) 0 = Low 1 = Discrete output
Levels	(input parameter) Bit wise representation of level of discrete (bit 15 = MSB) 0 = Low 1 = High

DESCRIPTION

This function configures all discrete IO bits for either input (0) or output (1) and their level to Low (0) or High (1).

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example will configure all discrete bits and levels.

     S16BIT Card = 1;
     U16BIT Directions = 0x81f0
     U16BIT Levels =0x5c1e;
     Return_value = SetDiscAll(Card, Directions, Levels);
```

SEE ALSO

GetDiscOut()
GetDiscDir()
SetDiscDir()
ResetGroup()
GetDiscIn()
SetDiscAll()

SetDiscDir

PROTOTYPE

#include “Control.h”
short ***SetDiscDir*** *(short* Card*,*
short Discrete*,*
short Direction*);*

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Discrete	(input parameter) The Discrete line to set.
Direction	(input parameter) INPUT (0) will set the line to receive, while an OUTPUT (1) will configure the line to transmit.

DESCRIPTION

This function sets a discrete output.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_DISCRETE	Invalid discrete output number.
ERR_DLEVEL	Invalid discrete output level.
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example sets discrete 1 to receive signals for card 1.
      short card = 1;
      short discrete = 1;
      short direction = INPUT;
      short setdiscdir_return = 0;
      setdisdir_return = SetDiscDir (card, discrete, direction);
```

SEE ALSO

GetDiscOut()
GetDiscDir()
SetDiscDir()
ResetGroup()
GetDiscIn()

SetDiscOut

PROTOTYPE

#include “Control.h”
short ***SetDiscOut*** *(short* Card*,*
short Discrete*,*
short Level*);*

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Discrete	(input parameter) The Discrete line to set.
Level	(input parameter) Sets the level of the output. A zero will set it to low and a one will set it high.

DESCRIPTION

This function sets a discrete output.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_DISCRETE	Invalid discrete output number.
ERR_DLEVEL	Invalid discrete output level.
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example sets the discrete output level to high on discrete 1 of
//card 1.
      short card = 1;
      short discrete = 1;
      short level = 1;
      short setdiscout_return = 0;
      setdiscout_return = SetDiscOut (card, discrete, level);
```

SEE ALSO

GetDiscOut()
SetDiscDir()
GetDiscIn()
ResetGroup()
GetDiscDir()

SetIntCondition

PROTOTYPE

```
#include “CardInit.h”
short SetIntCondition (short Card,
                       unsigned char Type,
                       unsigned char Channel,
                       Long Condition);
```

HARDWARE

Any

PARAMETERS

Card (input parameter)
The card number given to the card in the Card Manager in the Control Panel.

Type (input parameter)
The interface type. Valid values:
- CHAN_TYPE_429
- CHAN_TYPE_UART

Channel (input parameter)
The channel number on the card.
- 1-16 for ARINC 429 Multi-IO SDK depending on Channel count of device.
- 1-4 for UART Channel (RS-232/422/485)
- N/A for Time tag rollover

Condition (input parameter)
Interface specific interrupt enable bit fields:

ARINC429: 32 Bit value

BIT	Definition
31-13	N/A
12	Protocol Word Received
11	Fail Warning Received
10	Function Test Received
9	Solo word Received
8	Start of Transmission Received
7	No data
6	End of Transmission Received
5	Normal operation mode Received
4	Data Match
3	Word Type
2	FIFO ¼ full, ½ full, or full
1	Parity Error
0	FIFO overflow

RS232/485 8 Bit Value

SetIntCondition (continued)

Time Tag

BIT	Definition
2	Disable Rollover Interrupt
3	Enable Rollover Interrupt

DESCRIPTION

This function will configure the interrupt conditions for various interfaces (ARINC429, RS232, RS485 or a Time Tag rollover).

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_SUCCESS	The function returned successfully.
ERR_CRDINIT	Card is not initialized.
ERR_IRQ	IRQ(s) in use.
ERR_CHAN_TYPE	Invalid channel type.
ERR_INT_COND	Invalid interrupt condition.
ERR_INT_HANDLER	Invalid interrupt condition.
ERR_RX	Invalid receiver defined.
ERR_SERCHNL	Invalid serial channel.

EXAMPLE

```
//This example configures an interrupt for ARINC429 Multi-IO SDK
//protocol on channel 1 when the receiver is full.

     Short card = 1;
     Short wStatus = 0;
     Unsigned char bChan = 1
     wStatus += SetIntCondition(Card, CHAN_TYPE_429, bChan,
                                INT_COND_429_RXFULL);
```

SEE ALSO

InstallHandler() **UninstallHandler()**
GetIntStatus()

SetLoopBack

PROTOTYPE

#include “Control.h”
short ***SetLoopBack*** *(short* Card*,*
short Group*,*
short LoopBack*);*

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Group	(input parameter) Transmitter / Receiver channel group.
Loopback	(input parameter) DD429_DISABLE (0), DD429_ENABLE (1) or DD429_ENABLE_ALT (2).

DESCRIPTION

This function enables or disables the loopback from a transmitter to the mutliple receivers in the same group. This function is not recommended for use. The function SetChannelLoopBack is recommended for use over SetLoopBack.

Note: *If the loopback is enabled, the receivers will not be able to receive any data from their external pins. The transmitter and receiver speeds must match in order for the loopback to work properly.*

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_CHNLGROUP	Invalid channel group.
ERR_LOOPBACK	Invalid loopback.
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example will enable the loopback for group 1 on card 1.
      short card = 1;
      short group = 1;
      short loopback = 1;
      short setloopback_return = 0;
      setloopback_return = SetLoopBack (card,
                                        group,
                                        loopback);
```

SetLoopBack (continued)

SEE ALSO

GetLoopBack() **ResetGroup()**

SetLoopBackMapping

PROTOTYPE

#include “Control.h”

S16BIT S**etLoopBackMapping** (S16BIT Card,
S16BIT Receiver,
S16BIT TxMapping);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receive channel number.
TxMapping	(input parameter) The transmitter channel number.

DESCRIPTION

This function will configure which Receivers are connected to a Transmitter. This function is only available for use with DDC’s ***AceXtreme***® Devices.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid receiver defined.
ERR_TX	Invalid transmitter defined.
ERR_NORES	Card not responding.
ERR_SUCCESS	The function returned successfully.

EXAMPLE

```
//This example configures transmitter 1 to receiver 1 in loopback
mode.

     S16BIT Card = 1;
     S16BIT Receiver = 1;
     S16BIT TxMapping = 1;
     Return_value = SetLoopBackMapping(Card, Receiver, TxMapping);
```

SetLoopBackMapping (continued)

SEE ALSO

GetChannelLoopBack()
GetLoopBackMapping()
SetChannelLoopBack()

SetRxChannelMode

PROTOTYPE

#include “Receive.h”
short **SetRxChannelMode** (short Card,
short Receiver,
short Mode);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The receiver.
Mode	(input parameter) Mode of Operation either FIFO_MODE (1) or MAILBOX_MODE (0)

DESCRIPTION

This function returns the speed of the receivers on a channel basis.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid receiver.
ERR_SUCCESS	The function returned successfully.
ERR_MODE	Invalide operating mode.

EXAMPLE

```
//This example configures which mode receiver 1 is will use.
      short card = 1;
      short Receiver = 1;
      short mode = MAILBOX_MODE
      short getrxmode_return = 0;
      getrxmode_return = SetRxChannelMode (card, Receiver, mode);
```

SEE ALSO

GetRxChannelMode()

SetRxChannelParity

PROTOTYPE

#include “Receive.h”
short **SetRxChannelParity** (short Card,
short Receiver,
short Parity);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The Receiver
Parity	(input parameter) DD429_NO_PARITY (0), DD429_ODD_PARITY (1), or DD429_EVEN_PARITY (2)

DESCRIPTION

This function sets the receiver’s parity setting.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid Receiver defined.
ERR_PARITY	Invalid parity.
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example sets the parity setting for receiver 2 of card 1.
      short card = 1;
      short receiver = 2;
      short setrxparity_return = 0;
      short parity = DD429_EVEN_PARITY;
      setrxparity_return = SetRxChannelParity (card, receiver, parity);
```

SEE ALSO

GetRxChannelParity()

SetRxChannelSpeed

PROTOTYPE

#include “Receive.h”
short **SetRxChannelSpeed** (short Card,
short Receiver,
short Speed);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager inthe Control Panel.
Receiver	(input parameter) The Receiver.
Speed	(input parameter) DD429_HIGH_SPEED (1) represents high speed and DD429_LOW_SPEED (0) represents low speed. DD429_ARINC575 (2) sets the receiver into ARINC 575 speed.

DESCRIPTION

This function sets the speed of the two receivers in a group.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid receiver.
ERR_SPEED	Invalid speed.
ERR_SUCCESS	The function returned successfully
ERR_ARINC575	ARINC 575 bit rate

EXAMPLE

```
//This example will set receiver 1  on card 1 to high speed.
      short card = 1;
      short Receiver = 1;
      short speed = 1;
      short setrxspeed_return = 0;
      setrxspeed_return = SetRxChannelSpeed (card, Receiver, speed);
```

SEE ALSO

GetRxChannelSpeed()

SetRxParity

PROTOTYPE

```
#include “Receive.h”
short SetRxParity (short Card,
                   short Receiver,
                   short Parity);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Receiver	(input parameter) The Receiver
Parity	(input parameter) DD429_NO_PARITY (0), DD429_ODD_PARITY (1), or DD429_EVEN_PARITY (2)

DESCRIPTION

This function sets the receiver’s parity setting.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RX	Invalid Receiver defined.
ERR_PARITY	Invalid parity.
ERR_SUCCESS	The function returned successfully

EXAMPLE

```
//This example sets the parity setting for receiver 2 of card 1.
      short card = 1;
      short receiver = 2;
      short setrxparity_return = 0;
      short parity = DD429_EVEN_PARITY;
      setrxparity_return = SetRxParity (card, receiver, parity);
```

SEE ALSO

GetRxParity() **EnableRx()**
GetRxStatus()

SetRxMode

PROTOTYPE

```
#include “Receive.h”
short SetRxMode (short Card,
                 short Group,
                 short Mode);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Group	(input parameter) The receiver group
Mode	(input parameter) Mode of Operation either FIFO_MODE (1) or MAILBOX_MODE (0).

DESCRIPTION

This function returns the speed of the receivers in a group. Filtering will be limited to Label only when using IRIG time tagging.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RXGROUP	Invalid receiver.
ERR_SUCCESS	The function returned successfully
ERR_MODE	Invalid operating mode

EXAMPLE

```
//This example configures which mode the group 1 is will use.
      short card = 1;
      short group = 1;
      short mode = MAILBOX_MODE
      short getrxmode_return = 0;
      getrxmode_return = SetRxMode (card, group, mode);
```

SEE ALSO

GetRxMode() **EnableRx()**
GetRxStatus()

SetRxSpeed

PROTOTYPE

```
#include “Receive.h”
short SetRxSpeed (short Card,
                  short Group,
                  short Speed);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Group	(input parameter) Corresponds to all receivers in a channel group.
Speed	(input parameter) DD429_HIGH_SPEED (1) represents high speed and DD429_LOW_SPEED (0) represents low speed. DD429_ARINC575 (2) sets the receiver into ARINC 575 speed.

DESCRIPTION

This function sets the speed of the two receivers in a group.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_RXGROUP	Invalid receiver group.
ERR_SPEED	Invalid speed.
ERR_SUCCESS	The function returned successfully
ERR_ARINC575	ARINC 575 bit rate

EXAMPLE

```
//This example will set group 1 (receivers 1 and 2) on card 1 to high
//speed.
      short card = 1;
      short group = 1;
      short speed = 1;
      short setrxspeed_return = 0;
      setrxspeed_return = SetRxSpeed (card,
                                      group,
                                      speed);
```

SetRxSpeed (continued)

SEE ALSO

GetRxSpeed()
GetRxStatus()
EnableRx()
ResetGroup()

SetTxParity

PROTOTYPE

```
#include “Transmit.h”
short SetTxParity (short Card,
                   short Transmitter,
                   short Parity);
```

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Transmitter	(input parameter) The transmitter.
Parity	(input parameter) DD429_NO_PARITY (0), DD429_ODD_PARITY (1), DD429_EVEN_PARITY (2)

DESCRIPTION

This function controls the transmitter parity setting. Your 32nd bit (or the 9th bit in alternate bit format) will be a data bit if no parity setting, or a parity bit if odd parity is used.

Note: *The parity bit received by each Rx channel is checked and a parity error is reported for the 32nd bit, instead of the actual parity bit received. If the received data word contains a parity bit equal to zero, this means that the received data contained the proper parity (odd). If a word is received that is even parity, the parity bit will be set to one to indicate an error.*

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_TX	Invalid transmitter.
ERR_PARITY	Invalid parity.
ERR_SUCCESS	The function returned successfully

SetTxParity (continued)

EXAMPLE

```
//This example sets up transmitter 1 on card 1 to have no parity.
      short card = 1;
      short transmitter = 1;
      short parity = DD429_NO_PARITY;
      short settxparity_return = 0;
      settxparity_return = SetTxParity (card,
                                        transmitter,
                                        parity);
```

SEE ALSO

GetTxParity()
GetTxStatus()
EnableTx()
SetRxParity()

SetTxSpeed

PROTOTYPE

#include “Transmit.h”
short **SetTxSpeed** (short Card,
short Transmitter,
short Speed);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Transmitter	(input parameter) The transmitter.
Speed	(input parameter) DD429_HIGH_SPEED (1) represents high speed and DD429_LOW_SPEED (0) represents low speed. DD429_ARINC575 (2) sets the transmitter into ARINC 575 speed.

DESCRIPTION

This function sets the transmitter speed. The speed is set as either high or low.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CRDINIT	Card is not initialized.
ERR_TX	Invalid transmitter.
ERR_SPEED	Invalid speed.
ERR_SUCCESS	The function returned successfully
ERR_ARINC575	ARINC 575 bit rate

EXAMPLE

```
//This example sets transmitter 1 on card 1 to high speed.
      short card = 1;
      short transmitter = 1;
      short speed = DD429_HIGH_SPEED;
      short settxspeed_return = 0;
      settxspeed_return = SetTxSpeed (card, transmitter, speed);
```

SEE ALSO

GetTxSpeed() **EnableTx()**
GetTxStatus()

UninstallFifoRxHostBuffer

PROTOTYPE

#include “receive.h”
S16BIT **UninstallFifoRxHostBuffer** (short Card);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.

DESCRIPTION

This function will unstall the a Host Buffer for ARINC 429 recevied FIFO traffic. This Host Buffer will improve performance on receiving data and can be used instead of the ReadRxQueueIrig() functions.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_SUCCESS	The function returned successfully.
ERR_CRDINIT	Card is not initialized.
ERR_CRDINIT	Card is not initialized.
ERR_RX_HBUF_INSTALL	Receive Host Buffer is notinstalled.
ERR_NORES	Hardware is not responding.

EXAMPLE

```
// This example uninstalls the receive Host Buffer for card 1.
     Short card = 1;
     Short wStatus = 0;
     wStatus += UninstallFifoRxHostBuffer(Card);
```

SEE ALSO

InstallFifoRxHostBuffer()
UninstallFifoRxHostBuffer()
EnableRxHostBuffer()
DisableRxHostBuffer()
ReadRxHostBuffer()

UninstallHandler

PROTOTYPE

#include “CardInit.h”
short **UninstallHandler** (short Card,
unsigned char Type);

HARDWARE

Any

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Type	The interface type. Valid values: CHAN_TYPE_429 CHAN_TYPE_UART CHAN_TYPE_717_PROG CHAN_TYPE_CAN

DESCRIPTION

This function will uninstall a user defined interrupt handler for a specified type of interface.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_CHAN_TYPE	Invalid channel type.
ERR_SUCCESS	The function returned successfully.
ERR_CRDINIT	Card is not initialized.

EXAMPLE

```
//This example installs an interrupt handler for card 1.
      Short card = 1;
      Short wStatus = 0;
      wStatus += UninstallHandler(Card, CHAN_TYPE_429);
```

SEE ALSO

InstallHandler()
GetIntStatus()
SetIntCondition()
acexArinc717Interrupts()

WriteUart

PROTOTYPE

```
#include “serial.h”
short WriteUart   (short Card,
                   unsigned char Chan,
                   unsigned char Reg,
                   unsigned char Data);
```

HARDWARE

BU-67211Ux, BU-67107F/M/i/T

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Chan	(input parameter) Serial I/O Channel. Valid Values: 1 -4 (depending on hardware)
Reg	(input parameter) UART configuration Register MIO_RHR MIO_THR MIO_DLL MIO_DLM MIO_IER MIO_ISR MIO_FCR MIO_LCR MIO_MCR MIO_LSR MIO_MSR MIO_485TADELAY MIO_SPR MIO_FCTR MIO_EFR MIO_TXCNT MIO_TXTRG MIO_RXCNT MIO_RXTRG MIO_XOFF1 MIO_XCHAR MIO_XOFF2 MIO_XON1 MIO_XON2
Data	(input parameter) 8-Bit UART Data

WriteUart (continued)

DESCRIPTION

This function provides the user write access to the two UART configuration / control /data registers. Refer to the Exar XR16L784 datasheet for configuration and usage details.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_SUCCESS	The function returned successfully.
ERR_CRDINIT	Card is not initialized.
ERR_SERREG	Invalid serial register.
ERR_SERCHNL	Invalid serial channel.

EXAMPLE

```
//This example writes to the MIO_LCR register on ch1 for card 1.
      Short card = 1;
      Short wStatus = 0;
      Unsigned char Chan = 1;
      wStatus += WriteUart(Card, Chan, MIO_LCR, 0x80);
```

SEE ALSO

DisableUart() **EnableUart()**
ReadUartConfig() **WriteUartConfig()**
ReadUart() **WriteUart()**

WriteUartConfig

PROTOTYPE

```
#include “serial.h”
short WriteUartConfig(short Card,
                      unsigned char Reg,
                      unsigned char Data);
```

HARDWARE

BU-67211Ux, BU-67107F/M/i/T

PARAMETERS

Card	(input parameter) The card number given to the card in the Card Manager in the Control Panel.
Reg	(input parameter) UART configuration Register: MIO_INT0 MIO_INT1 MIO_INT2 MIO_INT3 MIO_TIMERCTL MIO_TIMER MIO_TIMERLSB MIO_TIMERMSB MIO_8XMODE MIO_REG1 MIO_RESET MIO_SLEEP MIO_DREV MIO_DVID MIO_REG2
Data	8 Bit UART configuration Data.

DESCRIPTION

This function provides the user write access to the quad UART configuration registers. Refer to the XL16L784 datasheet for configuration details.

RETURN VALUE

ERR_NOCRD	No card is present.
ERR_SUCCESS	The function returned successfully.
ERR_SERREG	Invalid serial register.
ERR_CRDINIT	Card is not initialized.
ERR_SERIAL	Serial error.

WriteUartConfig (continued)

EXAMPLE

```
//This example writes a 1 to the MIO_INT register for card 1.
      Short card = 1;
      Short wStatus = 0;
      wStatus += WriteUartConfig(Card, MIO_INT, 0x00000001);
```

SEE ALSO

DisableUart()
EnableUart()
ReadUartConfig()
WriteUartConfig()
ReadUart()
WriteUart()

6 STRUCTURES

CHANCOUNT_p

Channel counts for a card.

SYNOPSIS

```
Typedef struct_CHANCOUNT
{
      U8BIT bTx;
      U8BIT bRx;
      U8BIT bGroup;
      U8BIT bDiscrete;
      U8BIT bAvionic;
      U8BIT bBoardModel;
      U8BIT b1553;
      U8BIT a429Prog;
      U8BIT bUart;
      U8BIT bRs232;
      U8BIT bRs485;
      U8BIT CanBus;
      U8BIT a717Rx;
      U8BIT a717Tx;
      U8BIT a717Prog;
} CHANCOUNT_t, *CHANCOUNT_p;
```

MEMBERS

U8BIT bTx;	Number of Transmitters
U8BIT bRx;	Number of Receivers
U8BIT bGroup;	Number of Groups
U8BIT bDiscrete;	Number of Discrete Channels
U8BIT bAvionic;	Number of Avionic Channels
U8BIT bBoardModel;	Device specific model
U8BIT b1553;	Number of MIL-STD-1553 channels
U8BIT a429Prog	Number of programmable 429 channels
U8BIT bUart;	Number of Serial-Based UART Channels
U8BIT bRs232;	Number of RS232 Channels
U8BIT bRs485;	Number of RS485 Channels
U8BIT CanBus;	Number of CanBUS Channels
U8BIT a717Rx;	Number of ARINC 717 Receive Channels.
U8BIT a717Tx;	Number of ARINC 717 Transmit Channels
U8BIT a717Prog;	Number of ARINC 717 Programmable Channels

DESCRIPTION

This structure defines the information used to maintain channel counts for a card.

DD429_TESTER_OPTIONS_TYPE

Struct to configure error injection on a per message basis.

SYNOPSIS

```
typedef struct _DD429_TESTER_OPTIONS_TYPE
{
    U16BIT s16InterWordBitGapError;
    U8BIT u8WordSizeError;
    U8BIT u8ParityError;
    U8BIT u8Bit33;

} DD429_TESTER_OPTIONS_TYPE;
```

MEMBERS

U16BIT s16InterWordBitGapError	Input parameter to set the Inter-Word Bit Gap error.
U8BIT u8WordSizeError	Input parameter to set the Word Size error.
U8BIT u8ParityError	Input parameter to set the Parit error.
U8BIT u8Bit33	Input parameter to set the Extra Bit error.

DESCRIPTION

This structure is only used by DDC's new DD-40x00x ARINC Cards. It is used to add error injection on a per message basis.

DD429_TX_MINOR_FRAME_PAYLOAD_TYPE

Struct to add data to a minor frame.

SYNOPSIS

```
typedef struct _DD429_TX_MINOR_FRAME_PAYLOAD_TYPE
{
    U32BIT u32Data;
    DD429_TESTER_OPTIONS_TYPE sTesterOptions;

} DD429_TX_MINOR_FRAME_PAYLOAD_TYPE;
```

MEMBERS

U32BIT u32Data — Input parameter to add the 32-bit ARINC word.

DD429_TESTER_OPTIONS_TYPE sTesterOptions — Input parameter to set the Error Injection.

DESCRIPTION

This structure is only used by DDC's new DD-40x00x ARINC Cards. It is used to add messages to a minor frame.

DD429TX_FRAME_INFO_TYPE

Struct to determine the current state of the output FIFO.

SYNOPSIS

```
typedef struc _DD429TX_FRAME_INFO_TYPE
{
    U8BIT u8PercentageFulll;
    U8BIT u8Reserved1;
    U8BIT u8Reserved1;
    U8BIT u8Reserved1;

} DD429TX_FRAME_INFO_TYPE;
```

MEMBERS

U8BIT u8PercentageFulll	Output parameter to store FIFO usage information
U8BIT u8Reserved1	RESERVED
U8BIT u8Reserved1	RESERVED
U8BIT u8Reserved1	RESERVED

DESCRIPTION

This structure is only used by DDC's new DD-40x00x ARINC Cards. It is used to determine how much of the output FIFO has been used up.

PARINC_717_PROGRMMABLE_CONFIG

Contains information for programming an ARINC 717 channel.

SYNOPSIS

```
typedef struct _ARINC_717_PROGRMMABLE_CONFIG
{
    ACEX_CONFIG_ID  sConfigID;
    U32BIT          u32ConfigOption;
    U8BIT           u8Channel;
    U8BIT           u8SlopeControl;
    U8BIT           u8WrapAroundMode;
    U8BIT           bStopTx;
    U8BIT           bRxAutoDetect;
    U8BIT           bReset;
    U8BIT           u8BufferMode;
    U8BIT           u8ProtocolType;
    U8BIT           u8Type;
    U8BIT           u8Speed;
    ARINC_717_STATE eState;
    U16BIT          u16FrameCount;
    U8BIT           bGetInterrupt;
    U8BIT           bEnableInterrupt;
    U32BIT          u32Interrupts;
} ARINC_717_PROGRMMABLE_CONFIG, *PARINC_717_PROGRMMABLE_CONFIG;
```

MEMBERS

ACEX_CONFIG_ID sConfigID	Stores ARINC Type and Channel Number. Read only.
U32BIT u32ConfigOption	Enables which configuration parameters should be applied.
U8BIT u8Channel	Channel Number.
U8BIT u8SlopeControl	ARINC 717 Slope Control Setting.
U8BIT u8WrapAroundMode	Reserved.
U8BIT bStopTx	Reserved.
U8BIT bRxAutoDetect	Reserved.
U8BIT bReset	Reset device? (True/False).
U8BIT u8BufferMode	Select Buffer Mode (Only Double Supported).
U8BIT u8ProtocolType	Select Harvard Bi-Phase or BPRZ Mode.
U8BIT u8Type	Select Transmit or Receive operation.
U8BIT u8Speed	Current Speed of this channel Reserved.
ARINC_717_STATE eState	Current State of this channel.
U16BIT u16FrameCount	Select number of Frames to Transmit (Tx Mode only)
U8BIT bGetInterrupt	Reserved.
U8BIT bEnableInterrupt	Reserved.
U32BIT u32Interrupts	Reserved.

DESCRIPTION

This structure contains configuration information and is used in conjunction with **acexArinc717ProgConfig**().

PCAN_BUS_CONFIG

Contains information for programming a CanBus channel.

SYNOPSIS

```
typedef struct _CAN_BUS_CONFIG
{
    ACEX_CONFIG_ID    sConfigID;
    U32BIT            u32ConfigOption;
    U8BIT             u8Channel;
    U8BIT             u8Speed;
    U8BIT             bInterrupt;
    U16BIT            u16TimerValue;
    U16BIT            u16MessageCountInt;
    U8BIT             u8EnableMonitor;
    U8BIT             u8EnableLoopback;
    CAN_BUS_RUN_STATE eState;
    U32BIT            u32FilterValues[CAN_BUS_RX_FILTER_MAX];
} CAN_BUS_CONFIG, *PCAN_BUS_CONFIG;
```

MEMBERS

ACEX_CONFIG_ID sConfigID;	Stores ARINC Type and Channel Number. Read only.
U32BIT u32ConfigOption;	Enables which configuration parameters should be applied.
U8BIT u8Channel;	Channel Number.
U8BIT u8Speed;	Select CanBus Speed. (Bit Rate)
U8BIT bInterrupt;	Enable Interrupts (True/False)
U16BIT u16TimerValue;	Interrupt Timer Rate (in ms)
U16BIT u16MessageCountInt;	Interrupt Message Count Threshold.
U8BIT u8EnableMonitor;	Enable Passive CanBus Monitor (True/False)
U8BIT u8EnableLoopback;	Enable Internal Loopback (True/False)
CAN_BUS_RUN_STATE eState;	Check the State of the channel.
U32BIT u32FilterValues;	Set/Change the Filter Settings.

DESCRIPTION

This structure contains configuration information and is used in conjunction with **acexCanBusConfig()**.

PDDC_IRIG_TX_TYPE

Struct to configure IRIG output signal.

SYNOPSIS

```
typedef struct _DDC_IRIG_TX_TYPE
{
    U16BIT u16IRIGBTxSupported;
    U16BIT u16Enable;
    U16BIT u16Seconds;
    U16BIT u16Minutes;
    U16BIT u16Hours;
    U16BIT u16Days;
    U16BIT u16Year;
    U32BIT u32Control;

} DDC_IRIG_TX_TYPE, *PDDC_IRIG_TX_TYPE;
```

MEMBERS

U16BIT u16IRIGBTxSupported	Stores information on IRIG support. Read only.
U16BIT u16Enable	Input parameter to enable IRIG output.
U16BIT u16Seconds	Input parameter to set the seconds field.
U16BIT u16Minutes	Input parameter to set the Minutes field.
U16BIT u16Hours	Input parameter to set the Hours field.
U16BIT u16Days	Input parameter to set the Days field.
U16BIT u16Year	Input parameter to set the year field.
U32BIT u32Control	Reserved.

DESCRIPTION

This structure is only used by DDC's new DD-40x00x ARINC Cards. It is used to set the IRIG transmit registers for outputting an IRIG signal

PHWVERSIONINFO

Contains the hardware version information.

SYNOPSIS

```
Typedef struct_HWVERSIONINFO
{
      U32BIT dwFwVersion;
      U32BIT dwHdlVersion;
      U32BIT dwDriverVersion;
      U32BIT dwSerialNumber;
      FAMILY dwFamilyNumber;
      U32BIT dwModelNumber;
      U8BIT  szModelName[32];
      U8BIT  szDriverVersion[16];
      U32BIT dwReserved1;
      U32BIT dwReserved2;
      U32BIT dwReserved3;
      U32BIT dwReserved4;
      U8BIT  szReserved[32];

} HWVERSIONINFO, *PHWVERSIONINFO;
```

MEMBERS

U32BIT dwFwVersion;	Firmware version on the Device
U32BIT dwHdlVersion;	Contains programmable device binary version
U32BIT dwDriverVersion;	Driver version currently being used
U32BIT dwSerialNumber;	Serial number of the device being used
FAMILY dwFamilyNumber;	Family type, EMACE = 0, E²MACE = 1
U32BIT dwModelNumber;	Number model number of device in decimal
U8BIT szModelName[32];	Model number as a string
U8BIT szDriverVersion[16];	Character equivalent of dwDriverVersion
U32BIT dwReserved1;	Reserved for future use
U32BIT dwReserved2;	Reserved for future use
U32BIT dwReserved3;	Reversed for future use
U32BIT dwReserved4;	Reserved for future use
U8BIT szReserved[32];	Reserved for future use

DESCRIPTION

This structure contains Hardware version information for the SDK.

PRX_HBUF_MESSAGE

Contains received message information read from the RX FIFO Host Buffer

SYNOPSIS

```
typedef struct _RX_HBUF_MESSAGE
{
      U32BIT u32Data;
      U32BIT u32StampHigh;
      U32BIT u32StampLow;
} RX_HBUF_MESSAGE, *PRX_HBUF_MESSAGE;
```

MEMBERS

U32BIT u32Data;	32-bit ARINC Data Word
U32BIT u32StampHigh	Upper 32-bits of Message Time Stamp
U32BIT u32StampLow	Lower 32-bits of Message Time Stamp

DESCRIPTION

This structure contains information about a single Receive 429 message reac from the FIF Host Buffer.

PSWVERSIONINFO

Contains the software version information.

SYNOPSIS

```
Typedef struct_SWVERSIONINFO
{
      U32BIT dwRtlVersion;
      U32BIT dwCoreVersion;
      U8BIT  szRtlVersion[32];
      U32BIT dwReserved1;
      U32BIT dwReserved2;
      U32BIT dwReserved3;
      U32BIT dwReserved4;

} SWVERSIONINFO, *PSWVERIONINFO;
```

MEMBERS

U32BIT dwRtlVersion;	SDK version for specific OS
U32BIT dwCoreVersion;	Core version of code
U8BIT szRtlVersion[32];	SDK version reported in a string format
U32BIT dwReserved1;	Reserved for future use
U32BIT dwReserved2;	Reserved for future use
U32BIT dwReserved3;	Reserved for future use
U32BIT dwReserved4;	Reserved for future use

DESCRIPTION

This structure contains software version information for the SDK.

7 ENUMERATION TYPES

The following types have been changed in release 3.1.2 of the ***DD-42992SX Multi-IO SDK*** for Windows, Linux, and Vx Works. The values of the enumeration types have remained the same, but their names have been changed. The following values have the prefix DD429_ added to the current name.

Original Name	New Name	Value
LOW_SPEED	DD429_LOW_SPEED	0
HIGH_SPEED	DD429_HIGH_SPEED	1
ARINC575	DD429_ARINC575	2
NO_PARITY	DD429_NO_PARITY	0
ODD_PARITY	DD429_ODD_PARITY	1
EVEN_PARITY	DD429_EVEN_PARITY	2
DISABLE	DD429_DISABLE	0
ENABLE	DD429_ENABLE	1
ENABLE_ALT	DD429_ENABLE_ALT	2
BITFORMAT_ORIG	DD429_BITFORMAT_ORIG	0
BITFORMAT_ALT	DD429_BITFORMAT_ALT	1

The names of the enumeration types will need to be changed when recompiling source code for a user application. A user application will not compile without changing the names of the enumeration types.

The changes above will only affect someone trying to recompile their application with version 3.1.2 or later of the ***DD-42992SX Multi-IO SDK***. If you are not recompiling, your application will continue to work as it did before. The names of enumeration types are only used by the compiler and linker.

8 ERROR MESSAGES

0 = The function completed successfully. ERR_SUCCESS.
-6 = IRQ(s) in use. ERR_IRQ.
-20 = No card present in socket. ERR_NOCRD.
-29 = Card in use. ERR_INUSE.
-31 = Device timeout occurred. ERR_TIMEOUT
-95 = NULL value. ERR_NULL
-96 = Invalid handle occurred. ERR_CRDSERV.

-100 = System initialization failed. ERR_INITFAIL.
-101 = Unknown card in socket. ERR_UNKNOWN.
-102 = Card not initialized yet. ERR_CRDINIT.
-103 = Card initialization failed. ERR_CRDINITFAIL.
-105 = Card not responding. ERR_NORES.
-106 = FLASH memory error. ERR_FLASH.
-111 = Invalid transmit queue size. ERR_TXQUEUESZ.
-113 = Invalid receive queue size. ERR_RXQUEUESZ.
-120 = Invalid channel group. ERR_CHNLGROUP.
-121 = Invalid loopback. ERR_LOOPBACK.
-122 = Invalid bit format. ERR_BITFORMAT.
-130 = Invalid transmitter. ERR_TX.
-131 = Invalid parity. ERR_PARITY.
-132 = Invalid speed. ERR_SPEED.
-133 = Invalid enabling. ERR_ENABLE.
-134 = Invalid mode. ERR_MODE
-135 = Invalid time tag. ERR_TIMETAG
-136 = Invalid time tag format. ERR_TTFORMAT
-137 = Invalid time tag roll over. ERR_TTRO
-138 = Invalid time tag resolution. ERR_TTRES
-140 = Invalid frequency. ERR_FREQ.
-141 = Invalid offset. ERR_OFFSET.

-150 = Invalid receiver. ERR_RX.
-151 = Invalid receiver group. ERR_RXGROUP.
-160 = Invalid label and SDI. ERR_LABELSDI.
-161 = Filter error. ERR_FILTER
-169 = Invalid avionic line. ERR_AVIONIC.
-170 = Invalid discrete output number. ERR_DISCRETE.
-171 = Invalid discrete output level. ERR_DLEVEL.
-175 = Serial error. ERR_SERIAL
-176 = Invalid serial channel. ERR_SERCHNL.
-177 = Invalid serial register. ERR_SERREG
-180 = Invalid interrupt condition. ERR_INT_COND
-181 = Invalid interrupt handler. ERR_INT_HANDLER
-185 = Invalid channel type. ERR_CHAN_TYPE

-200 = FIFO Overflow error. ERR_OVERFLOW
-200 = Invalid ARINC 717 Speed. ARINC_717_INVALID_SPEED
-201 = Invalid ARINC 717 Rate Slope. ARINC_717_INVALID_RATE_SLOPE
-202 = Invalid ARINC 717 Wrap-Around. ARINC_717_INVALID_WRAP_AROUND
-203 = Invalid ARINC 717 Buffer Mode. ARINC_717_INVALID_BUFFER_MODE
-204 = Invalid ARINC 717 Protocol Mode. ARINC_717_INVALID_PROTOCOL_MOD
-205 = Invalid ARINC 717 Type. ARINC_717_INVALID_TYPE_MODE
-206 = Invalid ARINC 717 Channel. ARINC_ERR_INVALID_717_CHANNEL
-207 = ARINC 717 Transmit Queue Error. ARINC_717_TX_QUEUE_ERR
-208 = Invalid ARINC 717 Tx Buffer Size. ARINC_717_TX_LOAD_BUFFER_SIZE
-209 = Invalid ARINC 717 State. ARINC_717_INVALID_STATE_ERR
-210 = Device does not meet the minimum FPGA revision. ERR_FPGA_REV.
-211 = Feature not Supported. ERR_FEATURE_NOT_SUPPORTED
-212 = Invalid Channel Number. ERR_INVALID_CHANNEL_NO
-213 = 429 Programmable Channels not reset. ERR_429_PROG_RESET
-214 = Error in ARINC 717 Configuration. ERR_717_PROG_CONFIG
-215 = Invalid Channel Type. ERR_INVALID_CH_TYPE
-216 = Error loading ARINC 717 Transmit Data. ERR_717_PROG_TX_LOAD
-217 = Invalid ARINC 717 State. ERR_717_PROG_SET_STATE
-218 = Error in ARINC 787 Interrupt configuration. ERR_717_PROG_INTERRUPT
-219 = Error in ARINC 717 Receive Data. ERR_717_PROG_RX_DATA
-220 = Error installed ARINC 429 RX Host Buffer. ERR_RX_HBUF_INSTALL
-221 = No ARINC 429 or UART RX Host Buffer Installed. ERR_NO_RX_HBUF
-222 = ARINC 429 RX Host Buffer Overflow. ERR_RX_HBUF_OVERFLOW
-223 = Invalid CanBus State. ERR_CAN_BUS_SET_STATE
-224 = CanBus Transmit Data Error. ERR_CAN_BUS_TX_DATA
-225 = CanBus Firmware Error. ERR_CAN_BUS_FIRMWARE
-230 = Error in Asynchronous Priority input parameter. ERR_ASYNC_PRIORITY
-231= Error in Resolution input parameter. ERR_RESOLUTION
-232 = Error in Frame Control Type input parameter.
ERR_FRAME_CONTROL_TYPE
-233 = Regiester Access error. ERR_REG_ACCESS
-234 = Error in Inter-Word Bit Gap input parameter. ERR_INTER_WORD_GAP
-235 = Error in Word Size input parameter. ERR_WORD_SIZE
-236 = Error in Parity input parameter. ERR_PARTIY_ERROR
-237 = Error in Extra-Bit (Bit 33 error) input parameter. ERR_BIT_33
-238 = Error in Amplitude input parameter. ERR_AMPLITUDE
-239 = Error creating Frame. ERR_TX_FRAME
-240 = Error in Transmit Frame Size. input parameter. ERR_TX_FRAME_SIZE
-241 = Error. Command cannot be executed while frame is running.
ERR_TX_FRAME_RUNNING
-242 = Error in Transmit Repeat Count parameter.
ERR_TX_FRAME_REPEAT_COUNT
-243 = Error, this Label or SDI is not found in the scheduled transmission queue.
ERR_LABELSDI_NOT_FOUND

-244 = Error, channel not available. ERR_CHANNELS
-250 = Error, this feature is not supported by this DDC Card.
ERR_FEATURE_NOT_SUPPORTED

-300 = Invalid CanBus Speed. CAN_BUS_INVALID_SPEED
-301 = Invalid CanBus Channel. CAN_BUS_INVALID_CHANNEL
-302 = Invalid CanBus State. CAN_BUS_INVALID_STATE_ERR

-500 = Function failed. ERR_FUNCTION.

1 = No CanBus Messages Pending CAN_NO_MESSAGE_PENDING
2 = CanBus Transitter is currently busy. CAN_BUS_TX_BUSY

9 APPENDIX A

9.1 IRIG-B Interface

IRIG-B encodes day of year, hour, minute, and second data on a 1 kHz carrier frequency, with an update rate of once per second. IRIG time code standardization allows the MIL-STD-1553 and ARINC 429 protocols on the Multi-IO card to be synchronized to a known reference time. Multi-IO IRIG uses either a digital or amplitude modulated coding, on an audio sine wave carrier, analog signal as an input.

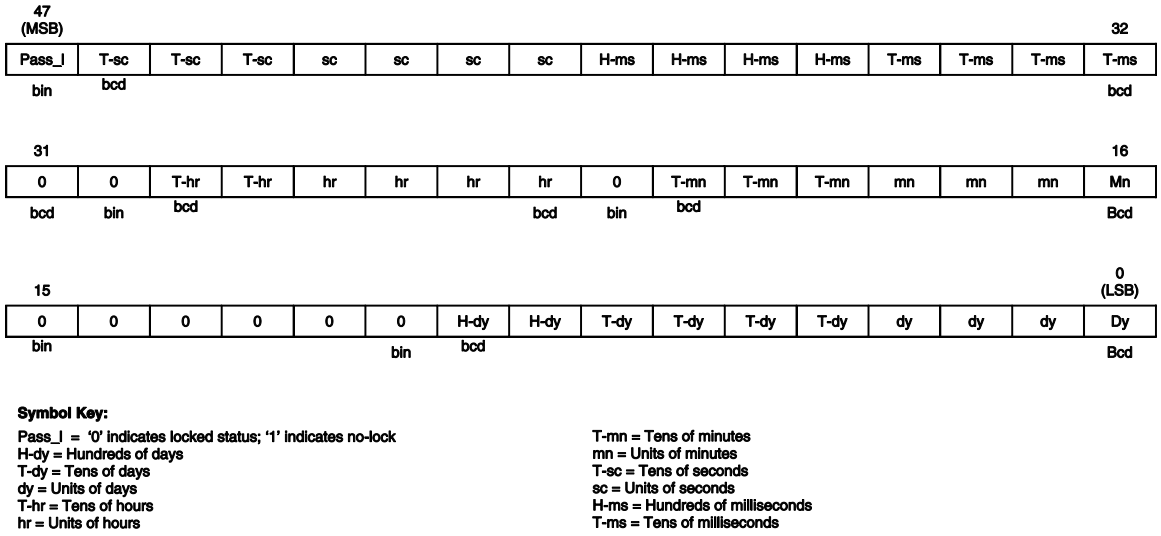


Figure 8. IRIG-B 48-Bit Format

Data Device Corporation

Leadership Built on Over 50 Years of Innovation

Military | Civil Aerospace | Space | Industrial

Data Device Corporation (DDC) is a world leader in the design and manufacture of high-reliability Connectivity, Power and Control solutions (Data Networking; Power Distribution, Control and Conversion; Motor Control and Motion Feedback) for aerospace, defense, space, and industrial applications. With awards for quality, delivery and support, DDC has served these industries as a trusted resource for more than 50 years... providing proven solutions optimized for efficiency, reliability, and performance. Data Device Corporation brands include DDC, Beta Transformer Technology Corporation, National Hybrid Inc., North Hills Signal Processing Corporation, Pascall Electronics Ltd., and XCEL Power Systems Ltd. DDC is headquartered in Bohemia, NY and has manufacturing operations in New York, California, Mexico, and the United Kingdom.

Beta Transformer Technology Corporation, a subsidiary of DDC and leader in high reliability transformer, magnetic and cable assembly solutions for the aerospace, defense, and space industries, offers field proven transformer solutions for the most demanding industrial environments… extreme temperature, shock, vibration, dust, fluid, and radiation. Beta Transformer developed many of the world's smallest transformers and inductors, and is recognized for superior quality and performance. Beta Transformer headquarters along with their main design and manufacturing operations are located in Bohemia, NY. Beta has expanded production capabilities through their manufacturing operations at Beta Transformer Mexico, S. DE R L. DE C.V., located in Ensenada, Mexico, and North Hills Signal Processing Corporation in H. Matamoros Tamaulipas, Mexico, both subsidiaries of Beta Transformer Technology Corporation.

XCEL Power Systems and Pascall Electronics are divisions of DDC Electronics, Ltd., a subsidiary of Data Device Corporation. DDC Electronics, Ltd. specializes in the design and manufacture of power supply solutions for extreme environments. With over 30 years of experience in the defense, aerospace and industrial sectors, DDC Electronics is a trusted source for complete solutions in the design, development and manufacture of electronic power conversion products – from single converters to complex multi- function conversion systems. DDC Electronics products are the first choice for power with In-Flight Entertainment & Connectivity (IFEC) and defense systems. There are more than 170,000 Pascall power supply units installed on commercial aircraft. XCEL and Pascall power supply units are in service with Ground, Air and Naval forces across the world, powering state of the art electronic systems, and trusted by industry leaders to deliver reliable proven performance in some of the most challenging environments to be found anywhere. DDC Electronics, Ltd. headquarters, along with the XCEL Power Systems design operations and the Pascall Electronics factory are located in the UK.

DDC Microelectronics, a division of Data Device Corporation and formerly the space microelectronics division of Maxwell Technologies, is a leading developer and manufacturer of innovative, cost-effective, space-qualified microelectronics solutions for satellites and spacecraft. DDC Microelectronics has provided space-qualified radiation-tolerant and radiation-shielded products, including semiconductors and single-board computers, to the space industry for more than two decades. DDC radiation mitigated power modules, memory modules, and single board computers incorporate powerful commercial silicon for superior performance and high reliability in space applications. DDC Microelectronics specializes in understanding the radiation performance of commercial semiconductors, qualifying selected components for use in space, integrating them with proprietary radiation mitigation technologies, and manufacturing and screening these products in a DLA approved MIL-PRF-38534 facility, located in southern California.

Your Solution Provider for... Connectivity, Power, and Control

Connectivity

Data Bus Solutions
DDC is the market leader in high reliability data bus solutions for MIL-STD-1553/1760, ARINC 429, Fibre Channel, Ethernet, CANbus, Serial I/O and other protocols, and is one of the few companies able to provide a full range of computers, boards, hybrids and ASIC solutions for aerospace, defense and space applications.

Power

Power Supplies
DDC supplies highly customized power products to the aerospace, defense, maritime and satellite communications industries.

Solid-State Power Controllers
DDC's programmable solid-state power controllers provide simple and reliable power management for aerospace and defense systems.

Control

Motor Controllers and Drives
DDC is the world leader in high reliability torque, speed, and position controllers and drives engineered to operate in demanding environments.

Motion Feedback
DDC is the world leader in the design and manufacture of Synchro/Resolver-to-Digital and Digital-to-Synchro/Resolver converters.

Certifications

Data Device Corporation is ISO 9001:2008, AS 9100 Rev C, EN 9100, and JIS Q9100 certified. DDC has been granted certification by the Defense Logistics Agency, Land & Maritime (DLA) for manufacturing Class D, G, H, and K hybrid products in accordance with MIL-PRF-38534. Industry documents used to support DDC's certifications and Quality system are MIL-STD-883, ANSI/NCSL Z540-1, IPC-A-610, MIL-STD-202, JESD-22, and J-STD-020.

Beta Transformer Technology Corporation (BTTC) and its subsidiaries are ISO 9001:2008 and AS 9100 Rev C certified. BTTC has been granted certification as a qualified source of transformers by the Defense Logistics Agency, Land & Maritime (DLA) and is listed on the QPL for products MIL-PRF 21038/27-01 through -31 Product Levels C, M and T.

DDC Electronics, Ltd.'s XCEL Power Systems and Pascall Electronics manufacturing operations are ISO 9001:2008, AS 9100 Rev C, EN9100 and ISO 14001:2004 certified.

Connectivity Power Control

Your Solution Provider for... Connectivity, Power, and Control

Contact Us

Inside the U.S. : Call 1-800-DDC-5757

Outside the U.S. : Call 1-631-567-5600

Operations

DDC Headquarters and Main Factory
105 Wilbur Place, Bohemia, NY 11716-2426
Tel: 1-800-DDC-5757 or (631) 567-5600
www.ddc-web.com
Data Device Corporation

DDC Microelectronics
13000 Gregg Street, Suite C, Poway, CA 92064
Tel:1-800-DDC-5757 or (631) 567-5600

Beta Transformer Technology Corporation
40 Orville Drive, Bohemia, NY 11716-2426
Tel: (631) 244-7393
www.BTTC-Beta.com

Beta Transformer Mexico, S. DE R L. DE C.V.
Avedina 20 De Noviembre
959 Zona Centro, Ensenada, Baja Mexico
Tel: (631) 244-7393

North Hills Signal Processing Corporation
Avedina Jose Escandon y Helquera No. 21
Km. 8.5 Carretera Lauro Villar
H. Matamoros Tamaulipas, Mexico
Tel: (631) 244-7393

DDC Electronics Ltd Headquarters
Westbridge Business Park, Cothey Way
Ryde, Isle of Wight, PO33 1QT, UK
Tel: +44 (0) 1983 817300

bsi. AS/EN 9100 Series Aviation, Space and Defense | ISO 9001 Quality Management | ISO 14001 Environmental Management

Sales Offices

DDC Headquarters and Main Factory
105 Wilbur Place, Bohemia, NY 11716-2426
Tel: 1-800-DDC-5757 or (631) 567-5600
www.ddc-web.com

United Kingdom: DDC U.K., Ltd Sales Office
James House, 27-35 London Road, Newbury,
Berkshire RG14 1JL, England
Tel: +44 1635 811140

France: DDC Electronique
84-88 Bld de la Mission Marchand
92411 Courbevoie Cedex, France
Tel: +33-1-41-16-3424

Germany: DDC Elektronik GmbH
Triebstrasse 3, D-80993 München, Germany
Tel: +49 (0) 89-15 00 12-11

Japan: DDC Electronics K.K.
Suidobashi Sotobori-dori Bldg, 8F, 1-5,
Koraku 1-chome,
Bunkyo-ku, Tokyo 112-0004, Japan
Tel: 81-3-3814-7688
www.ddcjapan.co.jp

Asia: DDC - RO Registered in Singapore
Blk-327 Hougang Ave 5 #05-164
Singapore 530327
Tel: +65 6489 4801

India: DDC Electronics Private Limited
C-31, C/O Quest Offices Pvt. Ltd.
10th Floor, Raheja Towers
M.G Road, Bangalore 560001, India
Tel: 91 80 46797 0368

The first choice for more than 50 years!

DDC is a world leader in the design and manufacture of high-reliability Connectivity, Power and Control solutions (Data Networking Components to Processor Based Subsystems, Space Qualified SBCs & Radiation Hardened Components; Power Distribution, Control & Conversion; Motor Control & Motion Feedback), has served the aerospace, defense, and space industries as a trusted resource for more than 50 years.