Condor Engineering CEI520A-1616

ARINC Interface for PCI

In Stock

Used and in Excellent Condition

Open Web Page

https://www.artisantg.com/82073-5

User’s Manual

CEI-100/CEI-200/CEI-x20

Copyrights

CEI-100/CEI-200/CEI-x20 User's Manual (1500-011)

Document Revision:	6.15
Document Date:	8 November 2016
Software Revisions:	
CEI-SW for CEI-100, CEI-200	4.52
CEI-x20-SW for CEI-220, CEI-420/420A, CEI-520/520A, CEI-620, CEI-820, CEI-820TX	4.50

Abaco Systems, Inc.
26 Castilian Drive, Suite B
Goleta, CA 93117
Main +1 805-965-8000 or 877-429-1553 (US-only)
Support +1 805-883-6097

support@abaco.com (email)

https://www.abaco.com/products/avionics

Additional Resources

For more information, please visit the Abaco Systems website at:

www.abaco.com

Contents and Tables

Contents

Figures

Tables

CHAPTER 1

Introduction

Features

The CEI-100, CEI-200, CEI-220, CEI-420, CEI-420A, CEI-520, CEI-520A, CEI-620, CEI-820, and CEI-820TX are intelligent interfaces between multiple ARINC-429/575 data buses and a compatible host computer. The CEI-100, CEI-200, and the CEI-220 are ISA-bus interface boards. The CEI-420 interfaces with the host using the PC/104 bus. The CEI-420A uses the PC/104 bus and includes a PC/104*Plus* pass-through connector. The CEI-520 (5V-only) and CEI-520A (3.3V and 5V) interface with the host using the 33 MHz PCI Revision 2.1 bus specification. The CEI-620 interfaces with the host using the CompactPCI bus specification. The CEI-820/820TX interfaces with the host using the PCI Mezzanine Card (PMC) bus specification.

The CEI-100, CEI-200, and CEI-x20 devices support ARINC-429/575 data buses. The CEI-220, CEI-420, CEI-420A, CEI-520, CEI-520A, CEI-620, and CEI-820 can also support other protocols including CSDB, ARINC 561, ARINC 568, and ARINC 573/717. Each of these boards has its own processor for interface control, data formatting, filtering, and time tagging. They have substantial dual-port RAM for buffering of data. The boards are powerful enough to support all of their channels running at maximum theoretical throughput simultaneously. The hardware buffering mechanisms guarantee valid data transfer between the ARINC buses and user programs. Powerful, easy-to-use application software and user interface libraries support the boards.

The distribution CD-ROM, available for most configurations of the hardware, contains a rich, programming toolkit. CEI-SW is the software distribution for the CEI-100 and CEI-200, and CEI-x20-SW is the software distribution for the CEI-220, CEI-420, CEI-420A, CEI-520, CEI-520A, CEI-620, CEI-820, and CEI-820TX.

The CEI-x20-SW distribution contains device drivers for Linux, VxWorks, and Windows NT/XP/Vista/Server 2008/7/8.0/8.1/Server 2012/10 (note

that the only Windows platforms supported by the CEI-220 and CEI-420/420A are Windows NT and 32-bit XP). The distribution includes source code for many sample programs, supporting ANSI C, C#.NET, Linux, VxWorks, and Visual Basic (VB6 and VB.NET). A LabWindows CVI function panel and sample program are also provided.

BusTools/ARINC, our Windows-based ARINC data bus analysis and simulation software, is available to control any of these products (except the CEI-100, CEI-200, CEI-820, and CEI-820TX). Visit our web site for additional information or to download demo software.

Hardware Specifications

This section shows hardware specifications for the CEI-100, CEI-200, CEI-220, CEI-420, CEI-420A, CEI-520, CEI-520A, CEI-620, CEI-820, and CEI-820TX.

Table 1. Hardware Specifications

Feature	CEI-100	CEI-200	CEI-220	CEI-420	CEI-420A	CEI-520/520A	CEI-620	CEI-820	CEI-820TX
Host Bus Interface	8 bit ISA	16 bit ISA	16 bit ISA	16 bit PC/104	16 bit PC/104 with optional PC/104*plus* pass through connector	(CEI-520) 5V PCI (32-bit) (CEI-520A) 3.3V or 5V PCI (32-bit)	5V cPCI (32-bit)	3.3V or 5V PMC (32-bit)	3.3V or 5V PMC (32-bit)
Dual Port RAM	4K or 8K bytes	16K or 32K	64K bytes	64K bytes	64K bytes	512K bytes	512K bytes	1M bytes	2M bytes
Transmit Channels	1	Up to 4	Up to 12	Up to 8	Up to 8	Up to 16	Up to 16	Up to 8	Up to 32
Receive Channels	2	Up to 8	Up to 12	Up to 8	Up to 8	Up to 16	Up to 16	Up to 8	1
ARINC Transmit Bit Rates	12.5 or 100 Kbs	12.5 or 100 Kbs or programmable	12.5 or 100 Kbs	12.5 or 100 Kbs	12.5 or 100 Kbs	12.5 or 100 Kbs	12.5 or 100 Kbs	12.5 or 100 Kbs or programmable	12.5 or 100 Kbs or programmable
ARINC Receive Low Speed	10.4 to 15.6 Kbs	10.4 to 15.6 Kbs or programmable	10.4 to 15.6 Kbs	10.4 to 15.6 Kbs	10.4 to 15.6 Kbs	10.4 to 15.6 Kbs	10.4 to 15.6 Kbs	10.4 to 15.6 Kbs or programmable	10.4 to 15.6 Kbs or programmable
ARINC Receive High Speed	83 to 125 Kbs	83 to 125 Kbs or programmable	83 to 125 Kbs	83 to 125 Kbs	83 to 125 Kbs	83 to 125 Kbs	83 to 125 Kbs	83 to 125 Kbs or programmable	83 to 125 Kbs or programmable
ARINC Transmit Slew Rate Adjustment	Manual	Manual	Automatic	Automatic	Automatic	Automatic	Automatic	Automatic	Automatic
Memory Footprint Required	4K or 8K bytes	32K bytes	4K bytes (paged access by host to full 64K bytes)	4K bytes (paged access by host to full 64K bytes)	4K bytes (paged access by host to full 64K bytes)	16M byte	16M byte	16M byte	16M byte
I/O port address required	Yes	Yes	No	No	No	No	No	No	No

Feature	CEI-100	CEI-200	CEI-220	CEI-420	CEI-420A	CEI-520/520A	CEI-620	CEI-820	CEI-820TX
On-board Processor	8-bit 80C88 @ 5 MHz	16-bit 80C186 @ 16 MHz	32-bit 80960 @ 12 MHz	32-bit 80960 @ 12 MHz	32-bit 80960 @ 16 MHz	32-bit 80960 @ 100 MHz	32-bit 80960 @ 100 MHz	32-bit 80960 @ 100 MHz	32-bit 80960 @ 100 MHz
Discrete Inputs	None	None	4 universal with software adjustable threshold	8, ground/open, low/high, TTL/CMOS	Up to 16, ground/open, low/high, TTL/CMOS (8 share the same lines as the discrete outputs)	Up to 16, ground/open, low/high, TTL/CMOS	Up to 16, ground/open, low/high, TTL/CMOS	Up to 16, ground/open, low/high, TTL/CMOS (share the same lines as the discrete outputs)	None
Discrete Outputs	None	None	4 Open/Ground, .5 amp max.	8 Open/Ground, .5 amp max.	Up to 8 Open/Ground, .5 amp max. (share the same lines as discrete inputs 9-16)	Up to 16 Open/Ground, .5 amp max.	Up to 16 Open/Ground, .5 amp max.	Up to 16 Open/Ground, .5 amp max. (share the same lines as the discrete inputs)	None
Variable ARINC receive threshold	No	No	Yes - software adjustable from 0 to 25 Vpp	No	No	Yes	Yes	Yes	No
Variable ARINC transmit voltage	No	No	Yes - software adjustable from 0 to 10 Vpp	No	No	Yes - software adjustable from 0 to 10 Vpp - all channels	Yes - software adjustable from 0 to 10 Vpp – channels 1 through 4 only	No	No
Power	2 watts	6 watts	5 watts	3 watts	3 watts	7 watts	7 watts	5 watts	7 watts
ARINC Connector(s)	15 pin female “D” connector	25 pin female “D” connector	37 pin “D” (1 male and 1 female)	40 pin and 20 pin IDC	40 pin and 20 pin IDC	68 pin SCSI-3 and 40 pin IDC	Two 50 pin Champ 0.8mm connectors	50 pin Champ 0.8mm and P14 mezzanine	68 pin SCSI-3 and P14 mezzanine

CHAPTER 2

Installation

Overview

Software for these products is distributed on CD-ROM. Updates are available on our web site (www.abaco.com/products/avionics).

Note: When installing a CEI-x20 device under Windows, you must install the software before installing the hardware. When installing a CEI-x20 device under Linux, you must install the hardware first.

It is recommended that you examine your system for resource conflicts before installation. When upgrading from an older version, uninstall the previous version of software before installing the new version. Software installation instructions are provided in the Quick Start Guide.

Installing the Board

Before handling a circuit board, allocate a static free workstation and be sure you are properly grounded. Then remove the board from its static protective bag and inspect it.

Caution: Be aware of the potential for damage to this (and any other) circuit card by static electricity. The board is shipped in a protective, anti-static bag and must be handled according to static prevention procedures. Failure to protect the board from static electricity may result in damage to the board. Such damage may be evident immediately or not for years.

Each board is tested and shipped with the default settings described in this manual. Resource conflicts or application requirements may cause you to make adjustments to the default board settings. For this reason, the following sections provide detailed information about features that may be configured on these boards through DIP-switches and jumper settings.

Setting the ISA Bus Base Address

The base memory address of the board defines the base segment address of the dual-port memory. The host computer sees the board at this address. The default segment address for the CEI-100 is 0xCC00 (hex), which translates to a physical address of 0xCC000. The default segment address for the CEI-200, CEI-220, CEI-420, and CEI-420A is 0xD000 (hex), which translates to a physical address of 0xD0000.

Make a note of the board address you program so you can make it known to the software. Use the physical address when installing the ISA hardware under VxWorks, Linux, or Windows NT or 32-bit XP. The address for both boards is switch-selectable within the upper memory block (UMB) area of the IBM PC. This is the address space between 640K and 1024K. Since other devices also use this space, select an area not already being used. Many common software programs such as Norton Utilities and Quarterdeck Manifest can help you identify software on your system. The default addresses were chosen to minimize conflict within a typical system; however, you may need to make a change.

Avoid selecting any addresses that directly conflict with other installed devices. The ARINC memory address space must not directly overlap with any other devices. In addition, you must avoid any indirect conflicts. This occurs when a 16-bit memory interface (such as the CEI-200/220/420/420A) is placed in the same 128 Kbytes region (i.e., segment address ranges: A000-C000 or C000-E000 or E000-10000) as an 8-bit memory interface. This is due to an inherent architectural limitation of the IBM PC.

Finally, when selecting a base memory address, be aware of a potential conflict with expanded memory managers (e.g., EMM386, QEMM386) also using the UMB. You can avoid this problem by configuring the memory manager to exclude the ARINC memory from its list of available mapping regions. For example, when starting EMM386, use the command line switch: X=D000-D7FF for a CEI-200 at the default segment address of D000. See your memory manager documentation for a complete discussion of these switches.

The base I/O address is also switch-selectable. I/O commands are used to control the CEI-100 and CEI-200 boards, while the CEI-220, CEI-420, and CEI-420A boards don’t use a base I/O address.

Both the base memory and base I/O address are selected through a single DIP-switch located on each board next to the end panel. Only certain address bits are selectable. All other bits are fixed as shown. When a switch is OFF, the corresponding address bit is a one and when it is ON the corresponding address bit is a zero.

ARINC 429 Slew Rate Configuration for CEI-100/CEI-200

Some jumpers may be installed on the CEI-100 and CEI-200 boards to allow proper selection of ARINC transmission slew rate. On CEI-x20 devices, *slew rate* selection is automatic. The slew rate refers to the rise and fall times for the ARINC signals.

To configure the slew rate for low-speed transmission (12.5K bps) for a channel, install a pair of jumpers to connect the proper capacitance. The bit rate (100K or 12.5K) is software-programmable. As a rule, it is possible to run at either bit rate without the jumpers in place. However, to meet the ARINC 429 specification exactly for low speed slew rate, you must install the appropriate jumpers.

Transmitting at low speed without the jumpers installed causes the ARINC waveform to be squarer than the specification defines. However, most ARINC 429 receivers can tolerate this. Transmitting at high speed with the jumpers in place makes the waveform so far out of specification that it is not useable.

Configuring the CEI-820TX

This section provides configuration information for the CEI-820TX.

CEI-820TX Input/Output Connector Pin Out

To couple the CEI-820TX to ARINC devices, a front bezel connector (P1) and a mezzanine PMC connector (P14) are provided. The front bezel connector is a 68-pin SCSI connector (AMP/Tyco part number 787394-7).

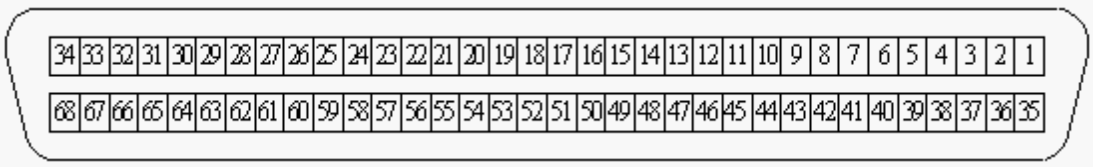


Figure 1. 68-pin Receptacle Connector (SCSI-3 compatible, view facing receptacles)

Table 2. CEI-820TX Connector Definitions (P1 and P14)

Signal	P1 Pin	P14 Pin	Signal	P1 Pin	P14 Pin
RX1A	1		RX1B	35	
TX1A	2	1	TX1B	36	2
TX2A	3	3	TX2B	37	4
TX3A	4	5	TX3B	38	6
TX4A	5	7	TX4B	39	8
TX5A	6	9	TX5B	40	10
TX6A	7	11	TX6B	41	12
TX7A	8	13	TX7B	42	14
TX8A	9	15	TX8B	43	16
TX9A	10	17	TX9B	44	18
TX10A	11	19	TX10B	45	20
TX11A	12	21	TX11B	46	22
TX12A	13	23	TX12B	47	24
TX13A	14	25	TX13B	48	26
TX14A	15	27	TX14B	49	28
TX15A	16	29	TX15B	50	30
TX16A	17	31	TX16B	51	32
Ground	18		Ground	52	
TX17A	19	33	TX17B	53	34
TX18A	20	35	TX18B	54	36
TX19A	21	37	TX19B	55	38
TX20A	22	39	TX20B	56	40
TX21A	23	41	TX21B	57	42
TX22A	24	43	TX22B	58	44
TX23A	25	45	TX23B	59	46
TX24A	26	47	TX24B	60	48
TX25A	27	49	TX25B	61	50
TX26A	28	51	TX26B	62	52
TX27A	29	53	TX27B	63	54
TX28A	30	55	TX28B	64	56
TX29A	31	57	TX29B	65	58
TX30A	32	59	TX30B	66	60
TX31A	33	61	TX31B	67	62
TX32A	34	63	TX32B	68	64

Notes:

Signal pairs “TXnA”/”TXnB” and “RXnA”/”RXnB” are the positive and negative lines of an ARINC differential pair.

The ARINC receive channel is available only on the front bezel connector (P1). Configurations with less than 32 transmit channels use the first (lowest number) defined channels.

One source for a breakout adapter for a 68-pin SCSI connector is Dynamic Engineering (www.dyneng.com). A three-foot SCSI cable, part number HDECable68-1, is available as well as a pass-through breakout adapter with screw terminals, part number HDEterm68. Some standard 68-pin SCSI cables built for computer peripheral use do not include wires for all 68 pins.

Configuring the CEI-820

This section provides configuration information for the CEI-820.

CEI-820 Outline Drawing

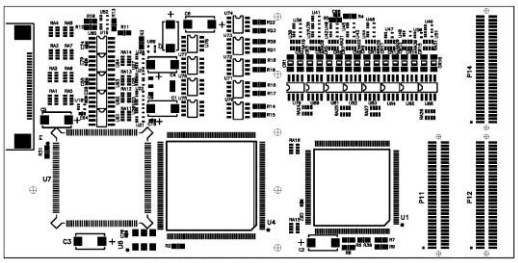


Figure 2. CEI-820 Outline Drawing

CEI-820 Input/Output Connector Pin Out

To couple the CEI-820 to ARINC devices and discrete inputs and outputs, a front panel connector (P1) and a mezzanine PMC connector (P14) are provided.

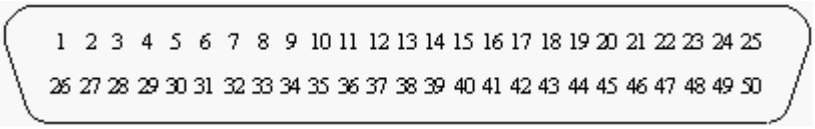


Figure 3. 50-pin P1 connector (AMP Champ 0.8 mm receptacle connectors - AMP part number 787096-1)

Table 3. CEI-820-xx and CEI-820-xxJ Connector Definitions (P1 and P14)

Signal	P1 Pin	P14 Pin	Signal	P1 Pin	P14 Pin
TX8A	1	9	TX8B	26	10
TX7A	2	11	TX7B	27	12
TX6A	3	13	TX6B	28	14
TX5A	4	15	TX5B	29	16
TX4A	5	17	TX4B	30	18
TX3A	6	19	TX3B	31	20
TX2A	7	21	TX2B	32	22
TX1A	8	23	TX1B	33	24
RX8A	9	25	RX8B	34	26

Signal	P1 Pin	P14 Pin	Signal	P1 Pin	P14 Pin
RX7A	10	27	RX7B	35	28
RX6A	11	29	RX6B	36	30
RX5A	12	31	RX5B	37	32
RX4A	13	33	RX4B	38	34
RX3A	14	35	RX3B	39	36
RX2A	15	37	RX2B	40	38
RX1A	16	39	RX1B	41	40
GND	17,19,23	41,47	GND	42,44,48	42,48
Discrete Input/Output 1	20	49	Discrete Input/Output 2	45	50
Discrete Input/Output 3	21	51	Discrete Input/Output 4	46	52
Discrete Input/Output 5	24	53	Discrete Input/Output 6	49	54
Discrete Input/Output 7	25	55	Discrete Input/Output 8	50	56
Discrete Input/Output 9		57	Discrete Input/Output 10		58
Discrete Input/Output 11		59	Discrete Input/Output 12		60
Discrete Input/Output 13		61	Discrete Input/Output 14		62
Discrete Input/Output 15		63	Discrete Input/Output 16		64

Notes:

P1 is a 50-pin, Champ 0.8 mm receptacle connector. The AMP part number is 787096-1.

Signal pairs “TXnA”/”TXnB” and “RXnA”/”RXnB” are the positive and negative lines of an ARINC differential pair.

The discrete inputs and outputs share the same lines. There are a total of 16 bi-directional discretes on the CEI-820, each of which can be treated as an input or an output. For example, you may choose to use discretes 1-4 as inputs and 5-16 as outputs.

Discrete inputs/outputs 9-16 are available only on the mezzanine connector (P14).

Optional, non-ARINC 429 protocols are assigned to the following channels for configurations that support them:

Protocol	Receive channels	Transmit channels
ARINC 573/717 Bipolar return-to-zero encoding	#1	#1
ARINC 573/717 Harvard bi-phase encoding	#1	#1 Since they use the same pins, BPRZ & HBP encoding cannot be used at the same time.
ARINC 561	See Table 4.	None

Table 4. CEI-820-44L Connector Definitions (P1 and P14)

Signal	P1 Pin	P14 Pin	Signal	P1 Pin	P14 Pin
reserved – do not connect	1	9	reserved – do not connect	26	10
reserved – do not connect	2	11	reserved – do not connect	27	12
reserved – do not connect	3	13	reserved – do not connect	28	14
reserved – do not connect	4	15	reserved – do not connect	29	16
ARINC 429 TX4A	5	17	ARINC 429 TX4B	30	18
ARINC 429 TX3A	6	19	ARINC 429 TX3B	31	20
ARINC 429 TX2A	7	21	ARINC 429 TX2B	32	22
ARINC 429 TX1A	8	23	ARINC 429 TX1B	33	24
reserved – do not connect	9	25	reserved – do not connect	34	26
ARINC 561 RX SYNC+	10	27	ARINC 561 RX SYNC-	35	28
ARINC 561 RX DATA+	11	29	ARINC 561 RX DATA-	36	30
ARINC 561 RX CLK+	12	31	ARINC 561 RX CLK-	37	32
ARINC 429 RX4A	13	33	ARINC 429 RX4B	38	34
ARINC 429 RX3A	14	35	ARINC 429 RX3B	39	36
ARINC 429 RX2A	15	37	ARINC 429 RX2B	40	38
ARINC 429 RX1A	16	39	ARINC 429 RX1B	41	40
GND	17,19,23	41,47	GND	42,44,48	42,48
Discrete Input/Output 1	20	49	Discrete Input/Output 2	45	50
Discrete Input/Output 3	21	51	Discrete Input/Output 4	46	52
Discrete Input/Output 5	24	53	Discrete Input/Output 6	49	54
Discrete Input/Output 7	25	55	Discrete Input/Output 8	50	56
Discrete Input/Output 9		57	Discrete Input/Output 10		58
Discrete Input/Output 11		59	Discrete Input/Output 12		60
Discrete Input/Output 13		61	Discrete Input/Output 14		62
Discrete Input/Output 15		63	Discrete Input/Output 16		64

For external wrap, connect each transmit signal to the corresponding receive signal (e.g., TX1+ to RX1+, TX1- to RX1-, etc.). Also, wrap discretes 1-4 to discretes 5-8 (e.g., discrete 1 to discrete 5, discrete 2 to discrete 6, etc.) and discretes 9-12 to discretes 13-16.

The mating connector for P1 is shown below:

Table 5. Mating Connector for P1

Manufacturer:	AMP
Connector:	787131-1
Metal backshell kit:	787233-1
Wire:	30 AWG

CEI-820 Transition Cable Pin Out

Table 6 defines the CONCEI-620 cable provided with each CEI-820. The cable is approximately three feet in length. There is a Champ 0.8mm 50-pin connector (AMP p/n 787131-1) at one end. The other end is a standard subminiature-D, 50-pin plug connector (AMP p/n 205212-3 or equivalent).

Table 6. CEI-820 Transition Cable Pin Out

P1	D50	Signal	P1	D50	Signal
25	17	Discrete Input/Output 7	50	50	Discrete Input/Output 8
24	49	Discrete Input/Output 5	49	33	Discrete Input/Output 6
23	32	GND	48	16	GND
21	47	Discrete Input/Output 3	46	31	Discrete Input/Output 4
20	30	Discrete Input/Output 1	45	14	Discrete Input/Output 2
19	13	GND	44	46	GND
17	28	GND	42	12	GND
16	11	RX1A	41	44	RX1B
15	43	RX2A	40	27	RX2B
14	26	RX3A	39	10	RX3B
13	9	RX4A	38	42	RX4B
12	41	RX5A	37	25	RX5B
11	24	RX6A	36	8	RX6B
10	7	RX7A	35	40	RX7B
9	39	RX8A	34	23	RX8B
8	22	TX1A	33	6	TX1B
7	5	TX2A	32	38	TX2B
6	37	TX3A	31	21	TX3B
5	20	TX4A	30	4	TX4B
4	3	TX5A	29	36	TX5B
3	35	TX6A	28	19	TX6B
2	18	TX7A	27	2	TX7B
1	1	TX8A	26	34	TX8B

Configuring the CEI-620

This section provides configuration information for the CEI-620.

CEI-620 Outline Drawing

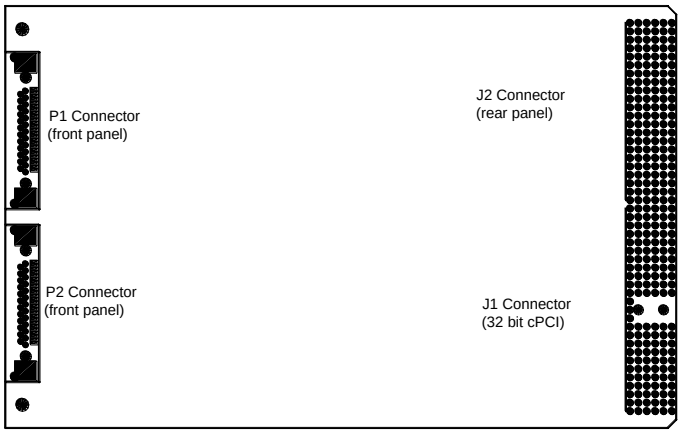


Figure 4. CEI-620 Outline Drawing

CEI-620 Input/Output Connector Pin Out

Use the cPCI J2 back plane connector to couple the CEI-620 to ARINC devices and discrete inputs and outputs for rear panel configurations. For front panel configurations, two connectors, P1 and P2, are provided. Each connector is identical.

Table 7. CEI-620 Rear Panel Input/Output Connector Definition

	Row a	Row b	Row c	Row d	Row e
22	Discrete Output 16	Discrete Output 12	Discrete Output 9	Discrete Output 6	Discrete Output 3
21	Discrete Output 15	Discrete Output 11	Discrete Output 8	Discrete Output 5	Discrete Output 2
20	Discrete Output 14	Discrete Output 10	Discrete Output 7	Discrete Output 4	Discrete Output 1
19	Discrete Output 13	Discrete Input 12	Discrete Input 9	Discrete Input 6	Discrete Input 3
18	Discrete Input 16	Discrete Input 11	Discrete Input 8	Discrete Input 5	Discrete Input 2
17	Discrete Input 15	Discrete Input 10	Discrete Input 7	Discrete Input 4	Discrete Input 1
16	Discrete Input 14	RX16B	RX16A	RX8B	RX8A
15	Discrete Input 13	RX15B	RX15A	RX7B	RX7A
14		RX14B	RX14A	RX6B	RX6A

	Row a	Row b	Row c	Row d	Row e
13		RX13B	RX13A	RX5B	RX5A
12		RX12B	RX12A	RX4B	RX4A
11		RX11B	RX11A	RX3B	RX3A
10		RX10B	RX10A	RX2B	RX2A
9		RX9B	RX9A	RX1B	RX1A
8		TX16B	TX16A	TX8B	TX8A
7		TX15B	TX15A	TX7B	TX7A
6		TX14B	TX14A	TX6B	TX6A
5		TX13B	TX13A	TX5B	TX5A
4		TX12B	TX12A	TX4B	TX4A
3		TX11B	TX11A	TX3B	TX3A
2		TX10B	TX10A	TX2B	TX2A
1		TX9B	TX9A	TX1B	TX1A

Notes:

Per the compact PCI specification, row “A” is closest to the edge of the circuit board and Row “E” is furthest from the edge of the circuit board.

Signal pairs “TXnA”/”TXnB” and “RXnA”/”RXnB” are the positive and negative lines of an ARINC differential pair.

Optional, non-ARINC 429 protocols are assigned to the following channels for configurations that support them:

Protocol	Receive channel(s)	Transmit channel(s)
ARINC 573/717 Bipolar return-to-zero encoding	#1	#1 TX+: pin E1 TX-: pin D1
ARINC 573/717 Harvard bi-phase encoding	#1	#1, #2 TX+: pin E1 TX-: pin E2

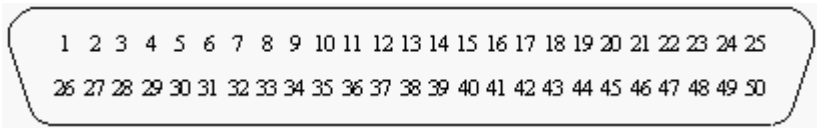


Figure 5. 50-pin Connector (AMP Champ 0.8 mm Receptacle Connectors - AMP Part Number 787096-1)

Table 8. Front View – ARINC 429 Input/Output Connector (P1 and P2)

P1 Connector					P2 Connector			
Pin	**Signal**	**Pin**	**Signal**		**Pin**	**Signal**	**Pin**	**Signal**
25	Discrete Output 7	50	Discrete Output 8		25	Discrete Output 15	50	Discrete Output 16
24	Discrete Output 5	49	Discrete Output 6		24	Discrete Output 13	49	Discrete Output 14
23	Discrete Output 3	48	Discrete Output 4		23	Discrete Output 11	48	Discrete Output 12
22	Discrete Output 1	47	Discrete Output 2		22	Discrete Output 9	47	Discrete Output 10
21	Discrete Input 7	46	Discrete Input 8		21	Discrete Input 15	46	Discrete Input 16
20	Discrete Input 5	45	Discrete Input 6		20	Discrete Input 13	45	Discrete Input 14
19	Discrete Input 3	44	Discrete Input 4		19	Discrete Input 11	44	Discrete Input 12
18	Discrete Input 1	43	Discrete Input 2		18	Discrete Input 9	43	Discrete Input 10
17	GND	42	GND		17	GND	42	GND
16	RX8A	41	RX8B		16	RX16A	41	RX16B
15	RX7A	40	RX7B		15	RX15A	40	RX15B
14	RX6A	39	RX6B		14	RX14A	39	RX14B
13	RX5A	38	RX5B		13	RX13A	38	RX13B
12	RX4A	37	RX4B		12	RX12A	37	RX12B
11	RX3A	36	RX3B		11	RX11A	36	RX11B
10	RX2A	35	RX2B		10	RX10A	35	RX10B
9	RX1A	34	RX1B		9	RX9A	34	RX9B
8	TX8A	33	TX8B		8	TX16A	33	TX16B
7	TX7A	32	TX7B		7	TX15A	32	TX15B
6	TX6A	31	TX6B		6	TX14A	31	TX14B
5	TX5A	30	TX5B		5	TX13A	30	TX13B
4	TX4A	29	TX4B		4	TX12A	29	TX12B
3	TX3A	28	TX3B		3	TX11A	28	TX11B
2	TX2A	27	TX2B		2	TX10A	27	TX10B
1	TX1A	26	TX1B		1	TX9A	26	TX9B

Notes:

P1 and P2 are 50-pin, Champ 0.8 mm receptacle connectors. They are AMP part number 787096-1.

Signal pairs “TXnA”/”TXnB” and “RXnA”/”RXnB” are the positive and negative lines of an ARINC differential pair.

Optional, non-ARINC 429 protocols are assigned to the following channels for configurations that support them:

Protocol	Receive channel(s)	Transmit channel(s)
ARINC 573/717 Bipolar return-to-zero encoding	#1	#1 TX+: pin P1-1 TX-: pin P1-26
ARINC 573/717 Harvard bi-phase encoding	#1	#1, #2 TX+: pin P1-1 TX-: pin P1-2

For external wrap, connect each transmit signal to the corresponding receive signal (e.g., TX1+ to RX1+, TX1- to RX1-, etc.) In addition, connect each discrete output to the corresponding discrete input.

The mating connectors for P1 and P2 are as shown in Table 9:

Table 9. Mating Connectors for P1 and P2

Manufacturer:	AMP
Connector:	787131-1
Metal backshell kit:	787233-1
Wire:	30 AWG

CEI-620 Transition Cable Pin Out

Table 10 defines the CONCEI-620 cable provided with each CEI-620. The cable is approximately three feet in length. There is a Champ 0.8mm 50 pin connector (AMP 787131-1) on one end. The other end is a standard subminiature-D, 50-pin plug connector (AMP p/n 205212-3 or equivalent).

Table 10. CEI-620 Transition Cable Pin Out

P1 Connector				P2 Connector		
P1	**D50**	**Signal**		**P2**	**D50**	**Signal**
25	17	Discrete Output 7		25	17	Discrete Output 15
24	49	Discrete Output 5		24	49	Discrete Output 13
23	32	Discrete Output 3		23	32	Discrete Output 11
22	15	Discrete Output 1		22	15	Discrete Output 9
21	47	Discrete Input 7		21	47	Discrete Input 15
20	30	Discrete Input 5		20	30	Discrete Input 13
19	13	Discrete Input 3		19	13	Discrete Input 11
18	45	Discrete Input 1		18	45	Discrete Input 9
17	28	GND		17	28	GND

P1 Connector				P2 Connector		
P1	**D50**	**Signal**		**P2**	**D50**	**Signal**
16	11	RX8A		16	11	RX16A
15	43	RX7A		15	43	RX15A
14	26	RX6A		14	26	RX14A
13	9	RX5A		13	9	RX13A
12	41	RX4A		12	41	RX12A
11	24	RX3A		11	24	RX11A
10	7	RX2A		10	7	RX10A
9	39	RX1A		9	39	RX9A
8	22	TX8A		8	22	TX16A
7	5	TX7A		7	5	TX15A
6	37	TX6A		6	37	TX14A
5	20	TX5A		5	20	TX13A
4	3	TX4A		4	3	TX12A
3	35	TX3A		3	35	TX11A
2	18	TX2A		2	18	TX10A
1	1	TX1A		1	1	TX9A
50	50	Discrete Output 8		50	50	Discrete Output 16
49	33	Discrete Output 6		49	33	Discrete Output 14
48	16	Discrete Output 4		48	16	Discrete Output 12
47	48	Discrete Output 2		47	48	Discrete Output 10
46	31	Discrete Input 8		46	31	Discrete Input 16
45	14	Discrete Input 6		45	14	Discrete Input 14
44	46	Discrete Input 4		44	46	Discrete Input 12
43	29	Discrete Input 2		43	29	Discrete Input 10
42	12	GND		42	12	GND
41	44	RX8B		41	44	RX16B
40	27	RX7B		40	27	RX15B
39	10	RX6B		39	10	RX14B
38	42	RX5B		38	42	RX13B
37	25	RX4B		37	25	RX12B
36	8	RX3B		36	8	RX11B
35	40	RX2B		35	40	RX10B
34	23	RX1B		34	23	RX9B
33	6	TX8B		33	6	TX16B
32	38	TX7B		32	38	TX15B
31	21	TX6B		31	21	TX14B

P1 Connector				P2 Connector		
P1	D50	Signal		P2	D50	Signal
30	4	TX5B		30	4	TX13B
29	36	TX4B		29	36	TX12B
28	19	TX3B		28	19	TX11B
27	2	TX2B		27	2	TX10B
26	34	TX1B		26	34	TX9B

Configuring the CEI-520 and CEI-520A

This section provides configuration information for the CEI-520 and the CEI-520A.

CEI-520/520A Outline Drawing

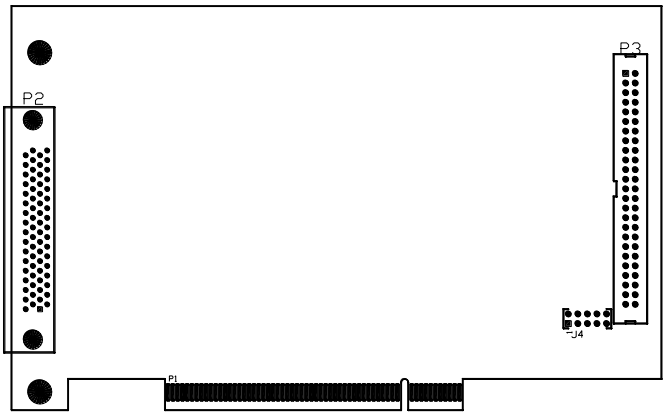


Figure 6. CEI-520/520A Outline Drawing

CEI-520/520A Input/Output Connector Pin Out

To couple the CEI-520/520A to ARINC devices using the rear panel, the following rear panel connector is provided (AMP/Tyco 787171-1).

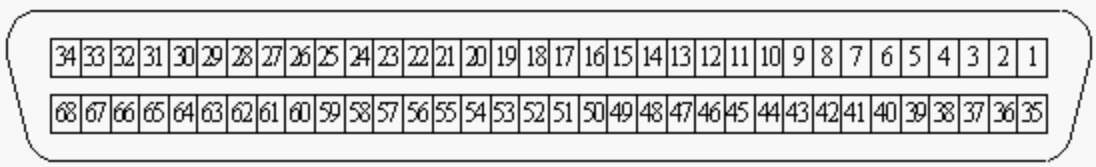


Figure 7. 68-pin Receptacle Connector (SCSI-3 compatible with rails and latch blocks, view facing receptacles)

Table 11. Front View – Standard CEI-520/520A ARINC 429 I/O Connector (P2)

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	TX1+	35	TX1-	18	RX1+	52	RX1-
2	TX2+	36	TX2-	19	RX2+	53	RX2-
3	TX3+	37	TX3-	20	RX3+	54	RX3-
4	TX4+	38	TX4-	21	RX4+	55	RX4-
5	TX5+	39	TX5-	22	RX5+	56	RX5-
6	TX6+	40	TX6-	23	RX6+	57	RX6-
7	TX7+	41	TX7-	24	RX7+	58	RX7-
8	TX8+	42	TX8-	25	RX8+	59	RX8-
9	TX9+	43	TX9-	26	GND	60	GND
10	TX10+	44	TX10-	27	RX9+	61	RX9-
11	TX11+	45	TX11-	28	RX10+	62	RX10-
12	TX12+	46	TX12-	29	RX11+	63	RX11-
13	TX13+	47	TX13-	30	RX12+	64	RX12-
14	TX14+	48	TX14-	31	RX13+	65	RX13-
15	TX15+	49	TX15-	32	RX14+	66	RX14-
16	TX16+	50	TX16-	33	RX15+	67	RX15-
17	GND	51	GND	34	RX16+	68	RX16-

Note: For external wrap, connect each transmit signal to the corresponding receive signal (e.g. TX1+ to RX1+, TX1- to RX1-, etc.).

Also available are the CEI-520-1208-C/CEI-520A-1208-C (which provide CSDB support) and the CEI-520-xxxx-J/CEI-520A-xxxx-J (which provide ARINC 573/717 support). The P2 connector pinouts for these configurations are shown in the following tables.

Table 12. Front View – CEI-520-1208-C/CEI-520A-1208-C I/O Connector (P2)

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	TX1+	35	TX1-	18	RX1+	52	RX1-
2	TX2+	36	TX2-	19	RX2+	53	RX2-
3	TX3+	37	TX3-	20	RX3+	54	RX3-
4	TX4+	38	TX4-	21	RX4+	55	RX4-
5	TX5+	39	TX5-	22	RX5+	56	RX5-
6	TX6+	40	TX6-	23	RX6+	57	RX6-
7	TX7+	41	TX7-	24	RX7+	58	RX7-
8	TX8+	42	TX8-	25	RX8+	59	RX8-
9	CSDB TX1+	43	reserved	26	GND	60	GND
10	CSDB TX1-	44	reserved	27	RX9+	61	RX9-

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
11	CSDB TX2+	45	reserved	28	RX10+	62	RX10-
12	CSDB TX2-	46	reserved	29	RX11+	63	RX11-
13	CSDB TX3+	47	reserved	30	RX12+	64	RX12-
14	CSDB TX3-	48	reserved	31	CSDB RX1+	65	CSDB RX1-
15	CSDB TX4+	49	reserved	32	CSDB RX2+	66	CSDB RX2-
16	CSDB TX4-	50	reserved	33	CSDB RX3+	67	CSDB RX3-
17	GND	51	GND	34	CSDB RX4+	68	CSDB RX4-

Table 13. Front View – CEI-520-xxxx-J/CEI-520A-xxxx-J I/O Connector (P2)

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	ARINC 573/717 BPRZ TX+ ARINC 573/717 HBP TX+	35	ARINC 573/717 BPRZ TX-	18	ARINC 573/717 RX+ (both BPRZ and HBP)	52	ARINC 573/717 RX- (both BPRZ and HBP)
2	ARINC 573/717 HBP TX-	36	reserved	19	reserved	53	reserved
3	TX3+	37	TX3-	20	RX3+	54	RX3-
4	TX4+	38	TX4-	21	RX4+	55	RX4-
5	TX5+	39	TX5-	22	RX5+	56	RX5-
6	TX6+	40	TX6-	23	RX6+	57	RX6-
7	TX7+	41	TX7-	24	RX7+	58	RX7-
8	TX8+	42	TX8-	25	RX8+	59	RX8-
9	TX9+	43	TX9-	26	GND	60	GND
10	TX10+	44	TX10-	27	RX9+	61	RX9-
11	TX11+	45	TX11-	28	RX10+	62	RX10-
12	TX12+	46	TX12-	29	RX11+	63	RX11-
13	TX13+	47	TX13-	30	RX12+	64	RX12-
14	TX14+	48	TX14-	31	RX13+	65	RX13-
15	TX15+	49	TX15-	32	RX14+	66	RX14-
16	TX16+	50	TX16-	33	RX15+	67	RX15-
17	GND	51	GND	34	RX16+	68	RX16-

Notes:

The ARINC 573/717 HBP and BPRZ transmitters cannot both be used simultaneously.

By default, the CEI-520/520A-xxJ configurations provide one 573/717 channel (TX1/RX1) and up to 14 ARINC 429 channels (TX3/RX3-TX16/RX16). Channel TX2/RX2 is not available. A special configuration of the CEI-520/520A-xxJ is also supported that allows ARINC 429-only operations on all available receivers/transmitters, with no channels available for 573/717. This mode causes a CEI-520/520A-xxJ to

effectively implement a CEI-520/520A-xx channel mapping. The documentation for the AR_LOADSLV routine describes how to enable this mode of operation.

The CEI-520/520A has a rear panel 68-pin receptacle connector (SCSI-3 compatible with rails and latch blocks). A mating connector (using a SCSI-3 cable plug connector with wire lacing termination) is shown in Table 14.

Table 14. CEI-520/520A P2 Mating Connector

Manufacturer	AMP
Connector	750913-7 or 1-750913-7
Metal back shell kit	Various options
Wire	28 or 30 AWG

The CEI-520/520A uses a 50-pin IDC-style connector (shown in Figure 8) for discrete input/output. Table 15 shows the pinout information for the discrete I/O connector.

2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46	48	50
1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39	41	43	45	47	49

Figure 8. 50-pin IDC-style Discrete I/O connector (3M Part Number 2550-6002-UB)

Table 15. Top View – Discrete Input/Output Connector (P3)

Pin	Signal	Pin	Signal
1	Discrete Input 1	2	Discrete Input 2
3	Discrete Input 3	4	Discrete Input 4
5	Discrete Input 5	6	Discrete Input 6
7	Discrete Input 7	8	Discrete Input 8
9	Discrete Input 9	10	Discrete Input 10
11	Discrete Input 11	12	Discrete Input 12
13	Discrete Input 13	14	Discrete Input 14
15	Discrete Input 15	16	Discrete Input 16
17	GND	18	GND
19	Discrete Output 1	20	Discrete Output 2
21	Discrete Output 3	22	Discrete Output 4
23	Discrete Output 5	24	Discrete Output 6
25	Discrete Output 7	26	Discrete Output 8

Pin	Signal	Pin	Signal
27	Discrete Output 9	28	Discrete Output 10
29	Discrete Output 11	30	Discrete Output 12
31	Discrete Output 13	32	Discrete Output 14
33	Discrete Output 15	34	Discrete Output 16
35	GND	36	
37		38	
39		40	
41		42	
43		44	
45		46	
47		48	
49		50	

Note: For external wrap, connect each discrete output to the corresponding discrete input.

An optional 15” adapter cable that brings all of the CEI-520/520A P3 signals out to a 37-pin D-sub female connector (mounted on a PC expansion card end-panel) is available. For details, search our web site for the part number CONM50-D37. A 30” version of the adapter cable is also available (CONM50-D37-30).

A mating connector for P3 on the CEI-520/520A is shown in Table 16.

Table 16. CEI-520/520A P3 Mating Connector

Manufacturer	3M
Connector	925110-50-R

CEI-520/520A Transition Cable Pin Out

Figure 9 shows how the pins are numbered on the cable connector.

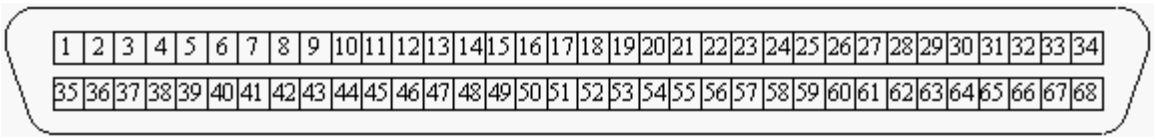


Figure 9. 68-pin connector for the CEI-520/520A Transition Cable

Consult the datasheet provided with each cable for pinout assignments.

Configuring the CEI-220

This section provides configuration information for the CEI-220.

CEI-220 Outline Drawing

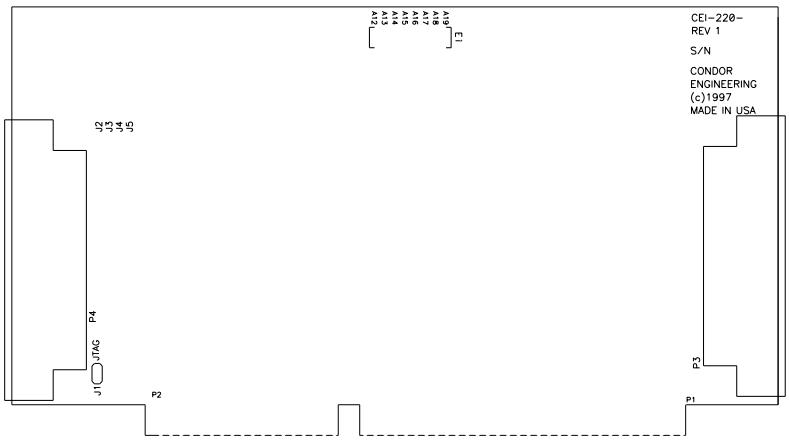


Figure 10. CEI-220 Outline Drawing

CEI-220 Base Memory Address

Jumpers A19-A12 set the base memory address for the CEI-220. An installed jumper selects that address bit as '0'. Table 17 demonstrates the mapping of switch positions to address bits and uses the default address of D000 (hex) as an example. The CEI-220 takes up 4Kb (i.e., 1000 Hex) of upper memory block memory space.

Table 17. Mapping of Switch Positions to Address Bits

Jumper Position	A19	A18	A17	A16	A15	A14	A13	A12
Address Bit	19	18	17	16	15	14	13	12
Jumper Setting	Off	Off	On	Off	On	On	On	On
Base address	1	1	0	1	0	0	0	0
Hex value	D				0			

Other sample settings for jumpers A19 through A12 respectively generate the following base memory addresses:

D800 = Off Off On Off Off On On On
E000 = Off Off Off On On On On On

CEI-220 Interrupts and Slew Rate

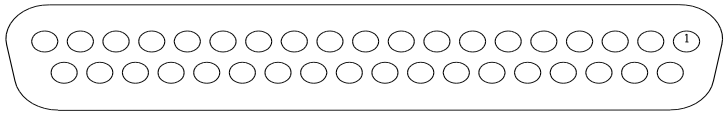
The CEI-220 interrupts are software-controlled. See the "Hardware Interrupt Control" section for information on selecting interrupts.

The CEI-220 slew rate is controlled automatically as a function of the selected bit rate.

CEI-220 ARINC Connector Pin Out

To couple the CEI-220 to ARINC devices, two, 37-pin Subminiature "D" type connectors are provided. The connector pin numbers are listed in Figure 11. The end-panel connector (P3) is female, and the rear connector (P4) is male.

To make an external wrap adapter, use a DB37 male connector and tie pin 1 to 20, 2 to 21, etc. through 18 to 37.



Pin	Signal	Pin	Signal
P3-1	TX1+	P3-20	RX1+
P3-2	TX1-	P3-21	RX1-
P3-3	TX2+	P3-22	RX2+
P3-4	TX2-	P3-23	RX2-
P3-5	TX3+	P3-24	RX3+
P3-6	TX3-	P3-25	RX3-
P3-7	TX4+	P3-26	RX4+
P3-8	TX4-	P3-27	RX4-
P3-9	TX5+ (Note 1)	P3-28	RX5+ (Note 1)
P3-10	TX5- (Note 1)	P3-29	RX5- (Note 1)
P3-11	TX6+ (Note 1)	P3-30	RX6+ (Note 1)
P3-12	TX6- (Note 1)	P3-31	RX6- (Note 1)
P3-13	TX7+ (Note 1)	P3-32	RX7+ (Note 1)
P3-14	TX7- (Note 1)	P3-33	RX7- (Note 1)
P3-15	TX8+	P3-34	RX8+

Pin	Signal	Pin	Signal
P3-16	TX8-	P3-35	RX8-
P3-17	TX9+	P3-36	RX9+
P3-18	TX9-	P3-37	RX9-
P3-19	GND (Note 1)		

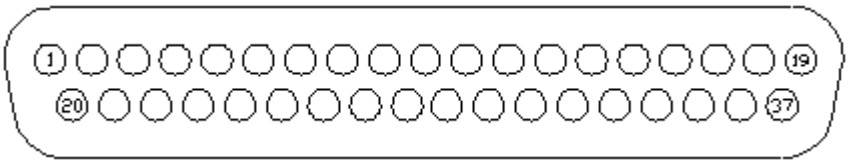
Figure 11. CEI-220 P3 Connector Pinout

Note See Table 18 for ARINC 561/568 configurations.

For configurations with ARINC 561/568 (six-wire) capability, pin connections are made as follows:

Table 18. CEI-220 P3 Connector Change for ARINC 6-wire Configurations

Pin	Signal
P3-9	ARINC 561 transmit data
P3-19	ARINC 561 transmit data ground
P3-11	ARINC 561 transmit clock
P3-19	ARINC 561 transmit clock ground
P3-13	ARINC 561 transmit sync
P3-19	ARINC 561 transmit sync ground
P3-28	ARINC 561 receive data +
P3-29	ARINC 561 receive data -
P3-30	ARINC 561 receive clock +
P3-31	ARINC 561 receive clock -
P3-32	ARINC 561 receive sync +
P3-33	ARINC 561 receive sync -



Pin	Function	Pin	Function
P4-1	TX10+	P4-20	RX10+
P4-2	TX10-	P4-21	RX10-
P4-3	TX11+	P4-22	RX11+
P4-4	TX11-	P4-23	RX11-
P4-5	TX12+	P4-24	RX12+

Pin	Function	Pin	Function
P4-6	TX12-	P4-25	RX12-
P4-7	DISOUT4	P4-26	DISINHI1
P4-8	GND	P4-27	DISINLO1
P4-9	DISOUT3	P4-28	DISINHI2
P4-10	GND	P4-29	DISINLO2
P4-11	DISOUT2	P4-30	DISINHI3
P4-12	GND	P4-31	DISINLO3
P4-13	DISOUT1	P4-32	DISINHI4
P4-14	GND	P4-33	DISINLO4
P4-15	****	P4-34	
P4-16	****	P4-35	
P4-17	****	P4-36	
P4-18	****	P4-37	****
P4-19	****		

**** Factory use only. Connecting these pins may damage the board.

Figure 12. CEI-220 P4 Connector Pinout

Configuring the CEI-420 and CEI-420A

This section provides configuration information for the CEI-420/420A.

CEI-420/420A Outline Drawing

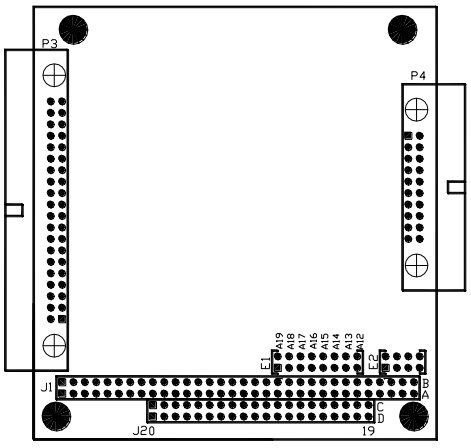


Figure 13. CEI-420/420A Outline Drawing

CEI-420/420A Base Memory Address

Jumpers A19-A12 on header E1 set the base memory address. An installed jumper selects that address bit as '0'. The following table demonstrates the mapping of switch positions to address bits. It uses the default address of D000 (hex) as an example. The CEI-420 takes up 4Kb (i.e., 1000 Hex) of upper memory block memory space.

Table 19. Mapping of Switch Positions to Address Bits

Jumper Position	A19	A18	A17	A16	A15	A14	A13	A12
Address Bit	19	18	17	16	15	14	13	12
Jumper Setting	Off	Off	On	Off	On	On	On	On
Base Address	1	1	0	1	0	0	0	0
Hex Value	D				0			

Other sample settings for jumpers A19 through A12 respectively generate the following base memory addresses:

D800 = Off Off On Off Off On On On
E000 = Off Off Off On On On On On

CEI-420/420A Interrupts and Slew Rate

The interrupts are software-controlled. See chapter 5, "Program Interface Library", for information on selecting interrupts.

The slew rate is automatically controlled as a function of the selected bit rate.

CEI-420/420A Power

For proper operation, the CEI-420/420A requires +12V, -12V and +5V power supplied via the PC/104 connector. The board will not function properly if any of these required voltages are not available.

CEI-420/420A ARINC Connector Pin Out

To couple the CEI-420/420A to ARINC devices use connector P3, a 40-pin IDC-style connector (AMP part number 102160-9). To connect the discrete I/O pins use connector P4, a 20-pin IDC-style connector (AMP part number 102160-4). The connector pin numbers are listed below.

Table 20. CEI-420/420A P3 Front View

39	37	35	33	31	29	27	25	23	21	19	17	15	13	11	9	7	5	3	1
40	38	36	34	32	30	28	26	24	22	20	18	16	14	12	10	8	6	4	2

↓ Board Edge ↓

Pin	Signal	Pin	Signal
1	TX1+	2	TX1-
3	TX2+	4	TX2-
5	TX3+	6	TX3-
7	TX4+	8	TX4-
9	TX5+	10	TX5-
11	TX6+	12	TX6-
13	TX7+	14	TX7-
15	TX8+	16	TX8-
17	RX1+	18	RX1-
19	RX2+	20	RX2-
21	RX3+	22	RX3-
23	RX4+	24	RX4-
25	GND	26	GND
27	RX5+	28	RX5-
29	RX6+	30	RX6-
31	RX7+	32	RX7-
33	RX8+	34	RX8-
35	GND	36	GND
37		38	
39		40	

For external wrap, connect each transmit signal to the corresponding receive signal (e.g., TX1+ to RX1+, TX1- to RX1-, etc). Use the AMP connector, part number 102387-9, with contacts 1-87309-4.

For configurations with ARINC 573/717 capability, pin connections are made as follows:

Table 21. CEI-420/420A P3 Pin Connections

Pin	Signal
P3-37	ARINC 573/717 HBP transmit data +
P3-38	ARINC 573/717 HBP transmit data -
P3-15	ARINC 573/717 BPRZ transmit data +
P3-16	ARINC 573/717 BPRZ transmit data -
P3-33	ARINC 573/717 receive data + (Either BPRZ or HBP)
P3-34	ARINC 573/717 receive data - (Either BPRZ or HBP)

19	17	15	13	11	9	7	5	3	1
20	18	16	14	12	10	8	6	4	2

↓ Board Edge ↓

Figure 14. CEI-420/420A P4 Front View

Table 22. CEI-420/420A P4 Pin Connections

CEI-420A			
PIN	SIGNAL	PIN	SIGNAL
1	DISOUT1/ DISIN9	2	DISOUT2/ DISIN10
3	DISOUT3/ DISIN11	4	DISOUT4/ DISIN12
5	DISOUT5/ DISIN13	6	DISOUT6/ DISIN14
7	DISOUT7/ DISIN15	8	DISOUT8/ DISIN16
9	GND	10	GND
11	DISIN1	12	DISIN2
13	DISIN3	14	DISIN4
15	DISIN5	16	DISIN6
17	DISIN7	18	DISIN8
19	GND	20	GND

CEI-420			
PIN	SIGNAL	PIN	SIGNAL
1	DISOUT1	2	DISOUT2
3	DISOUT3	4	DISOUT4
5	DISOUT5	6	DISOUT6
7	DISOUT7	8	DISOUT8
9	GND	10	GND
11	DISIN1	12	DISIN2
13	DISIN3	14	DISIN4
15	DISIN5	16	DISIN6
17	DISIN7	18	DISIN8
19	GND	20	GND

Note: For external wrap, connect each discrete output to the corresponding discrete input. Use AMP connector p/n 102387-4 with contacts 1-87309-4.

Configuring the CEI-200

This section provides configuration information for the CEI-200. Jumper J1 is factory-set and must *not* be modified.

CEI-200 Outline Drawing

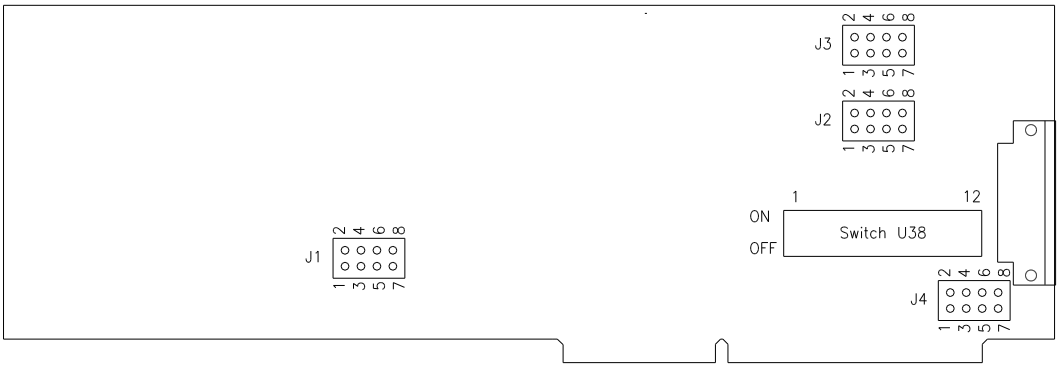


Figure 15. CEI-200 Outline Drawing

CEI-200 Base Memory Address

Switches 1 through 7 set the base memory address for the CEI-200. Switches 1 and 2 must be left in the ON position. They are for future software support. The following table demonstrates mapping of switch positions to address bits. It uses the default address of D000 (hex) as an example. The CEI-200 takes up 32Kb of memory space.

Table 23. CEI-200 Mapping of Switch Positions to Address Bits

Switch Position	1	2	3	4	5	6	7	-	-	-
Address Bit	21	20	19	18	17	16	15	14	13	12
Switch Setting	On	On	Off	Off	On	Off	On	-	-	-
Base address	0	0	1	1	0	1	0	0	0	0
Hex value	0		D				0			

Other sample settings for switches 1 through 7 respectively generate the following base memory addresses:

D800 = On On Off Off On Off Off
E000 = On On Off Off Off On On

CEI-200 Base I/O Address

Switches 8 through 12 set the base I/O address for the CEI-200. The following example demonstrates the default address of 380 (hex).

Table 24. Default Address of 380 (hex) Example

Switch Position	-	-	8	9	10	11	12	-	-	-
Address Bit	9	8	7	6	5	4	3	2	1	0
Switch Setting	-	-	Off	On	On	On	On	-	-	-
Base address	1	1	1	0	0	0	0	0	0	0
Hex value	3		8				0			

Other sample settings for switches 8 through 12 respectively generate the following base I/O addresses:

388 = Off On On On Off
390 = Off On On Off On

CEI-200 Interrupts and Slew Rate

Jumper blocks J2 and J3, located in the upper right corner of the board, are used to set the ARINC transmit data slew rates. With no jumpers in place, slew rates are set for the high (100Kbs) bit rate. To set up a channel for low speed transmission, attach two jumpers for each channel. For example, to set up channel 2 for low speed transmission, attach one jumper on J2 between locations 5-6 and one jumper between locations 7-8.

Table 25. Jumper blocks J2 and J3

J3 Channel

3A	4A	3B	4B
2	4	6	8
1	3	5	7

J2 Channel

1A	1B	2A	2B
2	4	6	8
1	3	5	7

↓ PC bus connector ↓

Jumper block J4 is used for interrupt control. To enable an interrupt, select one of the four-interrupt levels (IRQ3, IRQ4, IRQ5, or IRQ7) and install its corresponding jumper. For example, a jumper between locations 1 and 2 enables interrupts on IRQ3. Interrupts are not required for normal operation of the board.

Table 26. Jumper block J4

J4	Interrupt	IRQ3	IRQ4	IRQ5	IRQ7
		2	4	6	8
		1	3	5	7

↓ PC bus connector ↓

CEI-200 ARINC Connector Pin Out

To couple the CEI-200 to ARINC devices, a 25-pin Subminiature "D" type connector on the end panel is provided. The connector pin numbers are listed in Figure 16 and Table 27.

To build a loop back connector for the external wrap-around test, connect the following three pin groups: (1,2,3), (4,5,6), (7,8,9), (10,11,12), (14,15,16), (17,18,19), (20,21,22), and (23,24,25). This connects receivers 1 and 2 to transmitter 1, receivers 3 and 4 to transmitter 2, etc.

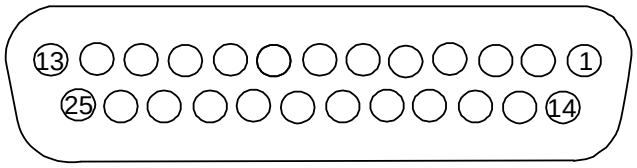


Figure 16. CEI-200 I/O Connector - Front View

Table 27. CEI-200 I/O Connector Pin Assignments

Pin	Function	Pin	Function
1	Receiver 1-A	14	Receiver 1-B
2	Receiver 2-A	15	Receiver 2-B
3	Transmitter 1-A	16	Transmitter 1-B
4	Receiver 3-A	17	Receiver 3-A
5	Receiver 4-A	18	Receiver 4-B
6	Transmitter 2-A	19	Transmitter 2-B
7	Receiver 5-A	20	Receiver 5-B
8	Receiver 6-A	21	Receiver 6-B
9	Transmitter 3-A	22	Transmitter 3-B
10	Receiver 7-A	23	Receiver 7-B

Pin	Function	Pin	Function
11	Receiver 8-A	24	Receiver 8-B
12	Transmitter 4-A	25	Transmitter 4-B
13	Ground		

Configuring the CEI-100

This section provides configuration information for the CEI-100.

CEI-100 Outline Drawing

Figure 17. CEI-100 Outline Drawing

CEI-100 Base Memory Address

Switches 1 through 6 on the dipswitch at U26 set the CEI-100 base address. The following example demonstrates the default address of CC00.

Table 28. Default Address of CC00 Example

Switch Position	-	1	2	3	4	5	6	-
Address Bit	19	18	17	16	15	14	13	12
Switch Setting	-	Off	On	On	Off	Off	On	-
Base address	1	1	0	0	1	1	0	0
Hex value	C				C			

Other sample settings for switches 1 through 6 respectively generate the following base segment addresses:

CE00 = Off On On Off Off Off
D000 = Off On Off On On On
D200 = Off On Off On On Off
E000 = Off Off On On On On

CEI-100 Base I/O Address

Switches 7 through 10 set the base I/O address for the CEI-100. The following example demonstrates the default address of 380 (hex).

Table 29. Default Address of 380 (hex) Example

Switch Position	-	-	-	7	8	9	10	-	-	-
Address Bit	9	8	7	6	5	4	3	2	1	0
Switch Setting	-	-	-	On	On	On	On	-	-	-
Base address	1	1	1	0	0	0	0	0	0	0
Hex value	3			8			0			

Other sample settings for switches 7 through 10 respectively generate the following base I/O addresses:

388 = On On On Off
390 = On On Off On
398 = On On Off Off

CEI-100 Interrupts and Slew rate

Jumper block JP1, located in the lower right corner of the board, sets the host interrupt level and the ARINC transmit data slew rate. With no jumpers in place, there is no host interrupt enabled, and the slew rate is set for the high (100Kbs) bit rate. To enable an interrupt, select one of the three interrupt levels (IRQ3, IRQ4, or IRQ5) and install its jumper. For example, a jumper between locations 1 and 2 enables interrupts on IRQ3. Interrupts are not required for normal operation of the board. To set the correct slew rate for the transmit channel to work at the low (12.5Kbs) bit rate, connect two jumpers – one between locations 7 and 8 and the other between locations 9 and A.

Table 30. Jumper block JP1

JP1

IRQ3	IRQ4	IRQ5	Slew rate	
2	4	6	8	10
1	3	5	7	9

↓ PC bus connector ↓

CEI-100 ARINC Connector Pin Out

To couple the CEI-100 to ARINC devices, use the provided 15-pin Subminiature "D" type connector on the end panel. The connector pin numbers are defined in Figure 18 and Table 31.

To build a loop back connector for the external wrap-around test, connect pins 1,3, and 5 and pins 2, 4, and 6. This connects the output of the transmitter to the input of the two receivers. The test pins are for discrete inputs, which are not currently supported in the firmware.

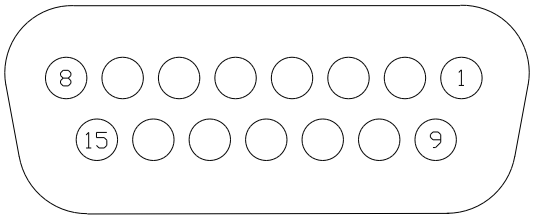


Figure 18. CEI-100 I/O Connector - Front View

Table 31. CEI-100 I/O Connector Pin Assignments

Pin	Function
1	Transmitter 1-A
2	Transmitter 1-B
3	Receiver 1-A
4	Receiver 1-B
5	Receiver 2-A
6	Receiver 2-B
7	Test In
8	Test In Ground
9	Test Out
10	Test Out Ground
11	
12	
13	
14	
15	

CEI-100/CEI-200 Software Installation

The software distributed with each CEI-100/200 board is called CEI-SW and is typically supplied on CD-ROM. A Quick Start, normally distributed with the CD, provides instructions for installation and testing on Windows NT/95/98 and DOS. The file INSTALL.TXT located in the README directory of the CD includes the same installation instructions and a detailed list of all included files.

DOS support is included in source code, libraries, and example executables. Files that are intended for a specific operating system are organized into folders named DOS or WIN32. DOS programs are excellent for troubleshooting because they talk directly to the board without the help of a device driver. A menu-driven program called CONDOR.EXE is included in the DOS examples directory of the CD. It is described in chapter 3, "CEI-100/200 ARINC Test Program".

To quickly verify that the board is connected properly, start the program CONDOR.EXE from DOS by typing (if configured at the default addresses):

```
CONDOR
```

If the board is using different addresses, include the addresses in the command. For example, for I/O address 388, memory address E000, type:

```
CONDOR /M E000 /I 388
```

If the program says "1 CEI-100 (CEI-200) ARINC controller(s) ready" on the status line at the bottom of the display, the board is properly installed and has been found by the software.

CEI-100/CEI-200 Self-test

When a CEI-100/200 board is installed in a new system, it is recommended you perform a self-test to validate the interface.

1. Start the program CONDOR.EXE as defined in the previous section. Command line switches are defined in Chapter 3, "CEI-100/200 ARINC Test Program".
2. Select SETUP MENU on the Main Menu and press **Enter**.
3. Select INITIALIZE/TEST BOARD on the Setup Menu.
4. Select EXECUTE SELF-TEST.
5. Select INTERNAL self-test.
6. Run the test for at least five minutes.

CEI-x20 Software Installation

The software distributed with each CEI-220, 420, 420A, 520, 520A, 620, 820, and 820TX board is called CEI-x20-SW and is typically supplied on CD-ROM. A *Quick Start* is distributed with the CD, providing instructions for installation and testing under Windows NT/XP/Vista/ Server 2008/7/8.0/8.1/Server 2012/10 (note that the only Windows platforms supported by the CEI-220 and CEI-420/420A are Windows NT and 32-bit XP). The file INSTALL.TXT located in the README folder includes the same installation instructions and a detailed list of all included files. Linux installation instructions can be found in the file, Linux_x20_install.txt, in the root directory on the distribution CD.

The software distribution includes a robust Application Programming Interface (API) library for each CEI-x20 device. The CEI-220, -420, and -420A support the *standard* API, while the CEI-520, -520A, -620, -820, and -820TX support the *enhanced* API. The enhanced API is a superset of the standard API. Refer to Chapter 5 for full details.

Under Windows, the standard API (CEI-220/420/420A) is provided as a 32-bit DLL, and both 32-bit and 64-bit DLLs are provided for the enhanced API (CEI-520/520A/620/820/820TX). The 32-bit standard and enhanced API DLLs are named CEI22032.DLL, and the enhanced 64-bit API DLL is named CEI22064.DLL. To interface with the API from a user application, link with one of the provided Microsoft-compatible or Borland-compatible LIB files. Link 32-bit applications with CEI22032.LIB, and link 64-bit applications with CEI22064.LIB.

The CEI-x20 Windows installer automatically determines the correct 32-bit API DLL for the device type being installed and copies it into the 32-bit Windows system folder. If running under 64-bit Windows, the correct 64-bit API DLL is copied into the 64-bit Windows system folder as well.

To determine which 32-bit API DLL is in use under 32-bit or 64-bit Windows, use Windows Explorer to browse to the file CEI22032.DLL in the 32-bit Windows system folder (typically ‘c:\winnt\system32’ or ‘c:\windows\system32’ under 32-bit Windows or ‘c:\windows\syswow64’ under 64-bit Windows). Under the Version or Details tab, the **File Description** field lists the board types supported by the DLL. The product list should include the board type you are trying to access. Similarly, to determine which 64-bit API DLL is in use under 64-bit Windows, inspect the file CEI22064.DLL in the 64-bit Windows system folder (typically ‘c:\windows\system32’ under 64-bit Windows).

Under Linux, the installer places a shared library under the /usr/lib directory. The library depends on the device type installed. Installation of the CEI-220/420 places libcei22032.so under /usr/lib. Installation of the CEI-520/620/820 places libceix2032.so under /usr/lib. Installation of the CEI-820TX places libcei820tx32.so under /usr/lib.

CEI-220/420/420A/520/520A/620/820/820TX Self-test

After installing a CEI-x20 board in a Windows system, you should perform a self-test to validate the interface.

1. Click the Start button. Then select Programs, then Abaco CEI-x20-SW, and then CEI-x20 Test Config.
2. Enter the board number and press **Enter**.
3. Verify that the test passes for each channel.

   The *wrap* test does not support non-ARINC-429 channels. The test fails if the selected board includes an ARINC-573/717 or CSDB channel. This doesn't indicate a problem with either the hardware or the software. See the listing of example programs for an example that exercises non-ARINC-429 channels.

CEI-220 Discrete Inputs

Each of the CEI-220 discrete inputs has a circuit as shown in Figure 19.

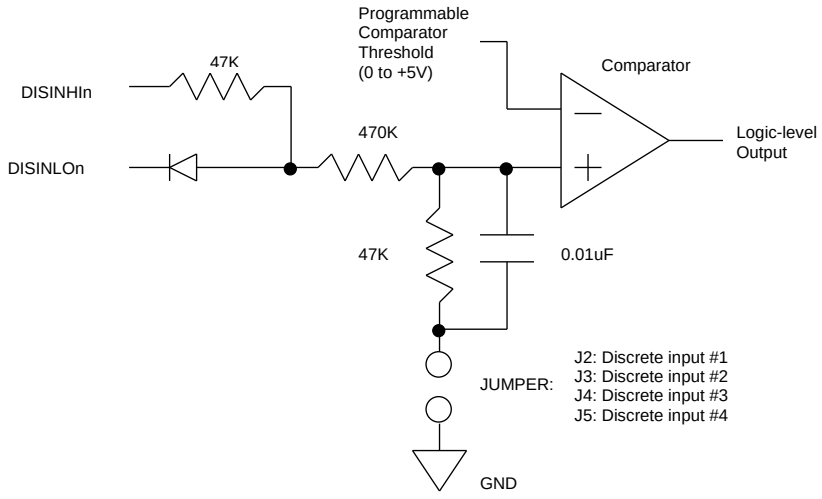


Figure 19. CEI-220 Discrete Inputs Circuit

The CEI-220 has four single-ended, unipolar discrete inputs, which can be configured for several different applications, as described in Table 32. The discrete input specifications are shown in Table 33. To use the discrete inputs, chose your operating mode from the first table, install jumpers as required, and program an input threshold.

All discrete inputs share the same threshold. This doesn't preclude using different operating modes. Program the threshold by passing a value to the AR_SET_CONFIG API routine. Calculate this value by taking the desired

threshold and dividing by the LSB value from the Table 33, for your chosen mode.

As you can see in Table 34, one value may be appropriate for various modes. The logical value of the discrete inputs is returned from the AR_GET_CONFIG API routine. The format is shown in Table 35.

Table 32. Typical Discrete Input Application

Mode	Typical Application
Avionics Open/Low	Input is driven to 0 Volts (Ground). Floats when not driven such as a switch, relay closures with a pole tied to ground, or an open collector driver. This mode handles wider voltage ranges than CMOS/TTL Open Collector Mode. This accommodates for large ground potential differences. The CEI-220 provides a 47K Ω pull-up resistor to the voltage applied to the DISINHIn input.
Avionics Open/High	Input is driven to a supply voltage. Floats when not driven. This includes switches or relays with a pole tied to a supply voltage.
Avionics High/Low	Input is driven between a supply voltage and 0 Volts.
CMOS/TTL	Input is driven by either a TTL or CMOS (5V or 3.3V) driver
CMOS/TTL Open Collector	Either an open collector or open drain logic device drives input. The CEI-220 provides a 47K Ω pull-up resistor to the voltage applied to the DISINHIn input. The maximum threshold is 4.98 Volts.

Table 33. CEI-220 Discrete Input Operating Modes Specification

Discrete Input Mode	Jumper[1] (J2-J5)	DISINHIn Connection	DISINLOn Connection	Typical Accuracy (Percent Full Scale)	Full Scale Threshold (Volts)[2,3,4]	Typical LSB Value (Volts)	Low Pass Filter (Hz)
Avionics Open/Low	Installed	Supply Voltage	Discrete Input	≤ 6%	59.766	60/256	340
Avionics Open/High	Installed	Discrete Input	Open	≤ 6%	59.766	60/256	340
Avionics High/Low	Installed	Discrete Input	Open	≤ 6%	59.766	60/256	340
CMOS/TTL	Not Installed	Discrete Input	Open	≤ 4%	4.980	5/256	None
CMOS/TTL Open Collector	Not Installed	Supply Voltage	Discrete Input	≤ 4%	4.980	5/256	None

Notes:
1. Jumper installed by default.
2. Inputs protected to ±100 Vdc.

3. All Four Inputs Discretes use the same threshold set by the AR_SET_CONFIG API routine.
4. Minimum Threshold value is 0 Volts with a code of 00h.

Table 34. Programming Examples

Signal Type	General	28V Avionics[2]	TTL/CMOS
Threshold (Volts)	V_{TH}	18	1.5
Programmed Value (Hex)	V_{TH}/LSB[1]	4D	4D
Jumper Setting		Installed	Open

Notes:
1. LSB value from Operating Modes Specification table.
2. Default Configuration from the factory.

Table 35. Read Discrete Input (AR_GET_CONFIG) Format

Discrete #	3	2	1	0
Value - condition	0 : $V_{IN}(3) < V_{TH}$ 1 : $V_{IN}(3) > V_{TH}$	0 : $V_{IN}(2) < V_{TH}$ 1 : $V_{IN}(2) > V_{TH}$	0 : $V_{IN}(1) < V_{TH}$ 1 : $V_{IN}(1) > V_{TH}$	0 : $V_{IN}(0) < V_{TH}$ 1 : $V_{IN}(0) > V_{TH}$

Notes:
1. There is ~4mV (jumper not installed) of internal hysteresis for clean switching.
2. V_{TH} Threshold Voltage.
3. $V_{IN}(0)$ Discrete # 0 Input Voltage.

CEI-420/420A/520/520A/620/820 Discrete Inputs

Each of the CEI-420/420A/520/520A/620/820 discrete inputs has a circuit as shown in Figure 20.

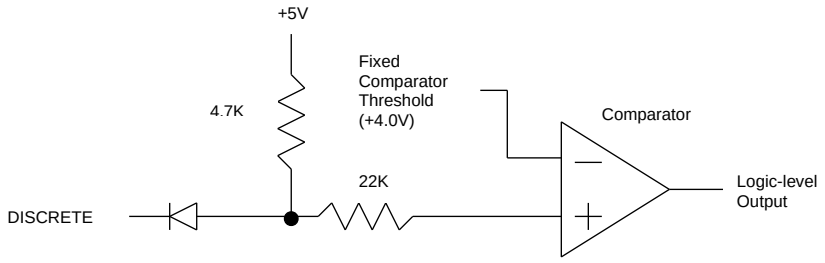


Figure 20. CEI-420/420A/520/520A/620/820 Discrete Input Circuit

The CEI-420 has eight single-ended, unipolar discrete inputs. The CEI-420A has 16 single-ended, unipolar discrete inputs, eight of which share the same lines as the eight discrete outputs. The CEI-520/520A/620 has up to sixteen single-ended, unipolar discrete inputs. The CEI-820 has 16 single-ended, bi-directional, unipolar discretes. Since discretes on the CEI-820 are bi-directional, they may be used as inputs or outputs. For example, you may choose to use discretes 1-4 as inputs and 5-16 as outputs. Only the first 8 discretes on the CEI-820 are available on the front panel (P2) connector, while all 16 discretes are available on the mezzanine (P14) connector. The discrete input specifications are shown in Table 36.

Table 36. Discrete Input Operating Modes Specifications

CEI-420/520/520A/620/820 Discrete Input Specifications	
Input voltage range	-20 volts to +100 volts continuous
Input threshold	Approximately 3.3 V. Because of the 4.7K pull up resistor, the discrete will switch high as long as the discrete input source does not source current near the threshold voltage. This allows for compatibility with TTL type signals.

CEI-220/420/420A/520/520A/620/820 Discrete Outputs

Each of the CEI-220/420/420A/520/520A/620/820 discrete outputs have a circuit as shown in Figure 21.

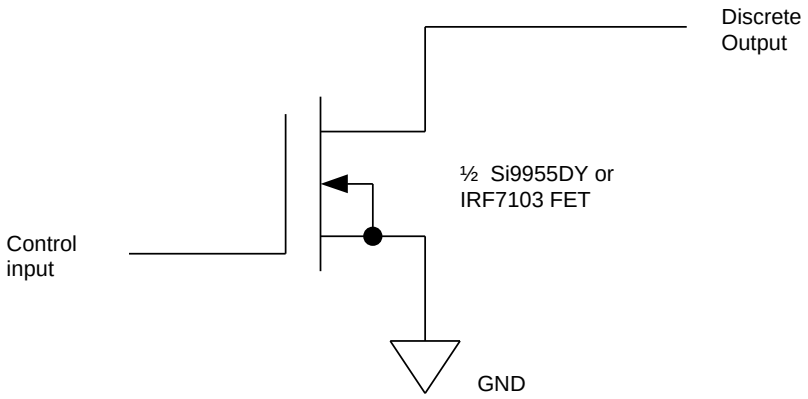


Figure 21. CEI-220/420/420A/520/520A/620/820 Discrete Outputs

Modify the discrete outputs on the CEI-220/420/420A/520/520A/620/820 by using the AR_SET_CONFIG API routine to write a value to their control register. The format is shown in Table 36. Each discrete output consists of Open Drain configured MOSFET switch (Drain connected to DISOUTn) with the gate control by the Discrete Output Control Register.

When the control bit is set to a "0", the discrete output is driven to ground. When the control bit is set to a "1", the discrete output floats. The CEI-220 supports four outputs. The CEI-420/420A support 8 outputs. The CEI-520/520A/620/820 supports up to 16 outputs. The MOSFET specifications are shown in Table 37. The discrete outputs are suitable for driving resistive loads, solenoids, relays, lamps, and other loads not exceeding the specifications in Table 37. See the previous section, "CEI-420/420A/520/520A/620/820 Discrete Inputs", for additional information regarding the CEI-820 discretes.

Table 37. Discrete Output Specifications

Maximum Drain-to-source (Volts)	Maximum On Resistance at 1A (Ohms)	Diode Forward Voltage Drop (Volts)	Continuous Drain Current (Amps)
50	0.20	1.2	1

Note: The CEI-220/420/420A/520/520A/620/820 use a MOSFET to drive the output discrete to ground when turned on.

CHAPTER 3

CEI-100/200 ARINC Test Program

General Information

The menu-driven control program, provided with the CEI-100 and CEI-200 interface, is CONDOR.EXE. It is used to control the channels on one of these boards. It provides a basic level of functionality for test purposes.

This program is NOT available for the CEI-220, CEI-420, CEI-420A, CEI-520, CEI-620, or CEI-820 and can be run only under DOS.

An optional, more-powerful control program called AVIATOR.EXE is also available. It enhances the utility of the CEI-100/200 by expanding your scope of control. It provides additional instrumentation and analytical tools, including simultaneous control of multiple channels on multiple boards, data logging, real-time display of data in engineering units, multiple buffering mechanisms, and powerful triggering logic. Strings of outgoing messages can be generated, repeated, or automatically stepped though a sequence. Strings of incoming messages can be filtered and captured for current or future analysis. A database is provided of the standard BCD and BNR message formats of both the ARINC 429-11 and ARINC 429-12 Specifications for the Mark 33 Digital Information Transfer System (DITS). AVIATOR.EXE also lets you load your own database in place of ARINC Specification data.

The CONDOR.EXE program is menu-driven. The Escape key always exits a submenu. Help can be obtained at any time by using the F1 key. The help file can be viewed using the arrow keys, Page up and Page down keys and the Home/End keys. To exit help use the Escape key.

This program automatically initializes the board and displays the opening screen illustrated in Figure 22.

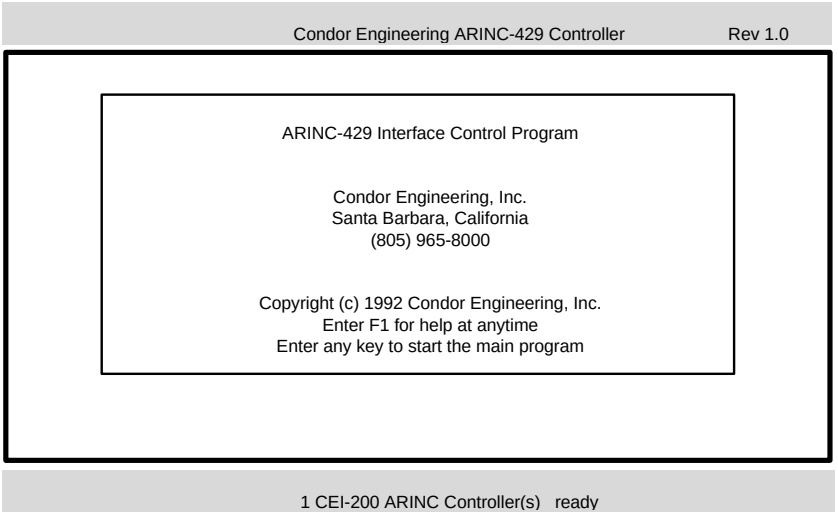


Figure 22. Welcome Screen

Note: If the message at the bottom of the screen indicates that no interface is installed, make sure that the proper dipswitch settings have been selected. If settings other than the default have been selected, make sure that the program was started with appropriate address switch arguments.

Accessing the Program

You can use the following command line switches to when calling the program. They can be used in any order. The default I/O address is 380 (hex). The default segment is CC00 for the CEI-100 and D000 (hex) for the CEI-200. This program automatically configures for the board type found at the selected I/O address. All command line switches and arguments must be separated by at least one space.

/M XXXX	Indicates that the board is installed at a segment address other than the default. It must be followed with a 4-digit HEX value for the segment argument.
/I XXX	Indicates that the board is installed at an I/O control address other than the default. This is a 3-digit HEX value for the address argument.
/X	Indicates that the board has expanded RAM installed (8K bytes on the CEI-100 or 32K bytes on the CEI-200) as opposed to the default.

/SA	Loads the ARINC control program. The main program then exits without entering the menu-driven functions.
/C name	Loads a previously saved configuration of the specified name when the program starts. A name must be specified. All boards defined in the configuration (and only those boards.) are automatically initialized.

Example 1

CONDOR /M D000 /I 388

In this example, the program assumes that the board was configured with the dipswitch set to segment address D000 and the I/O address 388.

Example 2

CONDOR /C AVIATOR.CFG

In this example, the program starts running using the previously saved configuration in file AVIATOR.CFG. This information includes the segment address, I/O address, and RAM size of the board.

Previously saved configuration data can also be entered after starting the program. The Setup menu provides this capability, as does the hot key Alt-R.

Hot Keys

The following keys are short cuts that instantly activate a function when hit during menu selection.

- ALT-R – Restores a configuration from disk.
- ALT-S – Saves current configuration to disk.
- ALT-O – Sets Octal as the default radix.
- ALT-X – Sets HEX as the default radix.
- ALT-D – Spawns a DOS shell (type Exit when done).

Main Menu

The Main Menu appears immediately following the Welcome Screen as shown Figure 23. In addition to a menu, each screen also contains information and message areas that apply to the current function. In the Main Menu screen shown below, you can see that the interface is properly installed because the status line summary shows one controller ready.

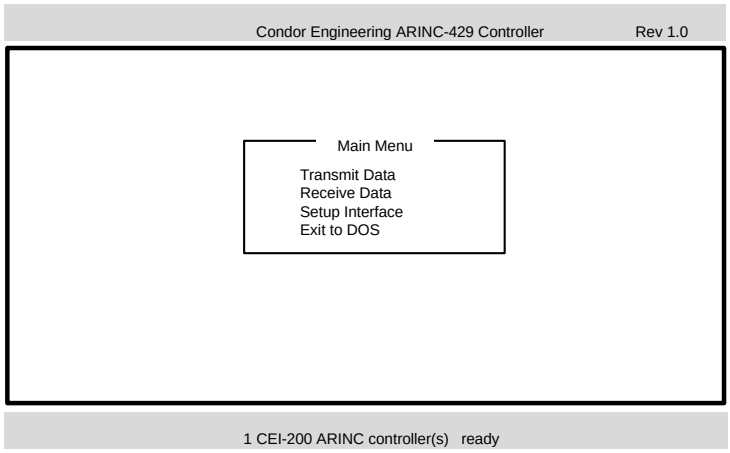


Figure 23. Main Menu

This menu contains three main functions:

- TRANSMIT DATA

  This function sets up and initiates transmissions on the ARINC 429 bus. It can send an ARINC message once or repetitively at a selected interval.

- RECEIVE DATA

  This function sets up reception of data on the ARINC 429 bus. Reception of data can be selectively triggered on label, data, or a combination of the two.

- SETUP INTERFACE

  This function sets up certain functions on the interface, including the ARINC transmit and receive bit rates (100Kbs or 12.5Kbs) and the parity (even or odd). You can save and restore current program settings from this menu. You can select the default radix for most input and output data as well as the self-test function.

- EXIT TO DOS

  This function exits the program and returns to DOS. You can also use the Esc key.

Setup Menu

The menu for setting up the interface is shown in Figure 24. A message at the bottom of the screen on the status line indicates whether a configuration has been loaded at this time.

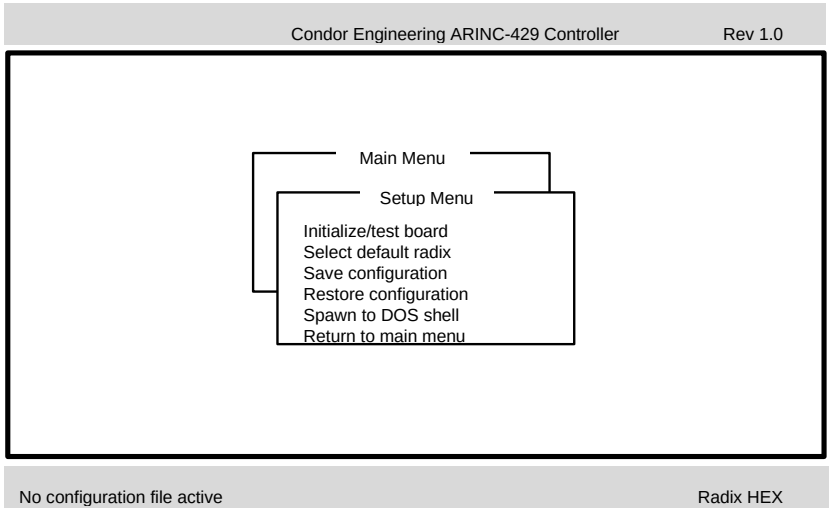


Figure 24. Setup Menu

This menu provides the functions that are used to set up the interface:

- INITIALIZE/TEST BOARD

  This is a sub-function to initialize parameters on the installed board. See "Board Setup" for details.

- SELECT DEFAULT RADIX

  Selects the default radix for most input and output. Valid choices are HEX or OCTAL.

- SAVE CONFIGURATION

  Saves the current configuration, which includes all transmit message data information; receive setup configuration, and all board configuration information, including ARINC setup.

- RESTORE CONFIGURATION

  Reloads a configuration saved via the above function. This can also be done from the command line with the /C switch.

- SPAWN DOS SHELL

  Spawn a DOS shell. Once in the shell, type EXIT to return to the program.

Board Setup

When you select INITIALIZE/TEST BOARD from the Setup Menu the Board Setup menu appears, where you can select additional parameters.

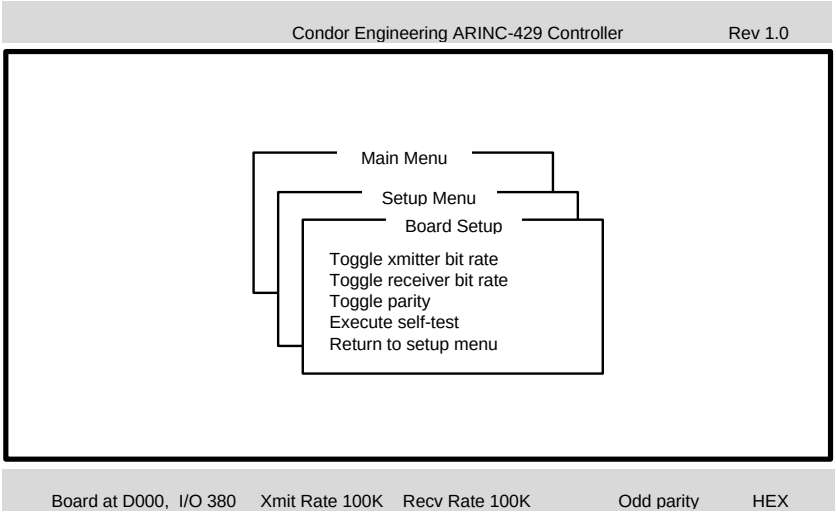


Figure 25. Board Setup Menu

The first three menu items on the Board Setup Menu are toggles for switching the ARINC transmit and receive bit rates between 12.5K and 100K, and toggling the ARINC parity between odd and even. The board shown in Figure 25 is set to operate at transmit and receive bit rates of 100K, using odd parity.

This menu also allows you to run the self-tests described in chapter 2, “Installation”. Execute self-test runs a wrap around test. The board can be tested through an external or an internal loop. The external test uses a wrap-around connector between the transmit and receive channels. The internal wrap around test checks the internal circuitry.

Once started, the test continues to run until you press a key. The tests use the parameters for which the board was configured. The system fails if the transmitter and receiver are configured for different bit rates. The screen in Figure 26 shows that five tests have been completed with no failures. Each test takes less than a minute.

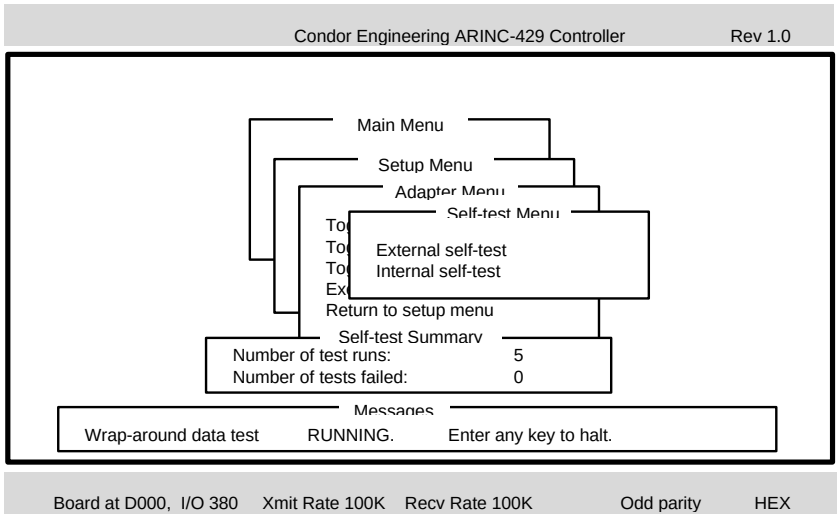


Figure 26. Self-test

Transmit Menu

The Transmit Menu is used to define a message to be transmitted, to select between single or repeated transmission, and to start the transmission process. The menu is shown in Figure 27.

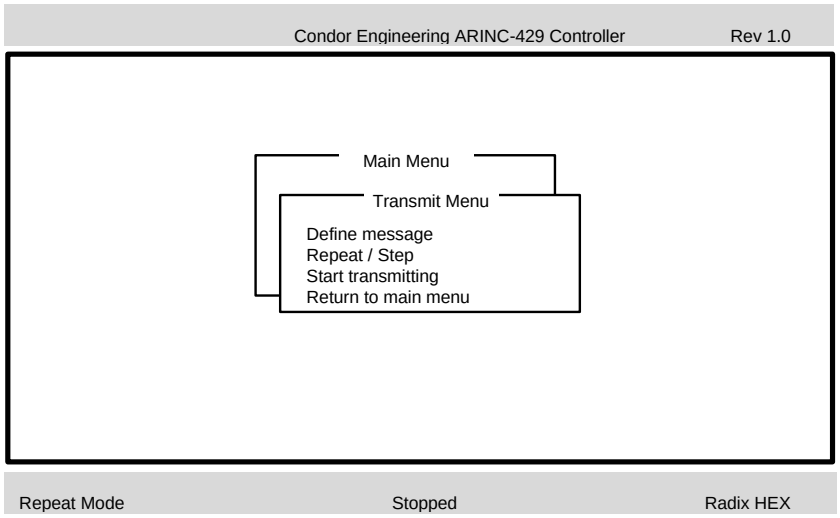


Figure 27. Transmit Menu

- DEFINE MESSAGE

  This function defines an ARINC message and has it transmitted on the 429 bus, once or at repetitive intervals. A message is considered one or more ARINC words (32 bits) of the same label sent out at the same time. A word comprises 8 bits for the label

and 23 bits of data. The remaining bit is parity and is set by the transmitter. See "Defining a Message" for more details.

- REPEAT/STEP

  This function toggles the mode of operation between continuous repetition of the message and a single transmission.

- START TRANSMITTING

  This function initiates the selected mode of transmission. See "Transmitting a Message" for more details.

Defining a Message

To initialize a message:

1. Move the selection-bar to the Define message and press **Enter**.

   The Transmit Message Definition Form (Figure 28) appears.

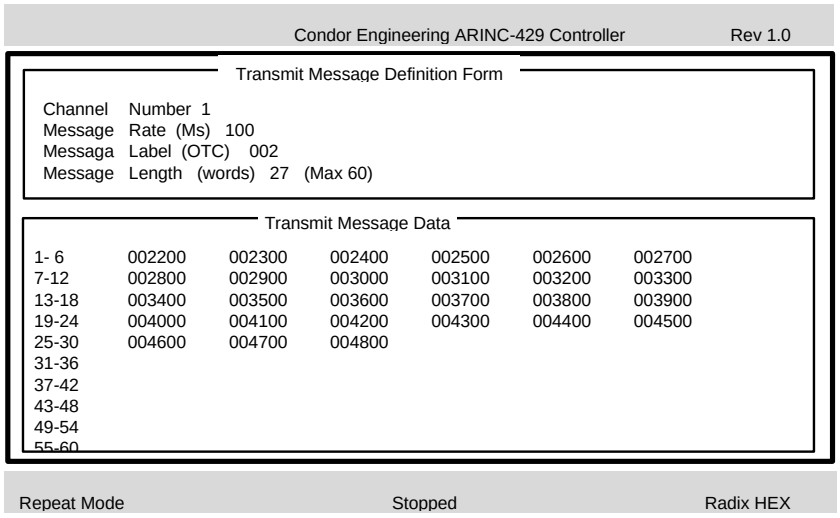


Figure 28. Transmit Message Definition Form

2. Select a valid transmit channel in the range 1 to 4, depending on the board configuration purchased.
3. Initialize the rate to send the message to the nearest one MS. The message rate field is ignored if STEP mode is selected, otherwise it must be defined.
4. Select a label that is attached automatically to all words in the message.
5. Define the message word count (32-bit words).
6. Initialize the message in the Transmit Message Data window, which presents the 23 data bits of each message word for editing. The

message can be sent out repetitively or once only by using the Repeat/Step toggle.

The form shown in Figure 28 shows a message of 27 ARINC words. A message up to 60 words long can be defined.

Transmitting a Message

Once you have defined the desired messages and selected the parameters using the Transmit and Setup Menus, you can initiate message transmission by selecting Start transmitting from the Transmit menu. The screen in Figure 29 appears until transmission is complete or until you press a key.

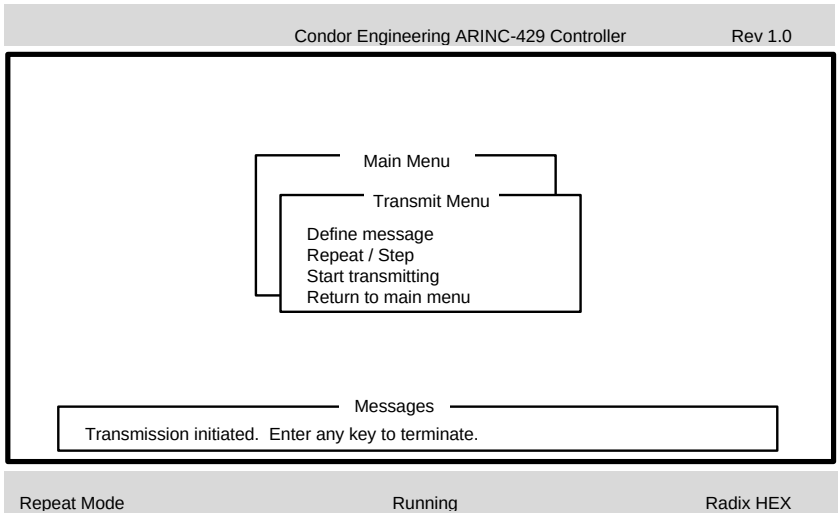


Figure 29. Screen during Transmission

Receive Menu

The receive function allows you to read ARINC data on any channel at the selected bit rate with any requested trigger. The data is stored in a circular buffer for subsequent viewing. Viewing can be in 'Normal' or 'ARINC' format (see below). The data display in the default radix as selected in the Setup menu.

Figure 30 shows the Receive Menu. The messages at the bottom of the screen show that the messages are being received on Channel 1, and appear in HEX format. No triggering has been selected yet, and striking any key can stop processing.

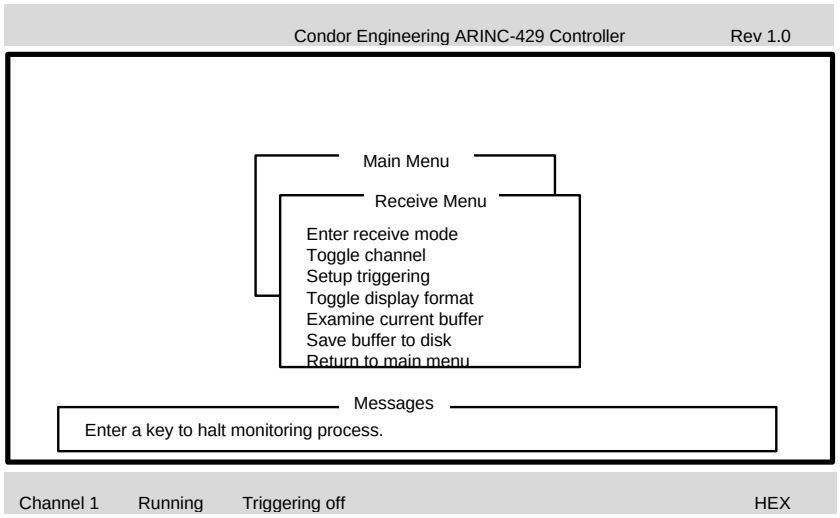


Figure 30. Receive Menu

The following functions are performed using the Receive Menu:

- ENTER RECEIVE MODE

  Starts receiving data on the current channel.

- TOGGLE CHANNEL

  Cycles the current receive channel through the available receive channels.

- SETUP TRIGGERING

  Triggers on an ARINC data condition that is of interest. One trigger condition is provided. The trigger condition can be used to start or stop buffering. See “Triggering” for details.

- TOGGLE DISPLAY FORMAT

  Toggles a data display between 'Normal' format, which includes display fields for label and data, and 'ARINC' format, which includes display fields for label, parity, SSM (Sign/Status Matrix), data and SDI.

- EXAMINE CURRENT BUFFER

  Presents the most recently stored data for examination.

- SAVE BUFFER TO DISK

  Transfers the current buffer to a file. See “Receiving Messages” for details.

Triggering

When you select, Setup triggering, the Triggering setup screen appears (see Figure 31). This screen displays the trigger, broken out bit-by-bit in logic analyzer format. Each bit position must have an X (don't care), a 1 (must be one), or a 0 (must be zero). If the trigger has an X in all bits, it is considered inactive and isn't tested in the triggering logic.

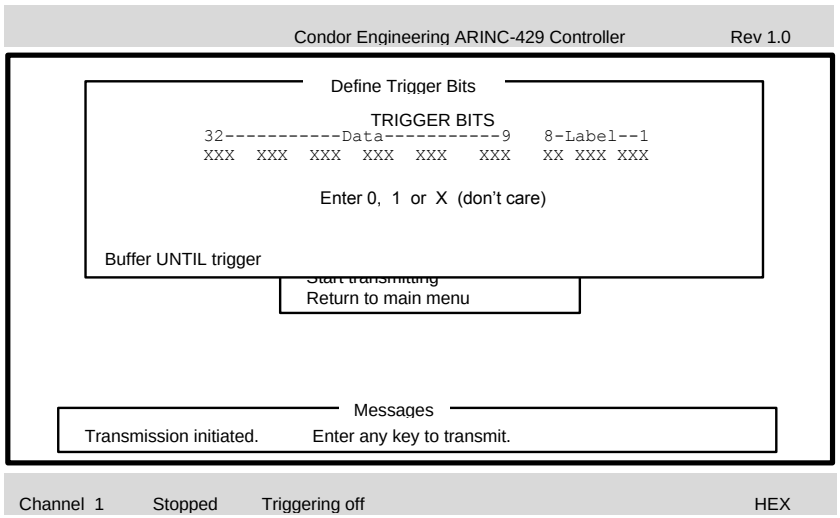


Figure 31. Trigger Bit Definition

Triggering Logic

As each data word is received, it is checked against the trigger condition. If the word meets the condition, the trigger status is considered TRUE, and a trigger has occurred. The trigger condition is met if, for each trigger bit specified as a 1, the corresponding bit in the data word is a 1, and for each trigger bit specified as a 0, the corresponding bit in the data word is a 0. For each bit in the trigger specified as an X, the corresponding bits in the receive data word are not evaluated.

The buffering mechanism is selected along with the trigger logic. Buffer UNTIL causes all words, up to and including the one that causes a trigger, to be buffered. At that point, the buffering is halted. Buffer AFTER causes all words, including and after the trigger, to be buffered. Toggle between the two conditions by pressing the space bar.

Receiving Messages

After the trigger is set up, you can start reception from the Receive Menu. Received messages are stored in a buffer until the trigger bits are detected in a message, or the messages are stored after the trigger bits are detected. You can then examine the buffer and store it to a file from the Receive Menu.

The optional program ANALYZER.EXE provides sophisticated forms of triggering, filtering, and data capture. ANALYZER.EXE also provides expanded capability for data translation and analysis.

CHAPTER 4

BusTools/ARINC Data Bus Analyzer

General Information

BusTools/ARINC™ is an optional ARINC 429 analysis and simulation utility which runs under select versions of Windows. It enhances the utility of an underlying ARINC 429 interface board by expanding your scope of control and by providing additional instrumentation and analytical tools. Additionally, BusTools/ARINC provides support for devices configured with ARINC 561, 573/717, or Commercial Standard Digital Bus (CSDB) channels.

BusTools/ARINC supports usage of up to four boards at the same time or independently and allows simultaneous control of all channels on each board.

Its data logging function streams data to disk or memory and replays it in a time-sequenced display. It provides multiple buffering mechanisms, including real-time display of data in engineering units. Strings of outgoing messages are generated, repeated, or automatically stepped through a sequence. Strings of incoming messages are filtered and captured for current or future analysis. A database of standard ARINC 429 translations is included. Translation among binary, hexadecimal, and engineering units is provided, as is a powerful user-defined label facility.

BusTools/ARINC Demo Software

A free demo version of BusTools/ARINC is available on our web site (at http://www.abaco.com/products/bt-arinc-bustools-software-analyzer). The demo software operates over a simulated ARINC 429 interface board, but is otherwise identical to the full version.

CHAPTER 5

Program Interface Library

Overview

An extensive software Application Programmers Interface (API) to the CEI-100/200/x20 ARINC interface boards is supplied. Functions are supplied to setup the interface and to transmit/receive data.

Data is transmitted by writing it into the transmit buffers. The ARINC interface continuously checks the transmit buffers for data, and when data is found, it is sent as soon as possible. On-board message scheduling is also available on the CEI-200 and all CEI-x20 boards.

When data is received from an input channel, the interface places it in that channel's buffer.

On the CEI-100 board, all three buffers (two receive and one transmit) are 512 ARINC-words in size. On the DTI-1073-4, all three buffers are 128 ARINC words in size.

Table 38 shows the transmitter and receiver buffer sizes for the various configurations of the CEI-200.

Table 38. CEI-200 Buffer Size Table

Number of Channels		16K Memory Configuration		32K Memory Configuration	
Rx	Tx	Rx	Tx	Rx	Tx
2	2	4 Kb	2 Kb	8 Kb	4 Kb
4	2	2 Kb	2 Kb	4 Kb	4 Kb
4	4	2 Kb	1 Kb	4 Kb	2 Kb
6	3	1 Kb	2 Kb	2 Kb	4 Kb
8	4	1 Kb	1 Kb	2 Kb	2 Kb

Table 39 lists the transmitter and receiver buffer sizes for various members of the CEI-x20 family, and many of the other features of these boards.

Table 39. CEI-x20 Family Features

Feature	CEI-220	CEI-420/420A	CEI-520/520A	CEI-620	CEI-820	CEI-820TX
RAM	64 KB	64 KB	512 KB	512 KB	1 MB	2MB
Max Rx/Tx	12/12	8/8	16/16	16/16	8/8	1/32
Rx Buffer per Channel	512 words	512 words	4096 words	4096 words	4096 words	8192 words
Merged Mode Rx Buffer (8 ch)*	2048 ARINC words	2048 ARINC words	16384 ARINC words	16384 ARINC words	16384 ARINC words	4096 words (only 1 rx channel)
Tx Buffer per Channel	512 words	512 words	1024 words	1024 words	1024 words	4096 words
717/573 Tx Buffer Size	512 words	512 words	4096 words	4096 words	4096 words	n/a
Scheduled Mode Buffer	120 Labels	120 Labels	120 Labels	120 Labels	120 Labels	120 Labels
Concurrent Burst and Scheduled Transmit	No	No	Yes	Yes	Yes	Yes
Dedicated Mode Buffer	Shared with Rx	Shared with Rx	Separate buffer	Separate buffer	Separate buffer	Separate buffer
Concurrent Buffered & Dedicated Receive	No	No	Yes	Yes	Yes	Yes
573/717 Frame Mode	No	No	Yes	Yes	Yes	n/a
Label Filtering	SDI, Label	SDI, Label	SDI, Label, eSSM	SDI, Label, eSSM	SDI, Label, eSSM	SDI, Label, eSSM
Block Transfers	No	No	Yes	Yes	Yes	Yes
Interrupt on Rx/Tx or Discretes	No	No	Yes	Yes	Yes	Yes
Interrupt Event Queue	No	No	Yes	Yes	Yes	Yes
Signal Tx on Rx data	No	No	Yes	Yes	Yes	Yes (up to 16 tx channels)

* In Merged mode, each ARINC word takes two words, one for the label and one for the time tag/channel identifier.

A library of utility routines provides the ability to write your own programs to interface with the ARINC boards. These routines are described in this chapter. They are written in C and are delivered in both Microsoft- and Borland-compiler-compatible formats. They can be called from other languages by adhering to the procedures defined in the applicable Borland or Microsoft documentation. Be sure to include the file UTILDEFS.H in all programs that call one or more utility routines, as it provides the function prototypes and the symbolic argument definitions.

DOS Programming

DOS large memory model import libraries are provided as follows:

- A Microsoft Visual C++ 1.52 library is provided for CEI-100 and CEI-200:
  - ARINCL.LIB: CEI-200/100 Microsoft, large.
- DOS is not currently supported by the CEI-x20 distribution, although older versions of the CEI-x20 distribution which contain DOS support for the CEI-220 and CEI-420/CEI-420A are available. Contact the factory for more information.

Windows, VxWorks, and Linux Programming

First, verify that the operating system you are planning to use is supported by your particular board type. To determine which operating systems are supported by each board type, refer to the file README.HTML. This file can be found in the root directory on the distribution CD or at Start → Programs → Abaco CEI-x20-SW → Release Notes. Programming information for each board type is given below.

CEI-100/200 Programming

For the CEI-100/200, a 32-bit Windows dynamic link library (.DLL) is provided. Any code that is linked with this library must define "_WIN32". In 32-bit Windows, all function prototypes are declared "_stdcall". Import libraries can be generated through third party tools for many popular compilers. A Microsoft Visual C++ 6.0 import library is provided with the distribution for convenience:

- CEI200.LIB: Microsoft Import Library.
- CEI200.DLL: Win32 DLL.

CEI-x20 Programming

The software distribution includes a robust Application Programming Interface (API) library for each CEI-x20 device. The CEI-220, -420, and -420A support the *standard* API, while the CEI-520, -520A, -620, -820, and -820TX support the *enhanced* API. The enhanced API is a superset of the standard API.

Under Windows, the standard API (CEI-220/420/420A) is provided as a 32-bit DLL and both 32-bit and 64-bit DLLs are provided for the enhanced API (CEI-520/520A/620/820/820TX). The 32-bit standard and enhanced API DLLs are named CEI22032.DLL, and the enhanced 64-bit API DLL is named CEI22064.DLL. To interface with the API from a user application, link with one of the provided Microsoft-compatible or Borland-compatible LIB files. Link 32-bit applications with CEI22032.LIB, and link 64-bit applications with CEI22064.LIB.

The CEI-x20 Windows installer automatically determines the correct 32-bit API DLL for the device type being installed and copies it into the 32-bit Windows system folder. If running under 64-bit Windows, the correct 64-bit API DLL is copied into the 64-bit Windows system folder as well.

To determine which 32-bit API DLL is in use under 32-bit or 64-bit Windows, use Windows Explorer to browse to the file CEI22032.DLL in the 32-bit Windows system folder (typically 'c:\winnt\system32' or 'c:\windows\system32' under 32-bit Windows or 'c:\windows\syswow64' under 64-bit Windows). Under the Version or Details tab, the **File Description** field lists the board types supported by the DLL. The product list should include the board type you are trying to access. Similarly, to determine which 64-bit API DLL is in use under 64-bit Windows, inspect the file CEI22064.DLL in the 64-bit Windows system folder (typically 'c:\windows\system32' under 64-bit Windows).

CEI-x20 products also support operation under Linux. The Linux installation resides in a separate folder on the distribution CD, and Linux installation instructions can be found in the file, Linux_x20_install.txt, in the root directory on the distribution CD. Under Linux, the installer places a shared library under the /usr/lib directory. The library installed depends on the selected device type. Installation of the CEI-220/420 places libcei22032.so under /usr/lib, installation of the CEI-520/620/820 places libceix2032.so under /usr/lib, and installation of the CEI-820TX places libcei820tx32.so under /usr/lib. All Linux libraries are compiled with the GNU C compiler (gcc).

Link your application to the shared library by including –lcei22032, –lceix2032, or –lcei820tx32 in your Makefile or on the command line. For example:

gcc –o test my_test_program.c –lceix2032

This creates a program called test by compiling and linking the user program my_test_program.c to the CEI-520/520A/620/820 build of the CEI-x20 Application Program Interface library.

VxWorks support for the CEI-x20 boards is described in chapter 6.

Programming the ARINC Interface

Programming the board is simple. For a host application to run, you must first initialize the board and load the firmware. This is done with a call to the subroutine AR_LOADSLV that is described in a following section. Next a host must set the characteristics of each transmit and receive channel with multiple calls to AR_SET_CONFIG (see the enhanced function AR_SETCHPARMS). Channel numbers are zero-based: the first channel is channel zero. Then, a typical host application would call AR_GO to start the interface processing ARINC data. It would then call AR_PUTWORD and AR_GETWORD (see AR_GETBLOCK and AR_PUTBLOCK) to send and receive ARINC data respectively. When communication is complete, the host program may call AR_RESET to suspend the interface processing ARINC data. Subsequently, the host program could again call AR_GO to restart the interface. AR_INIT_SLAVE may be invoked first to reset the slave queue pointers, which flush out any remaining data in the queues.

Unlike the CEI-100/200 boards, all of the boards in the CEI-x20 family support independent programming of each channel's parameters. Further, each channel may be reprogrammed at any time without affecting the programming of any other channel, and without interfering with transmission or reception of data on the other channels.

Refer to the AR_SET_PRELOAD_CONFIG documentation for special programming considerations when using the CEI-x20 API in conjunction with multiple processes and/or threads.

When calling the utility routines, it is important to verify for all routines that return a status value that the returned status indicates success. Otherwise, an important function may have failed and you may not become aware of it until an inappropriate time.

The ARINC 561 software interface on the CEI-220 and the CEI-820 is identical to the ARINC 429 interface. It appears as channel 4 to the software (counting the first channel as channel zero). The transmit configuration register isn't used and only the WRAP bit is used in the receive configuration register.

The ARINC 573/717 software interface on the CEI-420-70J and CEI-420-xxJ is nearly identical to the ARINC 429 interface. It appears as channel 7 to the software (counting the first channel as channel 0). The 573 interface on the CEI-520/520A-xxJ, CEI-620-xxJ, and CEI-820-xxJ appears as the first channel. For all devices equipped with 573/717 support, the receiver and transmitter configuration must be loaded by calling AR_SETCHPARMS before normal operation can begin.

ARINC 429 Data Format

Formatting of ARINC 429 32-bit *words* can be confusing to first-time ARINC users. This section will attempt to answer some common questions about the formatting of ARINC words by the CEI-100/200/x20 boards. See our "ARINC Protocol Tutorial" (available at http://www.abaco.com/download/arinc-protocol-tutorial) for a complete discussion of ARINC word formats.

Label Formatting

The ARINC 429 specification calls for the ARINC *label* bits to be transmitted on the ARINC bus in inverse order (bit 8 first, bit 1 last). The CEI-100/200/x20 hardware works with *labels* that are formatted in the normal bit order. During transmission, it automatically shifts the bits out on the bus in the correct (inverted) order. During reception, it formats and returns the bits in the proper order as well. There is no need for you to invert the *label* before or after transmission.

Transmission Order

ARINC convention numbers the bits of an ARINC word from 1 (LSB) to 32 (MSB). The least significant bit of each byte (except the label) is transmitted first; the label is transmitted ahead of the data in each case. The order of the bits transmitted on the ARINC bus is as follows:

8,7,6,5,4,3,2,1,9,10,11,12,13 … 32.

Most applications work with ARINC words formatted in engineering format; the bits are ordered from left to right, bits 32 … 1, where bit 32 is the most significant bit (parity) and bit 1 is the least significant bit (least significant bit of the label).

The CEI-100/200 boards provide a function (AR_REFORMAT) that converts a transmit ARINC word into transmission format. This function should be used to convert a transmit word before calling AR_PUTWORD to transmit the word.

All of the CEI-x20 boards accept (and produce) ARINC 429 words in engineering format; there is no need to call AR_REFORMAT. In fact, if the CEI-x20 library function AR_REFORMAT is called, it performs no operation; it returns the ARINC word unchanged.

ARINC Parity

The CEI-100/200/x20 hardware also handles the ARINC parity bit in a special way. You don't need to calculate parity for transmission. The hardware automatically calculates parity and overwrites bit 32 (parity) with the calculated parity. Use AR_SET_CONFIG to program the parity to ODD or EVEN for each transmitter. This parity function can be disabled using the routine AR_SET_RAW_MODE.

For received words, the CEI-100/200/x20 hardware calculates the parity. If the parity is ODD, it puts a zero in bit 32. Otherwise, it puts a one there. This is so that the user can easily check the parity bit for error. If the channel is set for ODD parity (normal), then a non-zero bit 32 indicates a parity error. This can be overridden using the routine, AR_SET_RAW_MODE, which can be used to disable parity altogether.

CEI-x20 Interrupt Support

Functions are included in the CEI-x20 API to allow configuration of hardware interrupt generation. The current API does not provide a user application interrupt handler function. If your application uses hardware interrupts, you have to supply an interrupt handler as needed by your operating environment (operating system and host int controller).

To support receiver 'await data' mode, limited internal interrupt handling capability was added to the CEI-x20 API for select boards and operating systems (refer to AR_SET_PRELOAD_CONFIG documentation for details). No user application interrupt handling capability is provided, and calling CEI-x20 interrupt support routines when internal interrupt handling is enabled should be avoided.

Receiver 'Await Data' Mode

A special receive mode ('await data' mode) is provided that allows a user program to wait until data is received on a particular channel. This mode is currently only supported by the CEI-820 under Red Hat Linux Enterprise 4.0 (kernel versions 2.6.17.13, 2.6.18, 2.6.20.14, 2.6.22.2, and 2.6.22.19 only).

When 'await data' mode is enabled for a particular receiver, calls to AR_GETWORD or AR_GETWORDT put the calling thread to sleep if no data word is available in the channel's sequential receive buffer. The sleeping thread is awakened when either (a) waiting is cancelled with a call to AR_CANCEL_DATA_WAIT, or (b) a word is received on the channel in question. If no data word is available in the channel's sequential receive buffer when a sleeping thread is awakened, the call to AR_GETWORD or AR_GETWORDT returns ARS_NODATA. Note that AR_GETWORD and AR_GETWORDT return failure if a thread is already waiting to receive a word on the channel in question (only one waiting thread per receive channel is supported).

The only routines that should be used to read data on a channel that is in 'await data' mode are AR_GETWORD or AR_GETWORDT. Other read routines (AR_GETNEXT, AR_GETNEXTT, AR_GETBLOCK) must not be used unless 'await data' mode is disabled. Only ARINC 429 and 561 receivers support 'await data' mode.

If 'await data' mode is enabled for a particular receiver and a thread is currently asleep (in AR_GETWORD or AR_GETWORDT) waiting to receive a word on the channel in question, then disabling 'await data' mode on that channel also cancels the pending wait.

Note that 'await data' mode cannot be enabled unless internal interrupt handling has also been enabled. Refer to AR_SET_PRELOAD_CONFIG documentation for a discussion of internal interrupt handling limitations.

Utility Routines – Summary

This section lists all of the functions in the API, alphabetically by name. Each of the following API utility routines is detailed in the remaining sections of this chapter.

Note: Use the prototypes in the UTILDEFS.H include file as the ultimate guideline when programming with these routines.

Special note for CEI-100/CEI-200 users only: The API routine prototypes listed in this chapter are specified using platform-independent data types. For example, CEI_UINT32 represents an unsigned 32-bit value and CEI_INT16 represents a signed 16-bit value. These platform-independent data types are not yet defined in the CEI-100/CEI-200 distribution (CEI-SW). When calling functions in the CEI-100/200 API, use appropriate native data types that are equivalent to the platform-independent data types shown in the function prototypes listed in this chapter.

ar_cancel_data_wait() Cancel a pending wait operation.

ar_close()	Disables access to the specified board, and releases host addressing resources.
ar_clr_rx_count()	Zeros out count of received data words.
ar_define_msg()	Defines a message for scheduling by the ARINC board.
ar_execute_bit()	Execute the desired built-in test operation
ar_get_boardname()	Return the name of the board as an ASCII string
ar_get_boardtype()	Return the board type
ar_get_config()	Gets the value of a bit in the control word for the ARINC configuration.
ar_get_error()	Returns a message string associated with a given error status code.
ar_get_label_filter()	Determine the label filter
ar_get_latest()	Gets the latest data from a particular label
ar_get_raw_mode()	Get raw mode (no parity) status
ar_get_rx_count()	Returns count of received data words for a channel.
ar_get_timercnt()	Gets the number of timer ticks that have occurred on interface (16 bits)
ar_get_timercntl()	Gets the number of timer ticks that have occurred on interface (32 bits)
ar_getblock()	Gets multiple ARINC words from a channels receiver buffer (CEI-520/520A/620/820/820TX-enhanced interface only)
ar_getfilter()	Gets the entire ARINC filter buffer for a specified receiver (CEI-520/520A/620/820/820TX-enhanced interface only)
ar_getnext()	Gets the next ARINC word from the interface. Wait if none.
ar_getnextt()	Gets the next ARINC word from the interface with time tag. Wait if no word is available.
ar_getword()	Gets an ARINC word from a receive buffer
ar_getwordt()	Gets an ARINC word from a receive buffer with time tag
ar_go()	Starts the interface processing ARINC data.

ar_init_dual_port()	Initializes all ARINC queue structures, buffers and word counts. Calls ar_reset(), flushes all data words and clears scheduled mode setup
ar_init_slave()	Initializes data structures on host and ARINC interface
ar_int_control()	Enables or disables interrupts from slave
ar_int_set()	Sets the interrupt number used to interrupt the host (CEI-220/420 only)
ar_int_slave()	Interrupt the slave.
ar_label_filter()	Initializes label filter(s)
ar_loadslv()	Loads and initializes an ARINC interface
ar_modify_msg()	Modifies the data in a scheduled message
ar_msg_control()	Selects transmission logic – scheduled or burst mode
ar_num_rchans()	Get the number of receive channels
ar_num_xchans()	Get the number of transmit channels
ar_putblock()	Puts multiple ARINC words into a channels transmit buffer (CEI-520/520A/620/820/820TX-enhanced interface only)
ar_putfilter()	Sets the entire filter buffer for an ARINC receiver channel (CEI-520/520A/620/820/820TX-enhanced interface only)
ar_putword()	Puts an ARINC word in the transmit buffer
ar_putword2x16()	Put 2 16-bit words into the burst transmit queue
ar_recreate_parity()	Recreate the parity bit for a received word
ar_reformat()	Converts output word to transmit format. Only needed for CEI-100/200 boards.
ar_reset()	Stops the interface from processing ARINC data
ar_reset_int()	Resets interrupt received from ARINC board
ar_reset_timercnt()	Resets the counter that registers the number of timer ticks on the interface
ar_set_config()	Sets ARINC configuration information
ar_set_control()	Function removed. See AR_SETCHPARMS() for a superior replacement function.

ar_set_preload_config()	Update a pre-load configuration setting (CEI-520/520A/620/820/820TX-enhanced interface only).
ar_set_raw_mode()	Disables parity function on the interface
ar_set_storage_mode()	Sets the data storage mode on the slave
ar_set_timerrate()	Sets the resolution of the slave receive timer
ar_setchparms()	Sets ARINC channel configuration information (CEI-520/520A/620/820/820TX-enhanced interface only)
ar_setinterrupts()	Sets ARINC channel interrupt mode (CEI-520/520A/620/820/820TX-enhanced interface only)
ar_sleep()	Temporarily suspend execution of the calling thread (CEI-x20 API only).
ar_timetag_control()	Enables or disables time tags on received ARINC data
ar_version()	Gets the API software and board firmware versions as an ASCII string
ar_xmit_sync()	Waits for the transmit queue to be empty

Utility Routines – By Function

This section lists all of the utility routines in the API, by function.

Board and API Initialization

ar_loadslv()	Loads and initializes an ARINC interface.

Board and API Information

ar_get_boardname()	Return the name of the board as an ASCII string.
ar_get_boardtype()	Return the board type.
ar_num_rchans()	Get the number of receive channels.
ar_num_xchans()	Get the number of transmit channels.
ar_version()	Gets the API software and board firmware versions as an ASCII string.

Error Reporting

ar_get_error()	Returns a message string associated with a given error status code.

Tick-Timer and Time Tag Functions

ar_get_timercnt()	Gets the number of timer ticks that have occurred on interface (16 bits).
ar_get_timercntl()	Gets the number of timer ticks that have occurred on interface (32 bits).
ar_reset_timercnt()	Resets the counter that registers the number of timer ticks on the interface.
ar_set_timerrate()	Sets the resolution of the slave receive timer.
ar_timetag_control()	Enables or disables time tags on received ARINC data.

Channel Parameter Definition/Setup

ar_clr_rx_count()	Zeros out count of received data words.
ar_define_msg()	Defines a message for scheduling by the ARINC board.
ar_set_config()	Sets ARINC channel configuration information.
ar_set_preload_config()	Update a pre-load configuration setting (CEI-520/520A/620/820/820TX-enhanced interface only).
ar_setchparms()	Sets ARINC channel configuration information (CEI-520/520A/620/820/820TX-enhanced interface only).
ar_timetag_control()	Enables or disables time tags on received ARINC data.

Channel Parameter Read-Back

ar_get_config()	Gets the value of a bit in the control word for the ARINC configuration.
ar_get_raw_mode()	Get raw mode (no parity) status.
ar_get_rx_count()	Returns count of received data words for a channel.

Mode Control

ar_go()	Starts the interface processing ARINC data.
ar_set_raw_mode()	Disables parity function on the interface.
ar_set_storage_mode()	Sets the data storage mode on the interface.
ar_msg_control()	Selects transmission logic – scheduled or burst mode.

Channel Data Read Functions

ar_get_latest()	Gets the latest data from a particular dedicated mode label.
ar_getblock()	Gets multiple ARINC words from a channels receiver buffer (CEI-520/520A/620/820/820TX-enhanced interface only).
ar_getnext()	Gets the next ARINC word from the sequential buffer. Wait if none.
ar_getnextt()	Gets the next ARINC word from the sequential buffer with time tag. Wait if no word is available.
ar_getword()	Gets an ARINC word from a sequential receive buffer.
ar_getwordt()	Gets an ARINC word from a sequential receive buffer with time tag.

Channel Data Write Functions

ar_define_msg()	Defines a message for scheduled transmission by the board.
ar_modify_msg()	Modifies the data in a scheduled message.
ar_putblock()	Puts multiple ARINC words into a channels transmit buffer (CEI-520/520A/620/820/820TX-enhanced interface only).
ar_putword()	Puts an ARINC word in the transmit buffer.
ar_putword2x16()	Put 2 16-bit words into the burst transmit queue.

Label Filtering Functions

ar_get_label_filter()	Determine the label filter.
ar_getfilter()	Gets the entire ARINC filter buffer for a specified receiver (CEI-520/520A/620/820/820TX-enhanced interface only).
ar_label_filter()	Initializes label filter(s).
ar_putfilter()	Sets the entire filter buffer for an ARINC receiver channel (CEI-520/520A/620/820/820TX-enhanced interface only).

Data Structure Initialization Functions

ar_init_dual_port()	Initializes all ARINC queue structures, buffers and word counts. Calls ar_reset(), flushes all data words and clears scheduled mode setup
ar_init_slave()	Initializes data structures on host and ARINC interface.

Hardware Interrupt Control

ar_int_control()	Enables or disables interrupts from slave.
ar_int_set()	Defines the hardware interrupt number used to interrupt the host (CEI-220/420 only).
ar_int_slave()	Interrupt the slave.

ar_reset_int()	Resets interrupt received from ARINC board.
ar_setinterrupts()	Sets ARINC channel interrupt mode (CEI-520/520A/620/820/820TX-enhanced interface only).

Miscellaneous Functions

ar_cancel_data_wait()	Cancel a pending wait operation.
ar_execute_bit()	Execute the desired built-in test operation.
ar_recreate_parity()	Recreate the parity bit for a received word. Not needed for any CEI-x20 board.
ar_reformat()	Converts output word to transmit format. Only needed for CEI-100/200 boards.
ar_sleep()	Temporarily suspend execution of the calling thread (CEI-x20 API only).

Board and Application Shutdown

ar_xmit_sync()	Waits for the transmit queue to be empty.
ar_reset()	Stops the interface from processing ARINC data.
ar_close()	Disables access to the specified board, and releases host addressing resources.

AR _CANCEL_DATA_WAIT

Syntax

CEI_INT16 ar_cancel_data_wait (CEI_INT16 board, CEI_INT16 channel)

Description

If 'await data' mode is enabled for the given receive channel and a thread is currently asleep (in AR_GETWORD or AR_GETWORDT) waiting to receive a word on the channel in question, then calling this function cancels the pending wait operation and awakens the waiting thread.

If no data word is available in the channel's sequential receive buffer when a sleeping thread is awakened, the call to AR_GETWORD or AR_GETWORDT returns ARS_NODATA.

A call to AR_CANCEL_DATA_WAIT may return before the waiting thread (if any) has fully processed the cancellation request.

Use AR_SET_CONFIG to disable 'await data' mode on a particular receiver. Note that AR_CANCEL_DATA_WAIT does not disable 'await data' mode on the channel in question.

Return Value

ARS_NORMAL	Operation completed successfully.
ARS_INVBOARD	Invalid board number.
ARS_INVARG	Invalid channel index.
ARS_BRDNOTLOAD	Board not initialized.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.
ARS_SIGNAL_FAILED	Failed to signal waiting thread.

Arguments

board	(input) board number.
channel	(input) receive channel.

AR _CLOSE

Syntax

CEI_INT16 ar_close (CEI_INT16 board)

Description

This routine should be called when the user application has finished using the board. This routine stops the on-board processor, releases memory mapping resources allocated by the host, and uninstalls the internal interrupt handler (if any). This must be the last utility subroutine executed, and it should be called before the user application exits back to the operating system.

If multi-process mode is active, this routine does not shut down the board if it is still opened by another process. Refer to AR_SET_PRELOAD_CONFIG documentation for details.

Return Value

ARS_NORMAL	Operation completed successfully.
ARS_INVBOARD	Invalid board number.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.
ARS_CONFIG_MUTEX	Failed to acquire/release the board configuration lock.
ARS_THREAD_WAITING	An 'await data' mode thread is waiting to receive a word - must cancel the wait operation before closing the board.
ARS_INT_ISR	Failed to uninstall internal interrupt handler.

Arguments

board	(input) board number.

AR_CLR_RX_COUNT

Syntax

void ar_clr_rx_count (CEI_INT16 board, CEI_INT16 channel)

Description

This routine resets the count of ARINC data words received by a particular channel to zero. The firmware maintains a count of ARINC data words received over the interface for each channel since the interface was initialized or since the count was reset. To read the current value of the count, see the routine AR_GET_RX_COUNT.

Return Value

None

Arguments

board	(input) board number.
channel	(input) receive-channel number.

AR_DEFINE_MSG

Syntax

CEI_INT16 ar_define_msg (CEI_INT16 board, CEI_INT16 channel, CEI_INT16 rate, CEI_UINT16 start, CEI_INT32 data)

Description

This routine defines a message to the ARINC interface for periodic re-transmission. It is applicable to scheduled transmission mode only (see AR_MSG_CONTROL). It defines a specific data word to go out at a specific rate. Up to 48 individual, one-word messages can be scheduled for re-transmission on each channel (120 for the CEI-x20 series of boards). Once a message has been defined, you can call AR_MODIFY_MSG to modify it. Messages can be defined either before or after the board has been started with AR_GO.

The message rate is defined as a number of clock ticks. The time represented by a clock tick is specified with the routine AR_SET_TIMERRATE. It is recommended that the nominal value be set to 1 MS or larger. Due to performance considerations, it is recommended that messages with the faster rates be defined before the slower ones.

To clear the scheduled mode buffer of all messages, call the function AR_INIT_DUAL_PORT. That function operates on all channels, so it will clear all message buffer settings.

If you have a CEI-520/520A/620/820/820TX, the enhanced API supports SIGNALED MESSAGES. In this mode, a specified ARINC receiver looks for a specified label. The filter can be setup to look for any combination of filtered bits as follows:

	eSSM	SDI	label
Bits:	30, 29, 28	9, 8	7, 6, 5, 4, 3, 2, 1, 0

The values FILTER_SIGNAL and FILTER_CHANNEL are used to signal a specific label previously setup by a call to AR_DEFINE_MSG(), and enabled by a call to AR_MODIFY_MSG(), which is transmitted by the signaled transmit channel as soon as possible after the specified label has been received.

To enable this mode, first call AR_DEFINE_MSG() to create the ARINC word that is to be transmitted when the desired label is received. When you call this function use a negative “rate” to indicate that you are creating a SIGNALED message. Then call AR_PUTFILTER() to define the label that SIGNALS the transmit channel. You may wish to call AR_GETFILTER() to read the current filter buffer, update the filter and call AR_PUTFILTER () to write the result back to the board.

You can change the scheduled mode label that responds to the SIGNAL by calling AR_MODIFY_MSG () with the new "msg_num" and a negative "rate".

Return Value

ARS_INVBOARD	The specified board is a CEI-100. (Supported by the CEI-200/x20 only).
ARS_BRDNOTLOAD	Board not loaded and initialized.
ARS_INVARG	Scheduled transmission mode has not been previously selected or an invalid channel was specified or the maximum number of messages has already been defined.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.
message_num	A return value in the range 0-47 (0-119 for the CEI-x20) indicates success and is the unique message number assigned to this message. Each channel assigns message numbers in this range.

Arguments

board	(input) board number of interest.
channel	(input) transmit channel of interest.
rate	(input) re-transmission rate of the message in clock ticks.
start	(input) initial delay (in ticks) from the current time until the message is transmitted for the first time. Subsequent transmissions of this data are at the rate defined in the previous argument. If you wish to schedule multiple labels for transmission at intervals that are relative to each other, I recommend that you stop all board processing with AR_RESET(), define the messages, then resume ARINC label processing by calling AR_GO().
data	(input) ARINC data to transmit, which must be reformatted into the same bit order as required by AR_PUTWORD.

AR_EXECUTE_BIT

Syntax

CEI_INT16 ar_execute_bit (CEI_INT16 board, CEI_INT16 test_type)

Description

This routine executes a user-specified built-in-test operation. Only the CEI-x20 API supports this function.

Return Value

ARS_NORMAL	Operation completed successfully.
ARS_INVBOARD	Invalid board number.
ARS_BRDNOTLOAD	The board must be launched before executing the requested test.
ARS_INVARG	Invalid test type parameter.
ARS_WRAP_DATA_FAIL	Wrap test invalid data failure occurred.
ARS_WRAP_DROP_FAIL	Wrap test missing data failure occurred.
ARS_FAILURE	The requested built-in-test failed.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

Arguments

board	(input) the board number.
test_type	(input) desired built-in-test operation. See "Supported Built-In Tests" for details.

Supported Built-In Tests

AR_BIT_BASIC_STARTUP	Performs all startup testing executed by the AR_LOADSLV routine (dual-port memory, the on-board processor, and the on-board FPGA are tested). The board is reinitialized during this test.
AR_BIT_PERIODIC	Verifies that the board is running by confirming that the on-board tick timer is being properly incremented. The board must be launched (by calling AR_GO) prior to executing this test.
AR_BIT_INT_LOOPBACK	Performs all start-up testing executed by the AR_BIT_BASIC_STARTUP operation. Then, an internal loopback test of all ARINC 429 channel pairs (that is, TX1->RX1, TX2->RX2, and so on) is executed. Note that partnerless receive/transmit channels in unbalanced configurations (for example, the last 4 transmitters on a 4RX/8TX board) are not included in the test. The board is reinitialized during this test.

AR_BIT_EXT_LOOPBACK	Same as AR_BIT_INT_LOOPBACK, but an external loopback is executed instead. A properly-connected loopback adapter must be externally attached.

Note:

All built-in-test operations (with the exception of AR_BIT_PERIODIC) re-initialize the board, and should therefore only be used at startup. Built-in test operations that re-initialize the board return failure if the board is open in multi-process mode (if supported) and more than one process has loaded the board. Refer to the documentation of AR_SET_PRELOAD_CONFIG for a discussion of multi-process limitations.

All loopback testing operations support balanced ARINC 429 channels only, as described above. All ARINC 573/717, ARINC 561, and CSDB channels are excluded from loopback testing. To test an ARINC 573/717 channel, refer to the sample application tst717.c (included with the CEI-x20 software distribution). To test a CSDB channel, refer to the CSDB API Wrap example. To test an ARINC 561 channel, refer to the CEI-x20 Wrap example.

AR_GET_BOARDNAME

Syntax

CEI_CHAR * ar_get_boardname (CEI_UINT32 board, CEI_CHAR * boardName)

Description

This function is used to decode the board type and return a NULL-terminated ASCII string, suitable for printing that describes the board. Only the CEI-x20 API supports this function, for all CEI-x20 boards.

If boardName is not a NULL pointer, the resulting ASCII string is copied to the buffer that boardName points to, which should be at least 120 characters long.

The function returns a pointer to the NULL-terminated ASCII string describing the board.

Return Value

Pointer to a string describing the specified board.

Arguments

board	(input) the board number.
boardName	(output) address to receive the board description.

AR_GET_BOARDTYPE

Syntax

CEI_INT16 ar_get_boardtype (CEI_INT16 board)

Description

This function is used to decode the board type and return a numeric code that describes the board. Only the CEI-x20 API supports this function, for all CEI-x20 boards.

Return Value

A numeric code describing the specified board. The possible values that this function can return are defined in UTILDEFS.H, in the "Board Type equates" section. The following list was current at the time this section of the manual was last updated:

```
CEI_100         2       /* CEI-100 board is selected HW                */
CEI_200         3       /* CEI-200 board is selected HW                */
CEI_220         4       /* CEI-220 board is selected HW                */
CEI_420         5       /* CEI-420 board is selected HW                */
CEI_520         6       /* CEI-520 board is selected HW                */
CEI_620         10      /* CEI-620 board is selected HW                */
CEI_820         11      /* CEI-820 board is selected HW                */
CEI_420A        13      /* CEI-420A board is selected HW               */
CEI_420A_12     14      /* CEI-420A (12 MHz) board is selected HW      */
CEI_420_70J     15      /* CEI-420-70J (12 MHz) reduced comp HW        */
CEI_820TX       17      /* CEI-820TX board is selected HW              */
CEI_520A        18      /* CEI-520A board is selected HW               */
IS_6WIRE        0x1000  /* CEI-x20 board with 6-wire support           */
IS_717          0x2000  /* CEI-x20 board with -717/573 support         */
IS_CSDB         0x4000  /* CEI-x20 board with CSDB support             */
```

A return value of 0 indicates an unknown board type or an invalid board number.

Arguments

board (input) the board number.

AR_GET_CONFIG

Syntax

CEI_INT32 ar_get_config (CEI_INT16 board, CEI_INT16 item)

Description

This routine reads configuration information about the ARINC interface. Use AR_SET_CONFIG to change device configuration information.

See the ARU_* definitions in the file UTILDEFS.H for the most current list of parameters supported by this function and the most up-to-date description of their function and parameters.

The CEI-x20 series of boards supports independent programming of each channel's parameters (baud rate, parity, etc.), so the configuration items that relate to multiple channels are supported by this function only for compatibility reasons.

Return Value

- If the board number is invalid, ARS_INVBOARD is returned.
- If the board is not initialized, ARS_BRDNOTLOAD is returned.
- If the requested item is invalid, ARS_INVHARCMD is returned.
- If a failure occurred while acquiring/releasing the board lock, ARS_BOARD_MUTEX is returned.
- If the requested item is ARU_Xn_RATE or ARU_Rn_RATE (where "n" is the transmitter number or receiver pair number, CEI-100/200 only), then this routine returns:

AR_HIGH	high rate (100Kbs)
AR_LOW	low rate (12.5Kbs)

- If the requested item is ARU_RnXm_PARITY (where "n" is the receiver pair number and "m" is the transmitter number, CEI-100/200 only), this routine returns:

AR_ODD	odd transmitter parity
AR_EVEN	even transmitter parity

- If the item is ARU_INTERNAL_WRAP, the return value is:

AR_WRAP_ON	internal wrap enabled
AR_WRAP_OFF	internal wrap disabled

Note: For the CEI-100/200, internal wrap mode causes the transmit channel to be internally connected to the two receive channels. Transmit data is received as transmitted on the first channel. It is received as the one's complement on the second channel.

For the CEI-x20 series boards, internal wrap mode causes the transmit channel to be internally connected to the same-numbered receive channel. Transmit data is received as transmitted.

- If the item is ARU_Rn_SDI_FILTER (where "n" is the receiver number 1-8), the return is:

AR_ON	SDI prefilter enabled
AR_OFF	SDI prefilter disabled

  For CEI-x20 boards, use ARU_RX_CHnn_SDI_FILTER instead.
- If SDI pre-filtering is enabled, only data with an SDI value that matches the filter value (see below) is stored into the on-board buffers.
- If the requested item is ARU_Rn_SDI_VALUE (where "n" is the receiver number 1-8), the active SDI filtering value is returned (in the range 0-3). For CEI-x20 boards, use ARU_RX_CHnn_SDI_VALUE instead.

CEI-x20 Parameters

The remaining parameters are supported by the CEI-x20 *only*. If these parameters are used with a CEI-100/200 board, they return an error and perform no operation.

- If the requested CEI-x20 item is ARU_RX_CHnn_BIT_RATE, where nn is the receiver channel (01 - 32), this function returns:

AR_HIGH	100 K baud receiver rate selected
AR_LOW	12.5 K baud receiver rate selected
- If the requested CEI-x20 item is ARU_TX_CHnn_BIT_ RATE, where nn is the transmitter channel (01 - 32), this function returns:

AR_HIGH	100 Kbaud receiver rate selected
AR_LOW	12.5 Kbaud receiver rate selected
- If the requested CEI-x20 item is ARU_RX_CHnn_PARITY, where nn is the receiver channel (01 - 32), this function returns:

AR_ODD	odd receiver parity
AR_OFF	receiver parity off
- If the requested CEI-x20 item is ARU_TX_CHnn_PARITY, where nn is the transmitter channel (01 - 32), this function returns:

AR_ODD	odd transmitter parity
AR_EVEN	even transmitter parity
- If the requested CEI-x20 item is ARU_RX_CHnn_SDI_FILTER, where nn is the receiver channel (01 - 32), this function returns:

AR_ON	SDI filtering is enabled
AR_OFF	SDI filtering is disabled
- If the requested CEI-x20 item is ARU_RX_CHnn_SDI_VALUE, where nn is the receiver channel (01 - 32), this function returns the value of the SDI filter (0-3).
- If the requested CEI-x20 item is ARU_TX_CHnn_SHUT_OFF, where nn is the transmitter channel (01 - 32), this function returns:

AR_ON	Transmitter has been shut off

AR_OFF	Transmitter is configured for normal transmission

- If the requested CEI-x20 item is ARU_DISCRETE_INPUTS, this function returns the current state of the 4/8/16 discrete inputs, as the lower 4/8/16 bits of the value. These discretes are active high sense.
- If the requested CEI-x20 item is ARU_DISCRETE_OUTPUTS, this function returns the current state of the 4/8/16 discrete outputs, as the lower 4/8/16 bits of the value. The discrete output values are set using the AR_SET_CONFIG routine.
- If the requested CEI-x20 item is ARU_TX_CHnn_HB_INJ, where nn is the transmitter channel (01 - 32), this function returns:

AR_ON	Extra transmit data bit is enabled
AR_OFF	Extra transmit data bit is disabled (normal operation)

- If the requested CEI-x20 item is ARU_TX_CHnn_LB_INJ, where nn is the transmitter channel (01 - 32), this function returns:

AR_ON	Transmit one too few data bits is enabled
AR_OFF	Transmit one too few data bits disabled (normal operation)

- If the requested CEI-x20 item is ARU_TX_CHnn_GAP_INJ, where nn is the transmitter channel (01 – 32), this function returns:

AR_ON	Transmit short gap is enabled
AR_OFF	Transmit short gap disabled (normal operation)

- If the requested CEI-x20 item is ARU_RX_CHnn_WAIT_PENDING, where nn is the receiver channel (01 – 32), this function returns:

AR_TRUE	There is currently a pending ‘await data’ mode wait operation on this receiver
AR_FALSE	There is currently no pending ‘await data’ mode wait operation on this receiver

- If the requested CEI-x20 item is ARU_RX_CHnn_AWAIT_DATA, where nn is the receiver channel (01 – 32), this function returns:

AR_ON	Receiver ‘await data’ mode is enabled on this receiver
AR_OFF	Receiver ‘await data’ mode is not enabled on this receiver (default)

The remaining parameters are supported by the CEI-x20 only, and only if the PARAMETRIC option was purchased. If these parameters are used with a CEI-100/200 board, they return an error and perform no operation. If used with a CEI-x20 that doesn’t support the function, no operation is performed.

- If the requested CEI-220/520/520A/620/820 item requests a DAC value, this function returns the raw DAC setting. This setting is converted to a voltage via the following equation:

Vdac = 5.0 Volts * n / 256;

See AR_SET_CONFIG for discussion of CEI-220/520/520A/620/820 DAC settings and formulas.

- The following items reference the CEI-220 DAC values:

ARU_RX1_4_THRESH_VALUE:	Receiver 1-4 threshold.
ARU_RX5_8_THRESH_VALUE:	Receiver 5-8 threshold.
ARU_RX9_10_THRESH_VALUE:	Receiver 9-10 threshold.
ARU_RX11_12_THRESH_VALUE:	Receiver 11-12 threshold.
ARU_INPUT_THRESH_VALUE:	Discrete input threshold (20 * Vdac). Default is 1.56 Volts. Note that this particular value may be set on ANY CEI-220 board, even if the PARAMETRIC option was not purchased.
ARU_TRANSMIT_VOLT_VALUE:	Transmitter output voltage Level: 10 * Vdac. Default: 9.96 Volts.

- The following items reference CEI-520/520A/620/820 DAC values:

  ARU_DAC_VALUE_01 –
  ARU_DAC_VALUE_16

Note: See appendix C, "Parametric Voltage Programming", to determine the DAC register definitions for your board.

- The following items allow you to enable/disable variable transmitter voltage levels on the CEI-520/520A/620 on a per-channel basis:

  ARU_OUTPUT_LEVEL_ADJ_01 –
  ARU_OUTPUT_LEVEL_ADJ_32

- If the requested CEI-220/520/520A/620/820/820TX item is ARU_PARAMETERIC_SUPPORT, this function returns:

AR_ON	Variable parametrics supported
AR_OFF	Variable parametrics not supported

Arguments

board	(input) specifies the board of interest
item	(input) specifies the control function about which to return information. See "CEI-100/200 Items" and "CEI-x20 Items" for details.

CEI-100/200 Items

ARU_X1_RATE	transmit rate for transmitter 1.
ARU_X2_RATE	transmit rate for transmitter 2.
ARU_X3_RATE	transmit rate for transmitter 3.
ARU_X4_RATE	transmit rate for transmitter 4.

ARU_R12_RATE	receive rate for receivers 1 & 2.
ARU_R34_RATE	receive rate for receivers 3 & 4.
ARU_R56_RATE	receive rate for receivers 5 & 6.
ARU_R78_RATE	receive rate for receivers 7 & 8.
ARU_R12X1_PARITY	parity for transmitter 1.
ARU_R34X2_PARITY	parity for transmitter 2.
ARU_R56X3_PARITY	parity for transmitter 3.
ARU_R78X4_PARITY	parity for transmitter 4.
ARU_INTERNAL_WRAP	internal wrap mode (all channels).
ARU_R1_SDI_FILTER	receiver 1 SDI prefilter enable.
ARU_R2_SDI_FILTER	receiver 2 SDI prefilter enable.
ARU_R3_SDI_FILTER	receiver 3 SDI prefilter enable.
ARU_R4_SDI_FILTER	receiver 4 SDI prefilter enable.
ARU_R5_SDI_FILTER	receiver 5 SDI prefilter enable.
ARU_R6_SDI_FILTER	receiver 6 SDI prefilter enable.
ARU_R7_SDI_FILTER	receiver 7 SDI prefilter enable.
ARU_R8_SDI_FILTER	receiver 8 SDI prefilter enable.
ARU_R1_SDI_VALUE	receiver 1 SDI prefilter value.
ARU_R2_SDI_VALUE	receiver 2 SDI prefilter value.
ARU_R3_SDI_VALUE	receiver 3 SDI prefilter value.
ARU_R4_SDI_VALUE	receiver 4 SDI prefilter value.
ARU_R5_SDI_VALUE	receiver 5 SDI prefilter value.
ARU_R6_SDI_VALUE	receiver 6 SDI prefilter value.
ARU_R7_SDI_VALUE	receiver 7 SDI prefilter value.
ARU_R8_SDI_VALUE	receiver 8 SDI prefilter value.

CEI-x20 Items

ARU_RX_CH01_BIT_RATE - ARU_RX_CH32_BIT_RATE	receiver 1 - 32 bit rate.
ARU_TX_CH01_BIT_RATE – ARU_TX_CH32_BIT_RATE	transmitter 1 - 32 bit rate.
ARU_RX_CH01_PARITY – ARU_RX_CH32_PARITY	receiver 1 - 32 parity.
ARU_TX_CH01_PARITY – ARU_TX_CH32_PARITY	transmitter 1 - 32 parity.
ARU_RX_CH01_SDI_FILTER – ARU_RX_CH32_SDI_FILTER	receiver SDI filtering On/Off.

ARU_RX_CH01_SDI_VALUE – ARU_RX_CH32_SDI_VALUE	receiver 1 - 32 SDI value (0-3).
ARU_TX_CH01_SHUT_OFF – ARU_TX_CH32_SHUT_OFF	transmitter 1- 32 shut off status.
ARU_RX1_4_THRESH_VALUE	receiver 1-4 threshold (220).
ARU_RX5_8_THRESH_VALUE	receiver 5-8 threshold (220).
ARU_RX9_10_THRESH_VALUE	receiver 9-10 threshold (220).
ARU_RX11_12_THRESH_VALUE	receiver 11-12 threshold (220).
ARU_INPUT_THRESH_VALUE	discrete onput threshold (220).
ARU_TRANSMIT_VOLT_VALUE	transmitter output level (220).
ARU_PARAMETERIC_SUPPORT	parametrics supported or not.
ARU_DISCRETE_INPUTS	current value of discrete inputs.
ARU_TX_CH01_HB_INJ – ARU_TX_CH32_HB_INJ	high-bit injection on/off.
ARU_TX_CH01_LB_INJ – ARU_TX_CH32_LB_INJ	low-bit injection on/off.
ARU_TX_CH01_GAP_INJ – ARU_TX_CH32_GAP_INJ	transmit short gap enabled/disabled.
CEI-520/520A/620/820 only values:	
ARU_RX_CH01_AWAIT_DATA – ARU_RX_CH32_AWAIT_DATA	is ‘await data’ mode currently enabled.
ARU_RX_CH01_WAIT_PENDING – ARU_RX_CH32_WAIT_PENDING	is there a pending ‘await data’ mode wait operation.
ARU_DAC_VALUE_01 - ARU_DAC_VALUE_16	raw DAC values.
CEI-520/520A/620 only values:	
ARU_OUTPUT_LEVEL_ADJ_01- ARU_OUTPUT_LEVEL_ADJ_32	enable adjustable ARINC transmit voltage levels.

AR_GET_ERROR

Syntax

CEI_CHAR * ar_get_error (CEI_INT16 status);

Description

Many of the ARINC functions return status values, which may indicate an error condition. This function, when passed such an error value, returns a pointer to a message string describing the error.

Look at the file UTILDEFS.H for the current list of possible error codes and explanations.

For the CEI-x20, you can call this function with a status value of ARS_LAST_ERROR to return extended error information if any is available. There is only one extended error buffer per process, so you should retrieve this information immediately after an error has occurred to prevent its contents from being over written. Not all errors generate extended error information. If no extended error information is available, ARS_LAST_ERROR returns the extended error information from the previous error that generated extended error information.

Return Value

A pointer to the error message character string. In 16-bit DOS, this is a FAR pointer (assumes large model).

Arguments

status	(input) status value returned by any of the ARINC utilities.

AR_GET_LABEL_FILTER

Syntax

CEI_INT16 ar_get_label_filter (CEI_INT16 board, CEI_UINT16 label);

Description

This function may be called to determine if a specified label has been filtered out, or if normal reception has been enabled. Label filtering is enabled by the functions AR_LABEL_FILTER, or for the CEI-520/520A/620/820/820TX enhanced API, the function AR_PUTFILTER.

Return Value

The value of the label filter, for all channels, is returned by this function. This value consists of a bit field; the least significant bit is set if the label is filtered out on channel 0, the next bit is set if the label is filtered out on channel 1, etc.

Arguments

board	(input) the board of interest.
label	(input) the label of interest.

AR_GET_LATEST

Syntax

void ar_get_latest (CEI_INT16 board, CEI_INT16 channel, CEI_UINT16 label, CEI_INT32 *data, CEI_CHAR *seqno);

Description

This routine retrieves ARINC data when "Dedicated" data storage mode is in effect (see AR_SET_STORAGE_MODE). It returns the most recently received data for a selected label and channel. It also returns an eight-bit sequence number that indicates the number of 32-bit ARINC words received for that combination of label and channel.

Note: For the CEI-100/200 and CEI-520/520A/620/820/820TX, if no data has been received, this routine returns a data word with zero in all bits except for the label field, which has the requested label value. Because the sequence number is stored in the label field on the hardware, the label is re-inserted by this routine after the sequence number has been extracted from the hardware data. The CEI-220/420/420A returns a data word containing all zeros if no labels have been received.

If you are using the CEI-520/520A/620/820/820TX enhanced API, the "DEDICATED' storage mode is always operational, and this function is enhanced as follows:

- The entry point, AR_GET_LATEST, is used to return the latest value for a label/channel. This function gets data from the "DEDICATED" storage mode Snap Shot buffer 'SnapRBuf'. It returns the most recently received data for a selected label and channel. It also returns the eight-bit sequence number that indicates the number of ARINC words received for that combination of label and channel.
- If the label requested is label 256 (an ARINC label that does not exist) or the value "ARU_ALL_LABELS", this function returns *all* 256 ARINC labels and sequence numbers for the specified channel, in sequence, in successive "data" and "seqno" array elements. This function assumes that the caller has allocated at least 256*4 bytes for "data" and at least 256 bytes for "seqno" when being used in this mode.

Return Value

None

Arguments

board	(input) board number.
channel	(input) receive-channel number.
label	(input) label number.
data	(output) address of the location to receive the 32 bit ARINC word(s). The returned ARINC data is in normal ARINC format.
seqno	(output) address of the location to receive the 8-bit sequence number(s).

AR_GET_RAW_MODE

Syntax

CEI_INT16 ar_get_raw_mode (CEI_INT16 board, CEI_INT16 direction, CEI_INT16 channel);

Description

This function is used to inquire about the parity enable status of a channel.

Return Value

AR_ON if raw-mode is on (parity is disabled), AR_OFF if raw-mode is off (parity is enabled).

Arguments

board	(input) board number.
direction	(input) channel type: ARU_XMIT or ARU_RECV
channel	(input) channel number.

AR_GET_RX_COUNT

Syntax

CEI_UINT32 ar_get_rx_count (CEI_INT16 board, CEI_INT16 channel);

Description

The software maintains a count of the number of ARINC data words received over the interface for each channel since the interface was initialized or since the count was reset (see AR _CLOSE). This routine returns that number.

Note: On the CEI-100 only, the returned value is 32 bits, but the data has only 16-bit accuracy. The most significant 16 bits are not used.

Return Value

Count of ARINC data words.

Arguments

board	(input) board number.
channel	(input) receive-channel number.

AR_GET_TIMERCNT

Syntax

CEI_UINT16 ar_get_timercnt (CEI_INT16 board);

Description

The ARINC interface is programmed to maintain an elapsed time count in "ticks". This routine returns the number of ticks that have occurred since the timer was initialized (see AR_LOADSLV) or since the counter was reset to zero (see AR_RESET_TIMERCNT). This value determines if a specific time interval has passed to help in the timed re-transmission of ARINC data. The resolution of a tick is programmable through the routine AR_SET_TIMERRATE.

Return Value

The number of timer ticks (16 bits wide).

Arguments

board	(input) board number from which to read the timer count.

AR_GET_TIMERCNTL

Syntax

CEI_UINT32 ar_get_timercntl (CEI_INT16 board);

Description

The ARINC interface is programmed to maintain an elapsed time count in "ticks". This routine returns the number of ticks that have occurred since the timer was initialized (see AR_LOADSLV) or since the counter was reset to zero (see AR_RESET_TIMERCNT). This value determines if a specific time interval has passed to help in the timed re-transmission of ARINC data. The resolution of a tick is programmable through the routine AR_SET_TIMERRATE. This routine is identical to AR_GET_TIMERCNT except that it returns 32 bits instead of 16.

Return Value

The number of timer ticks (32 bits wide).

Arguments

board	(input) board number from which to read the timer count.

AR_GETBLOCK

Syntax

CEI_INT32 ar_getblock (CEI_UINT32 board, CEI_UINT32 channel, CEI_INT32 MaxWords, CEI_INT32 Offset717, CEI_INT32 * ActualCount, CEI_INT32 * ARINCData, CEI_INT32 *TimeTag)

Description

This function is used to get multiple ARINC words from a channel's receiver buffer. It is supported on the CEI-520/520A/620/820/820TX, using the enhanced interface only.

All of the available ARINC words (or the specified number) are retrieved from the specified buffer and copied to the caller's buffer. If time tagging is enabled for this channel, the time tags are copied to the "TimeTag" destination. If time tagging isn't enabled, this pointer is not used. If this pointer is NULL, and time tags are enabled, the time tags are retrieved and discarded. ARINC-573/717 data contains a time tag in the upper bits of the data word. This time tag is returned as part of the 'ARINCData' array, *not* in the 'TimeTag' array.

If the buffer is empty, this routine returns ARS_NODATA.

If (ActualCount = MaxWords) when this function returns, there may or may not be more data in the buffer. Subsequent calls are required to determine if more words are available in the buffer.

Return Value

ARS_NODATA	No ARINC data is available.
ARS_GOTDATA	One or more ARINC words were read.
ARS_INVARG	Board not initialized.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

Arguments

board	(input) the board number.
channel	(input) the receive channel number.
MaxWords	(input) the maximum number of 32-bit ARINC words to return. If timetags are enabled this is the number of labels to return; an equal number of timetags are returned if TimeTag is not NULL.
Offset717	(input) beginning offset in frame if this is a 573/717 receiver in Frame mode, ignored otherwise. 'MaxWords' are copied from the receiver buffer to the users buffer, and the 'TimeTag' pointer is ignored.

ActualCount	(output) the number of words returned (only valid when AR_GETBLOCK returns ARS_GOTDATA).
ARINCdata	(output) address to receive the data words (data words are stored only when AR_GETBLOCK returns ARS_GOTDATA).
TimeTag	(output) address to receive the time tags (timetags are stored only when AR_GETBLOCK returns ARS_GOTDATA).

AR_GETFILTER

Syntax

CEI_INT32 ar_getfilter (CEI_UINT32 board, CEI_UINT32 channel, CEI_CHAR * Filter)

Description

Only the CEI-520/520A/620/820/820TX enhanced API supports this function.

This function reads the label interrupt and filter buffer from the board for a specified channel. Each receiver channel has an 8K-byte filter buffer, which is used by the firmware to control storage of received labels and to determine if a particular label is to cause an interrupt. Each element of the filter buffer consists of a bit field defined as follows:

FILTER_SEQUENTIAL	0x10	If CLEAR add label to Sequential receive buffer
FILTER_SNAPSHOT	0x20	If CLEAR add label to Snapshot receiver buffer
FILTER_INTERRUPT	0x40	If SET add entry to interrupt queue & set IRQ
FILTER_SIGNAL	0x80	If SET signal channel to transmit label
FILTER_CHANNEL	0x0F	Transmit channel number (0-15) (mask) to signal

The filter buffer for a single channel is defined as follows:

FilterCtrl[MAX_ESSM][MAX_SDI][MAX_LABEL]

and accessed as

FilterCtrl[eSSM][SDI][label]

where the bits of the ARINC word are split up as follows:

eSSM	SDI	label
30, 29, 28	9, 8	7, 6, 5, 4, 3, 2, 1, 0

An entry is made into the interrupt queue if specified by the filter buffer, even if hardware interrupts are not enabled.

To write the label interrupt and filter buffer, see AR_PUTFILTER.

The CEI-520/520A/620/820/820TX enhanced API supports SIGNALED MESSAGES. The values FILTER_SIGNAL and FILTER_CHANNEL are used to signal a specific label previously setup by a call to AR_DEFINE_MSG(), and enabled by a call to AR_MODIFY_MSG(), to be transmitted by the signaled channel as soon as possible after the specified label has been received. See AR_PUTFILTER () and AR_MODIFY_MSG () for more information.

Return Value

ARS_INVBOARD	Board number invalid.
ARS_BRDNOTLOAD	Board not initialized.
ARS_INVARG	Non-implemented, ARINC 717 channel, CSDB channel or invalid mode.
ARS_NO_HW_SUPRT	Board doesn't support this function.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.
ARS_NORMAL	Successful completion.

Arguments

board	(input) board number.
channel	(input) receive-channel number.
Filter	(output) address to receive filter buffer. Buffer must be at least 8K bytes long.

AR_GETNEXT

Syntax

CEI_INT16 ar_getnext (CEI_INT16 board, CEI_INT16 channel, CEI_INT32 *destination);

Description

This utility tries to get data from an input channel. If none is there, it continues trying for up to 1/4 of a second before giving up.

The system clock is used to time out this function. For the CEI-x20 API under Windows, the Sleep() function is used to avoid tying up the processor while waiting for a label to arrive.

Return Value

ARS_NODATA	No word available.
ARS_GOTDATA	One ARINC word (4 bytes) read.
ARS_INVBOARD	Invalid board number.
ARS_BRDNOTLOAD	Board not initialized.
ARS_INVARG	Invalid parameter value.
ARS_NO_HW_SUPRT	Non-implemented channel.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

Arguments

board	(input) board number of interest.
channel	(input) channel from which to get the data.
destination	(output) address that is to receive the data. The returned ARINC data is always in normal ARINC format.

AR_GETNEXTT

Syntax

CEI_INT16 ar_getnextt (CEI_INT16 board, CEI_INT16 channel, CEI_INT32 *destination, CEI_INT32 *timetag);

Description

This utility tries to get data with time tag from an input channel. If none is there, it continues trying for up to 1/4 of a second before giving up. This routine should only be used when time tags have been enabled by a previous call to AR_TIMETAG_CONTROL. If this precaution is not observed, this function returns two successive labels. If two labels aren't available in the buffer, it times out.

The system clock is used to time out this function. For the CEI-x20 API under Windows, the Sleep() function is used to avoid tying up the processor while waiting for a label to arrive.

Return Value

ARS_NODATA	No word available.
ARS_GOTDATA	One ARINC word with timetag read.
ARS_INVBOARD	Invalid board number.
ARS_BRDNOTLOAD	Board not initialized.
ARS_INVARG	Invalid parameter value.
ARS_NO_HW_SUPRT	Non-implemented channel.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

Arguments

board	(input) board number of interest.
channel	(input) channel from which to get the data.
destination	(output) address in the caller's program that is to receive the data. The returned ARINC data is written at this location. The returned ARINC data is always in normal ARINC format.
timetag	(output) address in the caller's routine that is to receive the timetag associated with the data.

AR_GETWORD

Syntax

CEI_INT16 ar_getword (CEI_INT16 board, CEI_INT16 channel, CEI_INT32 *destination);

Description

This routine gets the next available 32-bit ARINC word from the requested channel's buffer and puts it in the desired destination. If this routine successfully returns a word (status is ARS_GOTDATA), there may or may not be more words in the buffer. This function removes the data from the buffer and writes it in the specified destination.

Subsequent calls are required to determine if more words are in the buffer. When it returns ARS_NODATA, the buffer is empty. This is a circular FIFO buffer that is constantly being maintained by the ARINC interface as data comes in. The data returned into "*destination" depends on the channel type as follows:

ARINC 429 Receiver Buffer Data Format

All ARINC 429 receivers return data in the following format:

Table 40. ARINC 429 Receive buffer

31 - 8	7 - 0
Data (MSB-LSB)	Label (MSB-LSB)

Note: Per ARINC convention, the label is transmitted (MSB-first) followed by the data (LSB-first). The API and the board hardware operate exclusively on engineering-formatted data.

ARINC 573/717 Receiver Buffer Data Format

Table 41. ARINC 573/717 Receive Buffer

31 – 16	15	14	13 - 12	11 - 0
Time Tag inserted by firmware	Sync Word Flag 0 = data 1 = sync	X	Sub Frame	Data (MSB-LSB)

Note: The firmware reads all 32-bits from the FPGA, but stores only the lower 16-bits into the dual-port RAM buffer if the channel is running in ARINC 573/717 mode. The upper 16 bits are set to the current time tag value. The firmware uses the Sync Word Flag and the Sub Frame fields to determine the location in the receive buffer to store the data words. The Sub Frame number recorded by the FPGA and the firmware is one greater than the actual value (0 is recorded as 1, etc.). The *enhanced* API uses the Sync Word fields to arrange the data in memory when in FRAME mode.

CSDB Receiver Buffer Data Format

Table 42. CSDB Receive Buffer

31 - 16	15	14 - 8	7 – 0
X – API returns zeros in this field	PARITY 0=valid 1=invalid If Receiver Parity Detection is disabled for this receiver, this field should be ignored. If Receiver Parity Detection is enabled for this receiver, a zero bit indicates the message was received with valid parity, where a one bit indicates the message was received with invalid parity.	X – API returns zeros in this field	DATA 8 bit data word

Return Value

ARS_NODATA	No word available.
ARS_GOTDATA	One ARINC word (4 bytes) read.
ARS_INVBOARD	Invalid board number.
ARS_BRDNOTLOAD	Board not initialized.
ARS_INVARG	Invalid channel number.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.
ARS_NO_HW_SUPRT	Non-implemented channel.
ARS_THREAD_WAITING	An 'await data' mode thread is already waiting to receive a word.
ARS_WAIT_FAILED	Receiver 'await data' mode is enabled and the wait operation failed.

Arguments

board	(input) board number of interest.
channel	(input) channel from which to get the data.
destination	(output) address that is to receive the data. The returned ARINC data is in normal ARINC format.

AR_GETWORDT

Syntax

CEI_INT16 ar_getwordt (CEI_INT16 board, CEI_INT16 channel, CEI_INT32 *destination, CEI_INT32 *timetag);

Description

This routine gets the next available 32-bit ARINC word from the requested channel's buffer and puts it in the desired destination along with its timetag. It is used only if time tags have been selected by a previous call to AR_TIMETAG_CONTROL.

If it successfully returns data, there may or may not be more data in the buffer. It only means there was at least one ARINC word plus timetag in the buffer. This routine removes it from the buffer and returns it to you. Subsequent calls would be required to determine if more words are in the buffer. If it returns ARS_NODATA, the buffer is empty. This is a circular FIFO buffer that is constantly maintained by the interface board.

See AR_GETWORD for a description of the data formats returned by this function.

Return Value

ARS_NODATA	No word available.
ARS_GOTDATA	Read one ARINC word with timetag.
ARS_INVBOARD	Invalid board number.
ARS_BRDNOTLOAD	Board not initialized.
ARS_INVARG	Invalid channel number.
ARS_NO_HW_SUPRT	Non-implemented channel.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.
ARS_THREAD_WAITING	An 'await data' mode thread is already waiting to receive a word.
ARS_WAIT_FAILED	Receiver 'await data' mode is enabled and the wait operation failed.

Arguments

board	(input) board number of interest.
channel	(input) channel from which to get the data.
destination	(output) address in the caller's program that is to receive the data. The returned ARINC data is written to this location. The returned ARINC data is in normal ARINC format.
timetag	(output) address in the caller's routine that is to receive the timetag associated with the data.

AR_GO

Syntax

CEI_INT16 ar_go (CEI_INT16 board);

Description

Sends a command to the ARINC interface causing it to begin processing ARINC data. Once the interface is started, it goes into an endless loop processing ARINC data until the routine AR_RESET is executed (AR_RESET is called by AR _CLOSE). No ARINC data is received or sent until the board is in the "GO" state achieved by this routine.

This is not completely true for an ARINC 573/717 transmit channel. Since 573/717 is a stream protocol, once the board has been initialized via a call to AR_LOADSLV, the 573/717 channel begins transmitting zeros, at the default baud rate. It continues transmitting zeros until provided with proper transmit data by the application.

Return Value

ARS_NORMAL	Command was successful.
ARS_INVBOARD	Invalid board number.
ARS_BRDNOTLOAD	Board not initialized.
ARS_CHAN_TIMEOUT	Timed out waiting for response from the on-board processor.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.
ARS_WINRTFAIL	Command failed, WinRT error.

Arguments

board (input) number of the board of interest.

AR_INIT_DUAL_PORT

Syntax

CEI_INT16 ar_init_dual_port (CEI_INT16 board);

Description

This function is defined for the CEI-x20 only. It doesn't exist in the CEI-100/200 libraries. This routine first calls AR_RESET. It then initializes all of the all ARINC related queue structures, buffers, and word counts. All data structures are initialized, for all channels. It has the side effect of flushing out the data buffers. If you are using scheduled transmission mode, this routine clears your message setup for all channels.

If the board supports ARINC 573/717, the FRAME mode parameters are not altered.

Return Value

ARS_NORMAL	The interface was properly initialized.
ARS_INVBOARD	A board number out of the maximum range was specified.
ARS_BRDNOTLOAD	The requested board was not loaded first with a call to AR_LOADSLV.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

Arguments

board (input) number of the board of interest

AR_INIT_SLAVE

Syntax

CEI_INT16 ar_init_slave (CEI_INT16 board);

Description

This routine first calls AR_RESET. It then initializes all of the ARINC transmitter and receiver queue head and tail pointers. It has the side effect of flushing out the data buffers. If you are using scheduled transmission mode, this routine *does not* modify your message setup.

Return Value

ARS_NORMAL	The interface was properly initialized.
ARS_INVBOARD	A board number out of the maximum range was specified.
ARS_BRDNOTLOAD	The requested board was not loaded first with a call to AR_LOADSLV.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

Arguments

board (input) board number of interest.

AR_INT_CONTROL

Syntax

CEI_INT16 ar_int_control (CEI_INT16 board, CEI_INT16 channel, CEI_INT16 flag);

Description

This routine enables or disables interrupts from the slave processor. Interrupts can be enabled for any or all of the receive channels. If enabled, the interface interrupts the host upon receiving an ARINC data word on one of the enabled channels. The host software must reset the interrupt from within its interrupt handler using AR_RESET_INT. The host then must poll the enabled receive channels with AR_GETWORD to determine which channels contain data. This routine may be invoked while the interface is running.

For the CEI-100/200, you must select the interrupt number by setting the jumpers on the board before this function can be successfully used.

Call this function to enable the hardware interrupt for CEI-100/200 and CEI-x20 boards. For the CEI-220/420, you must program the interrupt number using the AR_INT_SET function before calling this function. For the enhanced API (CEI-520/620/820), the AR_GETFILTER, AR_PUTFILTER, and AR_SETINTERRUPTS routines provide additional functionality.

This function merely enables interrupts. The current API doesn't provide a user application interrupt handler function. If your application uses hardware interrupts, you have to supply an interrupt handler as needed by your operating environment (operating system and host int controller).

If AR_SET_PRELOAD_CONFIG was used to enable internal interrupt handling, the user application must not call any CEI-x20 interrupt support routines (including this one).

Return Value

ARS_NORMAL	Success.
ARS_INVBOARD	Invalid board number.
ARS_INVARG	Invalid channel index.
ARS_NO_HW_SUPRT	Function not currently supported.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

Arguments

board	(input) board number of interest.
channel	(input) receive channel of interest.
flag	(input) if equal to ARU_ENABLE_INT, enables interrupts for the board/channel combination. Otherwise, the interrupts are disabled.

AR_INT_SET

Syntax

CEI_INT16 ar_int_set (CEI_INT16 board, CEI_INT32 int_num);

Description

This routine is only available for the CEI-220/420. It specifies the interrupt number to be used by the board when interrupting the host. This routine may be invoked while the interface is running.

To change the interrupt number used by the CEI-100/200 boards, you must change the jumpers on the board.

The default interrupt, which is used by the CEI-220/420 if this routine has not been called, is interrupt 5. The CEI-520/520A/620/820/820TX boards are PCI/cPCI/PMC boards; the host computer BIOS or operating system assigns the actual interrupt number used by the board.

Return Value

ARS_NORMAL	Success.
ARS_INVBOARD	An un-initialized or invalid board was selected.
ARS_INVARG	An invalid channel was selected.
ARS_NO_HW_SUPRT	Not a CEI-220/420, or invalid interrupt number.

Arguments

board	(input) board number of interest.
int_num	(input) interrupt number. The CEI-220/420 support interrupts 3, 5, 6, 7, 9, 10, 11, 12, 14, and 15. The CEI-420A supports interrupts 3, 5, 6, 7, and 9.

AR_INT_SLAVE

Syntax

CEI_INT16 ar_int_slave (CEI_INT16 board);

Description

This function is intended to signal a hardware interrupt to the slave. This routine is only a placeholder. It has not been implemented.

Return Value

ARS_INVARG	Non-implemented function.

Arguments

board	(input) board number of interest.

AR_LABEL_FILTER

Syntax

CEI_INT16 ar_label_filter (CEI_INT16 board, CEI_INT16 channel, CEI_UINT16 label, CEI_INT16 action);

Description

The ARINC interface has the ability to filter data by label. This routine selects the labels to be filtered out. If filtering is enabled for a particular label, no received data on the specified channel with that label is stored in the ARINC buffer. Label filtering is disabled for all labels by default. Label filtering changes are effective immediately on completion of this routine.

Return Value

ARS_NORMAL	Success.
ARS_INVBOARD	An un-initialized or invalid board was selected.
ARS_INVARG	Invalid channel, label or action argument.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

Arguments

board	(input) board number of interest.
channel	(input) channel number of interest.
label	(input) label of interest. Valid range is 0-255. Also valid is ARU_ALL_LABELS, which takes the action for all labels.
action	(input) enables or disables filtering for this combination of board/channel/label. Valid values are:

ARU_FILTER_ON	enable filtering.
ARU_FILTER_OFF	disable filtering (default state is to not filter any labels).

AR_LOADSLV

Syntax

CEI_INT16 ar_loadslv (CEI_INT16 board, CEI_UINT32 base_seg, CEI_INT32 base_port, CEI_UINT16 ram_size);

Description

This routine loads the firmware into dual-port RAM (the interface runs it's firmware out of dual-port RAM).

In general, this must be the first utility subroutine executed; only AR_VERSION and AR_SET_PRELOAD_CONFIG may be called before calling AR_LOADSLV.

This routine resets the slave then performs a memory check on dual-port memory. If the memory test fails, the jumper configuration on the board (if applicable) may not match up with current settings. If the memory test passes, the firmware is loaded, but not started.

Upon successful completion of this routine, the following ARINC 429 defaults are in effect:

- Transmit and receive bit rates are 100K.
- Parity ODD.
- SDI and label filters disabled.
- Timer rate 5 milliseconds.

Each successful call to AR_LOADSLV must be matched with a call to AR_CLOSE.

This routine does not re-initialize an already-open board when multi-process mode is active. Refer to AR_SET_PRELOAD_CONFIG documentation for details.

Return Value

ARS_NORMAL	Operation completed successfully.
ARS_WINRTFAIL	Failed to load the board (WinRT error).
ARS_INVBOARD	Invalid board number.
ARS_BADLOAD	Memory test failed (board not loaded).
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.
ARS_CONFIG_MUTEX	Failed to acquire/release the board configuration lock.
ARS_MEMADERR	Failed to map the device memory regions.

ARS_HW_DETECT	Failed to detect supported device.
ARS_MEMWRERR	Error testing dual-port memory.
ARS_BRDNOTLOAD	The i960 failed to start.
ARS_NO_INT_SUPPORT	Internal interrupt handling was requested, but is not supported by the board and/or operating system.
ARS_INT_ISR	Failed to install internal interrupt handler.
ARS_ISR_ALREADY_INST	Internal interrupt handling was requested, but another process has already installed an internal interrupt handler for the board.

Arguments

board	(input) desired board number. The valid range for this argument is 0-5 for CEI-100/200 devices, or 0-15 for CEI-x20 devices. Under Windows, this logical device id value was specified when the software distribution was installed for the device in question.
base_seg	(input) base segment that the board is jumpered to, if applicable (e.g., CC00 is the default for CEI-100 and D000 is the default for CEI-200/220/420). Set this value to 0 under Windows (the base segment value, if used, is automatically read from the registry).
base_port	(input) base port address that the board is jumpered to, if applicable (e.g., 380 (hex) is the default). Set this value to 0 under Windows (the port address value, if used, is automatically read from the registry). For CEI-520/520A-xxJ configurations only: pass the value AR_CEI520J_573_DISABLE to disable 573/717 operations and use all supported channels for ARINC 429 traffic.
ram_size	(input) size in Kbytes of the system memory space, if applicable. The system's size depends on the configuration of the system purchased. The CEI-100 size is either 4 or 8 K and the CEI-200 is either 16 or 32 Kbytes. Valid values are 4, 8, 16, and 32. This value should be set to zero for all CEI-x20 boards.

AR_MODIFY_MSG

Syntax

CEI_INT16 ar_modify_msg (CEI_INT16 board, CEI_INT16 channel, CEI_INT16 msg_num, CEI_INT16 rate, CEI_INT32 data)

Description

This routine modifies the data and transmission rate of a message being scheduled by the ARINC interface. It is applicable only to Scheduled transmission mode (see AR_MSG_CONTROL). It can only be called to modify data for a message that has been defined with a previous call to AR_DEFINE_MSG. Once a call to this subroutine is completed, all subsequent scheduled transmissions will use the rate and data defined here.

To disable a previously scheduled message, call this routine with a value of zero for the rate argument. The message may be re-enabled later by setting the rate argument to a non-zero value. If the new rate is the same as the old rate, the label is re-activated in the same relative timing position, relative to all of the other scheduled mode labels.

If you have a CEI-520/520A/620/820/820TX, the enhanced API supports SIGNALED MESSAGES. In this mode, a specified ARINC receiver looks for a specified label; the filter can be setup to look for any combination of filtered bits as follows:

	eSSM	SDI	label
Bits:	30, 29, 28	9, 8	7, 6, 5, 4, 3, 2, 1, 0

The values FILTER_SIGNAL and FILTER_CHANNEL are used to signal a specific label previously setup by a call to AR_DEFINE_MSG(), and enabled by a call to AR_MODIFY_MSG(), which will be transmitted by the signaled transmit channel as soon as possible after the specified label has been received.

To enable this mode first call AR_DEFINE_MSG() to create the ARINC word that is to be transmitted when the desired label is received. When you call this function use a negative *rate*. Then call AR_PUTFILTER() to define the label that SIGNALS the transmit channel. You may wish to call AR_GETFILTER() to read the current filter buffer, update the filter and call AR_PUTFILTER() to write the result back to the board.

You can change the scheduled mode label that responds to the SIGNAL by calling this function with the new “msg_num” and a negative rate.

Return Value

ARS_INVBOARD	The selected board is a CEI-100 or the board index is invalid.
ARS_INVARG	Scheduled transmission mode has not been previously selected, an invalid channel number has been specified or an invalid message number has been specified.
ARS_NORMAL	Normal successful return.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

Arguments

board	(input) board number of interest.
channel	(input) transmit channel to use.
msg_num	(input) message number to update. This must be the value, which was returned by a previous call to AR_DEFINE_MSG.
rate	(input) update rate for the message in clock ticks.
data	(input) new ARINC data, which must be formatted into the same bit order as required by AR_PUTWORD.

AR_MSG_CONTROL

Syntax

CEI_INT16 ar_msg_control (CEI_INT16 board, CEI_INT16 control)

Description

This routine selects the type of transmission logic used by the ARINC board. Two types of transmission logic are supported by the interface:

- Burst mode (default)
- Scheduled mode.

In Burst mode, the slave processor provides an individual circular buffer for each transmit channel. When data is placed into the transmit buffer by the host computer, it is transmitted immediately by the slave. Data is transmitted as fast as possible until all data in the transmission buffer is sent.

In Scheduled mode, the ARINC interface is responsible for the repetitive transmission of the ARINC data. Once a message is defined to the board and the board is started, the on-board processor handles all message scheduling and transmission, freeing the host computer to perform other processing.

Burst mode is supported by all devices, and scheduled mode is supported by the CEI-200/220/420/420A/520/520A/620/820/820TX.

When using the Enhanced API with a CEI-520/520A/620/820/820TX, each transmitter supports both Scheduled and Burst transmission modes concurrently. No call to AR_MSG_CONTROL is necessary when using the Enhanced API. AR_MSG_CONTROL resets the contents of the scheduled message list for each specified channel when called under the Enhanced API.

To enable scheduled transmission mode in conjunction with a CEI-200/220/420/420A board, this routine must be called. It can be called at any time after AR_LOADSLV and prior to starting the board with AR_GO. It must also be called prior to AR_DEFINE_MSG. The two transmission modes (Scheduled and Burst) are mutually exclusive on the CEI-200/220/420/420A. When switching between modes, the function AR_INIT_DUAL_PORT should be called to clear the previous setup from RAM before starting the board with AR_GO.

Return Value

ARS_INVBOARD	The selected board is a CEI-100 or the board index is invalid.
ARS_INVARG	An invalid argument was used.
ARS_NORMAL	Normal successful return.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

Arguments

board (input) The board number of interest.

control (input) Flag to control transmission mode. Valid values are:

AR_ON	enable Scheduled transmission mode (all channels).
AR_OFF	disable Scheduled transmission mode (all channels).

For the CEI-x20 boards only:

ARU_CH01_BURST_MODE – ARU_CH32_BURST_MODE	Enables Burst Mode for the specified channel.
ARU_CH01_SCHEDULED_MODE – ARU_CH32_SCHEDULED_MODE	Enables Scheduled Mode for the specified channel.

AR_NUM_RCHANS

Syntax

CEI_INT16 ar_num_rchans (CEI_INT16 board)

Description

This function returns the number of receiver channels that are implemented on the specified board. This function is present only in the CEI-x20 API; the CEI-100/200 boards do not support this function.

Return Value

ARS_INVBOARD	Board number out of range.
ARS_BRDNOTLOAD	Board has not been initialized.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

If no error was detected, this function returns the number of receiver channels implemented on the specified board.

Arguments

board	(input) board number.

AR_NUM_XCHANS

Syntax

CEI_INT16 ar_num_xchans (CEI_INT16 board)

Description

This function returns the number of transmitter channels that are implemented on the specified board. This function is present only in the CEI-x20 API; the CEI-100/200 boards do not support this function.

Return Value

ARS_INVBOARD	Board number out of range
ARS_BRDNOTLOAD	Board has not been initialized.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

If no error was detected, this function returns the number of transmitter channels implemented on the specified board.

Arguments

board	(input) board number.

AR_PUTBLOCK

Syntax

CEI_INT32 ar_putblock (CEI_UINT32 board, CEI_UINT32 channel, CEI_INT32 maxWords, CEI_INT32 Offset717, CEI_INT32 * ARINCData, CEI_INT32 * ActualCount)

Description

This function puts multiple ARINC words into a specified channel's transmitter buffer (CEI-520/520A/620/820/820TX enhanced interface only).

The entry point, AR_PUTBLOCK, puts multiple words into a specified channel's sequential transmit buffer. This routine puts multiple ARINC words in the transmit queue for a particular channel. When this routine returns, the data has not necessarily been sent, it has been put only in the buffer. If there is other data in the transmit buffer ahead of it, the new data is transmitted in turn.

Return Value

ARS_NORMAL	Success.
ARS_INVBOARD	Invalid board number.
ARS_BRDNOTLOAD	Board not initialized.
ARS_INVARG	Invalid argument value.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.
ARS_XMITOVRFLO	There was not enough room in the transmit buffer for all of the ARINC words specified. 'ActualCount' contains the number of words actually transferred to the transmit buffer.

Arguments

board	(input) board number.
channel	(input) the channel number.
maxWords	(input) maximum number of 32-bit ARINC words to copy into the transmit buffer.
Offset717	(input) beginning offset in frame if this is a 573/717 transmitter, ignored otherwise. "MaxWords" is copied from the users buffer into the transmitter buffer.
ARINCData	(input) address that supplies the data words.
ActualCount	(output) address that receives the number of words actually transferred.

AR_PUTFILTER

Syntax

CEI_INT32 ar_putfilter (CEI_UINT32 board, CEI_UINT32 channel, CEI_CHAR * Filter)

Description

Only the CEI-520/520A/620/820/820TX enhanced API supports this function.

This function writes the label interrupt and filter buffer to the board. Each element of the filter buffer consists of a bit field defined by the call to AR_GETFILTER.

The CEI-520/520A/620/820/820TX enhanced API supports SIGNALED MESSAGES. The values FILTER_SIGNAL and FILTER_CHANNEL in the filter buffer are used to signal a specific label previously setup by a call to AR_DEFINE_MSG(), and enabled by a call to AR_MODIFY_MSG(), to be transmitted by the signaled channel as soon as possible after the specified label has been received. See AR_GETFILTER() and AR_MODIFY_MSG() for more information.

Return Value

ARS_INVBOARD	Board number invalid.
ARS_BRDNOTLOAD	Board not initialized.
ARS_INVARG	Non-implemented, ARINC 717 channel, CSDB channel or invalid mode.
ARS_NO_HW_SUPRT	Board doesn’t support this function.
ARS_NORMAL	Successful completion.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

Arguments

board	(input) board number.
channel	(input) receive-channel number.
Filter	(input) address to supply filter buffer. Buffer must be at least 8K bytes long.

AR_PUTWORD

Syntax

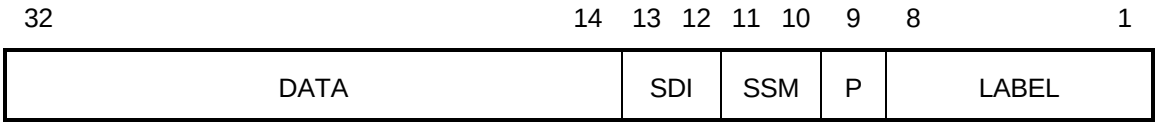
CEI_INT16 ar_putword (CEI_INT16 board, CEI_INT16 channel, CEI_INT32 xmit_data);

Description

This routine puts an ARINC word in the transmit buffer for a particular channel.

ARINC 429 Transmit Data Format

For the CEI-100/200 the ARINC 429 word must be in the transmission format:



To convert a word from standard ARINC format to this format, see AR_REFORMAT. This routine puts the data in an ARINC transmit queue. When this routine returns, the data has not necessarily been put out on the ARINC bus. It has been put only in a buffer. If other data is in the transmit buffer ahead of it, it is transmitted in turn.

The CEI-x20 automatically performs all necessary conversion to and from transmission mode. For this product, this function should be called with a word in standard ARINC format. For the CEI-x20, since no conversion is required, the AR_REFORMAT function performs no operation, it simply returns without modifying the specified ARINC word. This maintains capability with applications written for the CEI-100/200.

ARINC 573/717 Transmit Data Format

The CEI-x20 supports protocols besides ARINC 429. For board channels configured as ARINC 573/717, the transmit word must be formatted by the application as follows:

Table 43. ARINC 573/717 Transmit Buffer

31 - 12	11 - 0
X	Data (MSB-LSB)

Note: The firmware reads 16-bits from dual-port RAM, zero extended (*enhanced* API) or 32-bits (*standard* API), and outputs the entire value to the FPGA, even through only the lower 12 bits contain useful data. The host is responsible for loading the transmit buffer with data at a rate such that the buffer never becomes empty, and must place the sync words in the data stream at the proper places, if a proper 573/717 data stream is to be maintained. The ENHANCED API provides a transmit mode for 573/717 channels called FRAME mode. In this mode the transmit buffer is filled by the host, and then the firmware cycles through the data, transmitting each word in turn. When the end of the buffer is reached the firmware returns to the beginning and transmits the data again and again. The AR_PUTBLOCK function can be used to update sections or the entire transmit buffer.

CSDB Transmit Data Format

For board channels configured as CSDB, the transmit word must be formatted by the application as follows:

Table 44. CSDB Transmit Buffer

31 - 16	15	14	13 – 8	7 - 0
Transmit length (message length - 1 in bits)	PARITY ENABLE 0=disabled 1=enabled (when enabled, the parity of the eight bit data word is calculated and the parity bit is added to the transmission)	PARITY SELECT 0=odd 1=even	Not used	DATA 8 bit data word

Transmit length is a value one less than the overall length of the CSDB message. All transmitted words contain at least a start-bit, eight data-bits, a parity-bit (if enabled) and a stop-bit, regardless of the value of Transmit length. This field is useful for controlling the number of stop-bits in a continuous transmission. It is also useful for programming *gaps* of stop-bits for transmit scheduling. Typical Transmit length values are shown in the following table:

Table 45. Typical CSDB Message Lengths

N-1	Start	Data Bits	Parity	Stop Bits
10	1	8	1	1
11	1	8	1	2
9	1	8	0	1
10	1	8	0	2

Note: Since ARINC data rates are relatively slow, a moderately fast PC can generate data at a much faster rate than data is transmitted.

Return Value

ARS_NORMAL	Success.
ARS_INVBOARD	Invalid board number.
ARS_BRDNOTLOAD	Board not initialized.
ARS_INVARG	Invalid argument value.
ARS_XMITOVRFLO	Failure – transmit buffer overflow. The interface is not transmitting data as fast as data is being put in the buffer. The interface may be in a reset state or else the data rate is too high. Thus, the data wasn't put in the buffer. If the interface is running, you can retry the call until success is achieved.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

Arguments

board	(input) board number of interest.
channel	(input) channel to transmit on.
xmit_data	(input) 32-bit ARINC word to transmit.

AR_PUTWORD2X16

Syntax

CEI_INT16 ar_putword2x16 (CEI_INT16 board, CEI_INT16 channel, CEI_INT16 lsw, CEI_INT16 msw);

Description

This routine is an alternate entry point to AR_PUTWORD that takes an ARINC word as two 16-bit quantities, rather than one 32-bit value.

Return Value

Refer to AR_PUTWORD return values.

Arguments

board	(input) board number of interest.
channel	(input) channel to transmit on.
lsw	(input) lower 16-bits of the ARINC word to transmit.
msw	(input) upper 16-bits of the ARINC word to transmit.

AR_RECREATE_PARITY

Syntax

void ar_recreate_parity (CEI_UCHAR * arinc_word);

Description

This function determines the parity bit (e.g., the MSB of the 32-bit ARINC word) based on the even/odd parity of each of the four bytes that make up the ARINC word. It is normally not needed when using the CEI-x20 boards, since parity is automatically calculated when transmitting a word, and the state of the received parity can be determined by checking the most significant word.

Return Value

None.

Arguments

arinc_word	(input/output) pointer to the 32-bit ARINC word.

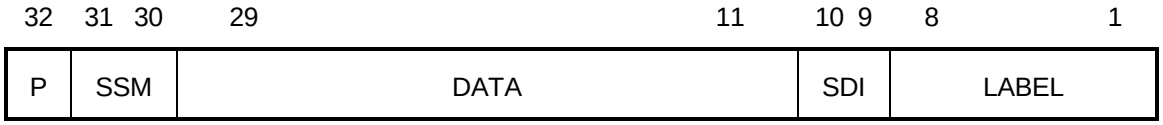
AR_REFORMAT

Syntax

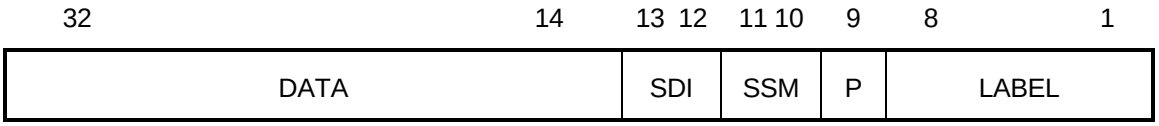
void ar_reformat (void *lsword, void *msword);

Description

Converts an ARINC word to transmission format. It takes data in the conventional ARINC format:



and converts it into transmission format:



There is an alternate method of calling this routine. It can be called with the address of a 32-bit ARINC word as the first argument and a NULL pointer as the second argument (e.g., "0L" works.).

In the CEI-x20 libraries, this function performs no conversion on the specified ARINC word since the CEI-x20 operates exclusively on standard ARINC format words.

Return Value

None. No errors are possible.

Arguments

lsword	(input/output) address of the least significant 16 bits of the ARINC word to reformat.
msword	(input/output) address of the most significant 16 bits of the ARINC word to reformat.

AR_RESET

Syntax

CEI_INT16 ar_reset (CEI_INT16 board);

Description

This function causes the interface board to stop processing ARINC data. Care should be taken not to reset the interface if there is still data in the transmit buffer(s) to be sent.

Use AR_XMIT_SYNC to determine if more ARINC data remains to be transmitted, and wait for it to be loaded into the transmitter.

Return Value

ARS_NORMAL	Command was successful.
ARS_WINRTFAIL	Command failed, WinRT error.
ARS_INVBOARD	Board number out of range.
ARS_INVARG	Board not initialized.
ARS_CHAN_TIMEOUT	Firmware communications failure.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

Arguments

board (input) board number of interest.

AR_RESET_INT

Syntax

CEI_INT16 ar_reset_int (CEI_INT16 board);

Description

This function resets an interrupt from the ARINC interface. This routine should be called from your interrupt handler. It is applicable only to interrupt-driven applications that have met the following conditions:

- Set an interrupt jumper on the interface board (CEI-100/200 only).
- Enabled interrupts with a call to AR_INT_CONTROL.

For the CEI-220/420, the function AR_INT_SET must have been called to specify the interrupt number that is used by the board, since the CEI-220/420 does not have jumper specified interrupts.

Return Value

ARS_NORMAL	Command was successful.
ARS_WINRTFAIL	Command failed, WinRT error.
ARS_INVBOARD	Board number out of range.
ARS_INVARG	Board not initialized.
ARS_CHAN_TIMEOUT	Firmware communications failure.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

Arguments

board (input) board number of interest.

AR_RESET_TIMERCNT

Syntax

void ar_reset_timercnt (CEI_INT16 board);

Description

This routine resets the count of timer ticks (i.e., back to zero) that is kept by the interface. For more information on timers, see AR_SET_TIMERRATE.

Return Value

None.

Arguments

board	(input) board number of interest.

AR_SET_CONFIG

Syntax

CEI_INT16 ar_set_config (CEI_INT16 board, CEI_INT16 item, CEI_UINT32 value);

Description

This routine defines configuration information about the ARINC interface. See AR_GET_CONFIG to read configuration information.

Multiple channel selections are provided for compatibility with the CEI-100/200 only. All members of the CEI-x20 family support individual channel parameters.

Return Value

ARS_NORMAL	Success.
ARS_INVBOARD	Invalid board number.
ARS_BRDNOTLOAD	Board not initialized.
ARS_INVHARVAL	Invalid value argument for item.
ARS_INVHARCMD	Invalid item selection.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

Arguments

board	(input) board number.
item	(input) control function about which to set information:

CEI-100/200 Items

ARU_X1_RATE	transmit rate for transmitter 1.
ARU_X2_RATE	transmit rate for transmitter 2.
ARU_X3_RATE	transmit rate for transmitter 3.
ARU_X4_RATE	transmit rate for transmitter 4.
ARU_R12_RATE	receive rate for receivers 1 & 2.
ARU_R34_RATE	receive rate for receivers 3 & 4.
ARU_R56_RATE	receive rate for receivers 5 & 6.
ARU_R78_RATE	receive rate for receivers 7 & 8.
ARU_R12X1_PARITY	parity for transmitter 1.
ARU_R34X2_PARITY	parity for transmitter 2.
ARU_R56X3_PARITY	parity for transmitter 3.
ARU_R78X4_PARITY	parity for transmitter 4.

ARU_INTERNAL_WRAP	internal wrap mode (all channels).
ARU_R1_SDI_FILTER	receiver 1 SDI prefilter enable.
ARU_R2_SDI_FILTER	receiver 2 SDI prefilter enable.
ARU_R3_SDI_FILTER	receiver 3 SDI prefilter enable.
ARU_R4_SDI_FILTER	receiver 4 SDI prefilter enable.
ARU_R5_SDI_FILTER	receiver 5 SDI prefilter enable.
ARU_R6_SDI_FILTER	receiver 6 SDI prefilter enable.
ARU_R7_SDI_FILTER	receiver 7 SDI prefilter enable.
ARU_R8_SDI_FILTER	receiver 8 SDI prefilter enable.
ARU_R1_SDI_VALUE	receiver 1 SDI prefilter value.
ARU_R2_SDI_VALUE	receiver 2 SDI prefilter value.
ARU_R3_SDI_VALUE	receiver 3 SDI prefilter value.
ARU_R4_SDI_VALUE	receiver 4 SDI prefilter value.
ARU_R5_SDI_VALUE	receiver 5 SDI prefilter value.
ARU_R6_SDI_VALUE	receiver 6 SDI prefilter value.
ARU_R7_SDI_VALUE	receiver 7 SDI prefilter value.
ARU_R8_SDI_VALUE	receiver 8 SDI prefilter value.

CEI-x20 Items

ARU_RX_CH01_BIT_RATE – ARU_RX_CH32_BIT_RATE	receiver 1 - 32 bit rate.
ARU_TX_CH01_BIT_RATE – ARU_TX_CH32_BIT_RATE	transmitter 1 - 32 bit rate.
ARU_RX_CH01_PARITY – ARU_RX_CH32_PARITY	receiver 1 - 32 parity.
ARU_TX_CH01_PARITY – ARU_TX_CH32_PARITY	transmitter 1 - 32 parity.
ARU_RX_CH01_SDI_FILTER – ARU_RX_CH32_SDI_FILTER	receiver SDI filtering on/off.
ARU_RX_CH01_SDI_VALUE – ARU_RX_CH32_SDI_VALUE	receiver 1 - 32 SDI value (0-3).
ARU_TX_CH01_SHUT_OFF – ARU_TX_CH32_SHUT_OFF	transmitter 1- 32 shut off status.
ARU_RX1_4_THRESH_VALUE	receiver 1-4 threshold (220 only).
ARU_RX5_8_THRESH_VALUE	receiver 5-8 threshold (220 only).
ARU_RX9_10_THRESH_VALUE	receiver 9-10 threshold (220 only).
ARU_RX11_12_THRESH_VALUE	receiver 11-12 threshold (220 only).
ARU_INPUT_THRESH_VALUE	discrete input threshold (220 only).
ARU_TRANSMIT_VOLT_VALUE	transmitter output level (220 only).

ARU_DAC_VALUE_01 – ARU_DAC_VALUE_16	set DAC value (non-220/420 only).
ARU_OUTPUT_LEVEL_ADJ_01 – ARU_OUTPUT_LEVEL_ADJ_32	enable/disable variable transmitter voltage (520/520A/620 only).
ARU_DISCRETE_OUTPUTS	set value of discrete outputs.
ARU_TX_CH01_HB_INJ – ARU_TX_CH32_HB_INJ	high-bit injection on/off.
ARU_TX_CH01_LB_INJ – ARU_TX_CH32_LB_INJ	low-bit injection on/off.
ARU_TX_CH01_GAP_INJ – ARU_TX_CH32_GAP_INJ	transmit short gap enabled or disabled.
ARU_RX_CH01_AWAIT_DATA – ARU_RX_CH32_AWAIT_DATA	enable/disable receiver 'await data' mode.

value (input) value to set the item:

CEI-100/200/x20 Values

- If the requested item is, ARU_Xn_RATE or ARU_Rn_RATE (where "n" is the transmitter number or receiver pair number), then valid values are:

AR_HIGH	high rate (100Kbs, 50Kbs for a CSDB channel).
AR_LOW	low rate (12.5Kbs).

- If the requested item is ARU_RnXm_PARITY (where "n" is the receiver pair number and "m" is the transmitter number), then valid values are:

AR_ODD	odd parity.
AR_EVEN	even parity.

  See AR_SET_RAW_MODE for information about disabling parity.

- If the requested item is ARU_INTERNAL_WRAP, then valid values are:

AR_WRAP_ON	internal wrap enabled
AR_WRAP_OFF	internal wrap disabled

Note: For the CEI-100/200, internal wrap mode causes the transmit channel to be internally connected to the two receive channels. Transmit data is received as transmitted on the first channel. It is received as the one's complement on the second channel.

For the CEI-x20, internal wrap mode causes the transmit channel to be internally connected to the same-numbered receive channel. Transmit data is received as transmitted.

- If the item is ARU_Rn_SDI_FILTER (where "n" is the receiver number), then valid values are:

AR_ON	SDI prefilter enabled
AR_OFF	SDI prefilter disabled

Note: If SDI pre-filtering is enabled, only data with an SDI value that matches the filter value (see below) is buffered.

- If the requested item is ARU_Rn_SDI_VALUE (where "n" is the receiver number), the valid value is an SDI in the range 0-3.

CEI-x20 Values

The following parameters are supported only by CEI-x20 boards. If these parameters are used with a CEI-100/200 board, they will return an error and perform no operation.

- If the requested CEI-x20 item is ARU_RX_CHnn_BIT_RATE where nn is the receiver channel (01 - 32), the valid values are:

AR_HIGH	100 Kbaud receiver rate selected.
AR_LOW	12.5 Kbaud receiver rate selected.

- If the requested CEI-x20 item is ARU_TX_CHnn_BIT_ RATE where nn is the transmitter channel (01 - 32), the valid values are:

AR_HIGH	100 Kbaud receiver rate selected.
AR_LOW	12.5 Kbaud receiver rate selected.

Note: The CEI-x20 API defines AR_HIGH as 50 Kbaud when setting the baud rate for a CSDB channel, and AR_LOW as 12.5 Kbaud. The function ar_setchparms() is recommended for setting the parameters for CSDB and ARINC 573/717 channels, rather than this function.

- If the requested CEI-x20 item is ARU_RX_CHnn_PARITY where nn is the receiver channel (01 - 32), the valid values are:

AR_ODD	receiver parity ON.
AR_OFF	receiver parity OFF.

- If the requested CEI-x20 item is ARU_TX_CHnn_PARITY where nn is the transmitter channel (01 - 32), the valid values are:

AR_ODD	odd transmitter parity.
AR_EVEN	even transmitter parity.

- If the requested CEI-x20 item is ARU_RX_CHnn_SDI_FILTER where nn is the receiver channel (01 - 32), the valid values are:

AR_ON	SDI filtering is enabled.
AR_OFF	SDI filtering is disabled.

- If the requested CEI-x20 item is ARU_RX_CHnn_SDI_VALUE where nn is the receiver channel (01 - 32), the valid values are the value of the SDI filter (0-3).
- If the requested CEI-x20 item is ARU_TX_CHnn_SHUT_OFF where nn is the transmitter channel (01 - 32), the valid values are:

AR_ON	Transmitter has been shut off.
AR_OFF	Transmitter is configured for normal transmission.

- If the requested CEI-x20 item is ARU_DISCRETE_OUTPUTS, this function accepts a new value for the 4/8/16 discrete outputs, as the lower 4/8/16 bits of the value. These discretes are active high sense. They are discussed in chapter 2, “Installation”.
- If the requested CEI-x20 item is ARU_TX_CHnn_HB_INJ where nn is the transmitter channel (01 - 32), the valid values are:

AR_ON	Extra transmit data bit is enabled.
AR_OFF	Extra transmit data bit is disabled (normal operation).

- If the requested CEI-x20 item is ARU_TX_CHnn_LB_INJ where nn is the transmitter channel (01 - 32), the valid values are:

AR_ON	Transmit one too few data bits is enabled.
AR_OFF	Transmit one too few data bits disabled (default).

- If the requested CEI-x20 item is ARU_TX_CHnn_GAP_INJ where nn is the transmitter channel (01 - 32), the valid values are:

AR_ON	Transmit short gap is enabled.
AR_OFF	Transmit short gap disabled (default).

- If the requested CEI-x20 item is ARU_RX_CHnn_AWAIT_DATA where nn is the receiver channel (01 - 32), the valid values are:

AR_ON	Enable receiver ‘await data’ mode (if supported).
AR_OFF	Disable receiver ‘await data’ mode (default).

CEI-x20 Parametric Values

The following items are supported by the CEI-220/520/520A/620/820 *only*, and ONLY if the PARAMETRIC option was purchased. If these parameters are used with a CEI-100/200/420 board, they return an error and perform no operation.

- If the requested CEI-220/520/520A/620/820 item requests a DAC value, this function returns the raw DAC setting. This setting is converted to a voltage via the following equation:

  Vdac = 5.0 Volts * n / 256

  Receiver threshold values are calculated using the following formulas:

Threshold = 10 * (2.5V – Vdac)	CEI-220/520/520A/620
Threshold = 10 * (Vdac – 2.5V)	CEI-820

The default DAC settings (threshold values) are defined as follows:

n = 0x72 (threshold = ±2.734V)	CEI-220
n = 0x70 (threshold = +3.125V)	CEI-520/520A/620 (pos)
n = 0x90 (threshold = -3.125V)	CEI-520/520A/620 (neg)
n = 0x90 (threshold = ±3.125V)	CEI-820

The following CEI-220 items use DAC values:

ARU_RX1_4_THRESH_VALUE	rx 1-4 threshold (220)
ARU_RX5_8_THRESH_VALUE	rx 5-8 threshold (220)
ARU_RX9_10_THRESH_VALUE	rx 9-10 threshold (220)
ARU_RX11_12_THRESH_VALUE	rx 11-12 threshold (220)
ARU_INPUT_THRESH_VALUE	discrete input threshold = 20 * Vdac (default is 1.56 Volts, 220 only)
ARU_TRANSMIT_VOLT_VALUE	xmtr output voltage level = 10 * Vdac (default is 9.96 Volts, 220 only)

The following items reference CEI-520/520A/620/820 DAC values:

ARU_DAC_VALUE_01 – ARU_DAC_VALUE_16	raw DAC value (refer to Appendix C for board-specific DAC definitions)

Note: See appendix C, “Parametric Voltage Programming“, to determine the DAC register definitions for your board.

- If the CEI-520/520A/620 item is ARU_OUTPUT_LEVEL_ADJ_nn where nn is the transmitter channel (01 - 32), the valid values are:

AR_ON	enable variable transmitter voltage levels
AR_OFF	disable variable transmitter voltage levels

AR_SET_CONTROL

Description

This routine has been removed from the CEI-x20 API. See the new function AR_SETCHPARMS as a replacement function.

AR_SET_PRELOAD_CONFIG

Syntax

CEI_INT16 ar_set_preload_config (CEI_INT16 board, CEI_INT16 item, CEI_UINT32 value);

Description

Call this routine before calling AR_LOADSLV to update the value of a particular load configuration setting. This routine should not be called after the calling process has executed AR_LOADSLV.

Only the CEI-520/520A/620/820/820TX enhanced API supports this function.

If 'item' is ARU_CONCURRENCY_MODE, the 'value' parameter specifies the board concurrency mode. One of three modes may be selected: AR_CONC_NONE, AR_CONC_MULTITHRD, or AR_CONC_MULTIPROC. Note that some modes are only supported on certain boards and operating systems (see table below).

Table 46. Supported Concurrency Modes

Concurrency Mode	Supported Board(s)	Supported Operating System(s)
AR_CONC_NONE	all CEI-x20 boards	all operating systems supported by the CEI-x20 API
AR_CONC_MULTITHRD or AR_CONC_MULTIPROC	CEI-520/520A, CEI-620, CEI-820	Windows XP/Vista/7/Server 2008/8.0/8.1/Server 2012/10, Red Hat Linux Ent 4 (kernels 2.6.17.13, 2.6.18, 2.6.20.14, 2.6.22.2, and 2.6.22.19 only)

The default concurrency mode, AR_CONC_NONE, provides no multi-thread protection and no multi-process support. The user application must ensure that at most one thread is calling into the API at any given time, and only a single process may interface with a particular board.

If AR_CONC_MULTITHRD concurrency mode is selected, thread protection is provided internally within the API. The user application may call into the API from multiple threads, but all threads must belong to a single process. The main user application thread should initialize the board with a call to AR_LOADSLV before other threads attempt to call into the API.

If AR_CONC_MULTIPROC concurrency mode is selected, thread protection is provided internally within the API and multiple processes may interface with a single board. If any process requests multi-process mode, all other processes must also request multi-process mode.

Some coordination is required when using multiple processes in conjunction with a single CEI-x20 device. A single "board" process should be launched first, followed later by one or more "channel" processes. The "board" process should load the board (AR_LOADSLV), configure board-specific parameters (e.g., AR_SET_TIMERRATE, AR_TIMETAG_CONTROL, etc.), and launch the board (AR_GO). After the "board" process launches the board using AR_GO, it is then permissible to launch one or more "channel" processes. A "channel" process should first attach to the board (AR_LOADSLV) and then execute operations strictly confined to the particular channel(s) that the process is associated with (e.g., AR_GETWORD, AR_PUTWORD, AR_SET_CONFIG, etc.). When finished, the "channel" process should call AR_CLOSE and terminate. The "board" process should remain running until all "channel" processes have terminated. After that, the "board" process should call AR_CLOSE and terminate. All processes must exercise caution when modifying board settings that may impact other concurrently-running processes.

Note that board setup/initialization is only executed in AR_LOADSLV if no other processes have the board open. If another process has the board open (that is, if another process has opened the board using AR_LOADSLV but hasn't yet closed the board using AR_CLOSE), AR_LOADSLV attaches to the device without re-initializing the board or modifying board settings. Similarly, AR_CLOSE only shuts down the board if no other processes have the board open. If another process has the board open, AR_CLOSE detaches from the board without shutting it down. Thus, board settings are preserved across "channel" process invocations of AR_LOADSLV and AR_CLOSE. Note the importance of matching every call to AR_LOADSLV with a call to AR_CLOSE in multi-process mode.

Multi-process mode is only required when accessing a single board from multiple processes. If multiple boards are installed, AR_CONC_NONE concurrency mode can be used as long as only one process and thread interfaces with each board.

If 'item' is ARU_INSTALL_INT_HANDLER, the 'value' parameter indicates whether or not an internal interrupt handler should be installed when the calling process executes AR_LOADSLV. If 'value' is AR_FALSE (the default value), no int handler should be installed. If 'value' is AR_TRUE, AR_LOADSLV attempts to install an internal interrupt handler if no internal interrupt handler has already been installed. Note that internal interrupt handling is only supported by the CEI-820 and is only available under Red Hat Ent 4 (kernels 2.6.17.13, 2.6.18, 2.6.20.14, 2.6.22.2, and 2.6.22.19 only). If internal interrupt handling is requested for an unsupported board type or operating system, AR_LOADSLV fails.

Internal interrupt handling functionality was added to support receiver 'await data' mode; no user application interrupt handling capability is provided. Either multi-process or multi-thread concurrency mode must be

selected if internal interrupt handling is requested. If used in conjunction with multi-process mode, only the single "board" process should install the internal interrupt handler.

If a process has enabled internal interrupt handling and successfully initialized the board (using AR_LOADSLV), the interrupt handler is automatically uninstalled when the process calls AR_CLOSE.

Return Value

ARS_NORMAL	Operation completed successfully.
ARS_INVBOARD	Invalid board number.
ARS_NO_OS_SUPPORT	Requested feature not supported under the current operating system.
ARS_NOT_SUPPORTED	Requested feature not currently available.
ARS_INVHARCMD	Invalid item parameter.
ARS_INVHARVAL	Invalid value parameter.

Arguments

board (input) board number.

item (input) the item to modify. Valid values are:

ARU_CONCURRENCY_MODE	board concurrency mode.
ARU_INSTALL_INT_HANDLER	enable/disable internal interrupt handling.

value (input) the value to set the item. If 'item' is ARU_CONCURRENCY_MODE, valid values are:

AR_CONC_NONE	no multi-thread or multi-process support (default).
AR_CONC_MULTITHRD	multi-thread concurrency mode (see Description section for details).
AR_CONC_MULTIPROC	multi-process concurrency mode (see Description section for details).

If 'item' is ARU_INSTALL_INT_HANDLER, valid values are:

AR_FALSE	no internal interrupt handler should be installed (default value).
AR_TRUE	install internal interrupt handler when board is loaded via AR_LOADSLV (see Description section for details).

AR_SET_RAW_MODE

Syntax

CEI_INT16 ar_set_raw_mode (CEI_INT16 board, CEI_INT16 direction, CEI_INT16 channel, CEI_INT16 control);

Description

Each transmit and receive channel can be configured to run in "raw" mode. In this mode, parity isn't used, and each 32-bit ARINC word is transmitted or received with the parity (32nd) bit unchanged. This is different from standard ARINC 429 data transfers, which call for the parity to be calculated. Raw mode is typically used for some older ARINC specifications such as ARINC 575. This routine is supported only on the CEI-200 and the CEI-x20.

See AR_SETCHPARMS for a superior replacement function.

Return Value

ARS_NORMAL	Operation completed successfully.
ARS_INVBOARD	Board is either not initialized or not supported for requested mode.
ARS_BRDNOTLOAD	Board not initialized.
ARS_INVARG	Invalid argument value.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

Arguments

board — (input) board number

direction — (input) type of channel specified in the channel argument (transmit or receive). Valid values are:

- ARU_XMIT — transmit channel.
- ARU_RECV — receive channel.

channel — (input) channel number. Valid arguments depend upon the number of channels configured on the board purchased. The first channel (transmit or receive) is number 0

control — (input) enables or disables raw mode. Valid values are:

- AR_ON — enable "raw" mode on this channel.
- AR_OFF — disable "raw" mode on this channel.

AR_SET_STORAGE_MODE

Syntax

CEI_INT16 ar_set_storage_mode (CEI_INT16 board, CEI_INT16 flag);

Description

There are three possible methods for storage of received data on the ARINC interface. The default method is Buffered Mode, which puts the data in a circular buffer with all data received on a particular channel stored chronologically in that channel's buffer. Each channel has its own individual buffer memory. You must use the utility AR_GETWORD or AR_GETNEXT to retrieve data sequentially from the buffer. If time tagging has been enabled, you must use AR_GETWORDT or AR_GETNEXTT to sequentially retrieve the data plus the time tag.

A permutation of Buffered Mode, called Merged Mode, merges all of the data for all receive channels into a single buffer. The storage areas for the individual receive channels are combined into one large buffer. Time tagging is automatically enabled. The buffer for receive channel 0 contains all of the data. You must use the utility AR_GETWORDT or AR_GETNEXTT for this channel to retrieve data sequentially from the buffer. On the CEI-100, the most significant bit of the time tag is overwritten with the channel. On the CEI-200, the most significant three bits of the time tag are overwritten with the channel number. On the CEI-x20, the most significant four bits of the time tag are overwritten with the channel number. This opposes the time tagging in Buffered Mode, which maintains that range.

An alternate storage method, Dedicated Mode, is available which puts received data in a dedicated location in memory, based on label and channel. Only the most recently received data is stored for any given label/channel combination. When this method is used, an 8-bit sequence number is maintained for each label/channel location. It is incremented each time a word is stored by the ARINC processor. You can retrieve data that is stored in this method by using the utility AR_GET_LATEST.

This routine defines the storage mode for all receive channels on the interface and must be called before starting the board with AR_GO.

Time tagging is only available for Buffered Mode and Merged Mode.

When using the Enhanced API for the CEI-520/520A/620/820/820TX, the board supports concurrent operation of both Buffered and Dedicated receive modes. Calling this function to enable Dedicated mode is ignored and returns ARS_NORMAL.

See AR_SETCHPARMS for a superior replacement for this function.

Return Value

ARS_NORMAL	Operation completed successfully.
ARS_INVBOARD	Invalid board number.
ARS_BRDNOTLOAD	Board not initialized.
ARS_INVARG	Invalid flag argument.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

Arguments

board (input) board number.

flag (input) Flag indicating type of storage mode to set:

ARU_BUFFERED	enables Buffered mode.
ARU_DEDICATED	enables Dedicated mode.
ARU_MERGED	enables Merged mode.

For the CEI-220/420/520/520A/620/820/820TX:

ARU_CH01_DEDICATED – ARU_CH32_DEDICATED	Sets specified channel to Dedicated Mode.
ARU_CH01_BUFFERED – ARU_CH32_BUFFERED	Sets specified channel to Buffered Mode.

AR_SET_TIMERRATE

Syntax

void ar_set_timerrate (CEI_INT16 board, CEI_INT16 rate);

Description

The ARINC board contains an onboard timer that can be programmed to interrupt at various frequencies (ticks). A 32-bit integer is incremented for each of the ticks. This tick count assists the host computer in accurately timing message retransmission. It is also used as a timetag when time tagging is enabled. The integer's value is read by using AR_GET_TIMERCNTL or cleared using AR_RESET_TIMERCNT.

For the CEI-100/200, this routine must be called after AR_LOADSLV but before AR_GO. The CEI-100 default timer rate is 0 (off). The CEI-200/x20 default timer rate is 20000 (5 milliseconds).

On the CEI-200/x20, the timer count is initialized with a 16-bit value. The value is given as a number of clocks where 4 clock cycles equals 1 microsecond. The default value timer rate is 20000 (decimal), which is equivalent to 5000 microseconds. Any valid 16-bit value can be programmed here. For example, if the timer rate were reprogrammed to a value of 4000, the value of a tick would be 1000 microseconds. The minimum acceptable timer rate parameter that should be programmed for a CEI-x20 board is 1440 (360 microseconds).

On the CEI-100, the timer count is selected from a table of possible values as defined below.

Increasing the timer rate also increases the overhead on the interface. For a lightly loaded interface, this is not important, but for a busy one it can impact performance, possibly causing ARINC data loss or timer tick loss.

The selected timer rate for the CEI-200 should not exceed 360 microseconds during high-speed operation of all twelve channels. As fewer channels are used, the resolution may increase.

The following limitations apply to the CEI-100:

- If time tagging isn't enabled and all three channels are running at the high bit rate, a maximum rate of 500 µsec may be used.

  If only two receivers are running, a maximum rate of 200 µsec may be used. A maximum of 100 µsec may be used if only one receiver is running.
- If time tagging is enabled, don't run all three channels simultaneously at the high bit rate or data may be lost.

When time tagging is enabled, a timer rate of 500 µsec may be used if only the two receivers are running. Otherwise, a timer rate of 100 µsec may be used if only one receiver is running and the transmitter is not.

- If all channels are running at the slow bit rate, any timer rate may be used, whether or not time tagging is enabled.

It is possible to increase CEI-200 timer resolution to ¼ of a microsecond by modifying the firmware to poll the timer counter register on receipt of an ARINC word. This can be done by purchasing the firmware development kit or by the customer service department upon request. The result would be to add slightly to firmware overhead and decrease the maximum timetag value.

Return Value

None.

Arguments

board (input) board for which to set rate.

rate (input) timer rate:

On the CEI-100, the following (hex) values are valid:

0	Off
50	100 us
52	200 us
55	500 us
70	1 ms
72	2 ms
75	5 ms
48	10 ms
4A	20 ms
4D	50 ms
68	100 ms
6A	200 ms
6D	500 ms
58	1000 ms

On the CEI-x20, any 16-bit value over 1440 is valid (see description).

AR_SETCHPARMS

Syntax

CEI_INT32 ar_setchparms (CEI_UINT32 cardnum, CEI_UINT32 channel, CEI_INT32 TransRcv, pAR_CHANNEL_PARMS def);

Description

This function is available both in the Standard library, and when you are using the Enhanced library for the CEI-520/520A/620/820/820TX. It is not available in the standard libraries for the CEI-100/200.

The entry point AR_SETCHPARMS sets up and defines the operating parameters and modes for a receiver or transmitter. This is the preferred method for initializing the 573/717 or the CSDB interfaces, and can be used to initialize 429 channels as well.

Return Value

ARS_NORMAL	Command completed without error.
ARS_INVBOARD	Board number out of range.
ARS_BRDNOTLOAD	ar_loadslv() has not been called.
ARS_INVARG	Channel out of range, or invalid argument value.
ARS_NO_HW_SUPRT	Channel type is not supported by this function.
ARS_CHAN_TIMEOUT	Timeout waiting for response from the on-board processor.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

Arguments

cardnum	(input) board number.
channel	(input) channel to initialize.
TransRcv	(input) sets the operating parameters for either the ARU_TRANSMITTER or ARU_RECEIVER.
def	(input) pointer to channel parameter definition structure.

The structure AR_CHANNEL_PARMS is documented in Appendix B, and is defined in the file UTILDEFS.H. This file is located in the **include** directory that is created when the software is installed on your computer. Refer to that file for the most up to date definition of this structure, and all of the parameters that it defines.

AR_SETINTERRUPTS

Syntax

CEI_INT32 ar_setinterrupts (CEI_UINT32 cardnum, CEI_UINT32 channel, CEI_INT32 TransRcv, CEI_UINT32 Count, CEI_UINT32 Mask);

Description

This function is available only when you are using the Enhanced API for the CEI-520/520A/620/820/820TX.

The entry point, AR_SETINTERRUPTS, is used to set the interrupt on "N" labels received or transmitted controls or to setup the interrupt on discrete input value change.

Entries are still made to the interrupt queue if enabled by this function, even if the hardware interrupt hasn't been enabled.

Return Value

ARS_INVBOARD	Board number invalid.
ARS_BRDNOTLOAD	Board not initialized.
ARS_NO_HW_SUPRT	Board does not support this function.
ARS_NORMAL	Successful completion of function.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

Arguments

cardnum	(input) board number of interest.
channel	(input) channel to modify (0-based).
TransRcv	(input) set up the ARU_TRANSMITTER, ARU_RECEIVER, or ARU_DISCRETES
Count	(input) for ARU_TRANSMITTER or ARU_RECEIVER – Number of labels between interrupts. If zero, no interrupts on "N" labels received/transmitted *or* on receive/transmit buffer over/underflow. For ARU_DISCRETES, this is the tick count for de-bouncing the discrete inputs.
Mask	(input) for ARU_TRANSMITTER or ARU_RECEIVER – Unused. For ARU_DISCRETES, this is the mask for detecting and interrupting on discrete changes.

AR_SLEEP

Syntax

void ar_sleep (CEI_UINT32 sleep_ms);

Description

Execution of the calling thread is suspended for the given number of milliseconds. A platform-dependent thread delay routine is used to implement this operation (for example, Sleep() under Windows). Timing accuracy of this operation is dependent upon the accuracy of the underlying operating system call.

This function is only available for CEI-x20 devices.

Return Value

None.

Arguments

sleep_ms	(input) Sleep duration (in milliseconds).

AR_TIMETAG_CONTROL

Syntax

CEI_INT16 ar_timetag_control (CEI_INT16 board, CEI_INT16 flag);

Description

The ARINC interfaces can generate a timetag for received data with a resolution equal to the timer rate that is programmed by the routine AR_SET_TIMERRATE, with a resolution of 32 bits. This timetag is stored in the receive buffer along with the data that it is tagging. Since the storage of a timetag effectively halves the size of the receive buffers (i.e., an ARINC word input is 32 bits plus a tag of 32 bits requires 64 bits of storage versus only 32 bits for a data-only buffer) the routine allows for selective enabling and disabling of the timetag. When time tagging is enabled, the interface automatically appends a timetag to each received data word as it is read off the bus. By default, time tagging is disabled.

The timetag indicates the time the data was put into the buffer by the interface. Since one interface controls up to sixteen channels, there may be some skew of the time tag, depending upon circumstances. For example, if two receive words come in simultaneously, the interface processes each one sequentially. Each word has a slightly different time tag. The skew should not be more than 360 microseconds, or the resolution of the time tag, whichever is greater. The CEI-100/200 receivers are interrupt-driven and the interrupts are handled in order of priority with receiver 8 being the highest and receiver 1 the lowest. The CEI-x20 receivers and transmitters are processed in "round-robin" fashion.

This routine must be called when the interface is in a reset state, prior to calling AR_GO (CEI-100/200 only). For CEI-x20 cards, this routine must only be called when you are certain that the receive buffers for all impacted channels are empty and the interface is not actively receiving traffic on the channel(s) in question. When time tagging is enabled, use AR_GETWORDT (or AR_GETNEXTT) to get the data. Don't use AR_GETWORD, which is for non-time tagged data and only reads 32 bits at a time. Using AR_GETWORD when time tagging is enabled results in alternately reading data and time tag.

See AR_SETCHPARMS for a superior replacement for this function.

Return Value

ARS_NORMAL	Success.
ARS_INVBOARD	Board is not initialized.
ARS_INVARG	Invalid flag argument.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

Arguments

board (input) board number of interest.

flag (input) action to take with time tags. Valid values:

ARU_ENABLE_TIMETAG	enable tagging of data.
ARU_DISABLE_TIMETAG	disable tagging of data.

CEI-x20:

ARU_CH01_ENABLE_TIMETAG – ARU_CH32_ENABLE_TIMETAG	Enable tagging of data on specified channel.
ARU_CH01_DISABLE_TIMETAG – ARU_CH32_DISABLE_TIMETAG	Disable tagging of data on specified channel.

AR_VERSION

Syntax

void ar_version (CEI_CHAR * verstr);

Description

This utility returns the API software and board firmware versions as an ASCII string.

If verstr is a non-NULL pointer, the API copies the API software and board firmware version string to the caller's buffer, which should be at least 120 characters long.

Return Value

None.

Arguments

verstr	(output) Pointer to the caller's buffer.

AR_XMIT_SYNC

Syntax

CEI_INT16 ar_xmit_sync (CEI_INT16 board, CEI_INT16 channel);

Description

This utility waits for all the data in the transmit buffer to be sent. It is useful in an application that is sending data out but doesn't want to halt the interface until everything has been sent. It waits up to 12 seconds only and returns an error if all of the labels in the buffer have not been sent.

This function returns all of the data that has been loaded into the transmitter once. 36-bit times are required following this event for the ARINC word to actually be transmitted onto the ARINC bus. For CEI-x20 boards, transmitters are double-buffered, so up to 72 bit times may be required after the buffer is emptied before the last label has been transmitted.

Return Value

ARS_NORMAL	Success.
ARS_INVBOARD	Invalid board number.
ARS_BRDNOTLOAD	Board not initialized.
ARS_INVARG	Invalid argument value.
ARS_NODATA	Board not started with a call to AR_GO.
ARS_NOSYNC	Time out before all data was sent.
ARS_BOARD_MUTEX	Failed to acquire/release the board lock.

Arguments

board	(input) board number of interest.
channel	(input) transmit channel.

CHAPTER 6

VxWorks Support

Overview

VxWorks is an embedded real-time operating system supporting flexible hardware configuration. The CEI-x20 API compiles and runs under Motorola PowerPC and Intel x86 VxWorks Board Support Packages. The CEI-820 and CEI-820TX are supported on both x86 and PowerPC platforms, while the CEI-220/420/420A/520/520A/620 are supported only on the x86 platform.

To use the CEI-x20 API with VxWorks you must first build a VxWorks image that includes our VxWorks driver. Next, download the compiled API module to the target machine. After downloading the API module, then compile, download, and run the user application. These three basic steps are described in the sections that follow.

It is also possible to incorporate the CEI-x20 API and user application code into your VxWorks image (thus removing the need to download these modules). The instructions given in the following sections can be extended to apply to an image-based build.

These instructions apply to both Tornado and Workbench environments, and are therefore written in a generic fashion which must be interpreted for your particular environment.

Building a VxWorks Image

To incorporate the CEI-x20 API with VxWorks, you must first rebuild your VxWorks image to include our VxWorks driver. This procedure uses the VxWorks Component Installation method.

The following steps describe how to update your VxWorks image. By default, '[*CEI-x20 Install Path*]' corresponds to the path 'c:\ Program Files \ Condor Engineering \ CEI-x20-SW'.

1. Copy the appropriate component installation file from the '[*CEI-x20 Install Path*] \ Source \ VxWorks \ Component Files' folder into '[*Tornado Path*] \ target \ config \ <*BSP Folder*>'. Use the following table to select the appropriate component installation file for your environment, platform, and board type.

Env	Plat	Board Type	Component Installation File
VxWorks 5.5	x86	CEI-220, CEI-420/420A	51_ABACO_x86_LEGACY_ISA.cdf
		CEI-520/520A, CEI-620, CEI-820, CEI-820TX	51_ABACO_x86_55_PCI.cdf
	PPC	CEI-820, CEI-820TX	51_ABACO_PPC_55_PCI.cdf
VxWorks 6.0 - 6.5	x86	CEI-220, CEI-420/420A	51_ABACO_x86_LEGACY_ISA.cdf
		CEI-520/520A, CEI-620, CEI-820, CEI-820TX	51_ABACO_x86_RTP_6x_PCI.cdf
	PPC	CEI-820, CEI-820TX	51_ABACO_PPC_RTP_6x_PCI.cdf
VxWorks 6.6 - 6.8	x86	CEI-220, CEI-420/420A	51_ABACO_x86_LEGACY_ISA.cdf
		CEI-520/520A, CEI-620, CEI-820, CEI-820TX	51_ABACO_x86_RTP_66_PCI.cdf
	PPC	CEI-820, CEI-820TX	51_ABACO_PPC_RTP_6x_PCI.cdf

2. Copy the driver source files into '[*Tornado Path*] \ target \ config \ <*BSP Folder*>'. Use the table below to determine which files should be copied for your board type.

Board Type	Files
CEI-220, CEI-420/420A	[CEI-x20 Install Path] \ Source \ VxWorks \ mem_vxWorks.c [CEI-x20 Install Path] \ Source \ lowlevel.h [CEI-x20 Install Path] \ Include \ cei_types.h [CEI-x20 Install Path] \ Include \ utildefs.h
CEI-520/520A, CEI-620, CEI-820, CEI-820TX	[CEI-x20 Install Path] \ Source \ VxWorks \ Common Driver \ *.*

3. Open the workspace containing your VxWorks target image project, and access the Kernel Configuration setup for the VxWorks image.
4. Within the *hardware* component group, include the "*Abaco Avionics Boards*" component.
5. Modify parameter values associated with the "*Abaco Avionics Boards*" component as required for your target system and/or development environment. The default parameter values support many configurations without modification, but customization of one or more

parameters may be required for certain target systems and/or development environments. Details regarding usage of each configuration parameter are provided within the parameter description.

6. Open the C/C++ compiler parameter setup window. Remove the **-ansi** compiler option. Using the –D compiler option, define the constant CEIX20_TARGET_VXWORKS.
7. Rebuild your VxWorks kernel image and reboot the target machine with the updated image.
8. If you are installing a CEI-520/520A, CEI-620, CEI-820, or CEI-820TX, open a shell to the target and invoke the function *abacoBoardShow*. This routine lists all detected Abaco Avionics products in increasing order by device ID. If you have only a single board installed, it is always device 0.

Now that the VxWorks image has been updated to include driver support for your CEI-x20 board, build and download the CEI-x20 API module. This is described in the section “Building the CEI-x20 API”.

x86 PCI BIOS Configuration

If you are installing a CEI-520/520A/620/820/820TX on an x86 target, the above procedure assumes that the x86 BIOS configures PCI boards by writing memory region addresses into the board’s configuration space. If your BIOS does not do this, you need to write memory addresses into the appropriate PCI configuration registers prior to invoking abacoInitPCI(). For more information on PCI and VxWorks, see “The Peripheral Component Interconnect (PCI) Bus and vxWorks” WTN-49 at ‘http://www.wrs.com/csdocs/product/t2/technote/WTN49.pdf’.

ISA Memory Mapping

If you are installing a CEI-220/420/420A, on-board jumper settings determine the base memory address. The default factory address setting is 0xd0000. If you have multiple ISA boards, they must be located at 0x4000 increments. VxWorks requires that memory regions do not overlap and that they are on even page boundaries. The x86 page size is 0x1000, so multiple boards should be at 0xd0000, 0xd4000, 0xd8000, and 0xdc000 when starting with the factory default. Specify the jumpered base address for board <n> using the ‘isa_base_addr<n>’ kernel configuration parameter.

Building the CEI-x20 API

The CEI-x20 API is built as a downloadable application. The following steps explain how to build the CEI-x20 API.

1. Create a new downloadable application project.
2. Add the API source file(s) to the project. Use the following table to determine which file(s) to add to the API project for your board type.

Board Type	Files
CEI-220, CEI-420/420A	[CEI-x20 Install Path] \ Source \ api220.c
CEI-520/520A, CEI-620, CEI-820, CEI-820TX	[CEI-x20 Install Path] \ Source \ api520.c [CEI-x20 Install Path] \ Source \ VxWorks \ mem_vxWorks.c

3. Open the C/C++ compiler parameter setup window. Remove the **-ansi** compiler option. Using the –I compiler option, add the include directories '[*CEI-x20 Install Path*] \ Include' and '[*CEI-x20 Install Path*] \ Source'. Add the '–mlongcall' compiler option (PPC targets only). Using the –D compiler option, define the constant CEIX20_TARGET_VXWORKS as well as any additional constants required for your board/platform as described in the table below.

Plat	Board Type	Additional Symbol(s) To Define
x86	CEI-220, CEI-420/420A	none
	CEI-520/520A, CEI-620, CEI-820	VXW_DRIVER_OPTION
	CEI-820TX	VXW_DRIVER_OPTION, CEIX20_TX32_RX32
PPC	CEI-820	VXW_DRIVER_OPTION, CEI_BIG_ENDIAN
	CEI-820TX	VXW_DRIVER_OPTION, CEI_BIG_ENDIAN, CEIX20_TX32_RX32

4. By default, the CEI-x20 API uses the taskDelay routine when a delay is required. To use a different delay routine, define the corresponding constant (using the –D option) shown below.

Delay Routine	Define Symbol
taskDelay	none – this routine is used by default
sysUsDelay	DELAY_USE_SYS_US_DELAY
sysMsDelay	DELAY_USE_SYS_MS_DELAY

5. Build the project and download it to the target system.

After you have built and downloaded the API, carry out a basic board test by downloading and running the provided sample program. This is described in the next section, "Building the Sample Program".

Building the Sample Program

The API distribution includes a sample program named vxw_wrap.c. You can use this program to test your installation. The program simply executes an internal wrap test on all receiver-transmitter channel pairs. You can also use vxw_wrap.c as a guide for programming with the CEI-x20 API.

Like the API, the sample program is also built as a downloadable application. The following steps explain how to build and run the sample program.

1. Create a new downloadable application project.
2. Add the file vxw_wrap.c from the folder '[*CEI-x20 Install Path*] \ Examples \ C \ VxWorks' to the project.
3. Open the C/C++ compiler parameter setup window. Remove the **-ansi** compiler option. Using the –I compiler option, add the include directory '[*CEI-x20 Install Path*] \ Include'. Add the '–mlongcall' compiler option (PPC targets only). Using the –D compiler option, define the constant CEIX20_TARGET_VXWORKS.
4. Build the project and download it to the target system.
5. Open a shell to the target and invoke the function **wrap(<board id>),** where **<board id>** represents the desired board index. The board index is always zero unless multiple Abaco Avionics cards are installed. If multiple cards are installed, use the **abacoBoardShow** command to determine which board index corresponds to the desired board. The **abacoBoardShow** command is only available for PCI, cPCI, and PMC cards.

   If everything is working correctly, the output of the sample application should resemble the trace shown Figure 32.

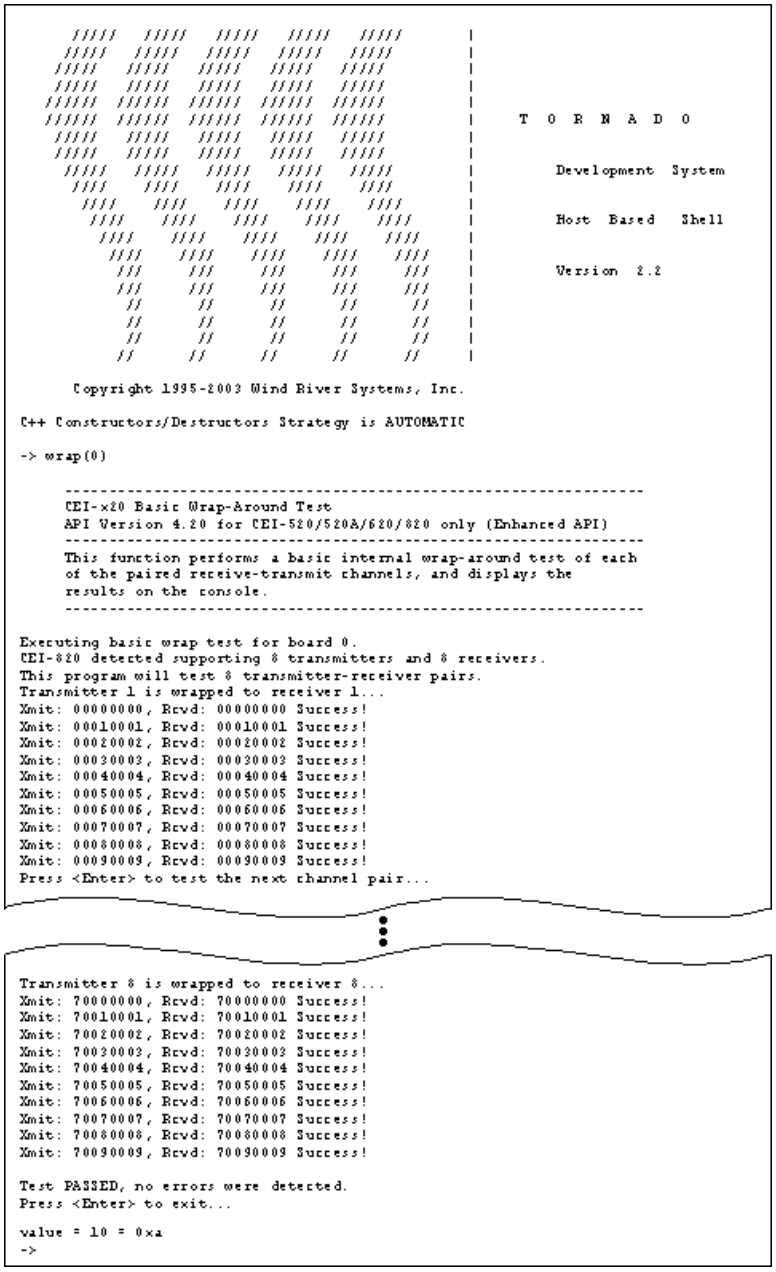


Figure 32. Sample Application Trace

CHAPTER 7

Description of CEI-100/200 ARINC Interface

Overview

If you need to write ARINC programs in a language or for an operating system other than that supported by the CEI application interface library, you may write your own utilities. This section documents the CEI-100 and the CEI-200 firmware interface. No special knowledge of DOS is required. Since the CEI-100/200 boards run completely out of dual-port RAM, you can access all ARINC data structures and data simply by reading/writing locations in that memory which is mapped into the upper memory block (UMB) of PC address space.

This section is *not* applicable to the CEI-x20, which uses a completely different interface. If the supplied libraries do not meet your needs, contact the factory for assistance.

All data structures are described here. Their addresses are given as byte offsets (in hex) from the base address of the board that is switch selectable as described in a previous section. For example, a data structure at offset 0430h on a CEI-200 based at the default segment 0D000h would be accessed using a far pointer to D000:0430.

Take care not to write to any locations in dual-port RAM not defined here. The result would be unpredictable. The executable code for the slave processor is in this RAM and overwriting that would have fatal consequences.

Loading the Board

Prior to using a board, its firmware must be loaded. You do this by entering the following command at the DOS command line for each CEI-100/200 board in the system. Typically, this would go in the AUTOEXEC.BAT file:

CONDOR /SA [/M XXXX] [/I XXX] [/X]

- The /SA switch tells it to load the control program and exit.
- The optional /M and /I switches specify non-default base segment and base I/O addresses, respectively. This action replaces the call to AR_LOADSLV that you would make if using the supplied utility libraries.
- The /X switch indicates the presence of extra memory (8K bytes on the CEI-100 or 32K bytes on the CEI-200.)

Controlling the Board

The board is controlled using standard I/O commands. Initially, the board is loaded and left in a reset state. All commands are byte output of value 0. The following offsets from the base I/O address have the function:

0 "Reset" - puts the board in a reset state.

1 "Go" - starts the board.

2 "Interrupt slave" - generates an interrupt on the slave. Not used in the current firmware.

3 "Reset interrupt" - resets an interrupt from the slave.

An assembly language example to start a board configured at the default I/O base address (0380h) follows:

```
mov     al,0
mov     dx,0381h
out     dx,al
```

Initializing the Board

After loading the board, and before you start it (with a "go"), you can initialize it by writing to specific control registers. You can't initialize it while it is running, since the firmware only executes the initialization sequence once per "go" command. To change parameters on a running system, the board must first be reset.

Programming the ARINC Channel Setup

An integrated control chip set which controls 2 receive channels controls all ARINC channels and 1 transmit channel. Each chip set is programmed with a separate control word that is formatted as follows:

Table 47. Control Word Format

Bit	Function
0	reserved
1	reserved
2	reserved
3	reserved
4	reserved
5	Internal Wrap-around enable (0 = enable, 1 = disable)
6	Receiver 1 SDI pre-filter (0=disable, 1= enable)
7	Receiver 1 SDI pre-filter mask LS bit
8	Receiver 1 SDI pre-filter mask MS bit
9	Receiver 2 SDI pre-filter (0=disable, 1=enable)
10	Receiver 2 SDI pre-filter mask LS bit
11	Receiver 2 SDI pre-filter mask MS bit
12	Transmit parity (0 = odd parity, 1 = even parity)
13	Transmit bit rate (0 = 100K Bps, 1 = 12.5K Bps)
14	Receiver bit rate (0 = 100K Bps, 1 = 12.5K Bps)
15	reserved

If SDI filtering is enabled on a channel, the received SDI must match that channel's two pre-filter mask bits. If internal self-test is enabled, the data for a transmit channel is tied internally to both of its corresponding receive channels. For example, on transmit channel 2, receive channel 3 gets the transmitted data and receive channel 4 gets the inverted data.

The control words are located in the following dual-port memory offset locations:

CEI-100 - **044Eh**

CEI-200 - **04A2h** (chip set 1 - receivers 1,2 transmitter 1)

04A4h (chip set 2 - receivers 3,4 transmitter 2)

04A6h (chip set 3 - receivers 5,6 transmitter 3)

04A8h (chip set 4 - receivers 7,8 transmitter 4)

Controlling the Timers

The ARINC boards have an on-board timer mechanism. The timer rate is determined by a programmable value, which is written at the timer control location:

CEI-100 - **044Ch**

CEI-200 - **0494h**

To determine what timer value to use, see AR_SET_TIMERRATE. Each time a timer "tick" occurs on the ARINC board, a 4-byte timer count variable is incremented (see AR_GET_TIMERCNT) which is maintained at the offset:

CEI-100 - **0448h**

CEI-200 - **049Ah**

Selecting the Receive Modes

There are three modes of ARINC data storage (see AR_SET_STORAGE_MODE). On the CEI-200, this is programmed with the storage mode control word at location:

04E0h

where 0 = buffered mode, 1 = dedicated mode and 2 = merged mode.

On the CEI-100, this is programmed with the slave status word at location:

0464h

where bits 3 and 4 define the mode: 0 = buffered mode, 1 = dedicated mode and 2 = merged mode. This word contains other data that must be maintained when updating these bits.

Enabling Time Tags

Timetags can be enabled or disabled (see AR_TIMETAG_CONTROL.) On the CEI-200 this is programmed using the timetag control word at location:

04DEh

where 0 = do not store timetags and 1 = store time tags.

On the CEI-100 this is programmed using the slave status word at location:

0464h

where if bit 2 is set, time tagging is enabled, otherwise it is disabled. This word contains other data that must be maintained when updating these bits.

Enabling Interrupts

The slave interrupts the host upon receipt of data on any particular channel (seeAR_INT_CONTROL). To enable this function on the CEI-200, write to the interrupt control status word at location:

04D8h

where the LS bit represents channel 1 and bit 7 represents channel 8. A 0 bit disables the interrupt function and a 1 enables the interrupt function for that channel.

On the CEI-100 the slave status word at location:

0464h

controls this with bit 0 for receiver 1 and bit 1 for receiver 2. The CEI-100 slave status word contains other information that must be preserved when modifying these bits.

This function can be used while the board is running.

Receiving Data

As data is received, it is stored in locations in dual-port RAM by the on-board processor. See AR_SET_STORAGE_MODE for a complete description of these modes. There are three storage modes: BUFFERED, DEDICATED, and MERGED.

In BUFFERED mode, each channel has its own circular buffer. As data comes in, the ARINC board continuously places it at the end of the buffer. It is your responsibility to retrieve data from the buffer fast enough so it is not overwritten. Each buffer has a four-word data structure that controls access to it:

word 0:	buffer segment
word 1:	wrap mask
word 2:	head pointer
word 3:	tail pointer

The buffer segment is the segment offset from the base of the board to the start of the buffer. For example, the buffer for a channel with a buffer segment value of 0C0h on a board based at segment CC00h would be addressed at segment CCC0h (real address CCC00h). This is the top of the circular buffer. The ARINC processor updates the head pointer as data is put in the buffer. You must update the tail pointer as data is read from the buffer to keep track of how much data has been removed. The head and tail pointers are both byte offsets from the top of the buffer. The head pointer points to the next location where data will be written. The tail pointer points to the next location to read. On initialization, both values are zero. When both values are the same, all data in the buffer is read. If the head pointer ever catches up to the tail pointer, an entire buffer of data is lost.

When the ARINC processor puts an ARINC word in the buffer, the LSB is written at the offset "head pointer". The MSB is at the offset "head pointer + 3". The ARINC processor adds the value 4 to the head pointer, performs the logical AND of the wrap mask and the head pointer, and stores the new head pointer value. The wrap mask is the size in bytes of the buffer minus 1. The buffer size must be a power of two.

The data structures are located at the following offset locations:

0430h receiver 1

0438h receiver 2

0440h receiver 3 (CEI 200 only)

0448h receiver 4 (CEI 200 only)

0450h receiver 5 (CEI 200 only)

0458h receiver 6 (CEI 200 only)

0460h receiver 7 (CEI 200 only)

0468h receiver 8 (CEI 200 only)

When time tagging is enabled, the slave processor appends a 4-byte timetag after each received word. When in this mode, you must read all 8 bytes (4 bytes of data and 4 bytes of time) each time the buffer is read. The timetag LSB is written at the offset "head pointer + 4". The MSB is written at offset "head pointer + 7". See AR_TIMETAG_CONTROL for more information.

When running in DEDICATED mode, data is stored at a dedicated location for each label within each channel's buffer. The circular buffer mechanisms (i.e., head/tail pointers and wrap mask) do not apply here. The location for the data for a particular label is simply calculated by multiplying the label by four. For example, the location for label 7 would be at offset 28 (decimal) from the beginning of that channel's buffer. The address of the beginning of the buffer is calculated in the same way as in buffered mode.

Since it would be redundant to store the label portion of an ARINC word at the location for that label, the firmware maintains an eight-bit sequence number in the LSB instead of label number. The sequence number is incremented each time a word for that label is stored there. The format of the data is:

byte 0	sequence number	
byte 1	ARINC data bits	8 15
byte 2	ARINC data bits	16 23
byte 3	ARINC data bits	24 31

Merged mode is handled in the same way as Buffered mode, except that all data is merged into the buffer for receive channel 1. It is possible to increase the buffer size for that channel to include the buffers for the other receive channels since all receive channels' buffers are contiguous in memory. You do this by modifying the wrap mask. For example, to combine the buffers for two consecutive receive channels into one large buffer, given that each buffer had a wrap mask of 01FFh, the wrap mask for the combined buffer would be 03FFh. The formula is:

n_channels * (wrap_mask + 1) - 1

Filtering Out Labels

An on-board filtering mechanism is available which causes the interface to throw away any undesired labels. This is implemented with a 256-byte flag array where each byte represents a label, and each bit represents a channel. Byte 0 contains the filtering flags for label 0, byte 1 contains the flags for label 1, etc. Bit 0 of a flag corresponds to the first receive channel. Bit 1 corresponds to the second channel, etc. A set bit causes that label/channel to be filtered out; a clear bit causes it to be stored. The flag array starts at location 0300h.

Transmitting Data

Each transmit channel has its own circular buffer which is handled the same way as in Buffered receive mode. When you desire to transmit data, you put data in the buffer starting at the offset pointed to by the head pointer and then update the head pointer to point to the next location. This is done by adding 4 and performing the bit-wise AND of the wrap mask and the head pointer and then writing the head pointer back to its dual-port RAM data structure location. The slave is constantly monitoring the transmit data structures. Any time it finds the head and tail pointers for a

channel are different, it knows it has to transmit data and does so. After each word is transmitted, the slave updates the tail pointer.

Since it is very possible to stuff data in the transmit buffer faster than it can be sent, check to see if the head pointer catches up to the tail pointer before any data is put in the buffer. If the head pointer catches up, an entire buffer of data is not sent.

If you have put data in a transmit buffer, wait until the head and tail pointers are equal before resetting the board. The slave updates the tail pointer after data has been removed from the buffer and put into the transmitter and the transmitter has been started. This time does not exactly coincide with the time the data goes out on the bus. It is starting to go out on the bus at this point, and is completely out no more than 32 bit times hence. (A bit time, of course, is determined by the programmed bit rate.)

The transmit buffer data structures are located at the following offsets and take the same format as the receive buffers:

CEI-100 - **0440h** - transmitter 1

CEI-200 - **0470h** - transmitter 1
0478h - transmitter 2
0480h - transmitter 3
0488h - transmitter 4

APPENDIX A

Modifying CEI-200 Base Bit Rate

Procedure

The CEI-200 provides multiple ARINC 429 channels. Standard high (100Kbs) and low (12.5Kbs) bit rates are supported by default. To use the hardware at different bit rates, the following procedures may be used. You must know what base address the board has been configured to use.

First, you must calculate a clock-scaling factor from the following table, based upon the desired bit rate:

Table 48. Calculating a Clock-Scaling Factor

Scaling	A	B	Bit Rate ("HI")	Max	Min
1:1	2	2	100K	125K	83K
1:2	4	4	50K	62.5K	41.5K
1:5	10	10	20K	25K	16.6K
1:9.5	19	19	10.5K	13.1K	8.7K
1:10	20	20	10K	12.5K	8.3K
1:10.5	21	21	9.5K	11.9K	7.9K
1:11	22	22	9.1K	11.3K	7.4K

Virtually any other scaling factor is achievable, based on the following formula:

(A+B)*2.5 = number of microseconds per bit.

In Table 48, a scaling of 1:1 is the default. The A and B values are used for scaling the clock on the CEI-200 as described in the following procedures. The Bit Rate "HI" setting is the actual bit rate (in bits/second) that corresponds to the "High" setting (as programmed using the utility libraries or menu selections.)

If the "Low" bit rate is programmed, the actual bit rate can be calculated by dividing the actual "High" bit rate by 8. The "Max" and "Min" values above represent the maximum calculated range within which the receivers detect valid data. The transmitters always transmit exactly at the defined bit rate.

1. From a user application:

   - After you call AR_LOADSLV, and before calling the routine AR_GO, you must program a new clock-scaling factor onto the board. The "A" value in Table 48 is a 16-bit value that must be written to the board at offset 0x490. The "B" value in Table 48 is a 16-bit value that must be written to the board at offset 0x492.

2. From the AVIATOR or CONDOR programs.

   - Start the applicable program (AVIATOR or CONDOR).
   - Create to DOS using the hot key ALT-D. If you don't have enough memory, free up enough memory by removing TSRs, etc.
   - From the DOS prompt, run the DEBUG program and manually modify the values on the board. Offset 0x490 contains a 16-bit value for the "A" value on the above chart and offset 0x492 contains the "B" value. The following example changes the clock to 10Kbs for a board configured at the default segment address of D000:

     DEBUG
     D D000:490
     E D000:490 14 0 14 0
     Q

Note: Notice the byte ordering per Intel architecture and hexadecimal notation for DEBUG input.

   - Finally, return to the application (type the **Exit** command) and proceed normally. In this example, the channel parameters as configured from the SETUP Menu must be set for the HIGH bit rate (the program says 100K, but it actually is as modified above).

APPENDIX B

CEI-x20 Structure Definitions

Structures

This chapter defines the structures that are used by the CEI-x20 API.

AR_CHANNEL_PARMS

The AR_CHANNEL_PARMS structure is used to pass parameters to the AR_SETCHPARMS function. These parameters are used to initialize ARINC receivers and transmitters, and the API returns information about the channel to the caller in this structure. This structure is defined as follows:

```
typedef struct ar_channel_parms {
CEI_INT32  BaudRate;        // Channel baud rate.
CEI_INT32  ChannelType;     // Channel type returned by function call.
CEI_INT32  InternalWrap;    // Internal wrap-around enable (receiver only)
CEI_INT32  IntCount;        // Interrupt enable
CEI_INT32  Parity;          // Parity definition
CEI_INT32  AutoSync;        // Autosync mode selection
CEI_INT32  HWEncoding;      // Hardware encoding method
CEI_INT32  Sync1;           // ARINC 573/717 Sync word #1
CEI_INT32  Sync2;           // ARINC 573/717 Sync word #2
CEI_INT32  Sync3;           // ARINC 573/717 Sync word #3
CEI_INT32  Sync4;           // ARINC 573/717 Sync word #4
CEI_INT32  NumSubFrames;    // Number of sub frames allocated by the API
CEI_INT32  OperateMode;     // Channel operational mode
CEI_INT32  TimeTagMode;     // Receiver channel time tag mode:
CEI_UINT32 Parametrics;     // Parametric enable/DAC settings
CEI_INT32  Reserved2;       // Reserved for future error injection use
} AR_CHANNEL_PARMS, *pAR_CHANNEL_PARMS;
```

The elements of the AR_CHANNEL_PARMS structure are defined as follows:

BaudRate

Defines the baud rate for the specified channel:

For ARINC 573/717 channels, baud rate/sub frame size:

ARU_717_SPEED_384 defines a sub frame size of 32 words.

ARU_717_SPEED_768 defines a sub frame size of 64 words.

ARU_717_SPEED_1536 defines a sub frame size of 128 words.

ARU_717_SPEED_3072 defines a sub frame size of 256 words.

ARU_717_SPEED_6144 defines a sub frame size of 512 words.

ARU_717_SPEED_12288 defines a sub frame size of 1024 words.

ARU_717_SPEED_24576 defines a sub frame size of 2048 words.

ARU_717_SPEED_49152 defines a sub frame size of 4096 words.

For CSDB channels:

ARU_CSDB_SPEED_4800 selects 4800 baud

ARU_CSDB_SPEED_9600 selects 9600 baud

ARU_CSDB_SPEED_12500 selects 12500 baud (same as AR_LOW)

ARU_CSDB_SPEED_19200 selects 19200 baud

ARU_CSDB_SPEED_38400 selects 38400 baud

ARU_CSDB_SPEED_50000 selects 50000 baud (same as AR_HIGH)

ARU_CSDB_SPEED_76800 selects 76800 baud

For all ARINC channels:

AR_HIGH select HIGH speed (100 Kbaud(50KB if CSDB))

AR_LOW select LOW speed (12.5 Kbaud)

In addition to the standard AR_HIGH and AR_LOW baud rate flags, the CEI-820/820TX also allows the user to specify any desired transmit/receive baud rate between 5K bps and

	200K bps. Simply specify the desired baud rate (in bps). For example, to set the baud rate of a particular transmit/receive channel to 98K bps, set 'BaudRate' to 98000.
ChannelType	The function returns the type of the specified channel: CHAN_NOT_IMPLEMENTED Channel is not implemented CHAN_ARINC429 Channel is an ARINC-429 channel CHAN_ARINC717 Channel is an ARINC-573/717 channel CHAN_CSDB Channel is a CSDB channel
InternalWrap	Defines the Internal wrap-around enable (receiver only): AR_WRAP_ON Enables internal wrap-around mode AR_WRAP_OFF Disables internal wrap-around mode
IntCount	Enables an interrupt when "IntCount" labels have been received or transmitted. If non-zero buffer full/empty interrupts are also enabled. If zero no interrupts are generated.
Parity	Selects the channel parity configuration: AR_EVEN, AR_ODD or AR_RAW
AutoSync	Selects Autosync or Raw mode for ARINC 573/717 receivers: ARU_717_AUTOSYNC or ARU_717_RAW
HWEncoding	Selects the hardware encoding for ARINC 573/717 channels only: ARU_717_H_BI_PHASE selects Harvard Bi-Phase Encoding ARU_717_BIPOLAR selects Bipolar R-T-Z Encoding NULL disables the transmitter

Sync1	Specifies the ARINC-573/717 receiver Sync word #1. This value is ignored for transmitters and non-573/717 channels.
Sync2	Specifies the ARINC-573/717 receiver Sync word #2. This value is ignored for transmitters and non-573/717 channels.
Sync3	Specifies the ARINC-573/717 receiver Sync word #3. This value is ignored for transmitters and non-573/717 channels.
Sync4	Specifies the ARINC-573/717 receiver Sync word #4. This value is ignored for transmitters and non-573/717 channels.
NumSubFrames	Returns the number of sub frames allocated by the API for ARINC-573/717 channels only.
OperateMode	Specifies the operational mode to enable, specify: ARU_BUFFERED or ARU_MERGED or ARU_FRAME (ARINC-573/717 channel only) FRAME mode is not supported on 573/717 channels running at baud rates above 12288. All baud rates are supported by 573/717 channels running in BUFFERED mode.
TimeTagMode	Specified the receiver channel time tag mode: ARU_ENABLE_TIMETAG Enable receiver time tagging ARU_DISABLE_TIMETAG Disable receiver time tagging Ignored if OperateMode is ARU_FRAME or ARU_MERGED
Parametrics	Parametric enable/DAC voltage settings: If this call is for ARU_TRANSMITTER and the board supports parametric transmit voltage, the value is 0x01-0xFF; API programs the transmit DAC with the supplied value. If value != 0xFF, the PARAMETRIC ENABLE bit is set in the channel control word. If specified as 0x00 the DAC value is not changed and the channel is programmed to disable variable output voltage.

	If this call is for ARU_RECEIVER, the value is 0xPPNN, where PP is the receiver positive threshold value and NN is the receiver negative threshold. On the CEI-820, the negative receiver threshold is ignored and automatically set to the additive inverse of PP (that is, NN = -PP). If specified as 0x0000 the DAC voltages are not modified. In either case, multiple channels might be sharing the associated DAC and will all be affected by the change in the DAC value(s) supplied by this argument.
Reserved2	Reserved for future error injection use.

APPENDIX C

Parametric Voltage Programming

Introduction

Some members of the CEI-x20 family provide the option of adjusting the output voltage levels, and adjusting the input threshold levels for the serial communications channels.

The parameters needed to program the DAC levels vary among the members of the CEI-x20 family; therefore, your code needs to be specific for the board type you are trying to control.

The voltage adjustments are provided by a group of Digital to Analog Converters (DAC) that are programmable through the API. These parameters are set using a call to AR_SET_CONFIG or AR_SETCHPARMS (preferred). You should use the information given here to determine the values that should be passed to the API functions.

All DAC output voltages are defined by the following relationship:

31 - 8	7-0
X	DAC output voltage = 5V * n / 256

This limits the range of valid DAC values to between 0 and 255. The values needed to program the DAC registers are defined in the following sections.

CEI-220 DAC Definitions

The DAC outputs used to control the functions of the CEI-220 board are defined as follows:

Table 49. CEI-220 DAC Register Scale Factors

DAC	Function	Scaling factor
#1	Receiver threshold channels 1 - 4 (sets symmetric positive and negative thresholds)	Thres = 10 * (2.5V – Vdac)
#2	Receiver threshold channels 5 - 8 (sets symmetric positive and negative thresholds)	Thres = 10 * (2.5V – Vdac)
#3	Receiver threshold channels 9 - 10 (sets symmetric positive and negative thresholds)	Thres = 10 * (2.5V – Vdac)
#4	Receiver threshold channels 11 - 12 (sets symmetric positive and negative thresholds)	Thres = 10 * (2.5V – Vdac)
#5	Discrete input threshold	Thres = 10 * Vdac
#6	ARINC transmit voltage for parametric testing	Differential output voltage = 2 * Vdac
#7	Spare	
#8	Spare	

CEI-420/420A DAC Definitions

The CEI-420 and the CEI-420A don't have adjustable output, input or discrete thresholds. Therefore, they have no DAC settings.

CEI-520/520A DAC Definitions

Table 50. CEI-520/520A DAC Register Scale Factors

DAC	Function	Scale factor
#1	Receiver threshold channels 1 – 4 positive threshold	Thres = 10 * (2.5V – Vdac)
#2	Receiver threshold channels 1 – 4 negative threshold	Thres = 10 * (2.5V – Vdac)
#3	Receiver threshold channels 5 – 6 positive threshold	Thres = 10 * (2.5V – Vdac)
#4	Receiver threshold channels 5 – 6 negative threshold	Thres = 10 * (2.5V – Vdac)
#5	Receiver threshold channels 7 – 8 positive threshold	Thres = 10 * (2.5V – Vdac)
#6	Receiver threshold channels 7 – 8 negative threshold	Thres = 10 * (2.5V – Vdac)
#7	Transmit voltage for parametric testing (channels 1 - 8)	Differential output voltage = 2 * Vdac
#8	Spare	
#9	Receiver threshold channels 9 - 12 positive threshold	Thres = 10 * (2.5V – Vdac)
#10	Receiver threshold channels 9 - 12 negative threshold	Thres = 10 * (2.5V – Vdac)
#11	Receiver threshold channels 13 - 14 positive threshold	Thres = 10 * (2.5V – Vdac)
#12	Receiver threshold channels 13 - 14 negative threshold	Thres = 10 * (2.5V – Vdac)
#13	Receiver threshold channels 15 - 16 positive threshold	Thres = 10 * (2.5V – Vdac)

DAC	Function	Scale factor
#14	Receiver threshold channels 15 - 16 negative threshold	Thres = 10 * (2.5V – Vdac)
#15	Transmit voltage for parametric testing (channels 9 - 16)	Differential output voltage = 2 * Vdac
#16	Spare	

CEI-620 DAC Definitions

Table 51. DAC Register Functions and Scaling Factors (CEI-620)

DAC	Function	Scale factor
#1	Receiver threshold channel 1 positive threshold	Thres = 10 * (2.5V – Vdac)
#2	Receiver threshold channel 1 negative threshold	Thres = 10 * (2.5V – Vdac)
#3	Receiver threshold channels 2 - 4 positive threshold	Thres = 10 * (2.5V – Vdac)
#4	Receiver threshold channels 2 - 4 negative threshold	Thres = 10 * (2.5V – Vdac)
#5	Receiver threshold channels 5 - 16 positive threshold	Thres = 10 * (2.5V – Vdac)
#6	Receiver threshold channels 5 - 16 negative threshold	Thres = 10 * (2.5V – Vdac)
#7	Transmit voltage for parametric testing (channels 1 - 4)	Differential output voltage = 2 * Vdac
#8	Secondary transmit voltage for parametric testing (useful for 561, set to 0xFF, not currently used)	Unipolar output voltage = 2.40 * Vdac

CEI-820 DAC Definitions

Table 52. DAC Register Functions and Scaling Factors (CEI-820)

DAC	Function	Scale factor
#1	Receiver #1 positive and negative threshold	Thres = 10 * (Vdac – 2.5V)
#2	Receiver #2 positive and negative threshold	Thres = 10 * (Vdac – 2.5V)
#3	Receiver #3 positive and negative threshold	Thres = 10 * (Vdac – 2.5V)
#4	Receiver #4 positive and negative threshold	Thres = 10 * (Vdac – 2.5V)
#5	Receiver #5 positive and negative threshold	Thres = 10 * (Vdac – 2.5V)
#6	Receiver #6 positive and negative threshold	Thres = 10 * (Vdac – 2.5V)
#7	Receiver #7 positive and negative threshold	Thres = 10 * (Vdac – 2.5V)
#8	Receiver #8 positive and negative threshold	Thres = 10 * (Vdac – 2.5V)
#9	Reserved for future use.	

Note: On the CEI-820, the negative receiver threshold is automatically set to the additive inverse of the positive receiver threshold.

APPENDIX D

Example CEI-x20 Programs

Introduction

The CEI-x20-SW distribution contains many example programs that can be used as templates for programming your own applications. Most of these example programs are written in "C" and assume ARINC 429 channels. Sample applications are provided to explain 573/717 and CSDB channel usage, and an example is also included that illustrates discrete input/output operation. There are also C#.NET, Visual Basic (VB6 and VB.NET), VxWorks, and LabWindows/CVI examples included with the software distribution.

.NET Example Programs

Two sample .NET projects are included with the CEI-x20 distribution: one supporting C#.NET and the other supporting VB.NET. Both examples implement a simple internal/external loopback test, and both interface with the CEI-x20 API .NET class library (also provided).

The supplied .NET examples and class library were developed using Microsoft Visual Studio 2005, but are also compatible with subsequent versions of Visual Studio which support .NET.

Visual Basic 6 Example Program

The CEI-x20 distribution includes an example project developed using Microsoft Visual Basic 6.0. The example implements a simple internal/external loopback test, and demonstrates general CEI-x20 programming in VB6. The file CEIx2032_vb6.txt contains all VB6-compatible function definitions and constants.

LabWindows CVI Support

A LabWindows/CVI sample program is included with the distribution, as well as a LabWindows/CVI function panel (CEI22032.fp) and include file (CEI22032.h). You may use these files as the basis for creating your own CVI applications. The 32-bit API DLL that was installed by the installation procedure is completely compatible with 32-bit versions of CVI. See "C/C++ Example Programs" below for descriptions of the standard examples included with the CEI-x20 software distribution.

C/C++ Example Programs

There are numerous C/C++ example programs included with the software distribution. Many of them are compiled and ready-to-run. Source code is included for all of the example programs.

Examine the example programs, and feel free to incorporate them in your own code. They can serve as templates for implementing your own applications. Briefly, the example "C" programs are defined as follows:

ARTEST.C	Simple internal/external wrap program.
BUFFER.C	Demo of the buffered, merged and dedicated receive modes.
BURST.C	Burst mode demo; ARINC words transmitted as quickly as possible.
DISCRETE.C	Illustrates usage of discrete input and outputs.
ENH717.C	Demo of the enhanced API support for ARINC 573/717 channels.
FILTER.C	Illustrates receiver label and SDI filtering.
MULTPROC.C	Illustrates multi-process support.
PARAWRAP.C	External wrap demo of adjustable ARINC levels for the CEI-220.
PARITY.C	Illustrates receiver/transmitter parity operation.
RAMP.C	Transmits some changing data, wraps the data back and verifies it.
SCHED.C	Demo of scheduled mode.
TESTCSDB.C	Demonstrates CSDB channel setup and TX/RX operations.
TST717.C	Demo of standard API support for ARINC 573/717 channels.
TSTX20.C	Demonstrates changing baud rates and parity settings.
WRAP.C	ARINC 429 wrap around test.

Note: Example programs are frequently added to the distribution; there may be new or different examples in the most current distribution than are listed here.