

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Visit our website - *Click HERE*

#MN-43000P1-001

DD-43000P1-300
ARINC 573 (717) Card
User's Manual

The information provided in this User's Manual is believed to be accurate; however, no responsibility is assumed by ILC Data Device Corporation for its use, and no license or rights are granted by implication or otherwise in connection therewith.

Specifications are subject to change without notice.

ILC DATA DEVICE CORPORATION®

105 Wilbur Place, Bohemia, New York 11716-2482

For Technical Support – 1-800-DDC-5757 ext. 7402
Headquarters – Tel: (516) 567-5600, Fax: (516) 567-7358
Northern New Jersey – Tel: (908) 687-0441, Fax: (908) 687-0470
West Coast – Tel: (714) 895-9777, Fax: (714) 895-4988
Europe – Tel: +44-(0)1635-811140, Fax: +44-(0)1635-32264
Asia/Pacific – Tel: +81-(0)3-3814-7688, Fax: +81-(0)3-3814-7689
World Wide Web – http://www.ilcddc.com

All rights reserved. No part of this User's Manual may be reproduced or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of ILC Data Device Corporation.

Rev. C January, 1999

© 1998 ILC Data Device Corp.

Table of Contents

Table of Contents ..... **iii**

Chapter 1 ..... **1**

1.0 Hardware Introduction 1

1.1 General Description 1

1.2 Specifications 2

1.2.1 Physical Characteristics 2
1.2.2 Environmental 2
1.2.3 Electrical 2
1.2.4 Protocols Supported 2
1.2.5 External Connector Pinouts 3
1.2.6 Loopback Test Connector 4

1.3 System Requirements 5

1.4 Version Availability 5

1.5 Accessory List 5

Chapter 2 ..... **7**

2.0 Installation 7

2.1 Overview 7

2.2 Configuring the ARINC 573/717 Card 7

2.2.1 I/O Base Address 8
2.2.2 Interrupt 8

2.3 Bootstrap Flash ROM 8

2.4 Installing the ARINC 573/717 Card 9

2.5 Software Introduction 11

2.5.1 General Description 11

2.6 Software Installation 11

2.6.1 Installation 11
2.6.2 Diagnostics 13
2.6.3 Firmware 14

Chapter 3.0 ..... **15**

3.0 Programmer's Reference Software 15

3.1 Card Commands 15

Get717CardAvailability( ) 15
Initialize (Filename) 15
RunProcessor( ) 15
StopProcessor( ) 15
WriteString(String) 16
ReadString(String) 16
SetIOAdr(Address) 16
WaitMillisecond(Time) 16
GetDLLPartNumber(DLLPN) 16
GetDLLVersionNumber(DLLVN) 16
GetFirmwarePartNumber(FirmPN) 16
GetFirmwareVersionNumber(FirmVN) 17

3.2 Transmitter Commands 17

SelectTransmitChannel(Channel) 17
DisableTransmitter(Channel) 17
ConfigureFdauTransmitter(Channel, Protocol, Speed, Type, Cpt, Sf, Wd) 17

ConfigureRS422Transmitter(Channel, Speed, Transmit, Parity, Size) 18
ConfigureInterCpuTransmitter(Channel, Type, Speed, Parity) 18
TransmitRS422(Channel, Number, Data Array) 18
TransmitFdau(Channel, Buffer, Number, Offset Array, Data Array) 19
3.3 Receiver Commands 19
SelectReceiveChannel(Channel) 19
DisableReceiver(Channel) 19
ConfigureFdauReceiver(Channel, Protocol, Speed) 19
ConfigureRS422Receiver(Channel, Speed, Parity, Size) 20
ConfigureInterCpuReceiver(Channel, Speed, Parity) 20
ReceptionStatusInterCpu(Channel, Count, Location) 20
ReceptionStatusFdau(Channel, Flag, Number, String) 21
ReceptionStatusRS422(Channel, Number) 21
ReadIntercpu(Channel, Flag, Word To Read, Data) 21
ReadFdau(Channel, Start, Number, Data) 21
ReadRS422(Channel, Number, Data) 22
BeginReception(Channel) 22
EndReception(Channel) 22
Chapter 4.0 23
4.0 Programming Example Software 23
4.1 General - Version 23
4.2 Loopback Test Transmitter - Harvard Biphase 24
4.3 Loopback Test Transmitters - Bipolar 25
4.4 Loopback Test Transmitters - RS422 26
4.5 Transmitter Test - Harvard Biphase 27
Chapter 5.0 29
5.0 Programmer's Reference Hardware 29
5.1 PC - ISA Bus Interface 29
5.1.1 MAP_MEM Register - Offset 0 30
Comments: 30
Examples: 30
Note: 30
5.1.2 IRQ_STATUS Register - Offset 1 31
Comments: 31
5.1.3 COMMAND Register - Offset 2 32
Comments: 32
Note: 32
5.1.4 IRQ_ENB Register - Offset 3 33
Comments: 33
5.1.5 PAGE Register - Offset 4 34
Comments: 34
5.1.6 FIFO Register - Offset 5 35
Comments: 35
5.1.7 FIFO_STATUS Register - Offset 6 35
Comments: 36
5.1.8 SIG_REG Register - Offset 7 36
Comments: 36
5.2 68332 Time Processor Unit 37
5.3 68332 Queued Serial Module - RS232/422 Transmitters/Receivers 38
5.4 Port F Status LED's 38
5.5 Transmitter Configuration 39
5.5.1 ARINC 429 39

5.5.2 BIPOLAR 40
5.5.3 ARINC 573 41
5.5.4 UART 42
5.6 Receiver Configuration 42
5.6.1 Bi-polar Inputs 43
5.6.2 Unipolar Inputs 43
Customer Contact Report 45
Index 47

List of Figures

FIGURE 1.1: ARINC 573/717 CARD BLOCK DIAGRAM 1
FIGURE 1.2: EXTERNAL CONNECTOR 3
FIGURE 1.3: LOOPBACK MATING CONNECTOR 4
FIGURE 2.1: SWITCH S1 WITH DEFAULT I/O ADDRESS 340H 8
FIGURE 2.2: JP12 WITH DEFAULT IRQ7 JUMPER INSTALLED 8
FIGURE 2.3: ARINC 573/717 CARD LAYOUT 10
FIGURE 2.4: ARINC 573/717 CARD LED'S 10
FIGURE 5.1: ARINC 429 SIGNAL LEVEL EXAMPLE 40
FIGURE 5.2: BIPOLAR SIGNAL LEVEL EXAMPLE 40
FIGURE 5.3: ARINC 573 SIGNAL LEVEL EXAMPLE 41
FIGURE 5.4: UART SIGNAL LEVEL EXAMPLE 42
FIGURE 5.5: BIPOLAR INPUT SIGNAL LEVEL EXAMPLE 43
FIGURE 5.6: UNIPOLAR SIGNAL LEVEL EXAMPLE 43

List of Tables

TABLE 1.1: EXTERNAL CONNECTOR J1 PIN ASSIGNMENTS 3
TABLE 1.2: ARINC 573/717 CARD LOOPBACK CONNECTOR FOR J1 4
TABLE 5.1: I/O ADDRESS OFFSETS 29
TABLE 5.2: TPU CHANNEL FUNCTION ASSIGNMENTS 37
TABLE 5.3: QSM PIN ASSIGNMENTS 38

Chapter 1

1.0 Hardware Introduction

1.1 General Description

The ARINC 573/717 card is an ISA-bus compatible plug-in card that supports four (4) transmit and four (4) receive channels using configurable protocols. The coprocessor's Time Processor Unit (TPU) and the configurable bus drivers provide flexible protocol support. The TPU supports these bit protocols: Bi-Polar (both RZ and NRZ), Harvard Bi-Phase, and RS422.

The card contains 256 Kbytes of shared RAM and 256 Kbytes of local RAM, a 2K x 8-bit bi-directional FIFO for command and data processing, and a 256K byte flash boot EPROM which performs a series of built in tests (BIT) at power-up. It also provides the kernel software for the coprocessor (which may be moved into "local" RAM for execution). The flash EPROM can be field programmed by the host PC when the program enable jumper is installed on the card. The host accesses the flash EPROM on a byte-wide basis; the coprocessor accesses it on a "short" or 16-bit wide basis.

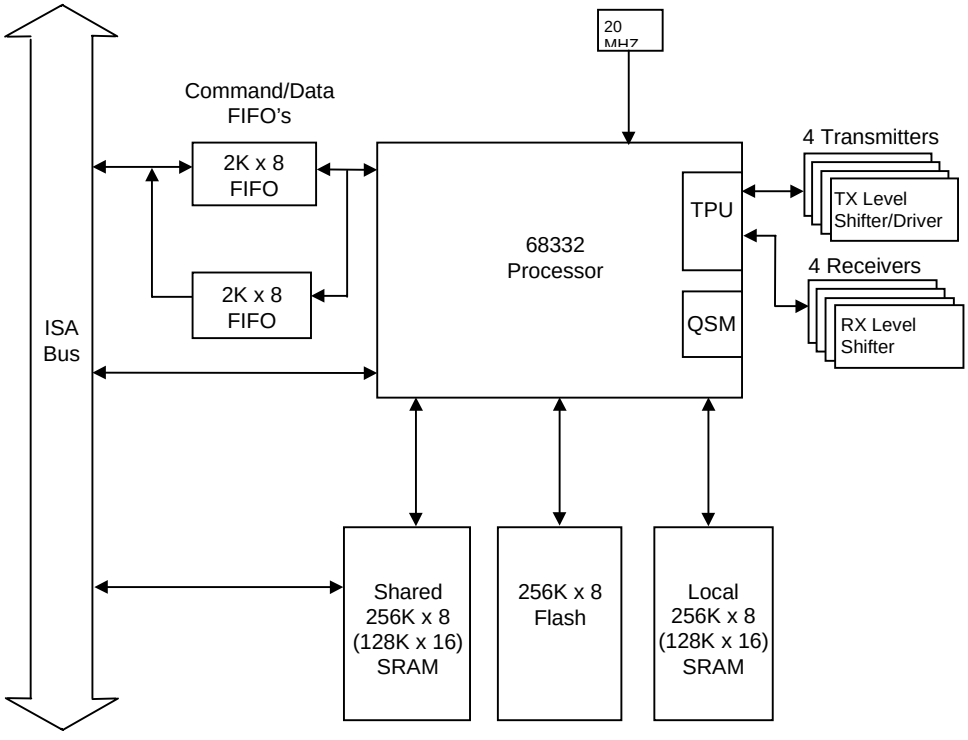


Figure 1.1: ARINC 573/717 Card Block Diagram

1.2 Specifications

1.2.1 Physical Characteristics

Card Size: 4.4” x 14.0” (AT size)

I/O Connector: DB-62

1.2.2 Environmental

Operating Temperature: 0 to 50° C at no more than 95% relative humidity, non-condensing

Storage Temperature: -30 to 50° C at no more than 99% relative humidity, non-condensing

1.2.3 Electrical

Operating Voltage:	±5 VDC and ± 12 VDC
Operating Current:	400 mA @ +5 VDC
(Typical)	110 mA @ -5 VDC
	130 mA @ +12 VDC
	120 mA @ -12 VDC

I/O Protection: All channels and discretes 1 through 5 are protected to 30 VDC

1.2.4 Protocols Supported

Harvard Bi-Phase ARINC 573/717 64, 128, 256, 512, and 1024 words per second.

Bi-Polar RZ ARINC 573/717 64, 128, 256, 512, and 1024 words per second.

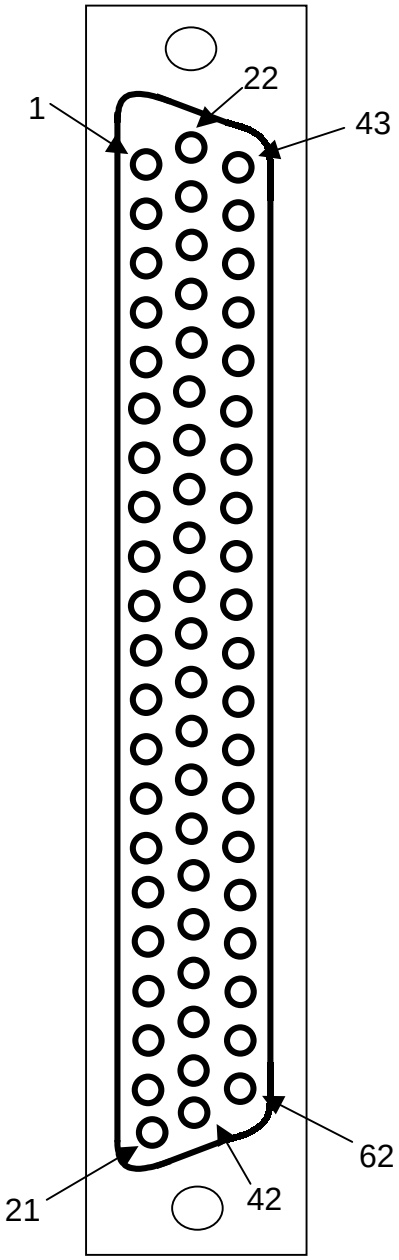
RS422 configurable baud rates

1.2.5 External Connector Pinouts

For I/O connections, the ARINC 573/717 card uses a single DB-62 connector (J1) that is located on the card's rear edge. Refer to figure 1.2 for illustration of this connector. This connector uses a 62 pin high density mating plug kit. One of these plug kits is included as a standard accessory to the card; the DDC part number for this connector is 5301-0025-0001. The manufacturer part number for this plug kit is AMP 748476-1. The ARINC 573/717 card provides a 62-pin, sub-D female interface connector (J1) for I/O interface:

Pin #	Name	Pin #	Name
1	GROUND	32	RESERVED
2	TX0A	33	RESERVED
3	TX0B	34	RESERVED
4	TX1A	35	UNUSED
5	TX1B	36	RESERVED
6	TX2A	37	RESERVED
7	TX2B	38	RESERVED
8	TX3A	39	RESERVED
9	TX3B	40	UNUSED
10	ALT_TX0B	41	RESERVED
11	RESERVED	42	RESERVED
12	ALT_TX1B	43	RESERVED
13	RESERVED	44	RESERVED
14	ALT_TX2B	45	UNUSED
15	RESERVED	46	DISCIN0
16	ALT_TX3B	47	DISCIN1
17	RESERVED	48	DISCIN2
18	UNUSED	49	DISCIN3
19	RX0A	50	DISCIN4
20	RX0B	51	DISCIN5
21	RX1A	52	DISCIN6
22	RX1B	53	DISCIN7
23	RX2A	54	DISCOUT0
24	RX2B	55	DISCOUT1
25	RX3A	56	DISCOUT2
26	RX3B	57	DISCOUT3
27	RESERVED	58	DISCOUT4
28	RESERVED	59	DISCOUT5
29	RESERVED	60	DISCOUT6
30	RESERVED	61	DISCOUT7
31	RESERVED	62	GROUND

Table 1.1: External Connector J1 Pin Assignments



Note mating connector part number:
#AMP 748476-1

Figure 1.2: External connector

1.2.6 Loopback Test Connector

The following describes the DDC-70098-1 loopback assembly for the ARINC 573/717 card. This loopback assembly is useful for diagnostics, and as a convenient way to wrap the transmit channels to the receive channels during software development.

Pin #	Name	Connected to	Name
1	Unused	Unused	Unused
2	TX0A	Pin 19	RX0A
3	TX0B	Pin 20	RX0B
4	TX1A	Pin 21	RX1A
5	TX1B	Pin 22	RX1B
6	TX2A	Pin 23	RX2A
7	TX2B	Pin 24	RX2B
8	TX3A	Pin 25	RX3A
9	TX3B	Pin 26	RX3B
10	Unused	Unused	Unused
11	Unused	Unused	Unused
12	Unused	Unused	Unused
13	Unused	Unused	Unused
14	Unused	Unused	Unused
15	Unused	Unused	Unused
16	Unused	Unused	Unused
17	Unused	Unused	Unused
18	Unused	Unused	Unused
27	Unused	Unused	Unused
28	Unused	Unused	Unused
29	Unused	Unused	Unused
30	Unused	Unused	Unused
31	Unused	Unused	Unused
32	Unused	Unused	Unused
33	Unused	Unused	Unused
34	Unused	Unused	Unused
35	Unused	Unused	Unused
36	Reserved	Pin 41	Reserved
37	Reserved	Pin 42	Reserved
38	Reserved	Pin 43	Reserved
39	Reserved	Pin 44	Reserved
40	Unused	Unused	Unused
45	Unused	Unused	Unused
46	DISCIN0	Pin 54	DISCOUT0
47	DISCIN1	Pin 55	DISCOUT1
48	DISCIN2	Pin 56	DISCOUT2
49	DISCIN3	Pin 57	DISCOUT3
50	DISCIN4	Pin 58	DISCOUT4
51	DISCIN5	Pin 59	DISCOUT5
52	DISCIN6	Pin 60	DISCOUT6
53	DISCIN7	Pin 61	DISCOUT7
62	Unused	Unused	Unused
Note: All unused pins must be left unconnected.			

Table 1.2: ARINC 573/717 Card Loopback Connector for J1

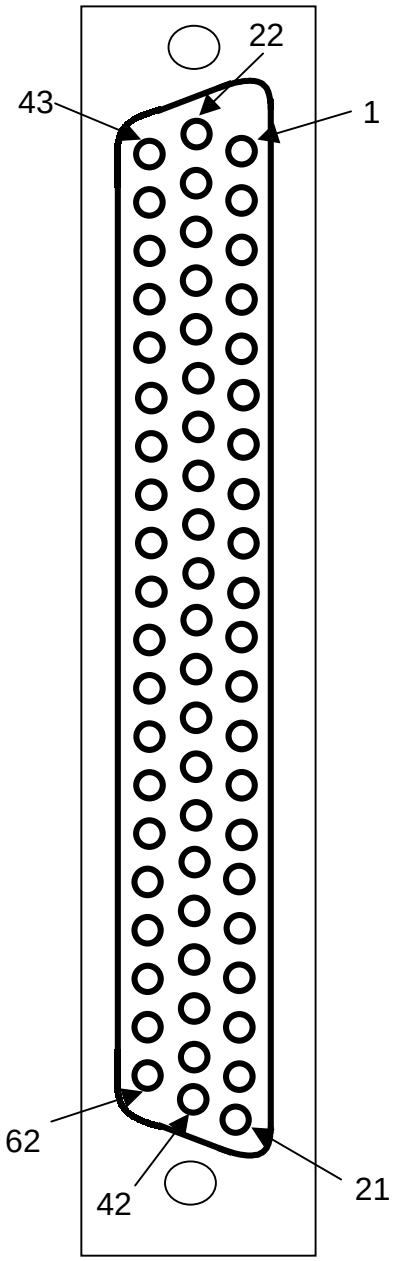


Figure 1.3: Loopback Mating Connector

1.3 System Requirements

The following minimum configuration is required to support operation of the ARINC 573/717 card:

1. IBM Compatible PC (386 or faster - depends upon operating system installed)
2. Microsoft Windows 3.1, 95, or 98

Memory requirements: 16k byte bank on 16k page boundary.

I/O requirements: 8 consecutive 16-bit addresses.

Interrupt requirements: uses single PC interrupt.

1.4 Version Availability

At the time this manual was published, this DLL was available in a 16-bit version appropriate for use with Windows 3.1 and Windows 95. Please call your DDC support representative if you require a 32-bit DLL for Windows 95 or NT.

1.5 Accessory List

1. Multi-Protocol Emulator Card (DD-43000P1-300)
2. Programmer's Toolbox Software (DD-43000S0-300)
3. Loopback connector assembly (DDC-70098-1)
4. HD-62 pin plug kit (5301-0025-0001)
5. User's Manual (This document)

This page

intentionally

left blank

Chapter 2

2.0 Installation

The information in this section enables you to install the ARINC 573/717 card in a PC. The card utilizes memory space, I/O address space, and interrupt resources in the host PC. It is necessary to configure both the card and your computer in order for the ARINC 573/717 card to operate properly. Switches and jumpers set the I/O and Interrupts. The base memory address is set by software (using the PAC717.ini file). The card uses 8 consecutive I/O addresses, a single interrupt, and 16K of PC memory (typically starting at D0000).

2.1 Overview

Installation of the ARINC 573/717 card involves the following procedures:

1. Configuring the card to an available I/O address and interrupt in your system
2. Installing the card's software onto your system's hard drive
3. Verifying that the card operates correctly following installation
4. Troubleshooting any installation problems
5. Wiring the card's I/O connector to your system.

2.2 Configuring the ARINC 573/717 Card

Caution: Never touch static sensitive electronic components without following proper electrostatic protection procedures.

1. Remove the ARINC 573/717 card from its shipping container and anti-static covering. Save this material in case the card must be returned.
2. Figure 2.3 illustrates the ARINC 573/717 card and identifies the jumpers and switches used in configuration.

Note: When selecting a base address and IRQ for this card, do not duplicate settings that are already assigned to other I/O devices in your system.

2.2.1 I/O Base Address

The I/O base address for the ARINC 573/717 card is selected with switch S1 (see Figure 2.3 for location). The factory default setting is 340h. Refer to the figure that follows for configuration information:

Base Address (BCD)			**3**				**4**					**0**
Binary	0	0	**1**	**1**	0	**1**	0	0	0	0	0	0
Open			X	X		X						
Closed					X		X	X	X	X		
S1			**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**		

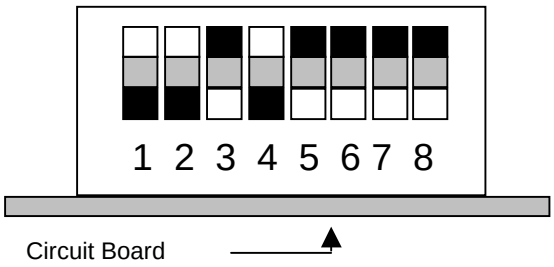


Figure 2.1: Switch S1 with Default I/O Address 340h

2.2.2 Interrupt

The interrupt assigned to the ARINC 573/717 card is determined by the jumper installed on JP12 (see Figure 2.3 for location). The available IRQ settings are 3, 4, 5, 6, 7, or none. The factory default setting is IRQ-7.

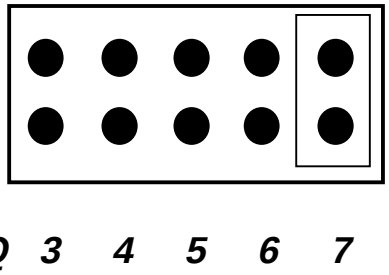


Figure 2.2: JP12 with Default IRQ7 Jumper Installed

2.3 Bootstrap Flash ROM

The flash ROM may be re-programmed from either local and/or shared RAM by installing the appropriate program enable jumpers and following the programming instructions for the 28F200BX-B as provided in the Intel Flash Memory Manual. Jumper JP3 (see Figure 2.3 for location) enables general programming of the flash ROM. To enable programming of the protected flash “boot-block”, you must also place a jumper on JP2. Removing both jumpers prevents accidental erasures. Refer to Chapter 3.0 for additional information on the bootstrap flash ROM.

2.4 Installing the ARINC 573/717 Card

Caution: Never install or remove the ARINC 573/717 card without first powering your system OFF.

1. Configure the card interrupt (JP12) and I/O address (S1) as previously described in *Configuring the ARINC 573/717 Card*, this chapter.
2. Power your system OFF and open the cover to your system's CPU and select an empty ISA slot on the motherboard.
3. Remove the screw that holds the blank card bracket for the selected ISA slot.
4. Insert the ARINC 573/717 card into the ISA slot.
5. Ensure that the card is pressed firmly into the bus connector and that its connector bracket is properly aligned with the chassis opening.
6. Install the screw removed in step 4 (held blank card bracket) into the I/O card bracket and tighten it in place.
7. Replace the CPU cover removed in step 3.
8. Power the system ON and continue the installation procedures and operation.

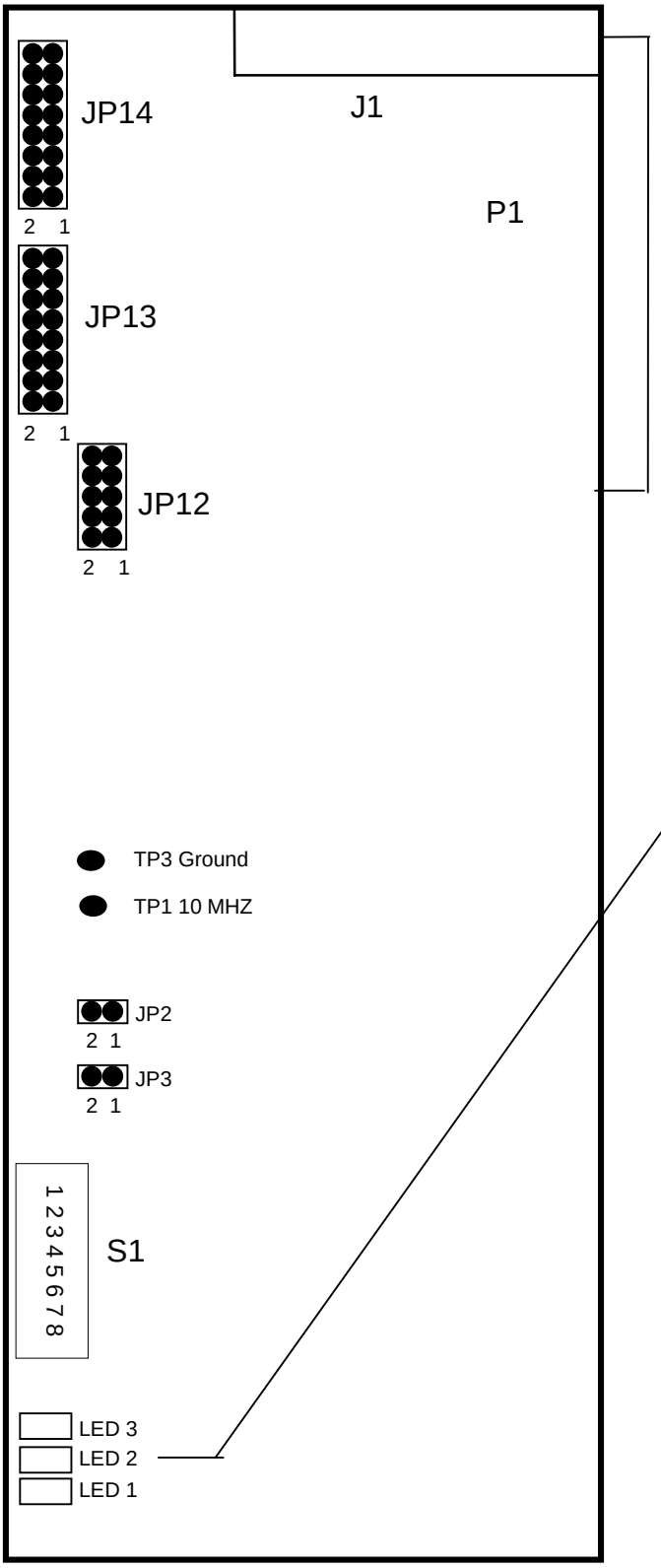
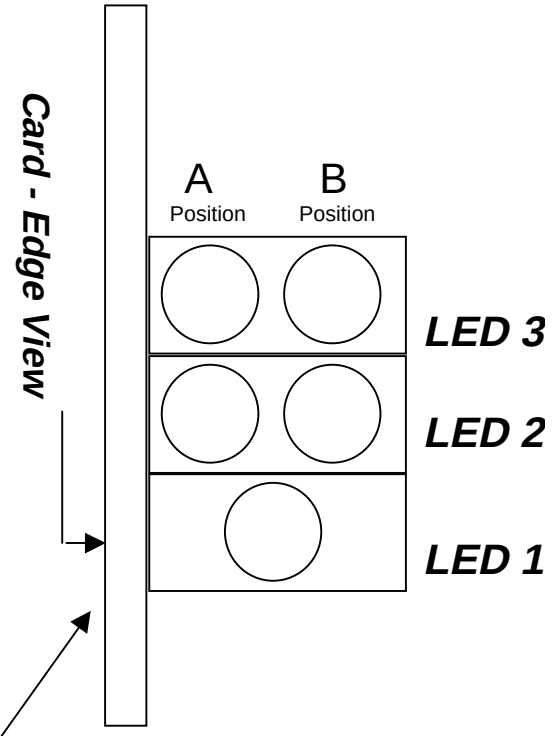


Figure 2.3: ARINC 573/717 Card Layout



LED 1 is the power indicator. It provides a general health indicator for +5V, +12V, and –12V. If any of these supplies are not available, the power light will not be on.

LED 2A, 2B, and 3A show that a receiver is active and is synchronized with incoming data. LED 2A is on when channel 1 is active. LED 2B is on when channel 2 is active. LED 3A is on when channel 3 is active.

LED 3B behaves differently from 2A, 2B, and 3A. This LED is turned on each time a subframe is received on an active channel, then turned off again. Typically this will flash at a 1Hz rate. However if channel 4 is active, this LED is set to operate like the other 3 LEDs.

Figure 2.4: ARINC 573/717 Card LED's

2.5 Software Introduction

2.5.1 General Description

DDC's ARINC 573/717-card supports four (4) transmit and four (4) receive channels with configurable protocols. Currently, these protocols are supported:

Harvard BI-Phase ARINC 573/717 64, 128, 256, 512, and 1024 words per second.

BI-Polar RZ ARINC 573/717 64, 128, 256, 512, and 1024 words per second.

RS422 configurable baud rates

Each transmitter can be configured to operate with the following outputs:

Single buffered output

Double buffered output

Test output with predefined data

Each receiver will automatically sync with the input data, if required.

The ARINC 573/717 card must first be given a RunProcessor command before any commands can be given. The default configuration is for a Harvard BI-Phase test output on transmit channel 1 and Harvard BI-Phase input on receive channel 1.

Note: For Purposes of this function set, Channels are numbered 1 – 4.

2.6 Software Installation

2.6.1 Installation

The software described in this section is contained on disks labeled with DDC part number DD-43000S0-300. Before you are able to use the 573/717 card, this software and one or more 573/717 card(s) must be installed as described in section 2.5 of this manual.

The diskette provided with the 573/717 card contains these files:

1. Install.bat
2. PAC717.DLL
3. PAC717.ini
4. PAC717.h
5. PAC717.lib
6. Loader.exe
7. PC_fdau.bin
8. Example code; Directory TST717.exe and associated files

To install your 573/717 software, place the provided diskette into your system's floppy drive. Use Windows 95/98 Explorer, Windows 3.1 File Manager, or DOS to change directories to DLL directory on floppy. Locate and run install.bat. The system will make a directory called PAC717 on your C drive and copy the following files to it.

1. PAC717.DLL
2. PAC717.ini
3. PAC717.h
4. PAC717.lib

5. Loader.exe
6. PC_fdau.bin
7. Example code; Directory TST717.exe and associated files

It will also copy the following files to the Windows directory on your C drive:

1. PAC717.DLL
2. PAC717.ini

Note: no changes are made to Windows 95 Registry.

The PAC717.ini file contains the following:

```
; File: pac717.Ini

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                       ;
; This is the INI file for the Pac 717 PC Fdau Card                     ;
; by ILC Data Device Corporation.  This file must be in ;
; the \Windows directory.                                                ;
;                                                                       ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

[Pac 717 PC Fdau]
BaseAddress = 0x340
SharedRamAddress = 0xd0000
```

2.6.2 Diagnostics

A small example program is supplied with the standard software that can be used to quickly check the general health of the card.

This program, TST717.exe, will run without modification. You should install the loopback connector assembly on the card prior to running the program. When run, this program turns on Transmitter 1 and checks to see that data is received on Receiver 1. This is also done for each of the three remaining channels.

When the program is started, a “>” prompt will appear on the screen. Type “H” and press enter. This sets channel 1 to 64 wps, channel 2 to 128 wps, channel 3 to 256 wps, and channel 4 to 1024 wps. Received data is tested to see that it matches transmitted data. If not, an error message will be displayed.

This is not a comprehensive test of all board functions, but does provide a method to do a quick check on the general condition of the card.

Please note that system configuration errors and system resource conflicts can make this check fail, even when the card is in proper operational condition. If you do get a failure, it is best to double-check card settings and make sure that upper memory space is reserved for the card.

Reserving memory from the operating system can be done in a number of ways depending on the operating system. This might be done in the config.sys file using a memory manager like EMM386. The command could look like:

Device=c:\emm386.exe noems x=DOOO-D3FF.

It can also be done in the Windows 3.x System.INI file in the [386Enh} section, add EMMExclude=DOOO-D3FF. In a Windows 95/98 system, it can be done using the device manager. With the device manager displayed, click on “Computer”. Then click “Properties”. Click on the “Reserve Resources” tab at the top of the window. Click on the radio button labeled “Memory”. Click on the button “Add…”. Enter start value of OOODOOOO and enter an end value of OOOD3FFF.

2.6.3 Firmware

Firmware Loading Procedure for DD-43000P1-300

The firmware on the DD-43000P1-300 is stored in non-volatile Flash memory. This allows the card to retain its programming between power cycles, but also allows for quick and easy field updating of the firmware if needed. This document describes the process for loading new firmware on the card.

To load the firmware on the DD-43000P1-300 board, insure that jumpers JP2 and JP3 are ON (installed). This allows the program signals to reach the flash ram. From a DOS prompt (A DOS window from within Windows sometimes locks up with loader.exe) go to the directory containing the download file, pc_fdau.bin. Assuming that the I/O address is 0x340, and that a useable shared ram address is 0xd0000, issue the following command.

```
loader 340 d0000 pc_fdau.bin
```

You will see a set of messages indicating that the flash is erasing and then a message indicating that the end of file was reached. These are normal.

```
Erase Block  [0]
Erase Block  [2000]
Erase Block  [4000]
Erase Block  [8000]
Erase Block  [0]
```

A message like the following indicates that an error happened.

```
Erase Block  [0]
Error Flash erase  stat=[0xd0]  clock=[0]
Error erasing Flash ROM
```

Chapter 3.0

3.0 Programmer's Reference Software

This chapter describes the DLL functions used to control the card operation. These functions are organized in 3 main categories; General Card Commands, Transmitter Commands, and Receiver Commands.

3.1 Card Commands

Use the functions that follow to configure and manipulate general card operations. These functions are available as direct calls to the windows DLL. The following parameter types are used:

int	=	16 bit short
string	=	char pointer or array
byte	=	char
long	=	32 bit int

Get717CardAvailability( )

Description: Returns 1 if card is available, 0 otherwise.

Parameters: none

Return Value: int

Initialize (Filename)

Description: Reads the specified file (default is PAC717.ini), and sets the DLL variables for board I/O and base memory values. This is called automatically the first time a DLL function is used, but this function can be used to specify a file other than the default name.

Parameters:	Filename	pointer to string containing .ini filename
Return Value:	int	0 indicates error
		1 indicates success

RunProcessor( )

Description: Sends command to start the 68332 processor.

Parameters: none

Return Value: none

StopProcessor( )

Description: Sends command to stop the 68332 processor.

Parameters: none

Return Value: none

WriteString(String)

Description: Sends a command character string to the card. All commands to the card are issued via strings. This can be used to issue commands that are not currently implemented in the DLL driver.

Parameters:	string	pointer to string to send
Return Value:	int	number characters sent

ReadString(String)

Description: Reads a response string from the card. All responses from the card are returned via strings. This can be used to read the raw responses from the card.

Parameters:	string	pointer to string to fill with response data
Return Value:	int	0 indicates error 1 indicates success

SetIOAdr(Address)

Description: Sets the internal I/O port base address for the DLL. This is automatically done at the initialization of the DLL.

Parameters:	int	address of I/O switch on card
Return Value:	int	0 indicates error 1 indicates success

WaitMillisecond(Time)

Description: provides a convenient way to wait for a specified number of milliseconds. Resolution is 1 msec.

Parameters:	long	time – in number of milliseconds
Return Value:	none	

GetDLLPartNumber(DLLPN)

Description: Returns the part number of the DLL into the specified string.

Parameters:	string	DLLPN in pointer to string
Return Value:	int	0 indicates error 1 indicates success

GetDLLVersionNumber(DLLVN)

Description: Returns the version number of the DLL into the specified string.

Parameters:	string	DLLVN in pointer to string
Return Value:	int	0 indicates error 1 indicates success

GetFirmwarePartNumber(FirmPN)

Description: Returns the part number of the firmware into the specified string.

Parameters:	string	FirmPN in pointer to string
Return Value:	int	0 indicates error 1 indicates success

GetFirmwareVersionNumber(FirmVN)

Description: Returns the version number of the firmware into the specified string.

Parameters:	string	FirmVN in pointer to string
Return Value:	int	0 indicates error 1 indicates success

3.2 Transmitter Commands

Use the following functions to configure and manipulate the card's transmit channels.

SelectTransmitChannel(Channel)

Description: Selects the current transmit channel for any commands which do not have a channel parameter as part of the parameter data.

Parameters:	int	channel 1 - 4
Return Value:	int	0 indicates error 1 indicates success

DisableTransmitter(Channel)

Description: Sets the given channel to a NULL transmitter value.

Parameters:	int	channel 1 - 4
Return Value:	int	0 indicates error 1 indicates success

ConfigureFdauTransmitter(Channel, Protocol, Speed, Type, Cpt, Sf, Wd)

Description: Sets the given channel to be a ARINC 573/717 transmitter with the given protocol (BI-Phase or BI-Polar), speed, and transmit buffer type. Superframe types use the Cpt, Sf, and Wd parameters to describe the # of cycles in a superframe and the location of the superframe cycle word.

Parameters:	int	channel 1 - 4
	int	protocol to use 1 Harvard BI-Phase 2 BI-Polar RZ
	int	speed to transmit (BITS per second) 768, 1536, 3072, 6144, 12288
	int	transmit type 1 Double buffered 2 Single buffered 3 Test output 4 Superframe
	int	Cycles per Superframe
	int	Subframe location of cycle word
	int	Word location of cycle word
Return Value:	int	0 indicates error 1 indicates success

ConfigureRS422Transmitter(Channel, Speed, Transmit, Parity, Size)

Description: Sets the given channel to be a RS422 transmitter. The transmit type can be used to generate test data output which will transmit continuously, or have a single output buffer which will transmit data only upon receipt of the Transmit command.

Parameters:	int	channel 1 - 4
	int	speed to transmit (BITS per second)
	int	transmit type 1 Double buffered 2 Single buffered 3 Test output
	int	parity 1 Odd 2 Even 3 None
	int	size specifies the length of the buffer. Max size is 1024 bytes.
Return Value:	int	0 indicates error 1 indicates success

ConfigureInterCpuTransmitter(Channel, Type, Speed, Parity)

Description: Sets the given channel to be a InterCpu type transmitter. The transmitter will continuously output test data.

Parameters:	int	channel 1 - 4
	int	lru type 1 MKS I 2 MKS II
	int	speed to transmit (BITS per second)
	int	parity 1 Odd 2 Even 3 None
Return Value:	int	0 indicates error 1 indicates success

TransmitRS422(Channel, Number, Data Array)

Description: Sends a stream of characters to the RS422 SINGLE buffered transmitter only.

Parameters:	int	channel 1 - 4
	int	number of characters to send
	byte*	array of data to send
Return Value:	int	0 indicates error 1 indicates success

TransmitFdau(Channel, Buffer, Number, Offset Array, Data Array)

Description: This overwrites the given ARINC 573/717 buffer location with the supplied data. The ARINC 573/717 is either a single or double buffered transmitter. The test transmitter is actually a prefilled double buffer. This allows data to be changed one word at a time.

Parameters:	int	channel 1 - 4
	int	buffer to change, 0 or 1
	int	number of items to change
	int*	array of offset values (1..max)
	int*	array of data values
Return Value:	int	0 indicates error 1 indicates success

For a given buffer, the transmit values in the buffer at each offset location will be set to the corresponding data value, for example, buffer {offset[i]} = data [i]. The transmit buffers contain 4 subframes of data. Therefore, the second word of the second subframe for a 64 wps transmitter would be offset 66.

3.3 Receiver Commands

Use the following commands to configure and manipulate the card's receive channels.

SelectReceiveChannel(Channel)

Description: Selects the current receive channel for any commands which do not have a channel parameter as part of the parameter data.

Parameters:	int	channel 1 - 4
Return Value:	int	0 indicates error 1 indicates success

DisableReceiver(Channel)

Description: Sets the given channel to a NULL receiver value.

Parameters:	int	channel 1 - 4
Return Value:	int	0 indicates error 1 indicates success

ConfigureFdauReceiver(Channel, Protocol, Speed)

Description: Sets the given channel to be a ARINC 573/717 receiver with the given protocol (BI-Phase or BI-Polar), and speed.

Parameters:	int	channel 1 - 4
	int	protocol to use 1 Harvard BI-Phase 2 BI-Polar RZ
	int	speed to receive (BITS per second) 768, 1536, 3072, 6144, 12288
Return Value:	int	0 indicates error 1 indicates success

ConfigureRS422Receiver(Channel, Speed, Parity, Size)

Description: Sets the given channel to be a RS422 receiver. Currently only the single buffered mode is functional.

Parameters:	int	channel 1 - 4
	int	speed to receive (BITS per second)
	int	parity 1 Odd 2 Even 3 None
	int	size specifies the length of the buffer. Max size is 1024 bytes.
Return Value:	int	0 indicates error 1 indicates success

ConfigureInterCpuReceiver(Channel, Speed, Parity)

Description: Sets the given channel to be a InterCpu type receiver. The receiver will automatically sync up to either MKS I or MKS II inputs.

Parameters:	int	channel 1 - 4
	int	speed to receive (BITS per second)
	int	parity 1 Odd 2 Even 3 None
Return Value:	int	0 indicates error 1 indicates success

ReceptionStatusInterCpu(Channel, Count, Location)

Description: Returns the current buffer count and receive location of the intercpu buffer. Every time an idle gap is detected, the buffer counter is incremented. Using this information, the current status of the receiver can be determined..

Parameters:	int	channel 1 - 4
	int*	current buffer count
	int*	current location
Return Value:	int	0 indicates error 1 indicates success

ReceptionStatusFdau(Channel, Flag, Number, String)

Description: Returns the current number of records available to be read on the ARINC 573/717 channel. If the syncFlag parameter is non-zero, the syncString value will be filled with information indicating the status of each buffer in memory. The syncString information packed bit information.

Parameters:	int	channel 1 - 4
	int*	syncFlag 0 no sync data requested 1 return sync data in syncString param
	int*	number records available
	string	sync string of packed data, i.e. 10 records of valid sync data will return as: 1111 1111 0000 0011 1234 5678 910 in two characters. The leftmost bit in each character is the lowest numbered record.
Return Value:	int	0 indicates error 1 indicates success

ReceptionStatusRS422(Channel, Number)

Description: Returns the number of characters received in the RS422 receive buffer.

Parameters:	int	channel 1 - 4
	int*	number records available
Return Value:	int	0 indicates error 1 indicates success

ReadIntercpu(Channel, Flag, Word To Read, Data)

Description: Returns the value of the given word in the intercpu receive buffer. If the newFlag parameter is non-zero, then the function will wait until new data has been received since receipt of the command to read the data.

Parameters:	int	channel 1 - 4
	int	newFlag 0 return data immediately 1 wait for new data
	int	word to read (1 through max)
	int*	data
Return Value:	int	0 indicates error 1 indicates success

ReadFdau(Channel, Start, Number, Data)

Description: Returns complete subframes of data from the channel. The ReceptionStatusFdau command can be used to determine how much data is available in the receive buffer. This command is used to return the data. The start parameter is used to begin data return from a given subframe offset from the start of the current receive buffer. The number parameter gives the number of subframes to return.

Parameters:	int	channel 1 - 4
	int	subframe start offset from the beginning of buffer
	int	number of subframes to read (max 3)
	int*	data
Return Value:	int	0 indicates error 1 indicates success

ReadRS422(Channel, Number, Data)

Description: Returns the given number of characters from the RS422 receive buffer. The receive buffer is a FIFO style buffer. The ReceptionStatusRS422 command can be used to determine how many characters are available.

Parameters:		
	int	channel 1 - 4
	int	number of characters to read (max 200)
	byte* data	

Return Value:		
	int	0 indicates error
		1 indicates success

BeginReception(Channel)

Description: This forces a clean restart of reception on a given channel. This is used to restart a ARINC 573/717 receiver. ARINC 573/717 receivers will eventually fill up with data and begin to overwrite the oldest data in the buffer. This command forces the buffer to a clean state with new data being received.

Parameters:		
	int	channel 1 - 4

Return Value:		
	int	0 indicates error
		1 indicates success

EndReception(Channel)

Description: This forces a ARINC 573/717 receiver to stop storing data in the receive buffer. This is useful when the it has been determined that enough data has been received for the current session. This will prevent old data from being overwritten when the receive buffer fills up.

Parameters:		
	int	channel 1 - 4

Return Value:		
	int	0 indicates error
		1 indicates success

Chapter 4.0

4.0 Programming Example Software

These programming example segments demonstrate some of the basic operations that are required to operate the ARINC 573/717 card. These examples use the general names of the DLL functions.

4.1 General - Version

The following provides the strings for the DLL and the numbers for the firmware product and versions:

```
Initialize ("pac717.ini")
GetDLLPartNumber (responseString)
print "DLL Part Number         = "; responseString
GetDLLVersionNumber (responseString)
print "DLL Version Number      = "; responseString
GetFirmwarePartNumber (responseString)
print "Firmware Part Number    = "; responseString
GetFirmwareVersionNumber (responseString)
print "Firmware Version Number = "; responseString
```

4.2 Loopback Test Transmitter - Harvard Biphase

The following demonstrates how to set up a Harvard Biphase test transmitter and receiver. The reception status is checked every 800 milliseconds and printed. After the last check, the first word (sync word) is read from the card for all records received.

```
TestStatus = PASS

StopProcessor
RunProcessor

delay (500 milliseconds)
ConfigureFdauTransmitter (1, 1, 768, 3, 0, 0, 0)   [Configure transmitter and receiver]
ConfigureFdauReceiver (1, 1, 768)
delay (3000 milliseconds)
BeginReception (1)
delay (1000 milliseconds)
for (i=1) to (5) do
  if (ReceptionStatusFdau (1, 0, numberRecords, syncString) = 0)   [Get Reception status]
    print "No Reception"
    TestStatus = FAIL
  else
    print "number Records = "; numberRecords
  endif
  delay (800 milliseconds)
next

print "NumberRecords = "; numberRecords

for (i=1) to (numberRecords) do          [Get all sync word 1, from received]
  N=ReadFdau (1, 1, 1, intArray)
  if (N <> 1)
    print "No Read  "; N
    TestStatus = FAIL
  else
    print "intArray[0] = "; str (intArray[0],8)
  endif
next
```

4.3 Loopback Test Transmitters - Bipolar

The following is the same as the Harvard Biplane test, except that the transmitter and receiver are Bipolar:

```
TestStatus = PASS

StopProcessor
RunProcessor

delay (500 milliseconds)

ConfigureFdauTransmitter (1, 2, 768, 3, 0, 0, 0)
ConfigureFdauReceiver (1, 2, 768)
delay (3000 milliseconds)
BeginReception (1)

delay (1000 milliseconds)
for (i=1) to (5) do
  if (ReceptionStatusFdau (1, 0, numberRecords, syncString) = 0)
    print "No Reception"
    TestStatus = FAIL
  else
    print "number Records = "; numberRecords
  endif
  delay (800 milliseconds)
next

print "NumberRecords = "; numberRecords

for (i=1) to (numberRecords) do
  N=ReadFdau (1, 1, 1, intArray)
  if (N <> 1)
    print "No Read  "; N
    TestStatus = FAIL
  else
    print "intArray[0] = "; str (intArray[0],8)
  endif
next
```

4.4 Loopback Test Transmitters - RS422

The following sets up a 200 byte transmit and receive buffer, transmits 20 bytes and then reads and compares the data:

```
TestStatus = PASS

Initialize ("pac717.ini")

StopProcessor
RunProcessor

delay (500 milliseconds)

ConfigureRS422Transmitter (1, 14400, 2, 1, 200)        [Configure channels]
ConfigureRS422Receiver (1, 14400, 1, 200)

delay (1000 milliseconds)

for (i=0) to (19) do                                   [Setup transmit data]
  xmitBytes[i] = i+3
  recvBytes[i] = 100
next

TransmitRS422 (1, 20, xmitBytes)                       [Transmit data]
delay (1000 milliseconds)
N=ReadRS422 (1, 20, recvBytes)                         [Read data]

if (N <> 1)
  print "Fail ReadRS422 "; N
  TestStatus = FAIL
else
  for (i=0) to (19) do
    if (xmitBytes[i] <> recvBytes[i])                  [Compare data]
      TestStatus = FAIL
      print "RS422 fail, expected "; xmitBytes[i]; "  received ";
recvBytes[i]
    endif
  next
endif
```

4.5 Transmitter Test - Harvard Biphase

The following sets up a ARINC 573/717 Harvard Biphase transmitter first as a test, then as a double buffer transmitter. The test configuration initializes the double buffer outputs to the test pattern. The test then sets the buffer 0 and 1 offsets for word #2 of subframes 1 and 2 to hex 123. This data is then received and compared.

```
TestStatus = PASS

StopProcessor
RunProcessor

delay (500 milliseconds)

! use this to load up the double buffers with data
ConfigureFdauTransmitter (1, 1, 768, 3, 0, 0, 0)

! now overwrite with the actual double buffer xmitter
ConfigureFdauTransmitter (1, 1, 768, 1, 0, 0, 0)

ConfigureFdauReceiver (1, 1, 768)
delay (3000 milliseconds)

! set word 2 to be 0x123 for both buffers (12 bits)
offset[0] = 2                ! subframe 1
offset[1] = 66               ! subframe 2
fdaudata[0] = 0x123
fdaudata[1] = 0x123
TransmitFdau (1, 0, 2, offset, fdaudata)
TransmitFdau (1, 1, 2, offset, fdaudata)

BeginReception (1)

delay (5000 milliseconds)

for (i=1) to (4) do
  N=ReadFdau (1, 1, 1, intArray)
  if (N <> 1)
    TestStatus = FAIL
  else
    if (intArray[0] = 0x247 and intArray[1] <> 0x123)
      TestStatus = FAIL
          print "1.. intArray[1] = "; str (intArray[1],16)
     endif
    if (intArray[0] = 0x5b8 and intArray[1] <> 0x123)
      TestStatus = FAIL
      print "2.. intArray[1] = "; str (intArray[1],16)
    endif
    if (intArray[0] = 0xa47 and intArray[1] = 0x123)
      TestStatus = FAIL
      print "3.. intArray[1] = "; str (intArray[1],16)
    endif
    if (intArray[0] = 0xdb8 and intArray[1] = 0x123)
      TestStatus = FAIL
      print "4.. intArray[1] = "; str (intArray[1],16)
    endif
  endif
next
```

[Configure transmitter and receiver]

[Build array containing the offset locations and the data to place there]

[Start reception]

[Wait until at least four subframes are received then read data]

[Compare data – only data for subframes 1 and 2 should be hex 123]

This page

intentionally

left blank

Chapter 5.0

5.0 Programmer's Reference Hardware

This section describes the low level interfaces and control registers of the ARINC 573/717 hardware. It is not normally necessary for the user to know or utilize the information contained in this section. Refer to chapter 3 for normal control of the card.

5.1 PC - ISA Bus Interface

Standard PC-Bus I/O instructions are used to control the ARINC 573/717 card. These control functions include starting and stopping the coprocessor, selecting shared memory pages, reading and writing the command FIFO, and testing various status and interrupt bits. Data transfers between the card and the host PC normally use shared RAM on the ARINC 573/717 card.

This card uses eight consecutive I/O address, starting at an address with a least-significant digit of 0 or 8
(e.g., 0x240, 0x368, etc.). The I/O base address is user-selected with switch S1 (refer to Figure 2.3 for location). The factory default address is 340h. In the register descriptions which follow, each address appears as an offset from the base address as defined by the settings on S1:

Offset	Read/Write	Function
0	R/W	MAP_MEM - Shared memory/FLASH memory map/enable register
1	R/W	IRQ_STATUS - FIFO/Host-Request status bits
2	R/W	COMMAND - Run/Stop and Clear-FIFO commands
3	R/W	IRQ_ENB - interrupt enables
4	R/W	PAGE_REG - Shared memory/FLASH memory page-select register
5	R/W	FIFO Read/Write data register
6	R	FIFO_STATUS register
7	R	SIG_REG - Board Signature register (0xA*n*) where *n* = revision number

Table 5.1: I/O Address Offsets

5.1.1 MAP_MEM Register - Offset 0

7	6	5	4	3	2	1	0
enable	**-**	**m19**	**m18**	**m17**	**m16**	**m15**	**m14**

enable — Enable shared RAM (or flash ROM)
This bit enables shared RAM or flash ROM on the PC bus. If off, shared RAM or flash ROM is invisible to the PC. Primarily, this bit allows multiple cards to share the same address space, one at a time, under software control.

m[19..14] — Shared RAM (or flash ROM)
The starting address for the shared RAM "window" is in the first 1 MB of address space. These six bits are the high-order bits of the 20-bit address: mmmm mmxx xxxx xxxx xxxx. The page will be 16 Kbytes in size. The specific page is selected with the page-register (see comments below).

Comments:

This register must be written to, at least once, in order to "wake up" the card. Until written to, the card remains "invisible" to the PC host at power up. This feature allows "backplane-only" PCs with a CPU card to boot properly. Without this feature, the shared RAM window encountered by the PCs Power-On Self Test (POST) could cause a failure in the boot process.

Normally, this register is set only once during card initialization. If multiple cards are used in the same PC, the enable bit can enable and disable a given card's shared RAM window.

If the flashrom bit is set (see below), this same map address applies to the flash ROM instead of shared RAM.

Examples:

Writing "0xB0" (10110000b) to this register enables the shared RAM window at address 0xC0000 through 0xC3FFF.

Writing "0xB4" (10110100b) to this register enables the shared RAM window at address 0xD0000 through 0xD3FFF. This is the factory default.

Writing "0xB7" (10110111b) to this register enables the shared RAM window at address 0xDC000 through 0xDFFFF.

Note:

The region selected for the 16 Kbytes window must be excluded from use by any memory-manager used by the PC (QEMM, EMM, etc.). For example, use the "X=" argument in QEMM386 or EMM386 in the **config.sys** file:

DEVICE = QEMM386.EXE RAM **X=D000-D3FF** (using the second example from above).

or:

DEVICE = EMM386.EXE RAM **X=D000-D3FF**

5.1.2 IRQ_STATUS Register - Offset 1

7	6	5	4	3	2	1	0
irqst	**-**	**-**	**b-i-t**	**cpu_rqst**	**cfifo_full**	**cfifo_hf**	**cfifo_ne**

irqst **Interrupt request**
This bit will be set if any of the four possible interrupts are enabled and set. It will be cleared only when all other interrupt status bits in the register are zero.

b-i-t **Built-In-Test bit** (controlled by the coprocessor)
This reflects the coprocessor's BIT output.

cpu_rqst **CPU request**
If this bit is set, the MC68332 coprocessor has posted a service request to the host PC. This bit is cleared by writing the corresponding bit (b3) to zero in the status register. This is the only writeable bit in the register.

cfifo_ful **Coprocessor-FIFO-full**
This interrupt status bit is set when the Coprocessor Command FIFO has been filled by the coprocessor.

cfifo_hf **Coprocessor-FIFO-half-full**
This interrupt status bit is set when the Coprocessor Command FIFO has been filled to the half-way point by the coprocessor.

cfifo_ne **Coprocessor-FIFO-not-empty**
This interrupt status bit is set whenever there is anything in the Coprocessor Command FIFO.

Comments:

The FIFO status bit interrupts (cfifo_ful, cfifo_hf, and cfifo_ne) are level-sensitive interrupts that are maskable. Normally, the not-empty and full bits generate interrupts. The not-empty bit tells the host that the coprocessor has written at least one byte to the Command FIFO. Under most circumstances, the full bit is an error condition indicating that the entire 2 Kbyte Command FIFO has been filled by the coprocessor without adequate response from the host PC. The half-full interrupt could be used to warn the host PC whenever it falls behind in its responses to the coprocessor FIFO requests.

The Coprocessor Request interrupt is directly set by the coprocessor when it needs attention. It is cleared when the host PC writes the bit back to the register.

5.1.3 COMMAND Register - Offset 2

7	6	5	4	3	2	1	0
flashrom	**-**	**-**	**-**	**-**	**hfifo_clr**	**cfifo_clr**	**run_reset**

flashrom Enables host PC access to the flash ROM, which is the boot-ROM for the coprocessor. When this bit is set, the shared RAM window accesses the flash ROM instead of the shared RAM. Setting this bit forces **run_reset** to zero, since the coprocessor cannot be running when the host PC accesses the flash ROM.

hfifo_clr **Host FIFO Clear**
Setting this bit resets and clears the Host Command FIFO. This bit is "OR'd" with the coprocessor's corresponding FIFO reset bit.

cfifo_clr **Coprocessor FIFO Clear**
Setting this bit resets and clears the Coprocessor Command FIFO. This bit is "OR'd" with the coprocessor's corresponding FIFO reset bit.

run_reset **Coprocessor Run control bit**
When set, this bit allows the coprocessor to run. The coprocessor always runs from the "reset" or "bootstrap" point. When this bit is set to zero, the coprocessor is held in reset mode. This bit is forced to zero (coprocessor reset) when the **flashrom** bit is set.

Comments:

Setting the flashrom bit allows the host PC to access the coprocessor's boot ROM. Normally, access to the boot ROM is read-only and is used by the host to checksum the coprocessor boot ROM during Built In Test (BIT).

The host PC (or the coprocessor) can program flash ROM only when the program enable jumper is in place on the card. Refer to the Intel Flash Memory Handbook for information on programming flash ROM. This component is a 28F200BX-B 2 Mbit "Boot Block" flash memory.

Note:

The host accesses flash ROM as a 256K x 8-bit memory while the coprocessor accesses flash ROM as a
128K x 16-bit ("big-endian") memory.

The host accesses flash ROM through the same window as does the shared RAM and uses exactly the same paging scheme.

Because the host PC "takes over" the address bus connected to flash ROM when the flashrom bit is set, the coprocessor cannot be running when host accesses are made. For this reason, the run_reset bit is forced to a reset mode by the hardware when the flashrom bit is set.

The Command Register may be read-back to reflect the data that was last written.

5.1.4 IRQ_ENB Register - Offset 3

7	6	5	4	3	2	1	0
int_enable	**-**	**-**	**-**	**co_rqst**	**co_fifo_ful**	**co_fifo_hf**	**co_fifo_ne**

int_enable **Interrupt enable**
This is the global interrupt enable for the ARINC 573/717 card. When set, any of the four interrupt sources may cause an interrupt.

co_rqst **Coprocessor Interrupt Enable**
When set, the coprocessor can interrupt the host PC with a request. This request is cleared by writing a zero bit to the **cpu_irqst** bit in the IRQ_STATUS register.

co_fifo_ful **Coprocessor FIFO Full Interrupt Enable**
When set, an interrupt will occur when the coprocessor writes the 2048th byte to the FIFO if the host PC has read none out.

co_fifo_hf **Coprocessor FIFO Half-full Interrupt Enable**
When set, an interrupt will occur when the coprocessor writes the 1024th byte to the FIFO if the host PC has read none out.

co_fifo_ne **Coprocessor FIFO Not-empty Interrupt Enable**
When set, a write to the FIFO by the coprocessor will generate an interrupt.

Comments:

These enable bits selectively enable and disable each interrupt. The interrupts are level interrupts, not edge interrupts. (See the comments in the IRQ_STATUS section.)

The actual card interrupt is determined by a jumper on JP12 (see Figure 2.3 for location). The factory default is IRQ-7. Available interrupt selections are IRQ 3, 4, 5, 6, 7, or none.

5.1.5 PAGE Register - Offset 4

7	6	5	4	3	2	1	0
-	**-**	**-**	**-**	**p3**	**p2**	**p1**	**p0**

p[3..0] **Page Number**

The page number controls which 16 Kbyte page appears in the 16 Kbyte window with a base address set by the MAP_MEM register. This page number applies to either the shared RAM or the flash ROM, depending upon the value of the **flashrom** bit.

Comments:

The **flashrom** bit selects the memory that is accessed through the window by the host PC. When this bit is a one, the host views flash ROM. When it is a zero, the host PC views shared RAM. In both cases, the page number determines which of the sixteen 16 Kbyte blocks of memory is visible.

From a hardware bus point of view, the host PC always accesses shared RAM one byte at a time. When a sixteen-bit access is made by the host PC, two consecutive one-byte accesses are made with the result being placed in the register. The sixteen-bit accesses made by the host PC are "little-endian", meaning that the least significant byte is read first at an even byte address. This is followed by the most significant byte at the next higher odd byte address. The coprocessor's 16-bit reads and writes are "big-endian".

Because all shared RAM data transfers between processors involve 16-bit or 32-bit values, the hardware resolves the "big-endian vs. little-endian" problem by byte swapping the data when going from the shared RAM to the PC host. This solves the problem for 16-bit values, but not for 32-bit values. For 32-bit values, one or the other processor must word swap the 32-bit value when reading or writing the shared RAM.

Because the host PC's x86 CPU is intrinsically a 16-bit processor, the swapping burden is placed upon the host. The host PC should write the high half of the 32-bit value first, followed by the low half, at increasing word addresses starting on a four-byte address boundary. The use of a 386/486/Pentium's 32-bit write causes the wrong word-order to be used when accessing shared RAM.

Due to byte-swapping when accessing the shared RAM, the host PC should always use 16-bit reads and writes. When accessing flash ROM, the host PC must always use byte reads and writes. No "byte-swapping" takes place. Nothing serious will happen if the above rules are not followed, but the resulting byte order may not appear as expected.

5.1.6 FIFO Register - Offset 5

7	6	5	4	3	2	1	0
d7	**d7**	**d5**	**d4**	**d3**	**d2**	**d1**	**d0**

d[7..0] **FIFO Data**
Writing to this address places a byte of data into the host FIFO. Reading from this address reads the coprocessor FIFO.

Comments:

Reading an empty FIFO is an indeterminate operation. The last byte read out will not necessarily be repeated. Writing to a full FIFO is a null operation without affecting the FIFO. The coprocessor's FIFO operations exactly mirror the host's operations. The FIFO's are basically designed as a command communications channel, as opposed to the shared RAM, which primarily passes data back and forth.

5.1.7 FIFO_STATUS Register - Offset 6

7	6	5	4	3	2	1	0
host_rst	**host_ff**	**host_hf**	**host_ne**	**co_rst**	**co_ff**	**co_hf**	**co_ne**

host_rst **Host FIFO reset bit**
Reflects the state of the host FIFO-clear bit

host_ff **Host FIFO full bit**
Set when the 2048th byte is written to the host PC's FIFO .

host_hf **Host FIFO half-full bit**
Set when the 1024th byte is written to the host PC's FIFO.

host_ne **Host FIFO not-empty bit**
Set when the 1st byte is written to the host PC's FIFO.

co_rst **Coprocessor FIFO reset bit**
Reflects the state of the coprocessor FIFO-clear bit.

co_ff **Coprocessor FIFO full bit**
Set when the 2048th byte is written to the coprocessor's FIFO.

co_hf **Coprocessor FIFO half-full bit**
Set when the 1024th byte is written to the coprocessor's FIFO.

co_ne **Coprocessor FIFO not-empty bit**
Set when the 1st byte is written to the coprocessor's FIFO.

Comments:

Only the coprocessor's FIFO status bits are capable of creating an interrupt to the host PC.

The half-full status bits should be monitored carefully by both the host PC and the coprocessor. If either side reaches the half-full state, it clearly indicates that the opposite processor is having a hard time keeping up with the requests. Action should be taken as necessary. Otherwise, the half-full condition will quickly become a full condition, which typically produces a fatal error. A write to this register is a null operation.

5.1.8 SIG_REG Register - Offset 7

7	6	5	4	3	2	1	0
1	**0**	**1**	**0**	**0**	**0**	**0**	**1**

0xA1 **Signature value 0xA*n***

Where *n* is the revision number of the hardware (shown as **1** here).

Comments:

This register simply returns a fixed hex value when read. Writing to this register is a null operation.

The hex value returned by the hardware identifies the board revision number. The application software uses this number to determine the features available for the card.

5.2 68332 Time Processor Unit

The pins of the Time Processor Unit (TPU) in the 68332 coprocessor are mapped to the Transmit and Receive pins as shown in the table below.

TPU Channel	Function
0	TX Channel 0 Leg A Data
1	TX Channel 0 Leg B Data
2	TX Channel 1 Leg A Data
3	TX Channel 1 Leg B Data
4	TX Channel 2 Leg A Data
5	TX Channel 2 Leg B Data
6	TX Channel 3 Leg A Data
7	TX Channel 3 Leg B Data
8	RX Channel 0 Leg A Data
9	RX Channel 0 Leg B Data
10	RX Channel 1 Leg A Data
11	RX Channel 1 Leg B Data
12	RX Channel 2 Leg A Data
13	RX Channel 2 Leg B Data
14	RX Channel 3 Leg A Data
15	RX Channel 3 Leg B Data

Table 5.2: TPU Channel Function Assignments

5.3 68332 Queued Serial Module - RS232/422 Transmitters/Receivers

There are two serial NRZ transmitters and two serial NRZ receivers on the ARINC 573/717 card which are separate from the 68332 TPU. These serial devices are implemented using the Queued Serial Module (QSM) in the 68332 coprocessor. They can be switched on a transmit/receive pair basis to be RS232 or RS422 transmitters/receivers. These are two-wire channels without any control lines (i.e., RTS, CTS, DTR, etc.). These channels are identified as SER_TX1, SER_TX2, SER_RX1, and SER_RX2 on the external connector.

In the table below, PQS*n* indicates a discrete Port QS output. Other QSM pin names have specific functions in the QSM. For information on programming the QSM, refer to the 68332 user's manual.

Port QS is referred to as Port D in earlier 68332 manuals.

QSM pins are assigned as follows:

QSM Pin	QSM Function	ARINC 573/717 Function
MISO/PQS0	MISO	Transmitter 2 Data
MOSI/PQS1	MOSI	Receiver 2 Data
SCK	unused	unused
PCS0/SS/PQS3	PQS3	Transmitter 2 Enable. A "1" enables the outputs of transmitter 2. A "0" places the outputs in a high-impedance state.
PCS1/PQS4	PQS4	Transmitter 2 RS232/RS422 Mode enable bit. A "1" sets transmitter 2 and receiver 2 into RS422 mode.
PCS2/PQS5	PQS5	Transmitter 1 Enable. A "1" enables the outputs of transmitter 1. A "0" places the outputs in a high-impedance state.
PCS3/PQS6	PQS6	Transmitter 1 RS232/RS422 Mode enable bit. A "1" sets transmitter 1 and receiver 1 into RS422 mode.
TXD/PQS7	TXD	Transmitter 1 Data
RXD	RXD	Receiver 1 Data

Table 5.3: QSM Pin Assignments

5.4 Port F Status LED's

PF7	PF6	PF5	PF4	PF3	PF2	PF1	PF0
LED4	**LED3**	**LED2**	**LED1**	**-**	**-**	**-**	**-**

There are four general-purpose LED's available as indicators for the 68332 coprocessor. These are driven directly by the Port F module in the 68332. A "1" lights an LED.

5.5 Transmitter Configuration

The transmitters are based upon ARINC 429 bus drivers with additional circuitry to control amplitude, slope, leg polarity and grounding. Input to the drivers comes from the TPU in the 68332 coprocessor. To relate TPU pins to specific transmitters, refer to *5.2 68332 Time Processor Unit* in this chapter.

Voltage levels can be varied from 2.5 Vp-p to 10 Vp-p. The slope can be switched between 0.4 V/μs (low speed) and 2.6V/μs (high speed).

There are two registers which control the transmitter behavior: the "Transmitter Config Register" and the "Transmitter Bi-polar Register". See the appropriate section in this manual for specific bit locations.

These two registers contain six parameters per transmitter channel:

TXBIPn	Bipolar/Unipolar Control
TXGNDBn	Ground Leg B of the Transmitter
TXnLEV	Voltage Level Control, +/-2.5 V or +/-5 V per leg.
ALT_TXnLEV	Voltage Level Control, set when using alternate output leg B.
TXSLPn	Slope Control, 2.6 V/μs or 0.4 V/μs
ALTTXSLPn	Slope Control, set when using alternate output leg B

The Alternate Output Leg B is for bipolar signals where neither leg goes negative with respect to ground. An example is ARINC 573 where the two legs alternate between 0 V and +5 V, resulting in a +/-5 V differential signal. When using the alternate leg B (i.e., ALT_TX0B instead of TX0B), the "ALT" control bits should mirror the primary control bits. For example, when using the transmitter 0 alternate leg, ALT_TX0LEV should be set to the same value as TX0LEV and ALT_TX0SLP should be set to the same value as TX0SLP.

The following examples show signal levels and register settings for some of the currently supported bit protocols using transmitter 0. All voltages shown are relative to ground.

Note: The following diagrams show only the required setting for the electrical characteristics and do not fully represent the indicated protocol.

5.5.1 ARINC 429

TXBIP0 = 1	Signal is bipolar.
TXGNDB0 = 0	Don't ground leg B of the transmitter.
TX0LEV = 1	Voltage level = +/-5 V per leg, +/-10 Vp-p.
ALT_TX0LEV:	Don't care, not using alternate output leg B.
TX0SLP = X	Set to a 1 for high speed 429, 0 for low speed.
ALT_TX0SLP:	Don't care, not using alternate output leg B.

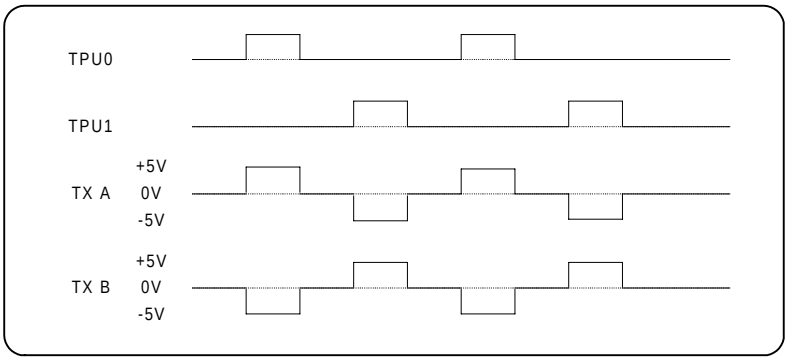


Figure 5.1: ARINC 429 Signal Level Example

5.5.2 BIPOLAR

TXBIP0 = 1 Signal is bipolar.
TXGNDB0 = 1 Ground leg B of the transmitter.
TX0LEV = 1 Voltage level = +/-5 V per leg, +/-5 Vp-p (leg B is grounded).
ALT_TX0LEV: Don't care, not using alternate output leg B.
TX0SLP = X Set to a 1 for high speed, 0 for low speed.
ALT_TX0SLP: Don't care, not using alternate output leg B

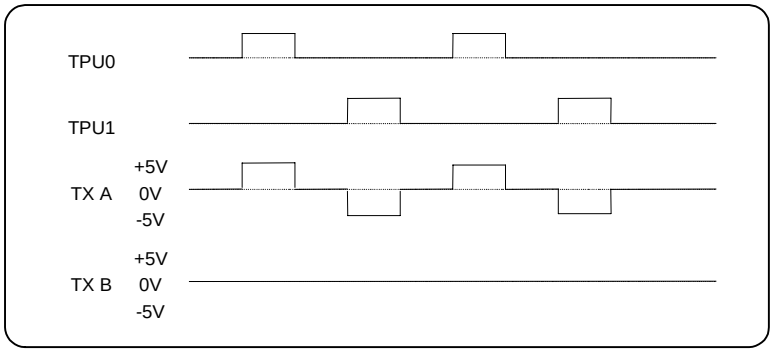


Figure 5.2: Bipolar Signal Level Example

5.5.3 ARINC 573

TXBIP0 = 1 Signal is bipolar.

TXGNDB0 = 0 Don't ground leg B of the transmitter.

TX0LEV = 1 Voltage level = 0,5 V per leg, +/-5 Vp-p.

ALT_TX0LEV = 1 Same as TXLEV0, using alternate output leg B.

TX0SLP = 0 Low speed.

ALT_TX0SLP = 0 Same as TXSLP0, using alternate output leg B.

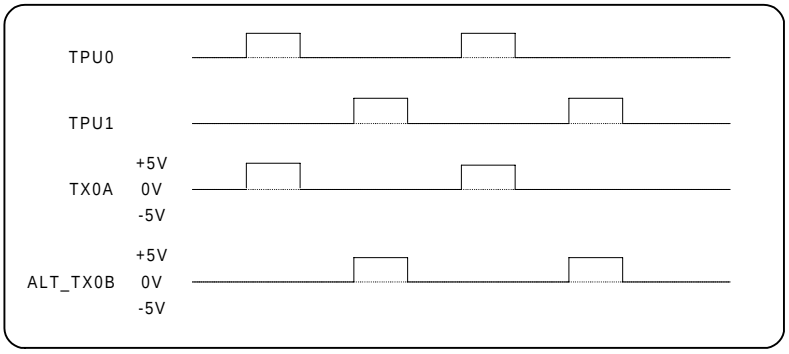


Figure 5.3: ARINC 573 Signal Level Example

5.5.4 UART

TXBIP0 =	**0**	Signal is unipolar.
TXGNDB0 =	**1**	Ground leg B of the transmitter.
TX0LEV =	**1**	Voltage level = 0,5 V on leg A, +5,0 Vp-p.
ALT_TX0LEV:		Don't care, not using alternate output leg B.
TX0SLP =	**X**	Set to a 1 for high speed, 0 for low speed.
ALT_TX0SLP:		Don't care, not using alternate output leg B.

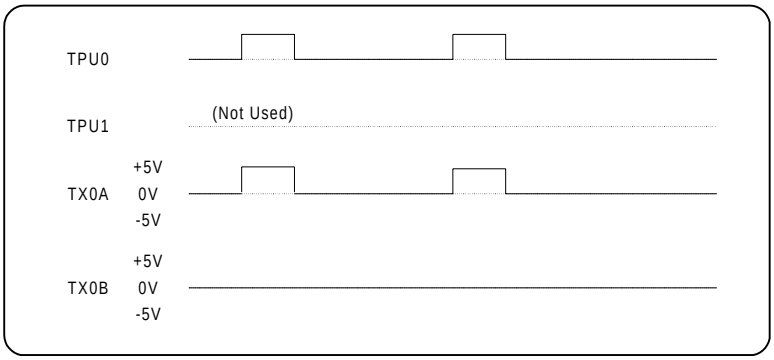


Figure 5.4: UART Signal Level Example

5.6 Receiver Configuration

The receivers are a pair of opto-isolated inputs whose outputs go to the TPU in the 68332 coprocessor. One opto outputs a "1" when leg A is more positive than leg B, the other opto outputs a "1" when leg B is more positive than leg A. If both inputs are equal, or the difference is less than approximately 2.0V, both optos output zeroes. Because the inputs are opto-isolated, common mode voltages and DC offsets have no effect on the signal. To see how the TPU pins correlate to specific receivers, refer to *68332 Time Processor Unit* in this section of the manual.

Receiver configuration is considerably simpler than is transmitter configuration. There are only two possible types of input: bipolar or unipolar. Each receiver has a bipolar control bit (RXBIP*n*) associated with it in the Receiver Configuration Register.

For single-ended inputs, input leg B can be grounded on the card using the RXGNDB*n* bit in the Receiver Configuration Register. Note that receiver ground bit polarity is opposite that of transmitter polarity.

The following examples illustrate the two types of configuration using Receiver 0. In these examples, a common mode voltage of 0 V and a differential voltages of +/-5 V are used.

5.6.1 Bi-polar Inputs

RXBIP0 = 1 Bi-polar input.
GNDRXB = 1 Don't ground input leg B.

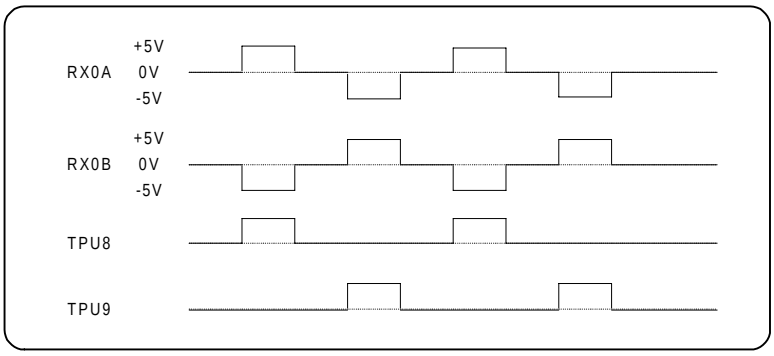


Figure 5.5: Bipolar Input Signal Level Example

5.6.2 Unipolar Inputs

RXBIP0 = 0 Unipolar input.
GNDRXB = 0 Ground leg B (assume single-ended input for this example).

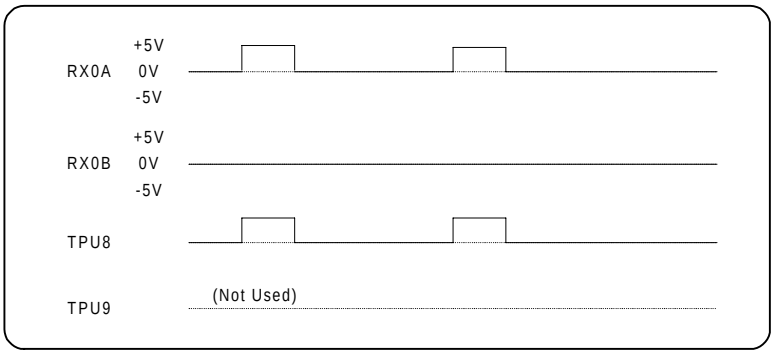


Figure 5.6: Unipolar Signal Level Example

This page

intentionally

left blank

Customer Contact Report

Date: _____

Name: _____
Company: _____
Address: _____
Phone: _____
Fax: _____
E-mail: _____

Product/Model Number: _____
Software Version: _____
Hardware Serial Number: _____

Briefly Describe the problem you are experiencing:

_____
_____
_____
_____
_____
_____

Please call Customer Support at 1-800-DDC-5757, FAX this form to (516) 567-7358 or contact us through our website at: www.ilcddc.com

This page

intentionally

left blank

Index

A

ARINC 429 39
ARINC 573 41
ARINC 573/717 card, description 1

B

base address, I/O 8
BeginReception 22
bipolar 40
Bi-Polar 2, 11
bi-polar, bit protocol 1
bi-polar, inputs receiver 43
block diagram 1
bootstrap flash ROM 8
bus drivers, ARINC 429 39

C

COMMAND register 32
commands, card 15
commands, receiver 19
commands, transmitter 17
config.sys file 30
ConfigureFdauReceiver 19
ConfigureFdauTransmitter 17
ConfigureInterCpuReceiver 20
ConfigureInterCpuTransmitter 18
ConfigureRS422 20
ConfigureRS422Transmitter 18

D

DisableReceiver 19
DisableTransmitter 17

E

electrical, specifications 2
EMM386 30
EndReception 22

F

FIFO register 35
FIFO_STATUS register 35
flash EPROM 1
flash ROM 8

G

Get717CardAvallabllity 15

H

Harvard Bi-Phase 2, 11
Harvard Bi-Phase, bit protocol 1

I

I/O base address 8, 29
I/O, base address 8
installation, ARINC 573/717 card 7, 9
interrupt enable 33
Interrupt, configuration 8
IRQ 8
IRQ_ENB register 33
IRQ_STATUS register 31
ISA 9
ISA bus interface 29

J

J1, pinouts 3
JP12, interrupt configuration 8
JP2, program enable 8
JP3, program enable 8
jumpers, program enable 8

L

LEDs, status 38
level sensitive interrupts 31
loopback test transmitter, bipolar 25
loopback test transmitter, Harvard biphase 24
loopback test transmitter, RS422 26

M

MAP_MEM, register 30

O

offset, I/O base address 29
outputs, transmitter 11

P

PAGE register 34
parameter types, C code 15
pinouts, J1 3
port F module 38
protocols 2, 11
pseudocode 23

Q

QEMM386 30
QSM 38
queued serial module 38

R

ReadFdau 21
ReadIntercpu 21
ReadRS422 22
ReadString 16

receiver, configuration 42
receivers, NRZ 38
ReceptionStatusFdau 21
ReceptionStatusInterCpu 20
ReceptionStatusRS422 21
ROM, flash 8
RS422 ..... 2, 11
RunProcessor 15

S

S1 29
S1, configuration 8
SelectReceiveChannel 19
selectTransmitChannel 17
SetIOAdr 16
SIG_REG register 36
size, card 2
specifications 2
status, LEDs 38
StopProcessor 15

T

temperature, operating 2
temperature, storage 2
time processor unit 1, 37
Time Processor Unit 39
TPU 1, 37
TransmitFdau 19
TransmitRS422 18
transmitter bi-polar register 39
transmitter config register 39
transmitter test, Harvard biphase 27
transmitters, NRZ 38

U

UART 42
unipolar, inputs 43

W

WriteString 16

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP