

GE
Intelligent Platforms

CEI-x30

User's Manual



Copyrights

Software Copyright © 2004-2011 GE Intelligent Platforms Embedded Systems, Inc. All rights reserved.

User's Manual Copyright © 2004-2011 GE Intelligent Platforms Embedded Systems, Inc.

This software product is copyrighted and all rights are reserved. The distribution and sale of this product are intended for the use of the original purchaser only per the terms of the License Agreement.

This User's Manual is copyrighted and all rights are reserved.

This document may not, in whole or part, be; copied; photocopied; reproduced; translated; reduced or transferred to any electronic medium or machine-readable form without prior consent in writing from GE Intelligent Platforms Embedded Systems, Inc.

Microsoft is a registered trademark of Microsoft Corporation.

Windows is a registered trademark of Microsoft Corporation.

VxWorks is a registered trademark of WindRiver Systems Corporation.

Integrity is a registered trademark of Green Hills Software Incorporated.

LabVIEW is a registered trademark of National Instruments Corporation.

GE Intelligent Platforms Embedded Systems, Inc. acknowledges the trademarks of other organizations for their respective products or services mentioned in this document.

CEI-x30-SW User's Manual (1500-048)

Software Revision: 3.10

Document Revision: 3.10

Document Date: 27 October 2011

GE Intelligent Platforms Embedded Systems, Inc.

6769 Hollister Ave.

Goleta, CA 93117

(805) 965-8000

(805) 963-9630 (fax)

support.avionics.ip@ge.com (email)

<http://defense.ge-ip.com/products/family/avionics>



Contents and Tables

Contents

Chapter 1	CEI-830	1
	Overview.....	1
	CEI-830 Specifications.....	1
	PMC/PCI Interface.....	2
	Transmit Channels	2
	Receiver Channels.....	2
	Avionics Discrete Input and Output	2
	IRIG Input and Output	2
	Typical Power Consumption	3
	Operating Temperature	3
	Weight	3
	PCI Memory Map.....	3
	I/O Connections.....	3
	Input /Output Connectors.....	3
	Input/Output Connector Pin-out	4
	IRIG-B Signal Connections	7
 Chapter 2	 R830RX.....	 8
	Overview.....	8
	R830RX Specifications	8
	PMC/PCI Interface.....	9
	Receiver Channels.....	9
	IRIG Input and Output	9
	Typical Power Consumption	9
	Operating Temperature	9
	Weight	9
	PCI Memory Map.....	10
	I/O Connections.....	10
	Mating Connectors	10
	Input /Output Connector Pin-out	10

Jumper Connections	12
IRIG-B Signal Connections.....	13
IRIG-B Generator Signal Connections.....	13
IRIG-B Receiver Signal Connections.....	13

Chapter 3 **CEI-530 14**

Overview.....	14
CEI-530 Specifications.....	14
PCI Interface.....	15
Transmit Channels	15
Receiver Channels.....	15
Avionics Discrete Input and Output	15
IRIG Input and Output	16
Typical Power Consumption	16
Operating Temperature	16
Weight	16
PCI Memory Map.....	16
I/O Connections.....	17
CEI-530 Outline Drawing.....	17
Mating Connectors	17
ARINC Input /Output Connector Pin-out.....	18
Discrete and IRIG Input/Output Connector Pin-out.....	19
IRIG-B Signal Connections	20

Chapter 4 **RAR-PCIE 22**

Overview.....	22
RAR-PCIE Specifications	22
PCI Express Interface.....	23
Transmit Channels	23
Receiver Channels.....	23
Avionics Discrete Input and Output	23
IRIG Input and Output	23
Typical Power Consumption	24
Operating Temperature	24
Weight	24
PCI Memory Map.....	24
I/O Connections.....	25
RAR-PCIE Outline Drawing	25
Mating Connectors	25
ARINC Input/Output Connector Pin-out.....	26
Discrete and IRIG Input/Output Connector Pin-out.....	27
IRIG-B Signal Connections	28

Chapter 5	CEI-430	30
	Overview.....	30
	CEI-430 Specifications.....	30
	PCI Interface.....	31
	Transmit Channels	31
	Receiver Channels.....	32
	Avionics Discrete Input and Output	32
	Differential Discrete Input and Output	32
	IRIG Input and Output	32
	Typical Power Consumption	32
	Operating Temperature	33
	Weight	33
	PCI Memory Map.....	33
	I/O Connections.....	34
	CEI-430 Outline Drawing.....	34
	Input/Output Connectors.....	34
	Input/Output Connector Pin-out	34
	IRIG-B Signal Connections	37
 Chapter 6	 CEI-430A	 38
	Overview.....	38
	CEI-430A Specifications.....	38
	PCI Interface.....	39
	Transmit Channels	39
	Receiver Channels.....	40
	Avionics Discrete Input and Output	40
	IRIG Input and Output	40
	Typical Power Consumption	40
	Operating Temperature	40
	Weight	40
	PCI Memory Map.....	41
	I/O Connections.....	42
	CEI-430A Outline Drawing.....	42
	Input / Output Connectors.....	42
	Input / Output Connector Pin-out	42
	IRIG-B Signal Connections	45
 Chapter 7	 AMC-A30	 46
	Overview.....	46
	AMC-A30 Specifications.....	46
	AMC/PCIe Interface	47
	Transmit Channels	47
	Receiver Channels.....	47

Avionics Discrete Input and Output	47
IRIG Input and Output	47
Typical Power Consumption	48
Operating Temperature	48
Weight	48
PCI Memory Map.....	48
I/O Connections	48
Input/Output Connectors.....	48
Input/Output Connector Pin-out	49
IRIG-B Signal Connections	50

Chapter 8 **RAR-EC 52**

Overview.....	52
RAR-EC Specifications.....	52
ExpressCard Interface	53
Transmit Channels	53
Receiver Channels.....	53
Avionics Discrete Input and Output	53
IRIG Input and Output	53
Typical Power Consumption	53
Operating Temperature	54
Weight	54
PCI Memory Map.....	54
I/O Connections.....	54
Mating Connectors	54
ARINC Input/Output Connector Pin-out.....	55
Transition Cable Pin-out	56
IRIG-B Signal Connections	57

Chapter 9 **RAR-CPCI..... 58**

Overview.....	58
RAR-CPCI Specifications.....	58
PCI Interface.....	59
Transmit Channels	59
Receiver Channels.....	59
Avionics Discrete Input and Output	59
IRIG Input and Output	60
Typical Power Consumption	60
Operating Temperature	60
Weight	60
PCI Memory Map.....	60
I/O Connections.....	61
RAR-CPCI Outline Drawing	61

	Mating Connectors	61
	ARINC Input/Output Connector Pin-out.....	61
	IRIG-B Signal Connections	64
Chapter 10	Windows Installation	65
	Software Installation for Windows	65
	Hardware Installation	66
	Hardware Installation with Windows 7/Vista/XP/2000/9x.....	66
	Hardware Installation with Windows NT 4.0.....	67
	Installation Verification	67
Chapter 11	VxWorks Installation.....	68
	Overview.....	68
	Building a VxWorks Image.....	68
	BIOS Initialization	70
	Using the Sample Program.....	70
	Building the API and Sample Program with Tornado.....	70
	Target-specific Compiler Directives	74
Chapter 12	Linux Installation.....	76
	Overview.....	76
	Software Installation.....	76
	Building Applications.....	77
	Automatic Installation (Builds LSP and API)	77
	Manual Installation.....	78
	Linux Driver Operation	78
	Troubleshooting.....	79
	Useful Linux system utilities	79
	Compilation Errors.....	79
	Run-time Errors.....	79
Chapter 13	Integrity® Support	81
	Introduction.....	81
	Integrity Installation.....	81
	Integrity PCI Driver Installation	82
	Building Integrity Applications	82
	Building the CEI-x30 API with Multi	85
Chapter 14	CEI-x30 Features	87
	Overview.....	87
	Enhanced CEI-x30 Interface	87
	ARINC 429 Protocol Support	87
	ARINC 429 Transmit Tri-state Support	88

ARINC 573/717 Protocol Support.....	88
CEI-x30 Timers	89
Receive Message Time-tagging and Timer Usage	90
IRIG 64-Bit Time Reference	90
Internal 64-Bit One Microsecond Time Reference	91
Internal 32-Bit Twenty Microsecond Time Reference.....	91
Internal 32-Bit One Millisecond Time Reference	91
CEI-x20 Compatible Time Reference	91
Receive Message Buffering Methods	92
Individual Circular Buffer Storage	92
Merged Circular Buffer Storage	92
Snapshot Buffer Storage	92
Interrupts and Triggers	93
ARINC 429 Receive Label Filtering and Interrupt Event.....	94
ARINC 429 Periodic Message Scheduling.....	95
Message Rate Skew.....	95

Chapter 15	BusTools/ARINC™ Data Bus Analyzer.....	98
	General Information	98
	BusTools/ARINC Demo Software.....	98

Chapter 16	Program Interface Library	99
	Overview.....	99
	API Source Files	99
	CDEV_API.C.....	99
	CDEV_API.H.....	99
	CDEV_GLB.H.....	100
	AR_ERROR.H.....	100
	CDEV_HW.H	100
	CEI_TYPES.H	100
	CDEV_WIN.C	100
	CDEV_VXW.C.....	100
	CDEV_LNX.C.....	100
	CDEV_INT.C.....	101
	CDEV_LRT.C.....	101
	CDEV_FW.H - Firmware Load Files.....	101
	Windows Libraries	102
	Time-tag Structure Definition	102
	Setting the Device Time.....	103
	Return Status Values	104
	Programming with the CEI-x30 API Interface	105
	Example Routines – Summary	106
	Tst_cnfg.c	106

Multiprocess_test.c.....	107
Visual Basic	108
Working with Unsigned Integers in Visual Basic	108
API Routines - Summary	109
Initialization and Control Routines.....	109
Device Control Routines.....	109
Termination Routines.....	110
Receive/Transmit Channel-level Configuration Routines	110
Device-level Configuration Routines	111
Receive Data Processing Routines	111
Transmit Data Processing Routines.....	112
Timer-related Routines.....	113
Information and Status Routines.....	113
Utility Routines	113
AR_BOARD_TEST.....	115
AR_BYPASS_WRAP_TEST.....	116
AR_CLR_RX_COUNT	117
AR_CLOSE.....	118
AR_CONVERT_TIME_TO_STRING	119
AR_DEFINE_MSG.....	120
AR_DEFINE_MSG_BLOCK.....	121
AR_ENH_LABEL_FILTER	123
Label Filtering	123
Interrupt Generation	123
AR_EXECUTE_BIT.....	125
AR_GET_573_FRAME.....	127
AR_GET_429_MESSAGE.....	129
AR_GET_BASE_ADDR.....	131
AR_GETBLOCK	132
AR_GETBLOCK_T.....	133
AR_GET_BOARDNAME.....	135
AR_GET_BOARDTYPE	136
AR_GET_CONFIG.....	137
AR_GET_DATA.....	141
AR_GET_DATA_XT	143
AR_GET_DEVICE_CONFIG.....	144
AR_GET_573_CONFIG.....	150
AR_GET_ERROR	153
AR_GETFILTER	154
AR_GET_LABEL_FILTER.....	156
AR_GET_LATEST.....	157
AR_GET_LATEST_T	158
AR_GETNEXT	159

AR_GETNEXTT.....	160
AR_GETNEXT_XT.....	161
AR_GET_RX_COUNT	162
AR_GET_SNAP_DATA	163
AR_GET_STATUS.....	164
AR_GET_STORAGE_MODE.....	165
AR_GET_TIME	166
AR_GET_TIMERCNTL.....	168
AR_GETWORD	169
AR_GETWORDT.....	170
AR_GETWORD_XT.....	172
AR_GO	173
AR_HW_INTERRUPT_BUFFER_READ.....	174
AR_INTERRUPT_QUEUE_READ	175
AR_INITIALIZE_API.....	176
AR_INITIALIZE_DEVICE.....	177
AR_HW_INTERRUPT_BUFFER_READ.....	178
AR_INTERRUPT_QUEUE_READ	179
AR_LABEL_FILTER.....	180
AR_LOADSLV.....	181
AR_MODIFY_MSG.....	183
AR_MODIFY_MSG_BLOCK.....	184
AR_NUM_RCHANS.....	186
AR_NUM_XCHANS.....	187
AR_PUT_429_MESSAGE.....	188
AR_PUT_573_FRAME.....	189
AR_PUTBLOCK	190
AR_PUTBLOCK_MULTI_CHAN.....	191
AR_PUTFILTER	193
AR_PUTWORD.....	195
AR_QUERY_DEVICE.....	196
AR_RESET	197
AR_RESET_TIMERCNT.....	198
AR_SET_CONFIG	199
AR_SET_DEVICE_CONFIG	204
AR_SET_573_CONFIG	209
AR_SET_MULTITHREAD_PROTECT.....	212
AR_SET_ISR_FUNCTION.....	213
AR_SET_PRELOAD_CONFIG.....	214
AR_SET_RAW_MODE.....	216
AR_SET_STORAGE_MODE.....	218
AR_SET_TIME.....	219
AR_SLEEP.....	221

AR_SET_TIMERRATE	222
AR_STOP	223
AR_VERSION	224
AR_WAIT	225

Chapter 17 **CEI-x30 Hardware Interface226**

Overview	226
PCI Configuration Space	227
PCI Device Identifiers and Resources	227
Host Memory Map	229
Device Interface Register Set (Common Memory)	230
Global Enable Register	230
DAC Control Register.....	231
Timer Registers	231
Update IRIG Generator Time Register.....	232
IRIG Sample Time Register.....	232
IRIG Sample Timestamp Registers	232
SRAM Access Address Register	233
SRAM Access Data Register	233
General Input Registers.....	233
Interrupt Queue Register.....	234
Channel Statistics Table.....	234
Channel Register Set.....	234
Channel Status Register	235
Channel Configuration Registers.....	236
Channel Configuration Register 1 – ARINC 429 Receive	236
Channel Configuration Register 1 – ARINC 573 Receive	237
Channel Configuration Register 2 – ARINC 573 Receive	239
Channel Configuration Register 3 – ARINC 573 Receive	239
Channel Configuration Register 1 – ARINC 429 Transmit.....	239
Channel Configuration Register 1 – ARINC 573 Transmit.....	241
Channel Configuration Register 1 – Discrete or Digital Output.....	243
Channel Configuration Register 1 – Differential Output	243
Channel Buffer Words	243
Channel Buffer Word 1 - Receive	244
Channel Buffer Word 2 - Receive	244
Channel Buffer Word 3 – Receive.....	244
Channel Buffer Word 4 – Receive.....	244
Channel Buffer Word 1, 2, and 3 - Transmit.....	245
Channel Buffer Word 4 - Transmit.....	246
Interrupt Queue	246
Message Scheduler Table.....	247
Snapshot Buffer.....	248

SRAM Memory Organization..... 249

 Label Filter Table 249

 Snapshot Buffer..... 250

 Individual Channel Buffers..... 250

ARINC 429 Receive Threshold 250

Avionics Discrete I/O 251

Differential Discrete I/O..... 251

Hardware Channel Assignments 252

Figures

Figure 1. CEI-830	1
Figure 2. 68-pin Front-Panel Receptacle Connector	6
Figure 3. R830RX	8
Figure 4. P1 68-pin Front-Panel (Bezel) Connector – View Facing Connector Pins.....	11
Figure 5. CEI-530	14
Figure 6. CEI-530 Outline Drawing.....	17
Figure 7. 68-pin Front-Panel (Bezel) Connector – View Facing Connector Pins.....	18
Figure 8. 50-pin IDC-50 I/O Connector – View Facing Connector Pins...	19
Figure 9. RAR-PCIE	22
Figure 10. RAR-PCIE Outline Drawing	25
Figure 11. 68-pin Front-Panel (Bezel) Connector – View Facing Connector Pins.....	26
Figure 12. 50-pin IDC-50 I/O Connector – View Facing Connector Pins.	27
Figure 13. CEI-430	30
Figure 14. CEI-430 Outline Drawing.....	34
Figure 15. P1 50-pin IDC ARINC Interface Connector – View Facing Connector Pins.....	35
Figure 16. P2 40-pin IDC I/O Connector – View Facing Connector Pins.	36
Figure 17. CEI-430A	38
Figure 18. CEI-430A Outline Drawing.....	42
Figure 19. P1 50-pin IDC ARINC Interface Connector – View Facing Connector Pins.....	43
Figure 20. P2 40-pin IDC I/O Connector – View Facing Connector Pins.	44
Figure 21. AMC-A30	46
Figure 22. RAR-EC	52
Figure 23. P1 Connector – View Facing Connector Pins.....	56
Figure 24. RAR-CPCI	58
Figure 25. RAR-CPCI Outline Drawing.....	61
Figure 26. P1/P2 50-pin Front-Panel (Bezel) Connectors – AMP Champ 0.8 mm Receptacle Connectors, View Facing Connector Pins	62
Figure 27. Integrity libbsp.gpj with cei_int_pci_drv.c Added	82
Figure 28. Example CEI-830 Integrity Application Project Setup.....	83
Figure 29. Example CEI-830 Integrity Application Project Option.....	83
Figure 30. Adding a MemoryPoolSize Entry	84
Figure 31. Modifying the Value for the DefaultStartIt Attribute	84
Figure 32. Example CEI-830 Integrity Library Project Setup.....	85
Figure 33. Example CEI-830 Integrity Library Project Options	86

Tables

Table 1. Power Consumption	3
Table 2. CEI-830 PCI Memory Map	3
Table 3. Input/Output Connectors	4
Table 4. CEI-830 I/O Connections.....	5
Table 5. CEI-830-xxxx-J I/O Connection for ARINC 573/717	6
Table 6. CEI-830-xxxxN I/O Connection for Discrete I/O	6
Table 7. IRIG Signal Formats	7
Table 8. Power Consumption	9
Table 9. R830RX PCI Memory Map.....	10
Table 10. P1 Input/Output Connector	10
Table 11. R830RX ARINC I/O Connections	11
Table 12. IRIG Signal Formats Supported.....	13
Table 13. Power Consumption	16
Table 14. CEI-530 PCI Memory Map	16
Table 15. P2 Input/Output Connector	17
Table 16. P3 Input/Output Connector	17
Table 17. CEI-530 Front Panel ARINC I/O Connections	18
Table 18. CEI-530 IDC-50 I/O Connections	20
Table 19. IRIG Signal Connections	20
Table 20. Power Consumption	24
Table 21. RAR-PCIE PCI Memory Map.....	24
Table 22. P2 Input/Output Connector	25
Table 23. P3 Input/Output Connector	25
Table 24. RAR-PCIE Front Panel ARINC I/O Connections	26
Table 25. RAR-PCIE IDC-50 I/O Connections.....	28
Table 26. IRIG Signal Connections	28
Table 27. PCI Stack Location Shunts	31
Table 28. Power Consumption	32
Table 29. CEI-430 PCI Memory Map	33
Table 30. CEI-430 Input/Output Connectors.....	34
Table 31. CEI-430 P1 ARINC I/O Connections.....	35
Table 32. CEI-430 P2 I/O Connections	36
Table 33. IRIG Signal Connections	37
Table 34. PCI Stack Location Shunts	39
Table 35. Power Consumption	40
Table 36. CEI-430A PCI Memory Map	41
Table 37. CEI-430 Input/Output Connectors.....	42
Table 38. CEI-430A P1 ARINC I/O Connections.....	43
Table 39. CEI-430A P2 I/O Connections	44
Table 40. IRIG Signal Connections	45
Table 41. Payload Power Consumption.....	48

Table 42. AMC-A30 PCI Memory Map.....	48
Table 43. Input/Output Connectors.....	49
Table 44. AMC-A30 I/O Connections.....	49
Table 45. AMC-A30-xxxx-J I/O Connection for ARINC 573/717	50
Table 46. IRIG Signal Formats	51
Table 47. Power Consumption	53
Table 48. RAR-EC PCI Memory Map	54
Table 49. P1 Input/Output Connector	54
Table 50. RAR-EC P1 ARINC I/O Connections.....	55
Table 51. RAR-EC-XX Transition Cable [37-Pin D-Subminiature] Connections	56
Table 52. IRIG Signal Connections	57
Table 53. Power Consumption	60
Table 54. RAR-CPCI PCI Memory Map.....	60
Table 55. P1/P2 Input/Output Connectors	61
Table 56. RAR-CPCI P1/P2 Front Panel I/O Connections	62
Table 57. RAR-CPCI Rear Panel Input/Output Connector Definition	63
Table 58. IRIG Signal Connections	64
Table 59. CEI-x30 PCI Configuration Space	227
Table 60. CEI-x30 Host Memory Map	229



CEI-830

Overview

The CEI-830 card is a multiple-channel ARINC interface available in several configurations. When configured as the CEI-830-1616, this product includes thirty-two ARINC 429 channels, (sixteen receivers and sixteen transmitters), in a PMC form-factor. Configurations are available supporting various ARINC 429 channel counts, ARINC 573/717, Avionics Discrete I/O, and IRIG time synchronization. A variety of bus adapter configurations are also available, supporting both front and rear-I/O access for PCI, PCI Express, and CompactPCI platforms.

CEI-830 Specifications

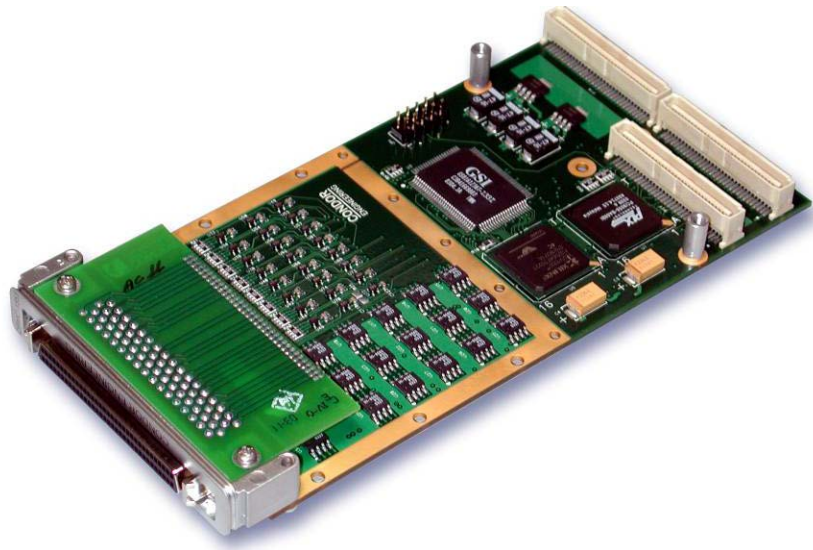


Figure 1. CEI-830

The CEI-830 is a multiple channel, multiple protocol interface built to the PMC standard IEEE-P1386.1.

PMC/PCI Interface

- Standard single-width CMC module per IEEE-P1386.1 draft standard
- +5V and +3.3V PCI signaling compatibility and universal keying
- 66 MHz, 32 bit PCI operation

Transmit Channels

- Up to sixteen independent differential serial transmit channels
- Automatic parity generation
- 2048 message transmit buffer for each channel
- Baud rate/slew rate software-programmable for each channel
- 1024 entry message table supporting scheduled message transmission for all channels

Receiver Channels

- Up to sixteen independent, differential receive channels
- 2048 message buffered mode receive FIFO buffer for each channel.
- 16384 message merged mode receive FIFO buffer
- Label/SDI message independent snapshot storage for each channel
- Independent merged/individual receive FIFO buffer operation.
- 64-bit, 1 μ sec time-tag is stored with each data element in the FIFO
- Parity error detection

Avionics Discrete Input and Output

- Four dedicated avionics-level discrete channels
- Output may switch to ground up to 500mA
- Fixed input threshold of 2.7 +/- 0.2 volts

IRIG Input and Output

- IRIG Time-code receiver and transmitter

Typical Power Consumption

Table 1. Power Consumption

+3.3V	+5V	+12V	-12V
500 mA	50 mA	100 mA (no TX Loads)	100 mA (no TX Loads)

Operating Temperature

-40 to +85 C

Weight

3.6 ounces

PCI Memory Map

The following table summarizes the PCI memory map interface definition for the CEI-830.

Table 2. CEI-830 PCI Memory Map

Region	Type	Size	Description
configuration	configuration	64 bytes	PCI configuration space
PCI BAR0	memory	512 bytes	PCI9056 memory-mapped local configuration registers
PCI BAR1	I/O	256 bytes	PCI9056 I/O-mapped local configuration registers, unused
PCI BAR2	memory	512K bytes	CEI-830 host interface
PCI BAR3	n/a	0	not used
PCI BAR4	n/a	0	not used
PCI BAR5	n/a	0	not used

I/O Connections

Input /Output Connectors

At publication of this document, the following mating connector was compatible with the 68-pin SCSI connector used on the CEI-830. GE

Intelligent Platforms Embedded Systems supplies the adapter cable CONSCSI3-6 for this connection.

Table 3. Input/Output Connectors

Part No.	Description	Manufacturer
1-5750913-7	Front Panel 68 pin SCSI-3	AMP/Tyco

Input/Output Connector Pin-out

The different CEI-830 product configurations have specific channel pin-out definitions based on the number of channels and protocols installed. Table 4 describes both the 68-pin front panel and P14 mezzanine I/O connector pin-out for the ARINC 429 version of the CEI-830 module. The exact ARINC 429 channel pin-out depends on the number of receivers and transmitters configured on your CEI-830. Table 6 describes the pin-out differences for the -J version of the CEI-830; these pins support ARINC 573/717 protocols on the pins used by the upper channels on non-J configurations. Table 6 describes the pin-out differences for the N configuration, supporting Discrete I/O on the pins assigned to the upper channels of the non-N configurations.

Additionally, the P14 mezzanine I/O connector routes the first fifteen ARINC 429 transmit and receive channels, the J configuration ARINC 573/717 channels, and up to two optional Discrete I/O channels.

To externally wrap ARINC signals, connect the transmitter signals to the respective receiver signals, TXnA to RXnA and TXnB to RXnB.

Figure 2 shows the view facing the receptacles of a 68-pin Front-Panel Receptacle Connector (SCSI-3-compatible with Rails and Latch Blocks).

Table 4. CEI-830 I/O Connections

Signal	Front Panel (P1)	P14 I/O Connector	Signal	Front Panel (P1)	P14 I/O Connector
RX1A	1	63	RX1B	35	64
RX2A	2	61	RX2B	36	62
RX3A	3	59	RX3B	37	60
RX4A	4	57	RX4B	38	58
RX5A	5	55	RX5B	39	56
RX6A	6	53	RX6B	40	54
RX7A	7	51	RX7B	41	52
RX8A	8	49	RX8B	42	50
RX9A	9	47	RX9B	43	48
RX10A	10	45	RX10B	44	46
RX11A	11	43	RX11B	45	44
RX12A	12	41	RX12B	46	42
RX13A	13	39	RX13B	47	40
RX14A	14	37	RX14B	48	38
RX15A	15	35	RX15B	49	36
RX16A	16	N/A	RX16B	50	N/A
TX1A	17	33	TX1B	51	34
TX2A	18	31	TX2B	52	32
TX3A	19	29	TX3B	53	30
TX4A	20	27	TX4B	54	28
TX5A	21	25	TX5B	55	26
TX6A	22	23	TX6B	56	24
TX7A	23	21	TX7B	57	22
TX8A	24	19	TX8B	58	20
TX9A	25	17	TX9B	59	18
TX10A	26	15	TX10B	60	16
TX11A	27	13	TX11B	61	14
TX12A	28	11	TX12B	62	12
TX13A	29	9	TX13B	63	10
TX14A	30	7	TX14B	64	8
TX15A	31	5	TX15B	65	6
TX16A	32	N/A	TX16B	66	N/A
IRIGRX+	33	3	IRIGRX-	67	4
IRIGTX	34	1	Gnd (note)	68	2

Note: The ground pins are provided as Discrete I/O return lines or for shielding, as necessary.

Table 5. CEI-830-xxxx-J I/O Connection for ARINC 573/717

Signal	Front Panel (P1)	P14 I/O Connector	Signal	Front Panel (P1)	P14 I/O Connector
ARINC 717 BPRZ RXA	14	37	ARINC 717 BPRZ RXB	48	38
ARINC 717 HBP RXA	15	35	ARINC 717 HBP RXB	49	36
Reserved	16	N/A	Reserved	50	N/A
...
ARINC 717 TXA (note)	30	7	ARINC 717 BPRZ TXB	64	8
ARINC 717 HBP TXB	31	5	Reserved	65	6
Reserved	32	N/A	Reserved	66	N/A

Note: The ARINC 573/717 TXA (High) signal is supported on this pin for both the BPRZ and HBP protocols. The selected protocol processed is based on the selection of the ARINC 717 HBP and BPRZ Encoding bits in the respective Transmit Channel Configuration register. See the API routine AR_SET_573_CONFIG for the method to define the ARINC 573/717 active encoding selection for this output pin.

Table 6. CEI-830-xxxxN I/O Connection for Discrete I/O

Signal	Front Panel (P1)	P14 I/O Connector	Signal	Front Panel (P1)	P14 I/O Connector
Discrete Input 1	15	35	Discrete Input 2	49	36
Discrete Input 3	16	N/A	Discrete Input 4	50	N/A
...
Discrete Output 1	31	5	Discrete Output 2	65	6
Discrete Output 3	32	N/A	Discrete Output 4	66	N/A

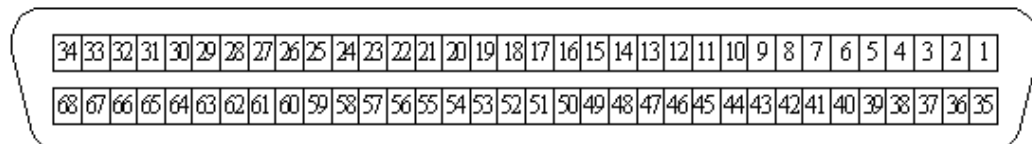


Figure 2. 68-pin Front-Panel Receptacle Connector

IRIG-B Signal Connections

IRIG-B time (AM or DC/TTL) may be received via signals IRIGRX+ and IRIGRX- (see Table 5). The following IRIG formats are accepted.

Table 7. IRIG Signal Formats

Format	Modulation Frequency	Frequency/Resolution	Coded Expressions
B	0, 1	0, 2	0, 1, 2, 3

Upon completion of the program load, the CEI-830 initiates IRIG B002 (DC/TTL) transmission from the onboard IRIG encoder via the IRIGTX signal (see Table 5). The IRIGTX signal can source/sink 16 mA at valid TTL levels.

To externally wrap the IRIG generator to the IRIG receiver, connect the IRIGTX signal to IRIGRX+ input, and connect the IRIGRX- input to Ground.



R830RX

Overview

The R830RX card is a receive-only version of the CEI-830 ARINC interface, in a PMC form-factor. Configurations are available supporting various ARINC 429 channel counts and IRIG time synchronization. A variety of bus adapter configurations are also available, supporting both front and rear-I/O access for PCI, PCI Express, and CompactPCI platforms.

R830RX Specifications

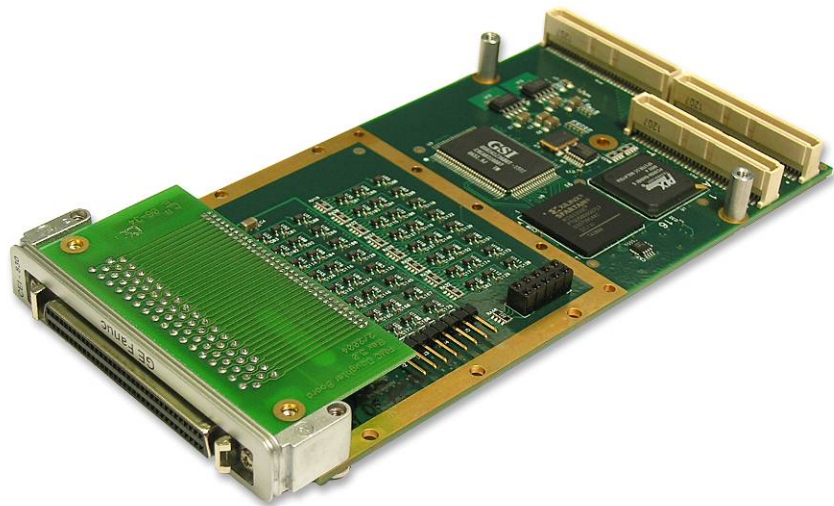


Figure 3. R830RX

The R830RX is a multiple channel ARINC receive-only interface built to the PMC standard IEEE-P1386.1.

PMC/PCI Interface

- Standard single-width CMC module per IEEE-P1386.1 draft standard
- +5V and +3.3V PCI signaling compatibility and universal keying
- 66 MHz, 32 bit PCI operation

Receiver Channels

- Up to thirty-two independent, differential receive channels
- 1024 message buffered mode receive FIFO buffer for each channel
- 16384 message merged mode receive FIFO buffer
- Label/SDI message independent snapshot storage for each channel
- Independent merged/individual receive FIFO buffer operation
- 64-bit, 1 μ sec time-tag is stored with each data element in the FIFO
- Parity error detection

IRIG Input and Output

- IRIG Time-code receiver and transmitter

Typical Power Consumption

Table 8. Power Consumption

+3.3V	5V
250mA	10mA

Operating Temperature

-40 to +85 C

Weight

3.6 ounces, maximum

PCI Memory Map

The following table summarizes the PCI memory map interface definition for the R830RX.

Table 9. R830RX PCI Memory Map

Region	Type	Size	Description
configuration	configuration	64 bytes	PCI configuration space
PCI BAR0	memory	512 bytes	PCI9056 memory-mapped local configuration registers
PCI BAR1	I/O	256 bytes	PCI9056 I/O-mapped local configuration registers, unused
PCI BAR2	memory	512K bytes	R830RX host interface
PCI BAR3	unused	0	not used
PCI BAR4	unused	0	not used
PCI BAR5	unused	0	not used

I/O Connections

Mating Connectors

At publication of this document, the following mating connector was compatible with the 68-pin (P1) SCSI connector provided on the R830RX front panel (bezel). GE Intelligent Platforms Embedded Systems supplies the adapter cable CONSCSI3-6 for this connection.

Table 10. P1 Input/Output Connector

Connector	Part No	Description	Manufacturer
P1	1-5750913-7	Front Panel 68 pin SCSI-3	AMP/Tyco

Input /Output Connector Pin-out

The following table describes both the 68-pin front panel and P14 mezzanine I/O connector pin-out for the R830RX. The exact ARINC 429 channel pin-out depends on the number of receivers configured on your R830RX. The P14 mezzanine I/O connector routes all thirty-two ARINC 429 receive channels. The figure below shows the view facing the receptacles of a 68-pin Front-Panel Receptacle Connector (SCSI-3-compatible with Rails and Latch Blocks).

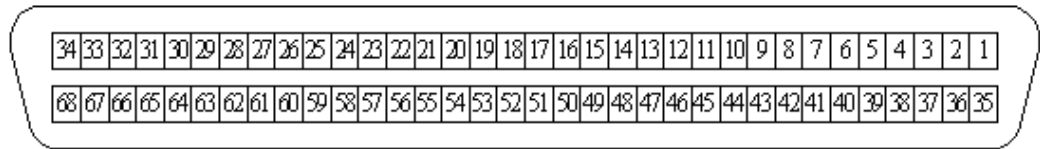


Figure 4. P1 68-pin Front-Panel (Bezel) Connector – View Facing Connector Pins

Table 11. R830RX ARINC I/O Connections

Signal	Front Panel (P1)	P14 I/O Connector	Signal	Front Panel (P1)	P14 I/O Connector
RX1A	1	2	RX1B	35	1
RX2A	2	4	RX2B	36	3
RX3A	3	6	RX3B	37	5
RX4A	4	8	RX4B	38	7
RX5A	5	10	RX5B	39	9
RX6A	6	12	RX6B	40	11
RX7A	7	14	RX7B	41	13
RX8A	8	16	RX8B	42	15
RX9A	9	18	RX9B	43	17
RX10A	10	20	RX10B	44	19
RX11A	11	22	RX11B	45	21
RX12A	12	24	RX12B	46	23
RX13A	13	26	RX13B	47	25
RX14A	14	28	RX14B	48	27
RX15A	15	30	RX15B	49	29
RX16A	16	32	RX16B	50	31
RX17A	17	34	RX17B	51	33
RX18A	18	36	RX18B	52	35
RX19A	19	38	RX19B	53	37
RX20A	20	40	RX20B	54	39
RX21A	21	42	RX21B	55	41
RX22A	22	44	RX22B	56	43
RX23A	23	46	RX23B	57	45
RX24A	24	48	RX24B	58	47
RX25A	25	50	RX25B	59	49
RX26A	26	52	RX26B	60	51
RX27A	27	54	RX27B	61	53
RX28A	28	56	RX28B	62	55
RX29A	29	58	RX29B	63	57
RX30A	30	60	RX30B	64	59
RX31A	31	62	RX31B	65	61

Signal	Front Panel (P1)	P14 I/O Connector	Signal	Front Panel (P1)	P14 I/O Connector
RX32A TXA ¹ IRIGTX+ ² IRIGRX+ ³	32	64	RX32B TXB ¹ IRIGTX- ² IRIGRX- ³	66	63
IRIGRX+	33	N/A	IRIGRX-	67	N/A
Gnd ⁴	34	N/A	Gnd ⁴	68	N/A

Notes:

1. When the jumper pin pairs J4 and J5 are shorted and Bit 4 of the Global Enable Register is set low (0), the on-board transmitter signals TXA and TXB will be enabled and physically shorted to these pins.
2. When the jumper pin pairs for J4 and J5 are shorted and Bit 4 of the Global Enable Register is set high (1), the on-board IRIG Generator signals will be enabled and physically shorted to these pins.
3. Specifically for rear-I/O configurations, when the jumper pin pairs for J2 and J3 are shorted, these I/O pins can then be used for either IRIG time code reception or as the thirty-second ARINC 429 receiver.
4. The ground pins are provided for shielding, as required.

Jumper Connections

The R830RX board contains a set of four special-purpose jumper pin-pairs. Each jumper pin-pair can be shorted using a standard 0.1 inch shunt (not provided); however, the preferred shorting method is wire-wrap or soldered wire. The jumper pin-pairs are located on the component side of the R830RX, labeled J2, J3, J4 and J5.

Jumper pin pairs J2 and J3 are provided specifically for use with IRIG-B signal reception on the R830RX rear-I/O configuration. When the jumper pins across the J2 and J3 pin pairs are shorted, the IRIG Receiver pins IRIGRX+ and IRIGRX- will be physically shorted to P14 pins 63 and 64, respectively. These pins can then be used for either IRIG time code reception or as the thirty-second ARINC 429 receiver, depending on the signal source on the external connection.

Jumper pin pairs J4 and J5 are provided specifically for use with the on-board IRIG generator. When the jumper pins across the J4 and J5 pin pairs are shorted and Bit 4 of the Global Enable Register is set high (1), the on-board IRIG Generator signals + and – will be enabled and physically shorted to ARINC Receive pins RX32A and RX32B, respectively. In this case, pins RX32A and RX32B will not support ARINC 429 reception and should not be connected to an external ARINC transmitter.

An alternate test-only transmitter connection using jumper pin pairs J4 and J5 is also provided. When jumper pin pairs J4 and J5 are shorted and Bit 4 of the Global Enable Register is set low (0), the on-board transmitter signals TXA and TXB will be enabled and physically shorted to ARINC Receive pins RX32A and RX32B, respectively. The output from this *test-only* transmitter is not compliant with ARINC 429 transmit specifications and in this case these I/O pins should not be connected to an off-board ARINC transmitter or receiver.

IRIG-B Signal Connections

IRIG-B time (AM or DC/TTL) may be received via signals IRIGRX+ and IRIGRX-. The following IRIG formats are accepted:

Table 12. IRIG Signal Formats Supported

Format	Modulation Frequency	Frequency/Resolution	Coded Expressions
B	0, 1	0, 2	0, 1, 2, 3

IRIG-B Generator Signal Connections

Upon completion of the program load, the R830RX initiates IRIG B002 (DC/TTL) transmission from the onboard IRIG encoder. In order to use the output from the IRIG-B generator, it must first be enabled by shorting the jumper pins for both J4 and J5 and setting Bit 4 of the Global Enable Register high (1), (see the routine AR_SET_DEVICE_CONFIG option ARU_IRIG_OUTPUT_ENABLE). Once enabled, the IRIGTX+/- signals can source/sink 16 mA at valid TTL levels on these pins; however, pins RX32A/RX32B will be unavailable for use as an ARINC 429 receiver.

IRIG-B Receiver Signal Connections

IRIG reception via front-I/O is always available on P1 pins 33 and 67. IRIG reception via rear-I/O is only available on P14 pins 63 and 64 when J2 and J3 jumper pins are shorted; however, pins 63 and 64 will then be unavailable for use as an ARINC 429 receiver.

To externally wrap the IRIG receiver to the IRIG generator you must first enable the IRIG-B generator as discussed above. You would then either connect the IRIGTX+ signal to IRIGRX+ input and the IRIGTX- signal to the IRIGRX- input (using front-I/O connections); or optionally (and for rear-I/O connections), short jumper pins J2 and J3.



CEI-530

Overview

The CEI-530 card is a multiple-channel ARINC interface, available in several configurations for the PCI platform. When configured as a CEI-530-1616, this card includes thirty-two ARINC 429 channels, (sixteen receivers and sixteen transmitters.) There are a variety of configurations available supporting various ARINC 429 channel counts, ARINC 573/717, Avionics Discrete I/O, and IRIG time synchronization.

CEI-530 Specifications

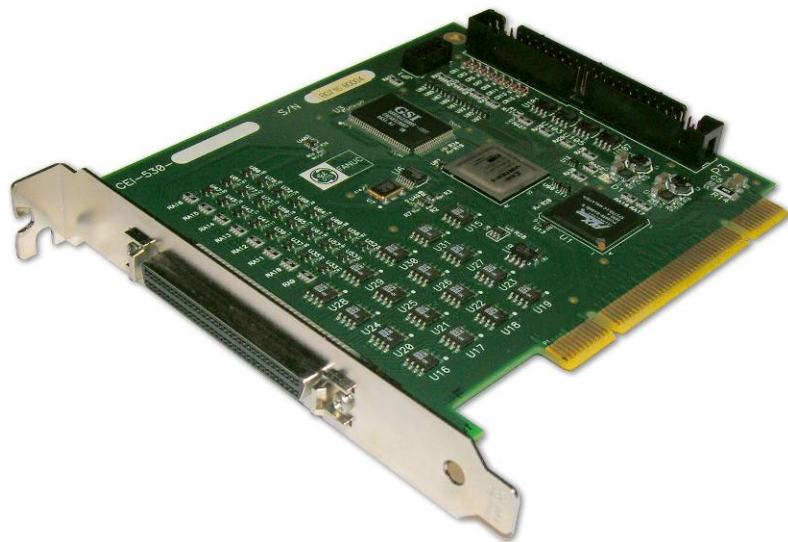


Figure 5. CEI-530

The CEI-530 is a multiple channel, multiple protocol interface built to the PCI Local Bus Specification, version 2.2.

PCI Interface

- Standard PCI Interface per the PCI Local Bus Specification Revision 2.2
- +5V and +3.3V PCI signaling compatibility and universal keying
- 66 MHz, 32 bit PCI operation

Transmit Channels

- Up to sixteen independent differential serial transmit channels
- Automatic parity generation
- 2048 message transmit buffer for each channel
- Baud rate/slew rate software-programmable for each channel
- 1024 entry message table supporting scheduled message transmission for all channels

Receiver Channels

- Up to sixteen independent, differential receive channels
- 2048 message buffered mode receive FIFO buffer for each channel
- 16384 message merged mode receive FIFO buffer
- Label/SDI message independent snapshot storage for each channel
- Independent merged/individual receive FIFO buffer operation
- 64-bit, 1 μ sec time-tag is stored with each data element in the FIFO
- Parity error detection

Avionics Discrete Input and Output

- Up to sixteen individual, avionics-level input and output discrete channels
- Output may switch to ground up to 500mA
- Power-up / reset default inactive
- Fixed input threshold of 2.7 +/- 0.2 volts

IRIG Input and Output

- IRIG Timecode receiver and transmitter

Typical Power Consumption

Table 13. Power Consumption

+3.3V	5V	+12V	-12V
500 mA	50 mA	100 mA (no TX Loads) 350mA (sixteen transmitters, max data rate, 400Ω load each)	100 mA (no TX Loads) 350mA (sixteen transmitters, max data rate, 400Ω load each)

Operating Temperature

-40 to +85 C

Weight

3.6 ounces, maximum

PCI Memory Map

The following table summarizes the PCI memory map interface definition for the CEI-530.

Table 14. CEI-530 PCI Memory Map

Region	Type	Size	Description
configuration	configuration	64 bytes	PCI configuration space
PCI BAR0	memory	512 bytes	PCI9056 memory-mapped local configuration registers
PCI BAR1	I/O	256 bytes	PCI9056 I/O-mapped local configuration registers, unused
PCI BAR2	memory	512K bytes	CEI-530 host interface
PCI BAR3	unused	0	not used
PCI BAR4	unused	0	not used
PCI BAR5	unused	0	not used

I/O Connections

CEI-530 Outline Drawing

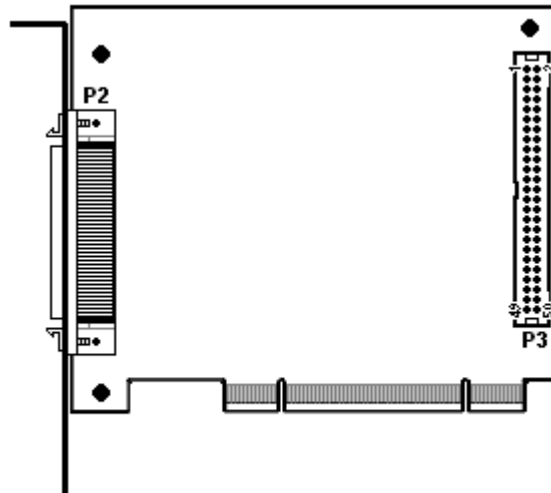


Figure 6. CEI-530 Outline Drawing

Mating Connectors

At publication of this document, the following mating connector was compatible with the 68-pin (P2) SCSI connector provided on the CEI-530 front panel (bezel). GE Intelligent Platforms Embedded Systems supplies the adapter cable CONSCSI3-6 for this connection.

Table 15. P2 Input/Output Connector

Connector	Part No	Description	Manufacturer
P2	1-5750913-7	Front Panel 68 pin SCSI-3	AMP/Tyco

At publication of this document, the following mating connectors were compatible with the 50-pin (P3) IDC ribbon connector used for the CEI-530 Discrete and IRIG I/O, located towards the rear of the CEI-530 PCB.

Table 16. P3 Input/Output Connector

Connector	Part No	Description	Manufacturer
P3	1-1658622-0	50 pin Ribbon 0.100 Centers	AMP/Tyco
	3425-6650	50 pin Ribbon 0.100 Centers	3M

ARINC Input /Output Connector Pin-out

Various CEI-530 product configurations have specific channel pin-out definitions based on the number of channels and protocols included. The following table describes the front panel connector pin layout for the CEI-530. The exact ARINC 429 channel pin-out depends on the number of receivers and transmitters configured on your CEI-530. For the -J version of the CEI-530, the ARINC 573/717 protocol support pins replace the channel 16 ARINC 429 I/O pins. Note the CEI-530 I/O pin assignments are pin-compatible with the CEI-520 I/O pin assignments.

To externally wrap ARINC signals, connect the transmitter signals to the respective receiver signals, TXnA to RXnA and TXnB to RXnB.

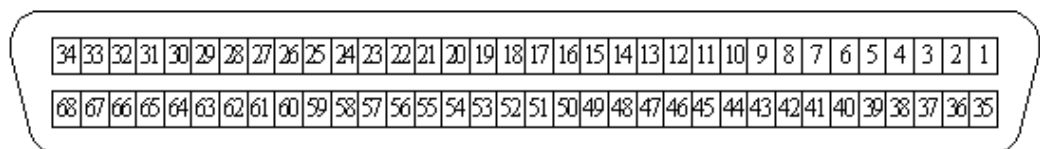


Figure 7. 68-pin Front-Panel (Bezel) Connector – View Facing Connector Pins

Table 17. CEI-530 Front Panel ARINC I/O Connections

Signal	P2 Pin	Signal	P2 Pin
TX1A	1	TX1B	35
TX2A	2	TX2B	36
TX3A	3	TX3B	37
TX4A	4	TX4B	38
TX5A	5	TX5B	39
TX6A	6	TX6B	40
TX7A	7	TX7B	41
TX8A	8	TX8B	42
TX9A	9	TX9B	43
TX10A	10	TX10B	44
TX11A	11	TX11B	45
TX12A	12	TX12B	46
TX13A	13	TX13B	47
TX14A	14	TX14B	48
TX15A	15	TX15B	49
TX16A ²	16	TX16B ²	50
Ground ¹	17	Ground ¹	51
RX1A	18	RX1B	52
RX2A	19	RX2B	53
RX3A	20	RX3B	54

Signal	P2 Pin	Signal	P2 Pin
RX4A	21	RX4B	55
RX5A	22	RX5B	56
RX6A	23	RX6B	57
RX7A	24	RX7B	58
RX8A	25	RX8B	59
Ground ¹	26	Ground ¹	60
RX9A	27	RX9B	61
RX10A	28	RX10B	62
RX11A	29	RX11B	63
RX12A	30	RX12B	64
RX13A	31	RX13B	65
RX14A	32	RX14B	66
RX15A	33	RX15B	67
RX16A ²	34	RX16B ²	68

- Notes:
- 1 The ground pins are provided for shielding, as required.
 - 2 The ARINC 573/717 signals are supported on the respective channel 16 pins for both the BPRZ and HBP protocols. The selected protocol processed is based on the selection of the ARINC 717 HBP and BPRZ Encoding bits in the respective Transmit Channel Configuration register. See the API routine `AR_SET_573_CONFIG` for the method to define the ARINC 573/717 active encoding selection for this output pin.

Discrete and IRIG Input/Output Connector Pin-out

All CEI-530 product configurations have the same Discrete and IRIG I/O pin-out definition for the IDC-50 I/O connector located at the rear of the board, pin-compatible with the CEI-520 IDC-50 Discrete I/O pin-out.

To externally wrap Discrete I/O channels, connect the output pins to the respective input pins, DOUTn to DINn.

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

Figure 8. 50-pin IDC-50 I/O Connector – View Facing Connector Pins

Table 18. CEI-530 IDC-50 I/O Connections

Signal	P3 Pin	Signal	P3 Pin
DIN1	1	DOUT8	26
DIN2	2	DOUT9	27
DIN3	3	DOUT10	28
DIN4	4	DOUT11	29
DIN5	5	DOUT12	30
DIN6	6	DOUT13	31
DIN7	7	DOUT14	32
DIN8	8	DOUT15	33
DIN9	9	DOUT16	34
DIN10	10	Ground	35
DIN11	11	Ground	36
DIN12	12	IRIGRX+	37
DIN13	13	IRIGRX-	38
DIN14	14	IRIGTX	39
DIN15	15	Ground	40
DIN16	16	Ground	41
Ground	17	Ground	42
Ground	18	N/C	43
DOUT1	19	N/C	44
DOUT2	20	N/C	45
DOUT3	21	N/C	46
DOUT4	22	N/C	47
DOUT5	23	N/C	48
DOUT6	24	N/C	49
DOUT7	25	Ground	50

Note: The ground pins are provided as Discrete I/O return lines or for shielding, as required.

IRIG-B Signal Connections

IRIG-B time (AM or DC/TTL) may be received via signals IRIGRX+ and IRIGRX- (Table 31). The following IRIG formats are accepted:

Table 19. IRIG Signal Connections

Format	Modulation Frequency	Frequency/Resolution	Coded Expressions
B	0, 1	0, 2	0, 1, 2, 3

Upon completion of the program load, the CEI-530 initiates IRIG B002 (DC/TTL) transmission from the onboard IRIG encoder via the IRIGTX signal (Table 32). The IRIGTX signal can source/sink 16 mA at valid TTL levels.

To externally wrap the IRIG generator to the IRIG receiver, connect the IRIGTX signal to IRIGRX+ input, and connect the IRIGRX- input to Ground.



RAR-PCIE

Overview

The RAR-PCIE card is a multiple-channel ARINC interface, available in several configurations for the PCI Express platform. When configured as a RAR-PCIE-1616, this card includes thirty-two ARINC 429 channels, (sixteen receivers and sixteen transmitters.) There are a variety of configurations available supporting various ARINC 429 channel counts, ARINC 573/717, Avionics Discrete I/O, and IRIG time synchronization.

RAR-PCIE Specifications



Figure 9. RAR-PCIE

The RAR-PCIE is a multiple channel, multiple protocol interface built to the PCI Express Base Specification 2.0.

PCI Express Interface

- Standard PCI Express Interface per the PCI Express Card Electromechanical Specification Revision 2.0

Transmit Channels

- Up to sixteen independent differential serial transmit channels
- Automatic parity generation
- 2048 message transmit buffer for each channel
- Baud rate/slew rate software-programmable for each channel
- Each channel includes tri-state capability
- 1024 entry message table supporting scheduled message transmission for all channels

Receiver Channels

- Up to sixteen independent, differential receive channels
- 2048 message buffered mode receive FIFO buffer for each channel
- 16384 message merged mode receive FIFO buffer
- Label/SDI message independent snapshot storage for each channel
- Independent merged/individual receive FIFO buffer operation
- 64-bit, 1 μ sec time-tag is stored with each data element in the FIFO
- Parity error detection

Avionics Discrete Input and Output

- Up to sixteen individual, avionics-level input and output discrete channels
- Output may switch to ground up to 500mA
- Power-up / reset default inactive
- Fixed input threshold of 2.0 +/- 0.2 volts

IRIG Input and Output

- IRIG Timecode receiver and transmitter

Typical Power Consumption

Table 20. Power Consumption

+3.3V	+12V
600 mA	140 mA (no TX Loads)
	Note: Each additional mA of transmit load current will add an additional mA of +12V supply current.

Operating Temperature

-40 to +75 C

Weight

3.8 ounces

PCI Memory Map

The following table summarizes the PCI memory map interface definition for the RAR-PCIE.

Table 21. RAR-PCIE PCI Memory Map

Region	Type	Size	Description
configuration	configuration	64 bytes	PCI configuration space
PCI BAR0	memory	512K bytes	RAR-PCIE host interface
PCI BAR1	unused	0	not used
PCI BAR2	unused	0	not used
PCI BAR3	unused	0	not used
PCI BAR4	unused	0	not used
PCI BAR5	unused	0	not used

I/O Connections

RAR-PCIE Outline Drawing

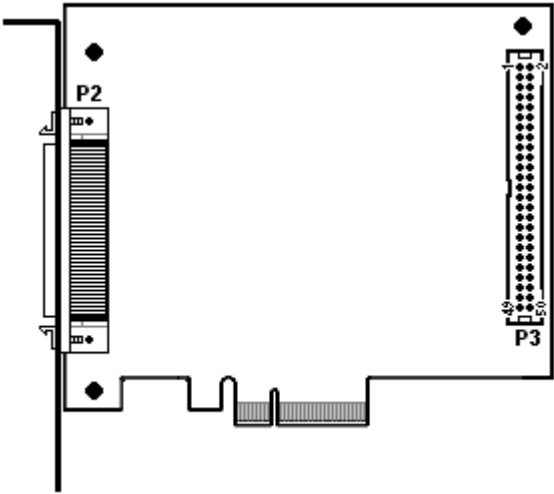


Figure 10. RAR-PCIE Outline Drawing

Mating Connectors

At publication of this document, the following mating connector was compatible with the 68-pin (P2) SCSI connector provided on the RAR-PCIE front panel (bezel). GE Intelligent Platforms Embedded Systems supplies the adapter cable CONSCSI3-6 for this connection.

Table 22. P2 Input/Output Connector

Connector	Part No	Description	Manufacturer
P2	1-5750913-7	Front Panel 68 pin SCSI-3	AMP/Tyco

At publication of this document, the following mating connectors were compatible with the 50-pin (P3) IDC ribbon connector used for the RAR-PCIE Discrete and IRIG I/O, located towards the rear of the RAR-PCIE PCB.

Table 23. P3 Input/Output Connector

Connector	Part No	Description	Manufacturer
P3	1-1658622-0	50 pin Ribbon 0.100 Centers	AMP/Tyco
	3425-6650	50 pin Ribbon 0.100 Centers	3M

ARINC Input/Output Connector Pin-out

Various RAR-PCIE product configurations have specific channel pin-out definitions based on the number of channels and protocols included. The following table describes the front panel connector pin layout for the RAR-PCIE. The exact ARINC 429 channel pin-out depends on the number of receivers and transmitters configured on your RAR-PCIE. For the -J version of the RAR-PCIE, the ARINC 573/717 protocol support pins replace the channel 16 ARINC 429 I/O pins. Note the RAR-PCIE I/O pin assignments are pin-compatible with the CEI-520 and CEI-530 I/O pin assignments.

To externally wrap ARINC signals, connect the transmitter signals to the respective receiver signals, TXnA to RXnA and TXnB to RXnB.

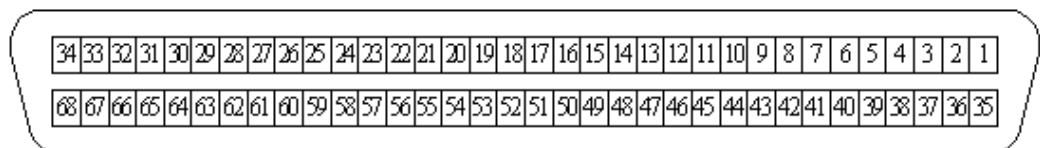


Figure 11. 68-pin Front-Panel (Bezel) Connector – View Facing Connector Pins

Table 24. RAR-PCIE Front Panel ARINC I/O Connections

Signal	P2 Pin	Signal	P2 Pin
TX1A	1	TX1B	35
TX2A	2	TX2B	36
TX3A	3	TX3B	37
TX4A	4	TX4B	38
TX5A	5	TX5B	39
TX6A	6	TX6B	40
TX7A	7	TX7B	41
TX8A	8	TX8B	42
TX9A	9	TX9B	43
TX10A	10	TX10B	44
TX11A	11	TX11B	45
TX12A	12	TX12B	46
TX13A	13	TX13B	47
TX14A	14	TX14B	48
TX15A	15	TX15B	49
TX16A ²	16	TX16B ²	50
Ground ¹	17	Ground ¹	51
RX1A	18	RX1B	52
RX2A	19	RX2B	53

Signal	P2 Pin	Signal	P2 Pin
RX3A	20	RX3B	54
RX4A	21	RX4B	55
RX5A	22	RX5B	56
RX6A	23	RX6B	57
RX7A	24	RX7B	58
RX8A	25	RX8B	59
Ground ¹	26	Ground ¹	60
RX9A	27	RX9B	61
RX10A	28	RX10B	62
RX11A	29	RX11B	63
RX12A	30	RX12B	64
RX13A	31	RX13B	65
RX14A	32	RX14B	66
RX15A	33	RX15B	67
RX16A ²	34	RX16B ²	68

- Notes:
- 1 The ground pins are provided for shielding, as required.
 - 2 The ARINC 573/717 signals are supported on the respective channel 16 pins for both the BPRZ and HBP protocols. The selected protocol processed is based on the selection of the ARINC 717 HBP and BPRZ Encoding bits in the respective Transmit Channel Configuration register. See the API routine `AR_SET_573_CONFIG` for the method to define the ARINC 573/717 active encoding selection for this output pin.

Discrete and IRIG Input/Output Connector Pin-out

All RAR-PCIE product configurations have the same Discrete and IRIG I/O pin-out definition for the IDC-50 I/O connector located at the rear of the board, pin-compatible with the CEI-520 and CEI-530 IDC-50 Discrete I/O pin-out.

To externally wrap Discrete I/O channels, connect the output pins to the respective input pins, DOUTn to DINn.

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

Figure 12. 50-pin IDC-50 I/O Connector – View Facing Connector Pins

Table 25. RAR-PCIE IDC-50 I/O Connections

Signal	P3 Pin	Signal	P3 Pin
DIN1	1	DOUT8	26
DIN2	2	DOUT9	27
DIN3	3	DOUT10	28
DIN4	4	DOUT11	29
DIN5	5	DOUT12	30
DIN6	6	DOUT13	31
DIN7	7	DOUT14	32
DIN8	8	DOUT15	33
DIN9	9	DOUT16	34
DIN10	10	Ground	35
DIN11	11	Ground	36
DIN12	12	IRIGRX+	37
DIN13	13	IRIGRX-	38
DIN14	14	IRIGTX	39
DIN15	15	Ground	40
DIN16	16	Ground	41
Ground	17	Ground	42
Ground	18	N/C	43
DOUT1	19	N/C	44
DOUT2	20	N/C	45
DOUT3	21	N/C	46
DOUT4	22	N/C	47
DOUT5	23	N/C	48
DOUT6	24	N/C	49
DOUT7	25	Ground	50

Note: The ground pins are provided as Discrete I/O return lines or for shielding, as required.

IRIG-B Signal Connections

IRIG-B time (AM or DC/TTL) may be received via signals IRIGRX+ and IRIGRX- (Table 31). The following IRIG formats are accepted:

Table 26. IRIG Signal Connections

Format	Modulation Frequency	Frequency/Resolution	Coded Expressions
B	0, 1	0, 2	0, 1, 2, 3

Upon completion of power-up initialization, the RAR-PCIE initiates IRIG B002 (DC/TTL) transmission from the onboard IRIG encoder via the IRIGTX signal (Table 32). The IRIGTX signal can source/sink 16 mA at valid TTL levels.

To externally wrap the IRIG generator to the IRIG receiver, connect the IRIGTX signal to IRIGRX+ input, and connect the IRIGRX- input to Ground.



CEI-430

Overview

The CEI-430 card is a multiple-channel ARINC interface, available in several configurations for the PC/104-*Plus* platform. When configured as the CEI-430-1212, this card includes twenty-four ARINC 429 channels, (twelve receivers and twelve transmitters). There are many configurations available supporting various ARINC 429 channel counts, ARINC 573/717, Avionics Discrete I/O, Differential Discrete I/O, and IRIG time synchronization.

CEI-430 Specifications

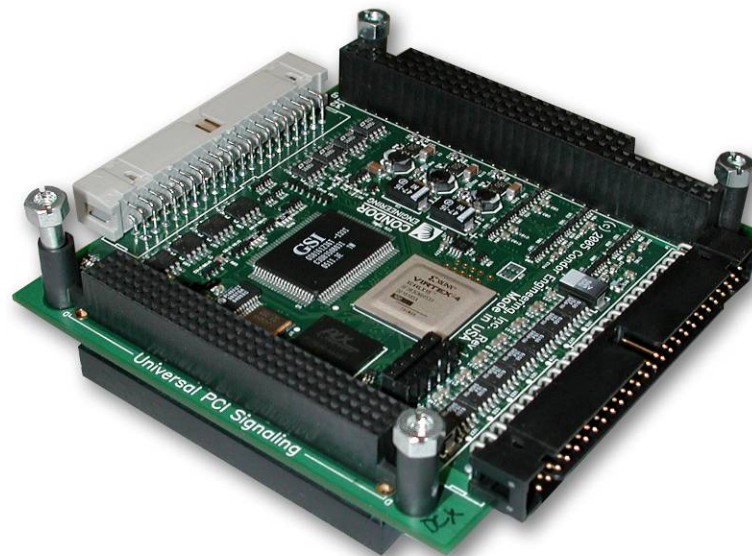


Figure 13. CEI-430

The CEI-430 is a multiple channel, multiple protocol interface built to the PC/104 Specification, version 2.5, PC/104-*Plus* Specification, version 2.0, and PCI-104 Specification, version 1.0.

PCI Interface

- Standard PCI Interface per the PCI Local Bus Specification Revision 2.2
- +5V and +3.3V PCI signaling compatibility and universal keying
- 33 MHz, 32 bNote:

Per the PC/104-*Plus* Specification, the PCI Stack location of the CEI-430 is determined by card jumper shunts S0/S1. Refer to Table 27 for jumper mapping.

Table 27. PCI Stack Location Shunts

Stack Location	Installed Shunts	
	S1	S0
1	IN	IN
2	IN	OUT
3	OUT	IN
4	OUT	OUT

Transmit Channels

- Up to twelve independent differential serial transmit channels
- Automatic parity generation
- 2048 message transmit buffer for each channel
- Baud rate/slew rate software-programmable for each channel
- 1024 entry message table supporting scheduled message transmission for all channels

Receiver Channels

- Up to twelve independent, differential receive channels
- 2048 message buffered mode receive FIFO buffer for each channel
- 16384 message merged mode receive FIFO buffer
- Label/SDI message independent snapshot storage for each channel
- Independent merged/individual receive FIFO buffer operation
- 64-bit, 1 μ sec time-tag is stored with each data element in the FIFO
- Parity error detection

Avionics Discrete Input and Output

- Up to sixteen bi-directional, avionics-level discrete channels
- Output may switch to ground up to 500mA
- Power-up / reset default inactive
- Fixed input threshold of 2.7 +/- 0.2 volts

Differential Discrete Input and Output

- Up to four RS-485 discrete channels
- Power-up / reset default inactive

IRIG Input and Output

- IRIG Time-code receiver and transmitter

Typical Power Consumption

Table 28. Power Consumption

5V	12V	-12V
300 mA	80 mA (no TX Loads) 200mA (twelve transmitters, maximum data rate, 400 Ω load each)	80 mA (no TX Loads) 200mA (twelve transmitters, maximum data rate, 400 Ω load each)

Operating Temperature

-40 to +85 C

Weight

3.6 ounces, maximum

PCI Memory Map

The following table summarizes the PCI memory map interface definition for the CEI-430.

Table 29. CEI-430 PCI Memory Map

Region	Type	Size	Description
configuration	configuration	64 bytes	PCI configuration space
PCI BAR0	memory	128 bytes	PCI9030 memory-mapped local configuration registers
PCI BAR1	unused	0	PCI9030 I/O-mapped local configuration registers, unused
PCI BAR2	memory	512K bytes	CEI-430 host interface
PCI BAR3	unused	0	not used
PCI BAR4	unused	0	not used
PCI BAR5	unused	0	not used

I/O Connections

CEI-430 Outline Drawing

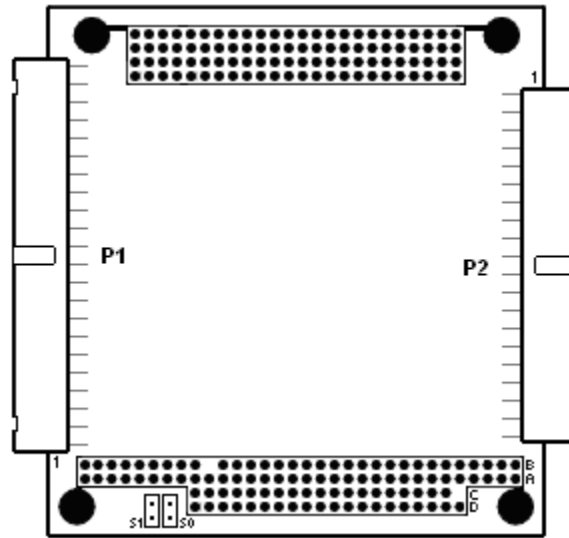


Figure 14. CEI-430 Outline Drawing

Input/Output Connectors

At publication of this document, the following mating connectors were compatible with the 50-pin (P1) and 40-pin (P2) IDC ribbon connectors used on the CEI-430. One each mating connector and retaining clip is provided for these connectors.

Table 30. CEI-430 Input/Output Connectors

Connector	Part No	Description	Manufacturer
P1	1-1658622-0	50 pin Ribbon 0.100 Centers	AMP/Tyco
	3425-6650	50 pin Ribbon 0.100 Centers	3M
P2	1-1658622-9	40 pin Ribbon 0.100 Centers	AMP/Tyco
	3417-6640	40 pin Ribbon 0.100 Centers	3M

Input/Output Connector Pin-out

Various CEI-430 product configurations have specific channel pin-out definitions based on the number of channels and protocols included. Table 28 describes the P1 connector pin layout for the CEI-430 module. The exact ARINC 429 channel pin-out depends on the number of receivers and

transmitters configured on your CEI-430. Note that for the -J version of the CEI-430, ARINC 573/717 protocol support replaces ARINC 429 Channel 12.

To externally wrap ARINC signals, connect the transmitter signals to the respective receiver signals, TXnA to RXnA and TXnB to RXnB.

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

Figure 15. P1 50-pin IDC ARINC Interface Connector – View Facing Connector Pins

Table 31. CEI-430 P1 ARINC I/O Connections

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

Notes:

1. For –J configurations, the ARINC 573/717 protocol replaces ARINC 429 channel 12. The selected protocol processed is based on the selection of the ARINC 717 HBP and BPRZ Encoding bits in the respective Transmit Channel Configuration register. See the API routine AR_SET_573_CONFIG for the method to define the ARINC 573/717 active encoding selection for these output pins.
2. The ground pins are provided for shielding, as necessary

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

Figure 16. P2 40-pin IDC I/O Connector – View Facing Connector Pins**Table 32. CEI-430 P2 I/O Connections**

Signal	P2 Pin	Signal	P2 Pin
Ground	1	Ground	2
ADISC1	3	ADISC2	4
ADISC3	5	ADISC4	6
ADISC5	7	ADISC6	8
ADISC7	9	ADISC8	10
ADISC9	11	ADISC10	12
ADISC11	13	ADISC12	14
ADISC13	15	ADISC14	16
ADISC15	17	ADISC16	18
Ground	19	Ground	20
DIFF1+	21	DIFF1-	22
DIFF2+	23	DIFF2-	24
DIFF3+	25	DIFF3-	26
DIFF4+	27	DIFF4-	28
Ground	29	Ground	30
IRIGRX+	31	IRIGRX-	32
IRIGTX	33	GND	34
Reserved	35	Reserved	36
Reserved	37	Reserved	38
Reserved	39	Reserved	40

Note:

The ground pins are provided as Discrete I/O return lines or for shielding, as required.

IRIG-B Signal Connections

IRIG-B time (AM or DC/TTL) may be received via signals IRIGRX+ and IRIGRX- (Table 31). The following IRIG formats are accepted:

Table 33. IRIG Signal Connections

Format	Modulation Frequency	Frequency/Resolution	Coded Expressions
B	0, 1	0, 2	0, 1, 2, 3

Upon completion of the program load, the CEI-430 initiates IRIG B002 (DC/TTL) transmission from the onboard IRIG encoder via the IRIGTX signal (Table 32). The IRIGTX signal can source/sink 16 mA at valid TTL levels.

To externally wrap the IRIG generator to the IRIG receiver, connect the IRIGTX signal to IRIGRX+ input, and connect the IRIGRX- input to Ground.



CEI-430A

Overview

The CEI-430A card is a multiple-channel ARINC 429 interface version of the CEI-430 with an emphasis on high ARINC 429 receive channel count, designed for the PC/104-*Plus* platform. When configured as the CEI-430-2404, this card includes twenty-eight ARINC 429 channels, (twenty-four receivers and four transmitters), and Avionics Discrete I/O. Configurations are available with additional support for ARINC 573/717 and IRIG time synchronization.

CEI-430A Specifications



Figure 17. CEI-430A

The CEI-430A is a multiple channel, multiple protocol interface built to the PC/104 Specification, version 2.5, PC/104-*Plus* Specification, version 2.0, and PCI-104 Specification, version 1.0.

PCI Interface

- Standard PCI Interface per the PCI Local Bus Specification Revision 2.2
- +5V and +3.3V PCI signaling compatibility and universal keying
- 33 MHz, 32 bit PCI operation

Note: Per the PC/104-*Plus* Specification, the PCI Stack location of the CEI-430 is determined by card jumper shunts S0/S1. Refer to Table 34 for jumper mapping.

Table 34. PCI Stack Location Shunts

Stack Location	Installed Shunts	
	S1	S0
1	IN	IN
2	IN	OUT
3	OUT	IN
4	OUT	OUT

Transmit Channels

- Two or four independent differential serial transmit channels
- Optional dedicated ARINC 573 / 717 transmit channel
- Automatic parity generation
- 2048 message transmit buffer for each channel
- Baud rate/slew rate software-programmable for each channel
- 1024 entry message table supporting scheduled message transmission for all channels

Receiver Channels

- Twenty-four independent, differential receive channels
- Optional dedicated ARINC 573 / 7171 receive channel
- 2048 message buffered mode receive FIFO buffer for each channel
- 16384 message merged mode receive FIFO buffer
- Label/SDI message independent snapshot storage for each channel
- Independent merged/individual receive FIFO buffer operation
- 64-bit, 1 μ sec time-tag is stored with each data element in the FIFO
- Parity error detection

Avionics Discrete Input and Output

- Sixteen bi-directional, avionics-level discrete channels
- Output may switch to ground up to 500mA
- Power-up / reset default inactive
- Fixed input threshold of 2.7 +/- 0.2 volts

IRIG Input and Output

- IRIG Time-code receiver and transmitter

Typical Power Consumption

Table 35. Power Consumption

5V
300 mA (no TX loads)
500 mA (four transmitters, 100KHz data rate, 400 Ω load each)

Operating Temperature

-40 to +85 C

Weight

3.6 ounces, maximum

PCI Memory Map

The following table summarizes the PCI memory map interface definition for the CEI-430A.

Table 36. CEI-430A PCI Memory Map

Region	Type	Size	Description
configuration	configuration	64 bytes	PCI configuration space
PCI BAR0	memory	128 bytes	PCI9030 memory-mapped local configuration registers
PCI BAR1	unused	0	PCI9030 I/O-mapped local configuration registers, unused
PCI BAR2	memory	512K bytes	CEI-430A host interface
PCI BAR3	unused	0	not used
PCI BAR4	unused	0	not used
PCI BAR5	unused	0	not used

I/O Connections

CEI-430A Outline Drawing

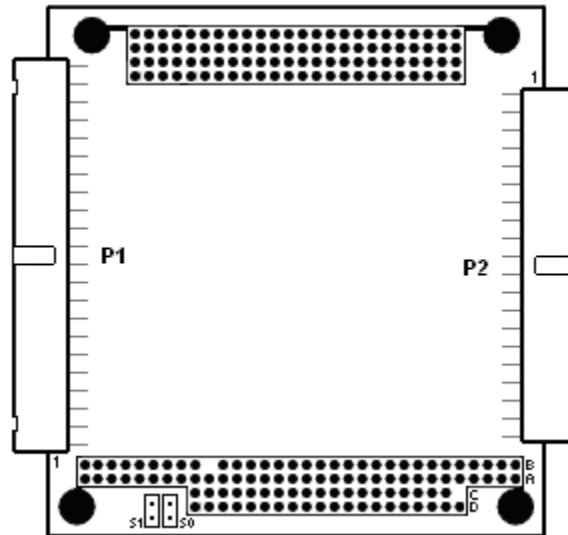


Figure 18. CEI-430A Outline Drawing

Input / Output Connectors

At publication of this document, the following mating connectors were compatible with the 50-pin (P1) and 40-pin (P2) IDC ribbon connectors used on the CEI-430A. One each mating connector and retaining clip is provided for these connectors.

Table 37. CEI-430 Input/Output Connectors

Connector	Part No	Description	Manufacturer
P1	1-1658622-0	50 pin Ribbon 0.100 Centers	AMP/Tyco
	3425-6650	50 pin Ribbon 0.100 Centers	3M
P2	1-1658622-9	40 pin Ribbon 0.100 Centers	AMP/Tyco
	3417-6640	40 pin Ribbon 0.100 Centers	3M

Input / Output Connector Pin-out

Tables 38 and 39 describe the connector pin layout for the CEI-430A module. The exact pin-out for the P2 connector depends on the number of ARINC 429 channels, ARINC 717 support, and IRIG support, configured on your CEI-430A.

To externally wrap ARINC signals, connect the P2 transmitter signals to the respective receiver signals, TXnA to RXnA and TXnB to RXnB.

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

Figure 19. P1 50-pin IDC Interface Connector – View Facing Connector Pins

Table 38. CEI-430A P1 ARINC I/O Connections

Signal	P1 Pin	Signal	P1 Pin
RX1A	1	RX1B	2
RX2A	3	RX2B	4
RX3A	5	RX3B	6
RX4A	7	RX4B	8
RX5A	9	RX5B	10
RX6A	11	RX6B	12
RX7A	13	RX7B	14
RX8A	15	RX8B	16
RX9A	17	RX9B	18
RX10A	19	RX10B	20
RX11A	21	RX11B	22
RX12A	23	RX12B	24
Ground	25	Ground	26
RX13A	27	RX13B	28
RX14A	29	RX14B	30
RX15A	31	RX15B	32
RX16A	33	RX16B	34
RX17A	35	RX17B	36
RX18A	37	RX18B	38
RX19A	39	RX19B	40
RX20A	41	RX20B	42
RX21A	43	RX21B	44
RX22A	45	RX22B	46
RX23A	47	RX23B	48
RX24A	49	RX24B	50

Notes:

The ground pins are provided for shielding, as necessary

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

Figure 20. P2 40-pin IDC I/O Connector – View Facing Connector Pins

Table 39. CEI-430A P2 I/O Connections

Signal	P2 Pin	Signal	P2 Pin
Ground	1	Ground	2
ADISC1	3	ADISC2	4
ADISC3	5	ADISC4	6
ADISC5	7	ADISC6	8
ADISC7	9	ADISC8	10
ADISC9	11	ADISC10	12
ADISC11	13	ADISC12	14
ADISC13	15	ADISC14	16
ADISC15	17	ADISC16	18
Ground	19	Ground	20
429TX1A	21	429TX1B	22
429TX2A	23	429TX2B	24
429TX3A	25	429TX3B	26
429TX4A	27	429TX4B	28
Ground	29	Ground	30
IRIGRX+	31	IRIGRX-	32
IRIGTX	33	GND	34
717RXA	35	717RXB	36
NC	37	717TXB	38
NC	39	717TXA	40

Note: The ground pins are provided as Discrete I/O return lines or for shielding, as required.

Pins 37 and 39 have no connection.

IRIG-B Signal Connections

IRIG-B time (AM or DC/TTL) may be received via signals IRIGRX+ and IRIGRX- (Table 40). The following IRIG formats are accepted:

Table 40. IRIG Signal Connections

Format	Modulation Frequency	Frequency/Resolution	Coded Expressions
B	0, 1	0, 2	0, 1, 2, 3

Upon completion of the program load, the CEI-430 initiates IRIG B002 (DC/TTL) transmission from the onboard IRIG encoder via the IRIGTX signal (Table 32). The IRIGTX signal can source/sink 16 mA at valid TTL levels.

To externally wrap the IRIG generator to the IRIG receiver, connect the IRIGTX signal to IRIGRX+ input, and connect the IRIGRX- input to Ground.



AMC-A30

Overview

The AMC-A30 product line is a multiple-channel ARINC interface, available in several configurations. When configured as the AMC-A30-1414, this product includes twenty-eight ARINC 429 channels, (fourteen receivers and fourteen transmitters), in an AMC form-factor. There are a variety of configurations available supporting various ARINC 429 channel counts, ARINC 573/717, Avionics Discrete I/O, and IRIG time synchronization.

AMC-A30 Specifications

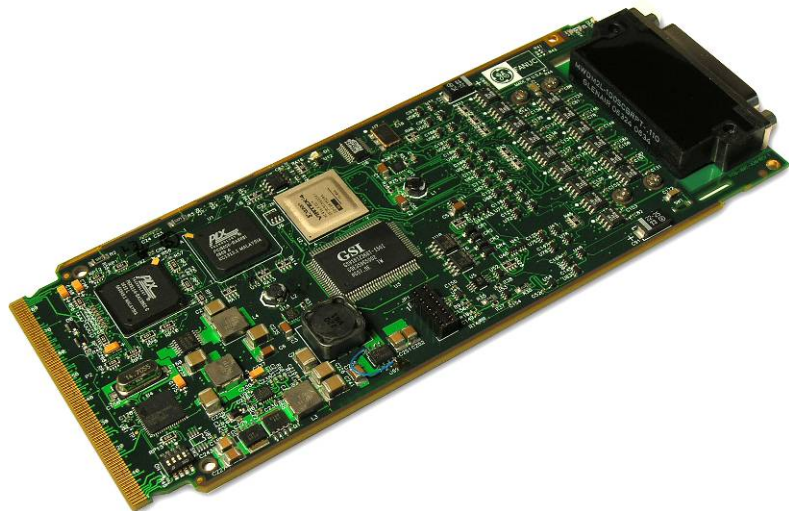


Figure 21. AMC-A30

The AMC-A30 is a multiple channel, multiple protocol interface built to the AMC.1 specification.

AMC/PCIe Interface

- Half-height, single-width AMC form factor
- AMC.1 compliant
- PCI Express (x4 lane) host interface

Transmit Channels

- Up to fourteen independent differential serial transmit channels
- Automatic parity generation
- 2048 message transmit buffer for each channel
- Baud rate/slew rate software-programmable for each channel
- 1024 entry message table supporting scheduled message transmission for all channels

Receiver Channels

- Up to fourteen independent, differential receive channels
- 2048 message buffered mode receive FIFO buffer for each channel.
- 16384 message merged mode receive FIFO buffer
- Label/SDI message independent snapshot storage for each channel
- Independent merged/individual receive FIFO buffer operation.
- 64-bit, 1 μ sec time-tag is stored with each data element in the FIFO
- Parity error detection

Avionics Discrete Input and Output

- Four bi-directional, avionics-level discrete channels
- Output may switch to ground up to 500mA
- Fixed input threshold of 2.7 +/- 0.2 volts

IRIG Input and Output

- IRIG Time-code receiver and transmitter

Typical Power Consumption

Table 41. Payload Power Consumption

+12V
370 mA (no TX Loads)

Operating Temperature

-40 to +85 C

Weight

4.8 ounces

PCI Memory Map

The following table summarizes the PCI memory map interface definition for the AMC-A30.

Table 42. AMC-A30 PCI Memory Map

Region	Type	Size	Description
configuration	configuration	64 bytes	PCI configuration space
PCI BAR0	memory	512 bytes	PCI9056 memory-mapped local configuration registers
PCI BAR1	I/O	256 bytes	PCI9056 I/O-mapped local configuration registers, unused
PCI BAR2	memory	512K bytes	AMC-A30 host interface
PCI BAR3	unused	0	not used
PCI BAR4	unused	0	not used
PCI BAR5	unused	0	not used

I/O Connections

Input/Output Connectors

At publication of this document, the following mating connector was compatible with the 100-pin Micro-D connector used on the AMC-A30.

Table 43. Input/Output Connectors

Part No.	Description	Manufacturer
MWDM2L-100PSS	100-pin Micro-D Connector	Glenair

Input/Output Connector Pin-out

The different AMC-A30 product configurations have specific channel pin-out definitions based on the number of channels and protocols installed. Table 44 describes the 100-pin front panel for the ARINC 429 version of the AMC-A30 module. The exact ARINC 429 channel pin-out depends on the number of receivers and transmitters configured on your AMC-A30. Table 45 describes the pin-out differences for the -J version of the AMC-A30; these pins support ARINC 573/717 protocols on the pins used by the upper channels on non-J configurations.

To externally wrap ARINC signals, connect the transmitter signals to the respective receiver signals, TXnA to RXnA and TXnB to RXnB.

Table 44. AMC-A30 I/O Connections

Signal	Pin	Signal	Pin	Signal	Pin
Reserved	1	TX10B	35	Reserved	69
Gnd (note)	2	TX8B	36	RX7A	70
Reserved	3	TX7B	37	Reserved	71
Discrete 4	4	TX5B	38	RX4A	72
Reserved	5	TX4B	39	Reserved	73
TX12B	6	TX2B	40	RX1A	74
Reserved	7	TX1B	41	Reserved	75
TX9B	8	Discrete 2	42	Reserved	76
Reserved	9	RX14B	43	Reserved	77
TX6B	10	RX12B	44	Reserved	78
Reserved	11	RX11B	45	IRIGRX+	79
TX3B	12	RX9B	46	Gnd (note)	80
Reserved	13	RX8B	47	TX14A	81
Gnd (note)	14	RX6B	48	TX13A	82
Reserved	15	RX5B	49	TX11A	83
RX13B	16	RX3B	50	TX10A	84
Reserved	17	RX2B	51	TX8A	85
RX10B	18	IRIG_TX	52	TX7A	86
Reserved	19	Reserved	53	TX5A	87
RX7B	20	Discrete 3	54	TX4A	88
Reserved	21	Reserved	55	TX2A	89
RX4B	22	TX12A	56	TX1A	90

Signal	Pin	Signal	Pin	Signal	Pin
Reserved	23	Reserved	57	Discrete 1	91
RX1B	24	TX9A	58	RX14A	92
Reserved	25	Reserved	59	RX12A	93
Reserved	26	TX6A	60	RX11A	94
Reserved	27	Reserved	61	RX9A	95
Reserved	28	TX3A	62	RX8A	96
Reserved	29	Reserved	63	RX6A	97
IRIGRX-	30	Gnd (note)	64	RX5A	98
Gnd (note)	31	Reserved	65	RX3A	99
TX14B	32	RX13A	66	RX2A	100
TX13B	33	Reserved	67		
TX11B	34	RX10A	68		

Note: The ground pins are provided as Discrete I/O return lines or for shielding, as necessary.

Table 45. AMC-A30-xxxx-J I/O Connection for ARINC 573/717

Signal	Pin	Signal	Pin	Signal	Pin
ARINC 717 BPRZ TXB	6	Reserved	43	Reserved	81
ARINC 717 HBP RXB	16	ARINC 717 BPRZ RXB	44	ARINC 717 HBP TXB	82
Reserved	32	ARINC 717 TXA (note)	56	Reserved	92
Reserved	33	ARINC 717 HBP RXA	66	ARINC 717 BPRZ RXA	93

Note: The ARINC 573/717 TXA (High) signal is supported on this pin for both the BPRZ and HBP protocols. The selected protocol processed is based on the selection of the ARINC 717 HBP and BPRZ Encoding bits in the respective Transmit Channel Configuration register. See the API routine AR_SET_573_CONFIG for the method to define the ARINC 573/717 active encoding selection for this output pin.

IRIG-B Signal Connections

IRIG-B time (AM or DC/TTL) may be received via signals IRIGRX+ and IRIGRX- (see Table 44). The following IRIG formats are accepted:

Table 46. IRIG Signal Formats

Format	Modulation Frequency	Frequency/Resolution	Coded Expressions
B	0, 1	0, 2	0, 1, 2, 3

Upon completion of the program load, the AMC-A30 initiates IRIG B002 (DC/TTL) transmission from the onboard IRIG encoder via the IRIGTX signal (see Table 44). The IRIGTX signal can source/sink 16 mA at valid TTL levels.

To externally wrap the IRIG generator to the IRIG receiver, connect the IRIGTX signal to IRIGRX+ input, and connect the IRIGRX- input to Ground.



RAR-EC

Overview

The RAR-EC card is a multiple-channel ARINC interface, available in several configurations for the ExpressCard form factor. When configured as a RAR-EC-74, this card includes eleven ARINC 429 channels, (seven receivers and four transmitters). There are a variety of configurations available supporting various ARINC 429 channel counts, ARINC 573/717, Avionics Discrete I/O, and IRIG time synchronization.

RAR-EC Specifications



Figure 22. RAR-EC

The RAR-EC is a multiple channel, multiple protocol interface built to the ExpressCard Specification, Release 1.2.

ExpressCard Interface

- PCI-Express Interface per the PCI Local Bus Specification Revision 2.2
- 54 mm ExpressCard format

Transmit Channels

- Up to four independent differential serial transmit channels
- Automatic parity generation
- 2048 message transmit buffer for each channel
- Baud rate/slew rate software-programmable for each channel
- 1024 entry message table supporting scheduled message transmission for all channels

Receiver Channels

- Up to seven independent, differential receive channels
- 2048 message buffered mode receive FIFO buffer for each channel
- 16384 message merged mode receive FIFO buffer
- Label/SDI message independent snapshot storage for each channel
- Independent merged/individual receive FIFO buffer operation
- 64-bit, 1 μ sec time-tag is stored with each data element in the FIFO
- Parity error detection

Avionics Discrete Input and Output

- Up to four bi-directional, avionics-level discrete channels
- Output may switch to ground up to 500mA
- Power-up / reset default inactive
- Fixed input threshold of 2.7 +/- 0.2 volts

IRIG Input and Output

- IRIG Timecode receiver and transmitter

Typical Power Consumption

Table 47. Power Consumption

+3.3V	1.5V
750 mA	0mA

Operating Temperature

-40 to +85 C

Weight

TBD ounces, maximum

PCI Memory Map

The following table summarizes the PCI memory map interface definition for the RAR-EC.

Table 48. RAR-EC PCI Memory Map

Region	Type	Size	Description
configuration	configuration	64 bytes	PCI configuration space
PCI BAR0	memory	128 bytes	PCI9030 memory-mapped local configuration registers
PCI BAR1	unused	0	PCI9030 I/O-mapped local configuration registers, unused
PCI BAR2	memory	512K bytes	RAR-EC host interface
PCI BAR3	unused	0	not used
PCI BAR4	unused	0	not used
PCI BAR5	unused	0	not used

I/O Connections

Mating Connectors

At publication of this document, the following mating connector was compatible with the Champ36 connector provided on the RAR-EC enclosure end-plate. GE Intelligent Platforms Embedded Systems supplies the adapter cable CONAREC-30 or CONAREC-180 for this connection.

Table 49. P1 Input/Output Connector

Connector	Part No	Description	Manufacturer
P1	2-5175677-5	Champ 36	AMP/Tyco
P1	0543063619	Champ 36	Molex

ARINC Input/Output Connector Pin-out

Various RAR-EC product configurations have specific channel pin-out definitions based on the number of channels and protocols included. The following table describes the P1 connector pin layout for the RAR-EC. The exact ARINC 429 channel pin-out depends on the number of receivers and transmitters configured on your RAR-EC. Note that for the -J version of the RAR-EC, ARINC 573/717 protocol support replaces the receive channel 7 and transmit channel 4 ARINC 429 I/O pins.

To externally wrap ARINC signals, connect the transmitter signals to the respective receiver signals, TXnA to RXnA and TXnB to RXnB.

Table 50. RAR-EC P1 ARINC I/O Connections

Signal	P1 Pin	Signal	P1 Pin
RX1A	1	RX1B	19
RX2A	2	RX2B	20
RX3A	3	RX3B	21
RX4A	4	RX4B	22
RX5A	5	RX5B	23
RX6A	6	RX6B	24
RX7A ²	7	RX7B ²	25
TX1A	8	TX1B	26
TX2A	9	TX2B	27
TX3A	10	TX3B	28
TX4A ²	11	TX4B ²	29
Discrete I/O 1	12	Discrete I/O 2	30
Discrete I/O 3	13	Discrete I/O 4	31
IRIG RX+	14	IRIG RX-	32
IRIG TX	15	Ground ¹	33
Reserved	16	Ground ¹	34
Reserved	17	Reserved	35
Reserved	18	Reserved	36

Notes:

- 1 The ground pins are provided as Discrete I/O return lines or for shielding, as required.
- 2 The ARINC 573/717 signals are supported on the respective transmit channel 4 and receive channel 7 pins for both the BPRZ and HBP protocols. The selected transmit protocol processed is based on the selection of the ARINC 717 HBP and BPRZ Encoding bits in the respective Transmit Channel Configuration register. See the API routine AR_SET_573_CONFIG for the method to define the ARINC 573/717 active encoding selection for this output pin.

Transition Cable Pin-out

An adapter cable is provided to transition from the 36 pin device I/O connector to a 37-pin Subminiature D receptacle connector, with the pin-out shown below.

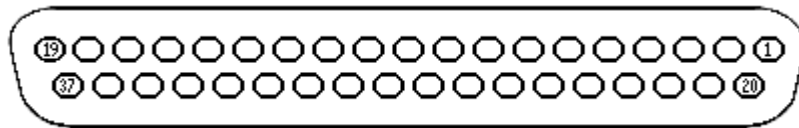


Figure 23. P1 Connector – View Facing Connector Pins

Table 51. RAR-EC-XX Transition Cable [37-Pin D-Subminiature] Connections

Adapter Pin-out	SIGNAL	Adapter Pin-out	SIGNAL
1	RX1A	20	RX1B
2	RX2A	21	RX2B
3	RX3A	22	RX3B
4	RX4A	23	RX4B
5	RX5A	24	RX5B
6	RX6A	25	RX6B
7	RX7A	26	RX7B
8	Reserved	27	Reserved
9	Reserved	28	Reserved
10	TX1A	29	TX1B
11	TX2A	30	TX2B
12	TX3A	31	TX3B
13	TX4A	32	TX4B
14	Discrete I/O 1	33	Discrete I/O 2
15	Discrete I/O 3	34	Discrete I/O 4
16	Reserved	35	Reserved

Adapter Pin-out	SIGNAL	Adapter Pin-out	SIGNAL
17	Ground	36	Ground
18	IRIG TX	37	IRIG RX-
19	IRIG RX+		

Notes:

- 1 The ground pins are provided as Discrete I/O return lines or for shielding, as required.
- 2 The ARINC 573/717 signals are supported on their transmit channel 4 and receive channel 7 pins for both the BPRZ and HBP protocols. The selected transmit protocol processed is based on the selection of the ARINC 717 HBP and BPRZ Encoding bits in the respective Transmit Channel Configuration register. See the API routine AR_SET_573_CONFIG for the method to define the ARINC 573/717 active encoding selection for this output pin.

IRIG-B Signal Connections

IRIG-B time (AM or DC/TTL) may be received via signals IRIGRX+ and IRIGRX-. The following IRIG formats are accepted:

Table 52. IRIG Signal Connections

Format	Modulation Frequency	Frequency/Resolution	Coded Expressions
B	0, 1	0, 2	0, 1, 2, 3

Upon completion of the program load, the RAR-EC initiates IRIG B002 (DC/TTL) transmission from the onboard IRIG encoder via the IRIGTX signal. The IRIGTX signal can source/sink 16 mA at valid TTL levels.

To externally wrap the IRIG generator to the IRIG receiver, connect the IRIGTX signal to IRIGRX+ input, and connect the IRIGRX- input to Ground.



RAR-CPCI

Overview

The RAR-CPCI card is a multiple-channel ARINC interface, available in several configurations for the CompactPCI platform. When configured as a RAR-CPCI-1616, this card includes thirty-two ARINC 429 channels, (sixteen receivers and sixteen transmitters). There are a variety of configurations available supporting various ARINC 429 channel counts, ARINC 573/717, Avionics Discrete I/O, and IRIG time synchronization.

RAR-CPCI Specifications



Figure 24. RAR-CPCI

The RAR-CPCI is a multiple channel, multiple protocol interface built to the PICMG 2.0 Revision 3 CompactPCI® Specification, PXI™ Specification Revision 2.0, and PXI™ Hardware Specification Revision 2.1.

PCI Interface

- Standard PCI Interface per the PCI Local Bus Specification Revision 2.2
- +5V and +3.3V PCI signaling compatibility and universal keying
- 66 MHz, 32 bit PCI operation

Transmit Channels

- Up to sixteen independent differential serial transmit channels
- Automatic parity generation
- 2048 message transmit buffer for each channel
- Baud rate/slew rate software-programmable for each channel
- 1024 entry message table supporting scheduled message transmission for all channels

Receiver Channels

- Up to sixteen independent, differential receive channels
- 2048 message buffered mode receive FIFO buffer for each channel
- 16384 message merged mode receive FIFO buffer
- Label/SDI message independent snapshot storage for each channel
- Independent merged/individual receive FIFO buffer operation
- 64-bit, 1 μ sec time-tag is stored with each data element in the FIFO
- Parity error detection

Avionics Discrete Input and Output

- Sixteen bi-directional, avionics-level discrete channels
- Output may switch to ground up to 500mA
- Power-up / reset default inactive
- Fixed input threshold of 2.7 +/- 0.2 volts

IRIG Input and Output

- IRIG Timecode receiver and transmitter

Typical Power Consumption

Table 53. Power Consumption

+3.3V	5V	+12V	-12V
500 mA	50 mA	100 mA (no TX Loads) 350mA (sixteen transmitters, max data rate, 400Ω load each)	100 mA (no TX Loads) 350mA (sixteen transmitters, max data rate, 400Ω load each)

Operating Temperature

-40 to +85 C

Weight

3.6 ounces, maximum

PCI Memory Map

The following table summarizes the PCI memory map interface definition for the RAR-CPCI.

Table 54. RAR-CPCI PCI Memory Map

Region	Type	Size	Description
configuration	configuration	64 bytes	PCI configuration space
PCI BAR0	memory	512 bytes	PCI9056 memory-mapped local configuration registers
PCI BAR1	I/O	256 bytes	PCI9056 I/O-mapped local configuration registers, unused
PCI BAR2	memory	512K bytes	RAR-CPCI host interface
PCI BAR3	unused	0	not used
PCI BAR4	unused	0	not used
PCI BAR5	unused	0	not used

I/O Connections

RAR-CPCI Outline Drawing

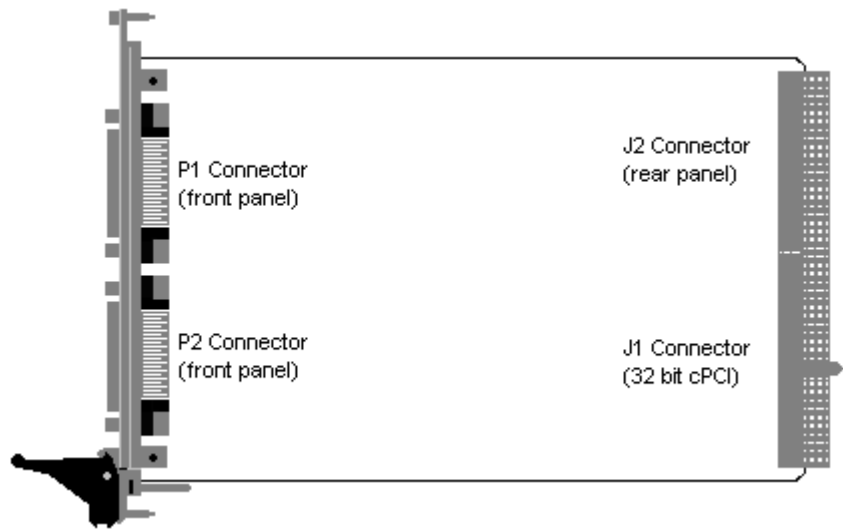


Figure 25. RAR-CPCI Outline Drawing

Mating Connectors

At publication of this document, the following mating connector was compatible with the 50-pin (P1 and P2) Champ connectors provided on the RAR-CPCI front panel (bezel). GE Intelligent Platforms Embedded Systems supplies the adapter cable CONCEI-620 for this connection.

Table 55. P1/P2 Input/Output Connectors

Connector	Part No	Description	Manufacturer
P1 and P2	787131-1	Front Panel 50 pin Champ	AMP/Tyco

ARINC Input/Output Connector Pin-out

Various RAR-CPCI product configurations have specific channel pin-out definitions based on the number of channels and protocols included. The following table describes the P1/P2 connector pin layout for the RAR-CPCI. The exact ARINC 429 channel pin-out depends on the number of receivers and transmitters configured on your RAR-CPCI. For the -J version of the RAR-CPCI, the ARINC 573/717 protocol support pins replace the channel 16 ARINC 429 I/O pins. Note the RAR-CPCI I/O connections are not pin-compatible with the CEI-620.

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

To externally wrap ARINC signals, connect the transmitter signals to the respective receiver signals, TXnA to RXnA and TXnB to RXnB.

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

Figure 26. P1/P2 50-pin Front-Panel (Bezel) Connectors – AMP Champ 0.8 mm Receptacle Connectors, View Facing Connector Pins

Table 56. RAR-CPCI P1/P2 Front Panel I/O Connections

P1 Connector						P2 Connector					
Pin	D50	Signal	Pin	D50	Signal	Pin	D50	Signal	Pin	D50	Signal
1	1	TX1A	26	34	TX1B	1	1	TX9A	26	34	TX9B
2	18	TX2A	27	2	TX2B	2	18	TX10A	27	2	TX10B
3	35	TX3A	28	19	TX3B	3	35	TX11A	28	19	TX11B
4	3	TX4A	29	36	TX4B	4	3	TX12A	29	36	TX12B
5	20	TX5A	30	4	TX5B	5	20	TX13A	30	4	TX13B
6	37	TX6A	31	21	TX6B	6	37	TX14A	31	21	TX14B
7	5	TX7A	32	38	TX7B	7	5	TX15A	32	38	TX15B
8	22	TX8A	33	6	TX8B	8	22	TX16A ²	33	6	TX16B ²
9	39	RX1A	34	23	RX1B	9	39	RX9A	34	23	RX9B
10	7	RX2A	35	40	RX2B	10	7	RX10A	35	40	RX10B
11	24	RX3A	36	8	RX3B	11	24	RX11A	36	8	RX11B
12	41	RX4A	37	25	RX4B	12	41	RX12A	37	25	RX12B
13	9	RX5A	38	42	RX5B	13	9	RX13A	38	42	RX13B
14	26	RX6A	39	10	RX6B	14	26	RX14A	39	10	RX14B
15	43	RX7A	40	27	RX7B	15	43	RX15A	40	27	RX15B
16	11	RX8A	41	44	RX8B	16	11	RX16A ²	41	44	RX16B ²
17	28	Ground ¹	42	12	Ground ¹	17	28	Ground ¹	42	12	Ground ¹
18	45	IRIGRX+	43	29	IRIGRX-	18	45	IRIGRX+	43	29	IRIGRX-
19	13	IRIGTX	44	46	Ground	19	13	IRIGTX	44	46	Ground
20	30	Ground ¹	45	14	Ground ¹	20	30	Ground ¹	45	14	Ground ¹
21	47	Ground ¹	46	31	Ground ¹	21	47	Ground ¹	46	31	Ground
22	15	Discrete IO 1	47	48	Discrete IO 2	22	15	Discrete IO 9	47	48	Discrete IO 10
23	32	Discrete IO 3	48	16	Discrete IO 4	23	32	Discrete IO 11	48	16	Discrete IO 12

P1 Connector						P2 Connector					
Pin	D50	Signal	Pin	D50	Signal	Pin	D50	Signal	Pin	D50	Signal
24	49	Discrete IO 5	49	33	Discrete IO 6	24	49	Discrete IO 13	49	33	Discrete IO 14
25	17	Discrete IO 7	50	50	Discrete IO 8	25	17	Discrete IO 15	50	50	Discrete IO 16

- Notes:
- 1 The ground pins are provided as Discrete I/O return lines or for shielding, as required.
 - 2 The ARINC 573/717 signals are supported on the respective channel 16 pins for both the BPRZ and HBP protocols. The selected protocol processed is based on the selection of the ARINC 717 HBP and BPRZ Encoding bits in the respective Transmit Channel Configuration register. See the API routine `AR_SET_573_CONFIG` for the method to define the ARINC 573/717 active encoding selection for this output pin.

Table 57. RAR-CPCI Rear Panel Input/Output Connector Definition

	Row a	Row b	Row c	Row d	Row e
1		TX9B	TX9A	TX1B	TX1A
2		TX10B	TX10A	TX2B	TX2A
3		TX11B	TX11A	TX3B	TX3A
4		TX12B	TX12A	TX4B	TX4A
5		TX13B	TX13A	TX5B	TX5A
6		TX14B	TX14A	TX6B	TX6A
7		TX15B	TX15A	TX7B	TX7A
8		TX16B	TX16A	TX8B	TX8A
9		RX9B	RX9A	RX1B	RX1A
10		RX10B	RX10A	RX2B	RX2A
11		RX11B	RX11A	RX3B	RX3A
12	Ground ¹	RX12B	RX12A	RX4B	RX4A
13	IRIGTX	RX13B	RX13A	RX5B	RX5A
14	IRIGRX-	RX14B	RX14A	RX6B	RX6A
15	IRIGRX+	RX15B	RX15A	RX7B	RX7A
16		RX16B	RX16A	RX8B	RX8A
17					
18					
19	Discrete IO 13				
20	Discrete IO 14	Discrete IO 10	Discrete IO 7	Discrete IO 4	Discrete IO 1
21	Discrete IO 15	Discrete IO 11	Discrete IO 8	Discrete IO 5	Discrete IO 2
22	Discrete IO 16	Discrete IO 12	Discrete IO 9	Discrete IO 6	Discrete IO 3

Notes:

- 1 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.
- 2 The ground pins are provided as Discrete I/O return lines or for shielding, as required.

IRIG-B Signal Connections

IRIG-B time (AM or DC/TTL) may be received via signals IRIGRX+ and IRIGRX-. The following IRIG formats are accepted:

Table 58. IRIG Signal Connections

Format	Modulation Frequency	Frequency/Resolution	Coded Expressions
B	0, 1	0, 2	0, 1, 2, 3

On completion of the program load, the RAR-CPCI initiates IRIG B002 (DC/TTL) transmission from the onboard IRIG encoder via the IRIGTX signal. The IRIGTX signal can source/sink 16 mA at valid TTL levels.

To externally wrap the IRIG generator to the IRIG receiver, connect the IRIGTX signal to IRIGRX+ input, and connect the IRIGRX- input to Ground.



Windows Installation

Software Installation for Windows

Although system resources may limit the number of boards installed on a system, the CEI-x30-SW distribution supports up to sixteen devices when installed under 32-bit and 64-bit versions of the Windows 7, Vista, XP, 2000, and NT operating systems. Windows 9x installations are limited to ten devices.

Prior to physically installing the CEI-x30 product, the CEI-x30-SW software distribution must be installed on your PC. Failure to properly install the software prior to the hardware may result in corruption of the Windows Device Manager Registry settings and require restoration to a previous configuration.

To install the software, follow these steps:

1. Exit all programs.
2. Insert the CEI-x30-SW CD into your CD drive.
3. If the installation does not automatically start after 10 seconds:
 - Click Start from the Windows Task Bar and select Run.
 - Use the Browse button to locate the Setup.exe file in the Setup\Disk1 folder.
 - Double-click the file Setup.Exe. Then, click OK to launch the setup program.
4. Follow the on-screen instructions for the installation.
5. Note which device number is allocated during the installation.
6. Following successful completion of the installation, **turn off the computer.**

Hardware Installation

Once the software has been successfully installed, follow these steps to install the hardware and Windows device driver:

With the computer powered off, install the CEI-x30 product into any available slot, PMC site, or PC/104-*Plus* stack location. The CEI-430 requires card jumpers to agree with the installed stack location. Refer to Chapter 2 for applicable jumper settings.

Hardware Installation with Windows 7/Vista/XP/2000/9x

To install the hardware under these Windows operating systems, follow these steps:

1. Power-up the PC.

Windows 7 device installation should occur automatically; for other Windows versions, the Windows Plug and Play hardware manager should detect the CEI-x30 device, and the *Found New Hardware* dialog box should automatically startup. Decline any request to query the Microsoft web site to obtain drivers for this device.

- For Windows XP, select the *Install the software automatically* option and click Next. Under the *Completing the Found New Hardware* dialog, click Finish.
- For Windows Vista, select the *Locate and install driver software (recommended)* option and click Next. Under the *Completing the Found New Hardware* dialog, click Finish.
- For Windows 2000 the *Install hardware device drivers* dialog will be displayed, select the "*Display a list of known drivers...*" option and click Next. Under the *Select a device driver* dialog, the hardware you are installing should appear in the Models window. Select the device you are installing and click Next. Under the *Completing the Found New Hardware* dialog, click Finish.
- For Windows Me/98/95, Windows should then display the Building Driver Information Database dialog for the *CEI-x30 DevX*, followed by the System Settings Change dialog. If you are prompted, click Yes to the request to restart your computer, (Windows XP/2000 does not prompt for a restart).

2. If Windows does not detect the new hardware, you must manually install the device driver:

- Click Start and point to Settings.
- Select the Control Panel.
- Select Add New Hardware.

3. If the device drivers aren't automatically detected and you are prompted for a path to the driver location, enter *C:\Windows\Inf* and click OK.
4. To check for proper driver installation under Windows 7, Vista, XP, and 2000, open the Device Manager as follows:
 - From the "Start->Run" command prompt, enter "devmgmt.msc" to open the Windows Device Manager.
 - Expand the GE Avionics Devices folder (or Condor Engineering Devices folder for older installations).
5. To check for proper driver installation on any older Windows version such as ME/98/95, open the Device Manager as follows:
 - Right mouse click on the My Computer Desktop icon:
 - Select Properties from the menu to open the System Properties window.
 - Select the Device Manager tab, scroll down to, and expand the WinRT folder.
6. Verify the device entry *Product Name* for your device is shown with no exclamation point overlaying the icon. If this is true, the device driver was properly installed. You have completed installation of the hardware.

Hardware Installation with Windows NT 4.0

To continue installing the hardware under Windows NT 4.0, follow these steps:

1. Power-up the PC. The device driver should be loaded automatically on power-up.
2. To check for proper driver installation, verify that the device driver **WinRT** appears in both the IRQ and Memory resource windows with the appropriate IRQ and address range assignments.

Installation Verification

To verify the device driver is properly installed, execute the Test Configuration program.

1. Click Start, and then Programs.
2. Select the **GE CEI-x30-SW** program group and click on the **Test Configuration** shortcut.

This program executes an internal wrap test on all available channels and notifies you of success or failure. If the program reports success on all channels tested, you are ready to use your new board.



VxWorks Installation

Overview

VxWorks is an embedded real-time operating system supporting flexible hardware configurations. The CEI-830 API compiles and runs under the Motorola PowerPC and Intel x86 VxWorks Board Support Packages.

The CEI-x30-SW API supports up to eight boards on a single VxWorks host, using device numbers 0-7 to designate the device. VxWorks assigns the device numbers based on the order it encounters the devices on the bus. The first device is device 0, the next device is 1, and so on. If you have only a single board in your system, it is always device 0. Use this value when programming using the supplied API.

To use the CEI-x30-SW API with VxWorks you must first build a VxWorks image supporting the respective CEI-x30 board configuration. Upon boot of this image, you may download and execute the client application. These basic steps are described in this chapter.

Building a VxWorks Image

To incorporate your CEI-x30 API with VxWorks you must rebuild your VxWorks image. The procedure provided for including the CEI-x30 API in your VxWorks image utilizes the VxWorks Component Installation method, and applies to both Tornado and Workbench development environments. Since these methods differ greatly, these instructions are written in a generic fashion and must be interpreted for your environment:

1. To install the generic GE Avionics common VxWorks driver source code for an Intel-based target, use the configuration definition file 51_GEFES_x86_RTP_PCI.cdf.; for a PowerPC-based product, use the file 51_GEFES_PPC_RTP_PCI.cdf. Copy the respective file from the folder \Program Files\Condor Engineering\CEI-x30-

SW\Source\VxWorks to
`[Tornado_directory_path]\target\config\comps\vxWorks.`

2. Copy the following source files from the folder \Program Files\Condor Engineering\CEI-x30-SW\Source\VxWorks to either folder `[Tornado_directory_path]\target\config\<BSP Folder>` or `[Tornado_directory_path]\target\config\comps\src`:

```

    cei_types.h
    CondorVxWRTPDrv.c
    CondorVxWRTPDrv.h
    gefes_ioctl.h
    lowlevel.h
    target_defines.h

```

3. You may choose to include the CEI-x30-SW API source files in the BSP kernel source folder, create a new project folder for the CEI-x30 API and application development, or reference the distribution installation Source and Include folders. If you are not using the distribution folder, copy the following two API source files from the folder \Program Files\Condor Engineering\CEI-x30-SW\Source to the folder of your choice:

```

    cdev_api.c          cdev_vxw.c

```

4. If you choose not to reference the Include folder in your compilation Include Path, copy the following C header files from the folder \Program Files\Condor Engineering\CEI-x30-SW\Include to that same folder:

```

    ar_error.h          cdev_api.h          cdev_fw.h
    cdev_glb.h          cdev_hw.h          fpga430.h
    fpga530.h           fpga630.h          fpga830.h
    fpga830rx.h         fpga_ec.h          fpgaA30.h
    fpgax30n.h          cei_types.h        target_defines.h

```

5. Open the workspace containing your VxWorks target image project, and access the Kernel Configuration setup for the VxWorks image.
6. Beneath the *hardware* component, right-click the “*GE Intelligent Platforms Avionics Products*” component and select *Include (quick include)*.
7. Modify the default values for any definitions as required for your target system. Examples of such modifications include:
 - For x86/Pentium kernel images for any VxWorks version prior to 6.6, change the default state of. “*Adds device memory into the sysPhysMemDesc table*” to TRUE.
 - To modify the maximum number of overall GE Avionics boards supported in a system, change the value of “*Defines the maximum*”

number of devices” to the desired number, up to a maximum number of 16.

8. If you choose to build the API outside of the BSP source folder, you will have to manually define the directive `VXW_PCI_X86` for an x86/Pentium target C source compilation/build or the directive `VXW_PCI_PPC` for a PowerPC target C source compilation/build, (normally defined in the configuration definition file).
9. Once you have your VxWorks kernel image built and installed on your target, open a shell to the target and invoke the function *geipBoardShow*. This routine lists all detected GE Avionics products in the device ID order that should be referenced from your application for the respective boards.

See the section, “Target-specific Compiler Directives” for more information on the various ways to customize the CEI-x30 API for your target BSP.

BIOS Initialization

The installation process assumes that the x86 BIOS configures PCI-based boards by writing the base memory address into the board’s configuration space. If your BIOS initialization doesn’t do this, you need to write the base memory address into the appropriate PCI configuration registers prior to calling “*gefesInitPCI*”. 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>

Using the Sample Program

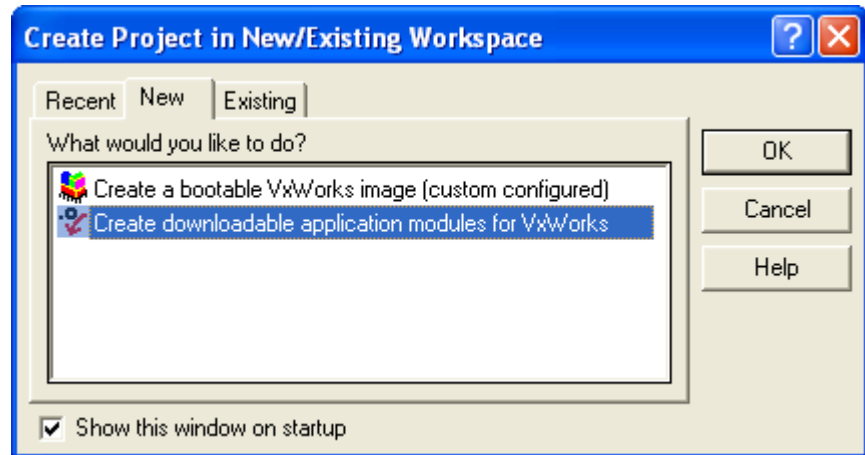
The CEI-x30 API distribution includes an example program named `TST_CNFG.C`. The source code is located within the Windows distribution Source folder or on the distribution disk in the `CEI-x30-SW\Source` directory. You can use this program to test your VxWorks installation, as it simply executes an internal wrap on all receiver-transmitter channel pairs. You can also use `TST_CNFG.C` as a guide for programming with the CEI-x30 API.

Building the API and Sample Program with Tornado

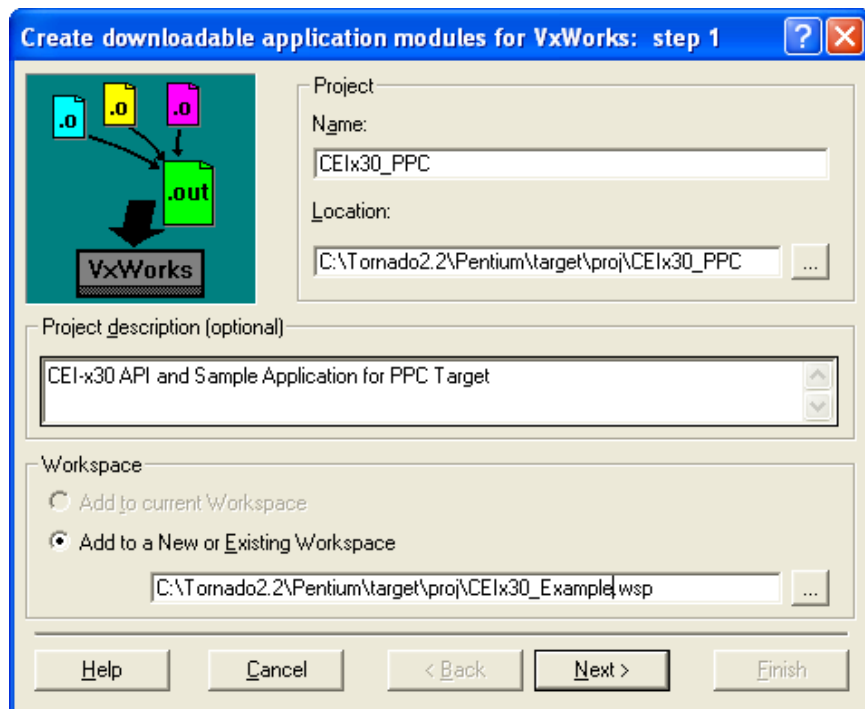
The API and sample program are built as a downloadable object and application within Tornado. The following steps explain how to build the

sample program for a PowerPC 604 target using Tornado 2.2, but can be easily adapted to Workbench, as well as other targets.

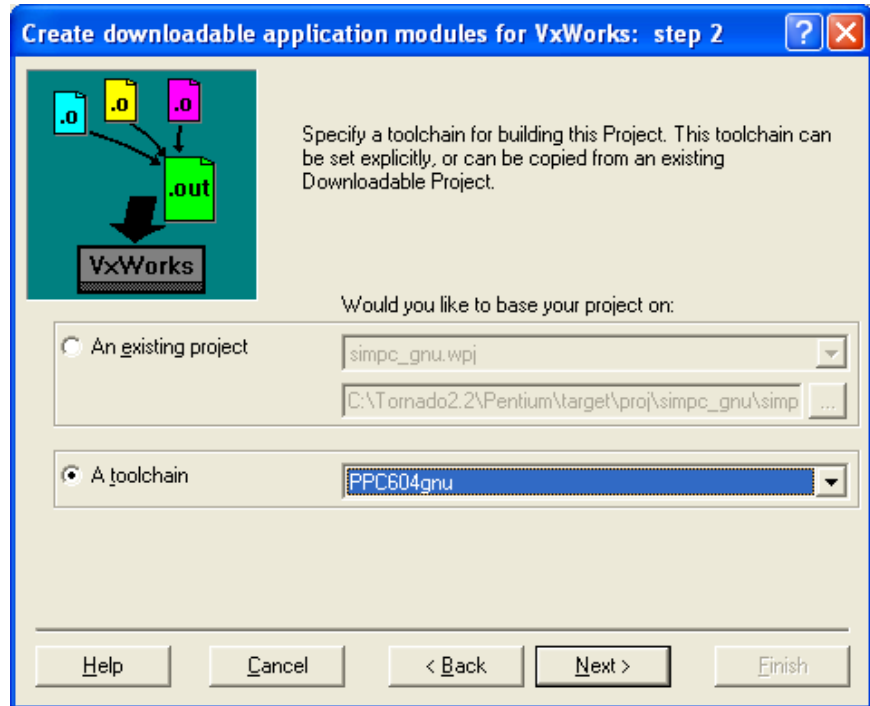
1. Create a new downloadable application project.



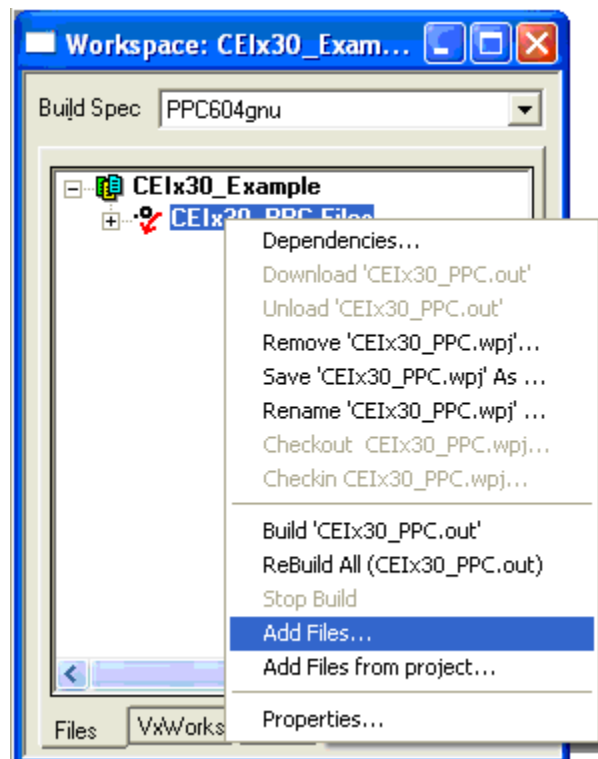
2. Fill out the project name, location, and workspace. Click **Next**.



3. Select the desired **tool chain**. For this example, we'll choose **PPC604gnu**. Click **Next**.

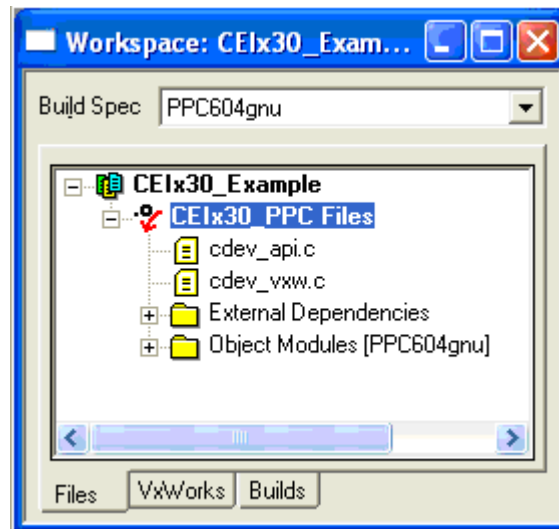


4. Verify the project, **Workspace**, and **tool chain** selections in the next window. Click **Finish**.
5. Right-click the project and select **Add Files...**

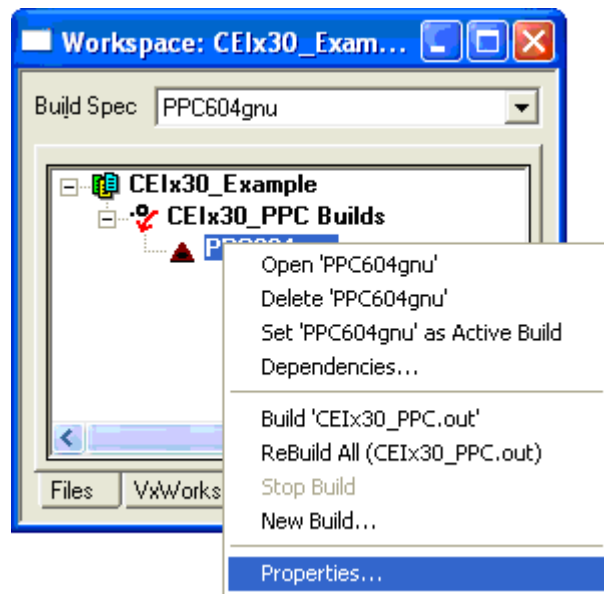


6. If you are only building the CEI-x30 API object, add the files CDEV_API.C and CDEV_VXW.C from the folder \Program

Files\Condor Engineering\CEI-x30-SW\Source to the project. If you wish to include the CEI-x30 Test Configuration program as part of the object, also include TST_CNFG.C in the source file list.



7. Click the Builds tab in the Workspace window. Right-click the build specification (PPC604gnu, in this case), and select **Properties**.



8. Select the C/C++ compiler tab and click on the “Include paths...” button. Now click on the “Add...” button and browse to the Include folder beneath where the CEI-x30-SW distribution is installed. Next define the constants required for your target. For example, you would add -DVXW_PCI_PPC to define the directive VXW_PCI_PPC for a PowerPC target. Click **OK** to close the build properties window.
9. Right-click the build specification and select **Rebuild All....**
10. Run the dependency checker if prompted.

The Build Output window should appear and indicate a successful build.

11. Assuming you have already connected to the target via target server, right-click the build specification and select **Download** to download the output file you created.
12. If you included the CEI-x30 Test Configuration program as part of the object, open a shell to the target via the Launch Shell button on the toolbar and from the shell prompt, type **wrap**.

This program executes an internal wrap test on all available channels and notifies you of success or failure. If the program reports success on all channels tested, you are ready to use your new board.

Target-specific Compiler Directives

The CEI-x30 API accounts for specific target requirements using compilation directives. There are a few directives that may be required for the board-specific VxWorks support provided with your target. Two alternatives to the standard *taskDelay* method to pause execution are provided for select board support packages, *sysUsDelay* and *sysMsDelay*. There are also differing methods required to map a CEI-x30 board's PCI memory regions, *sysMmuMapAdd*, *sysPciMemToLocalAdrs* and *sysBusToLocalAdrs*. You should determine the specific requirements for your target BSP and take the appropriate action prior to building the CEI-x30 API into your system.

The following compiler directives are defined to include both general and specific features required for compiling for various VxWorks target BSPs:

VXW_PCI_PPC	required for all VxWorks PowerPC targets
VXW_PCI_X86	required for all VxWorks x86/Pentium targets
NON_INTEL_WORD_ORDER	defines a Big Endian mapped target (typical for PowerPC).
DELAY_USE_SYS_US_DELAY	required when a delay should be implemented with <i>sysUsDelay</i> instead of <i>taskDelay</i> (applies to some Thales VMPC* targets).
DELAY_USE_SYS_MS_DELAY	required when a delay should be implemented with <i>sysMsDelay</i> instead of <i>taskDelay</i> (applies to some Motorola MCP* targets).
VXW_SYS_BUS_MAP	required when execution of the routine <i>sysBusToLocalAdrs</i> should be used to map PCI BARs

VXW_SYS_PCI_MAP	<p>(applies to some Thales VMPC* and Motorola MCP* targets).</p> <p>required when execution of the routine <i>sysPciMemToLocalAdrs</i> should be used to map PCI BARs (applies to some Thales VMPC* targets).</p>
NO_INT64	<p>required if the VxWorks Kernel version or BSP does not provide support for the 64-bit integer data type.</p>
TARGET_EMBEDDED	<p>bypasses parameter checking source lines during compilation. This can be defined for any embedded application towards the end of the integration phase to improve API routine throughput.</p>



Linux Installation

Overview

CEI-x30-SW provides support for all CEI-x30 products under most Linux Kernel 2.6 revisions. The install process builds the API as a shared library and installs the driver as a module. Application programs link with the shared library to access the respective device. Up to eight boards can be installed under supported Linux distributions. Refer to the files `Linux_support.txt` and `Linux_install.txt` located in the Linux distribution file for the latest information on installing and building the common driver along with the CEI-x30 API and example program.

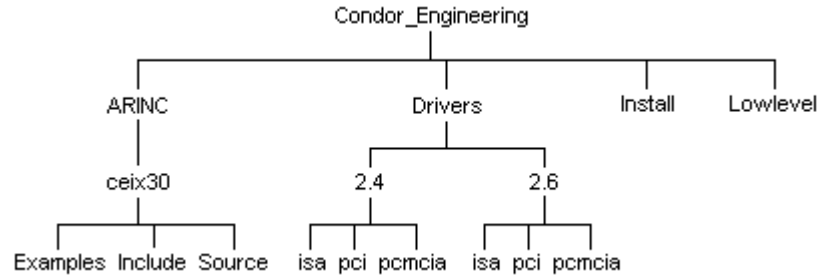
Software Installation

The Linux installation process requires your CEI-x30 hardware be installed prior to execution of the installation:

1. You must log on as "root" (you may use "su")
2. Copy the Linux distribution compressed tar file (`linux_x30_vnnn.tgz`) to the `/root` directory.
3. Uncompress and extract the installation file using the following:

```
tar -zxvf linux_x30_vnnn.tgz
```

After the tarball extraction completes, the following directory structure will be created:



Building Applications

Automatic Installation (Builds LSP and API)

Navigate to the Install directory and run the installation script by typing

`./install`

The PCI device driver builds and loads if the installation script detects a GE Intelligent Platforms Embedded Systems avionics PCI board in the "procs" file system. If the system does not have a "proc" file system, perform a manual install of the device driver.

These are the configuration arguments that are accepted by the "install" script:

1. To remove support for SYSFS (the SYS file system), include "no_sysfs" in the `./install` command line. If the system does not have the SYS file system or is based on kernel 2.6.10+ and does not agree with the "Proprietary/GPL" license, then SYSFS support must be removed.
2. To debug the kernel device driver(s), include the option `"debug_drv=<DEBUG LEVEL>"` in the `./install` command line. The debug statements will be printed out to the kernel message log. The `<DEBUG LEVEL>` provides increasing debug information with a range of "0" (none) to "3" (all).
3. To debug the low level library, include `"debug_ll"` in the `./install` command line. The debug statements will be printed to stdout.
4. To build only the device drivers and libraries, include `"no_install"` in the `./install` command line. If support for SYSFS has been removed, the `"ceidev.conf"` will not be generated. Need to follow the instructions that are displayed on the screen during the installation. Refer to `Linux_install.txt` in your distribution, sub-section 4 in the section "Manual Install" concerning the `"ceidev.conf"` file.

5. To build the low-level and API libraries as 32-bit libraries to run in 32-bit emulation mode for 64-bit systems, include "32bit" in the "./install" command line.
6. To disable hardware interrupt support in the kernel 2.6 PCI/ISA drivers, include "no_hwint" in the "./install" command line.
7. To disable using POSIX RT signals in the kernel 2.6 PCI/ISA drivers, include "no_hwint_signal" in the "./install" command line.
8. To disable using a "wait queue" for the kernel 2.6 PCI driver, include "no_hwint_waitqueue" in the "./install" command line.

The installation is finished. Check the "install" script output and the kernel message log for any errors. If there are no errors then the device driver(s) are loaded into the kernel, the low-level library is built as well as all detected API(s) distributions.

The installation installs the driver, builds and installs the API, (including the Linux common low-level interface), and compiles the example program. To test the installation, navigate to the Examples directory and execute the *tst_cnfg* application.

Manual Installation

Refer to the file *Linux_install.txt* in your distribution, section "Manual Install" concerning the manual installation of the Linux distribution and/or driver.

Linux Driver Operation

Linux compiles drivers as modules that dynamically link with the Linux kernel. The installation script automatically compiles the correct driver for the boards you are installing and the Linux kernel version. You can recompile using one of the Make files in the */Drivers/kernel/pci* directory, where *kernel* is either 2.4 or 2.6. The module installation script *load_pci* is supplied in the Driver folder, which loads the module. You can manually load the driver by typing *./load_pci*, and unload the driver by typing *./unload_pci*.

Installation automatically invokes the driver load script. However, if you reboot the system you need to re-execute this script. You can put the script in the *rc.local* initialization file, which should automatically execute on power-up. The installation instructions located in the distribution file *Linux_Install.txt* explain how to manually execute the script.

Troubleshooting

When installing any API distribution, you will need to be logged on as "root". Use the "su" command to gain "root" permissions. Root permissions are necessary when building device drivers and loading the modules into the kernel.

Useful Linux system utilities

dmesg: displays the kernel message log.

lsmod: displays the current modules loaded in the kernel.

lspci displays the PCI config space for all PCI devices

strace: displays the system calls that the driver or application calls.

gdb: the GNU debugger.

modinfo: displays the module information for a driver.

Compilation Errors

If there are compilation errors, check that the path to the kernel headers is valid. If different than the default ("/lib/modules/<KERNEL>/build"), include the path in the applicable driver's makefile by including a "-I" with the path. If there are system calls that cannot be resolved, check the "/proc/kallsyms" file to verify that they are compiled into the kernel.

Run-time Errors

Run-time error resolution may involve one or more of the following:

1. Check that the device driver, uceipci, is loaded with "lsmod".
2. Examine the kernel message log for error output from the device driver uceipci. Use "dmesg" and/or look directed at the kernel message log located in "/var/log/messages".
3. If there are version errors when loading the driver, the driver's version string (magic) may not coincide with current running kernel. Use "modinfo" to get the driver's magic number. Refer to the "/usr/src/linux/makefile" and "/usr/src/linux/.config".

4. To determine where an error may be occurring in the application or API libraries use "gdb". Make sure when compiling to provide the "-g" to GCC.
5. If a device driver fails to unload with "modprobe", use "rmmod".
6. If loading the 2.6 PCI device driver and receive errors indicating missing symbols with "sysfs" in the symbol name, then build the distribution without support for SYSFS.



Integrity® Support

Introduction

Green Hills Integrity is a secure, high-reliability real-time operating system (RTOS) intended for use in mission critical systems. The CEI-x30-SW distribution supports up to two CEI-830 devices with Integrity on PowerPC processors.

Integrity is flexible in how it builds the kernel and application software. You can build a monolith containing the kernel, BSP, and application software, or you can build a separate kernel/BSP and the application as a Dynamic Download. This Integrity distribution supports either method. This distribution provides the Integrity PCI driver source file, API source files, and an example application source file. You must compile and link the API to form a static library, which can then be linked with your application to achieve support for the CEI-830.

Integrity Installation

There are two options for installing CEI-x30-SW support for Integrity. If you are running the Multi IDE on a Windows system, you can install using the Windows installation and select the *Source installation only for VxWorks/Integrity* option at the Target Installation selection prompt. You can also copy the desired folders directly from the Installation CD-ROM. The installation contains the GE Intelligent Platforms Embedded Systems Avionics Integrity PCI driver, the source code for the API and driver interface, the API compiled as a static library for most PowerPC processors, the Example program, and documentation.

After installing your CEI-830, you need to copy the PCI device driver source into the integrity BSP project for your target systems and rebuild the kernel. The GE Intelligent Platforms Embedded Systems Avionics Integrity driver works with most PowerPC BSPs. You can build your

application using the static library supplied or create your own static library project.

Integrity PCI Driver Installation

You must install the PCI device driver as part of the BSP project in the default.gpj. The driver is a C file named `cei_int_pci_drv.c` that is BSP independent. Add that file *into* the libbsp.gpj project. Figure 27 shows the driver file installed in a Dy4 DMV181 project. To add the file, right-click the libbsp.gpj line and select the **Add File Into libbsp.gpj** option.

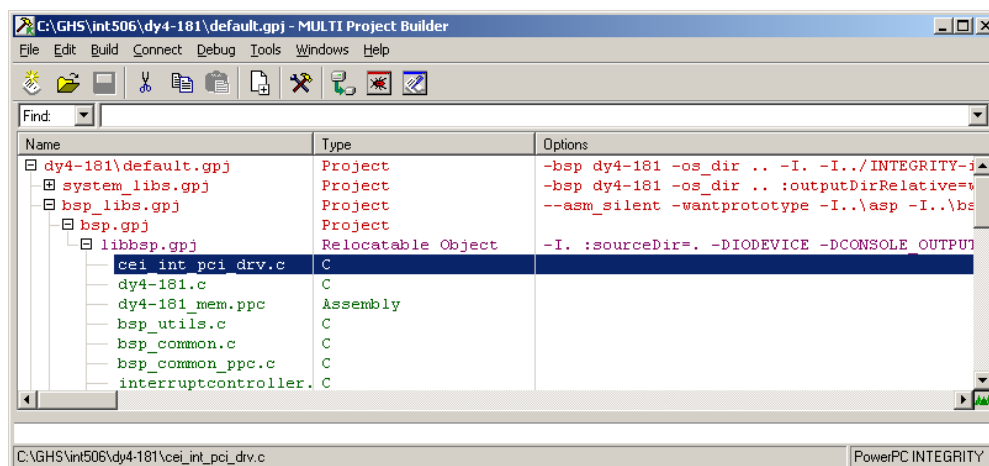


Figure 27. Integrity libbsp.gpj with `cei_int_pci_drv.c` Added

Building Integrity Applications

Once the PCI device driver has been built as part of the kernel, you are ready to build your application program.

The distribution includes the static API libraries for the CEI-x30 products named `libceix30_api.a` and `libceix30_api_k.a`; `libceix30_api_k.a` is built for use with monolith applications built into the kernel. If you are building a dynamic download, use `libceix30_api.a`. The figure below shows a typical Dynamic Download project using the CEI-x30 API library.

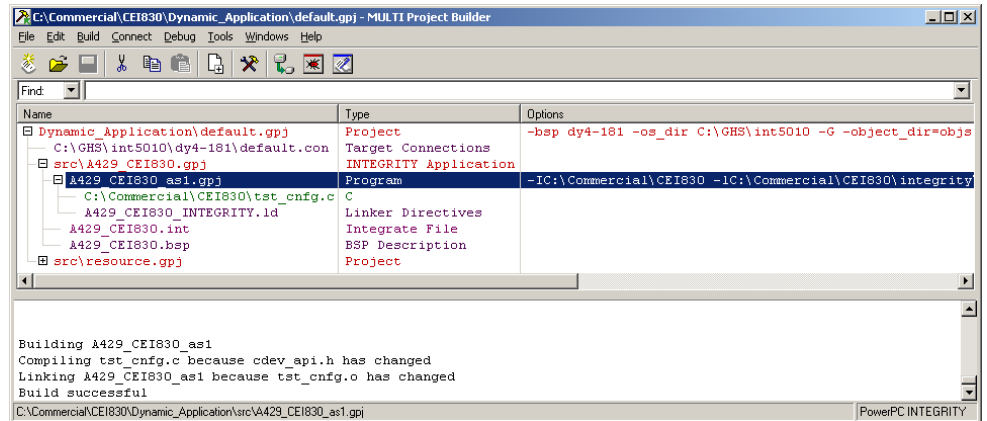


Figure 28. Example CEI-830 Integrity Application Project Setup

Build the application using the following steps:

1. Define the preprocessor symbol `INTERGITY_PCI_PPC`.
2. Link with either the `libceix30_api.a` (`-llibceix30_api`) or `libceix30_api_k.a` (`-llibceix30_api_k`) and include `libposix.a` and `libsocket.a`. You should review the Integrity POSIX chapter to make sure this POSIX option meets your application's needs.

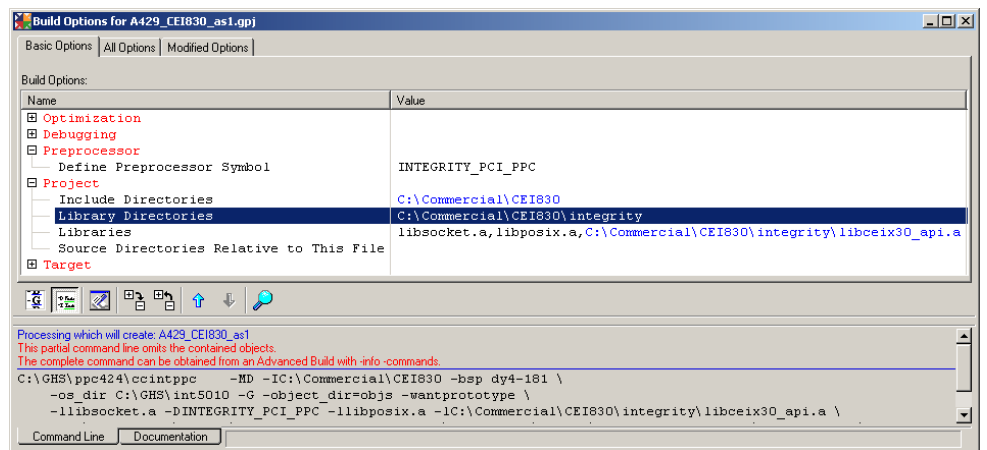


Figure 29. Example CEI-830 Integrity Application Project Option

3. Add sufficient `MemoryPoolSize` to create POSIX threads.

The example below uses `0x1000000`, but your application may need more. Add the `MemoryPoolSize` entry between the `Filename` and `Language` entries in the `Integrate` file options.

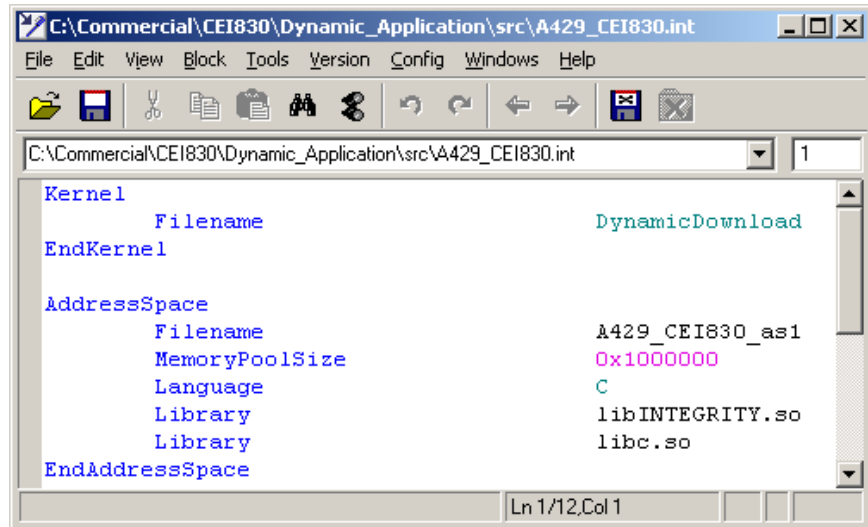


Figure 30. Adding a MemoryPoolSize Entry

4. Build the project, then download and run your application. If you desire the application to execute upon download, modify the value for the **DefaultStartIt** attribute to be “true” as follows:

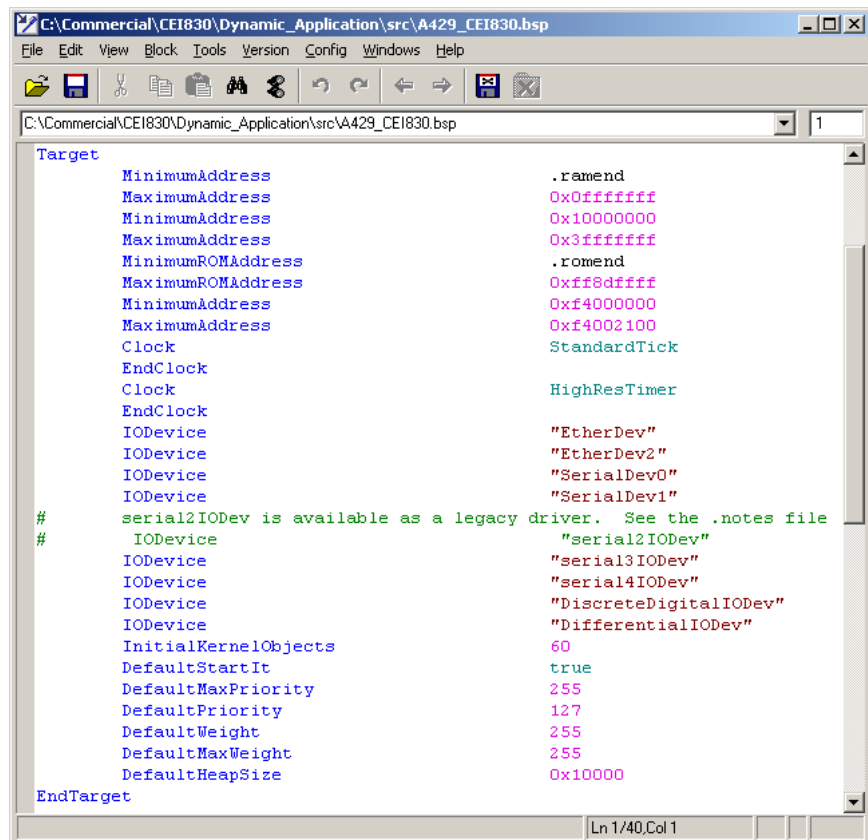


Figure 31. Modifying the Value for the DefaultStartIt Attribute

5. If you want to build a monolith project that includes both the kernel and application, add your application code into the project and link with the CEI-x30 API library following the steps outlined above.

Building the CEI-x30 API with Multi

The CEI-x30-SW Integrity Distribution contains a static library built for a generic PowerPC, which should suffice for most PowerPC systems. You can choose to rebuild this static library to customize API operation for your system. You may build a static library with the supplied source and include files using the following steps. Use the following source files:

- cdev_api.c
- cdev_int.c
- mem_integrity.c

The following list shows the include files (.h) needed to build the API.

ar_error.h	cdev_api.h	cdev_fw.h
cdev_glb.h	cdev_hw.h	fpga830.h
fpgax30n.h	cei_types.h	lowlevel.h

The preprocessor symbol INTEGRITY_PCI_PPC selects the Integrity target compilation in the source files.

Select a stand-alone project for a generic PowerPC. Select a PowerPC option matching your system.

For Project Type, select Library (empty). You can then add the C source files to the project and add the path to the include files.

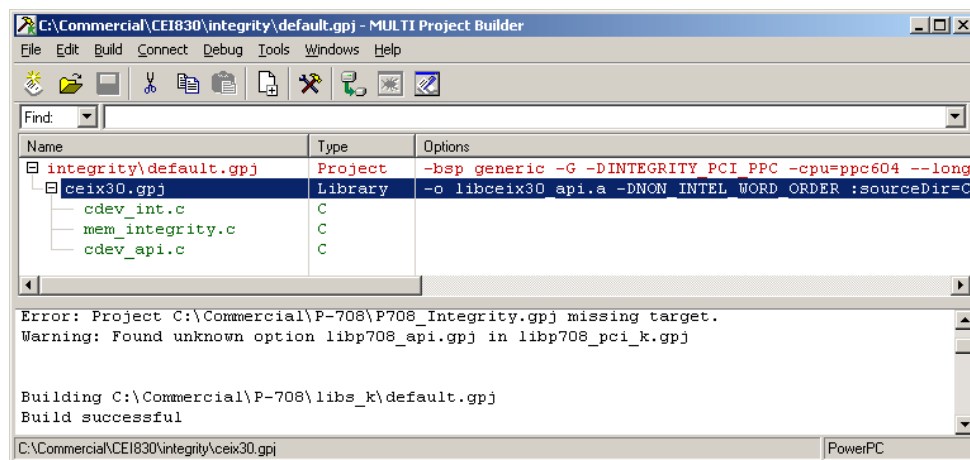


Figure 32. Example CEI-830 Integrity Library Project Setup

Define the pre-processor symbol INTEGRITY_PCI_PPC and NON_INTEL_WORD_ORDER (required if your Integrity BSP does not automatically handle Endian byte swapping). If you are building a library

to run in a Monolith, you also need to define the symbol `GHS_KERNEL`. You can name the output library to anything and use that to link with your application(s).

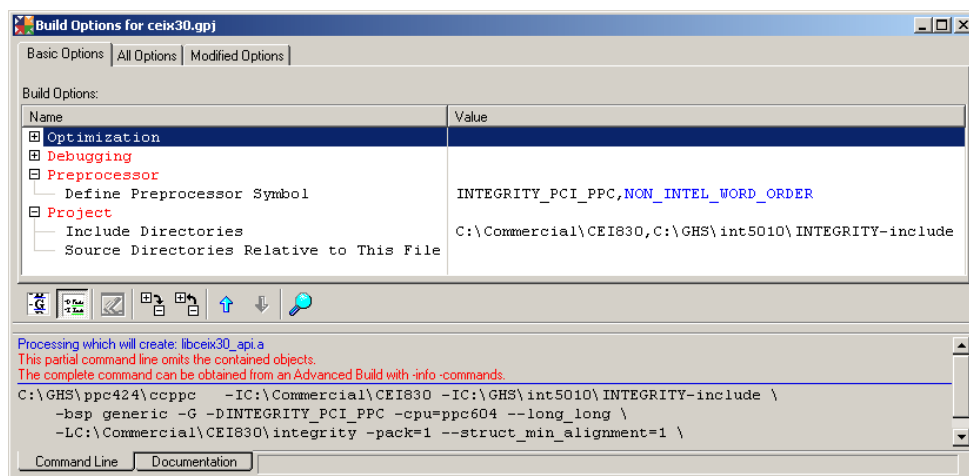


Figure 33. Example CEI-830 Integrity Library Project Options



CEI-x30 Features

Overview

The CEI-x30 products provide specialized features for receive message storage and time-tagging, timer usage, and transmit message scheduling. The following paragraphs document several of these features, and how they might be used in your ARINC application.

Enhanced CEI-x30 Interface

Beginning with the release of CEI-x30-SW Version 2.00, all CEI-x30 products provide enhanced features, including full exposure of all channel register sets to the host PCI interface, time-stamped Snapshot Buffer message storage, and Filter Table triggered hardware interrupt support. This enhanced feature set includes a larger firmware host interface, requiring a modification to the default PCI BAR2 memory definition stored in an onboard EEPROM. The process by which an existing CEI-x30 device EEPROM BAR2 size attribute is modified is handled via `AR_SET_DEVICE_CONFIG` invocation with the *item* parameter `ARU_HW_ENHANCE_UPDATE`.

ARINC 429 Protocol Support

Several aspects of the ARINC 429 protocol are handled by the CEI-x30 products.

The electrical transmission of ARINC 429 data over the bus is performed with the label field in reverse bit order. The transmit logic of the CEI-x30 product automatically reverses the bit order of the ARINC 429 message label (B0-B7) prior to transmission. The receiver logic of the CEI-x30 also reverses the bit order of the ARINC 429 message label prior to placing the data in the respective receive buffers. This ARINC 429 label modification

is fixed in the CEI-x30 processing and cannot be modified by the application. For more information on the ARINC 429 protocol, see the “ARINC 429 Protocol Tutorial”.

ARINC 429 message parity is defined in the MSB of the ARINC 429 message. The CEI-x30 transmission logic provides the capability to generate either odd or even parity based on the bit states of the first 31 bits of the ARINC 429 message. When disabled, the transmitter logic transmits the ARINC 429 message with the parity bit unaltered. When enabled, the CEI-x30 product overwrites the value of the parity bit in the 32-bit user-defined message with the calculated parity value.

The CEI-x30 reception logic provides the capability to detect the parity of ARINC 429 messages based on the bit states of each message. When disabled, the receiver logic provides the received ARINC 429 as it was received, (without modification). If enabled, the receiver logic modifies the state of the parity bit (B32) to be “0” if the parity was detected as odd and “1” if the parity was detected to be even.

The bus speed for both ARINC 429 transmission and reception may be programmed to any baud rate from 3.9Kbps to 800Kbps; however, the slew rate doesn’t provide for a good signal beyond 150Kbps.

The API routine `AR_SET_DEVICE_CONFIG` provides the method to set the transmit and receive channel bus speed and parity options for your device.

ARINC 429 Transmit Tri-state Support

Some CEI-x30 products have the capability to tri-state the output of transmit channels. Currently, only the RAR-PCIE has this capability. See chapters 15 and 16 for a description of how to use this feature.

ARINC 573/717 Protocol Support

Several aspects of the ARINC 573/717 HBP and BPRZ protocols are handled by the CEI-x30 products.

The electrical transmission of ARINC 573/717 data over the bus is performed at various bus speed/sub-frame size combinations resulting in the standard four-second frame duration. Each frame consists of four sub-frames comprised of a sub-frame sync word and subsequent data words. Each sync and data word is 12 bits long, transmitted in LSB-MSB order.

The transmit logic of a CEI-x30 board relies on the application to supply the frame data in a 16-bit unsigned integer array in which only the lower 12 bits of each 16-bit element are used. The application must supply the

applicable sub-frame sync words and data in the respective locations within this array for proper frame transmission.

The receiver logic of a CEI-x30 board supports both raw and auto-synchronized frame data reception. With raw data frame data reception, the data captured and provided to the application is organized in the least significant 12-bits of each element of a 16-bit unsigned integer array, based on the first detected bit transition. With auto-synchronized frame data reception, four application-provided sub-frame sync words are used by the CEI-x30 ARINC 717 receive logic to synchronize reception and data logging to a detected sub-frame sequence. The sub-frame detection is based both on the provided sub-frame sync word bit patterns and the specified sub-frame size.

The API routine `AR_SET_573_CONFIG` provides the method to set transmit and receive channel bus speed and frame size options for your device.

CEI-x30 Timers

The CEI-x30 products support two independent timers, a 64-bit one-microsecond timer and an optional IRIG timer. The one-microsecond timer is utilized for all ARINC 429 receive message time-tagging. It can be assigned to any 64-bit value by the host at any time.

Specified in microseconds from January 1st of the current year, received IRIG time is based on an external IRIG reference connected to the IRIG input of the CEI-x30 device (see the section, “IRIG B Signal Connections” for the procedure to connect the CEI-x30” device to the IRIG source). If the IRIG time reference is desired, but no external IRIG source is available, the CEI-x30 IRIG generator may be internally wrapped and used as the time source (see the `ARU_IRIG_WRAP_ENABLE` option of `AR_SET_CONFIG`); however, if the CEI-x30 IRIG generator is to be used by other data collection hardware in your system, it is best to externally connect the CEI-x30 IRIG output to its IRIG input. The CEI-x30 IRIG generator can be reset by the host application to any desired value using the standard IRIG time format, (see `AR_SET_TIME`).

A user-programmable compensation to the CEI-x30 IRIG time value can be defined when a consistent offset to the IRIG source time value is desired. This compensation should be used when a consistent skew in IRIG time-tagging is encountered between ARINC 429 events occurring on the CEI-x30 and other IRIG time-tagged components in your system. This offset can be specified via the `ARU_IRIG_SET_BIAS` option of `AR_SET_CONFIG`.

An IRIG DAC threshold adjustment procedure is provided that configures the CEI-x30 device IRIG receiver for optimal signal reception. This

procedure is usually not necessary; however, it may be required if IRIG timing appears unstable from a known good source.

First, test the stability of the IRIG signal by invoking `AR_GET_CONFIG` with the option `ARU_IRIG_CALIBRATED`. If this invocation returns a `FALSE` status, the adjustment should be invoked through use of the `AR_SET_CONFIG` routine with the `ARU_IRIG_QUICK_ADJUSTMENT` option. If the quick DAC adjustment is not successful, a more thorough adjustment may be performed. This adjustment is invoked through the `ARU_IRIG_ADJUST_THRESHOLD` option of the `AR_SET_CONFIG` routine. Execution of this IRIG adjustment may require at least one minute and should be performed only during the initialization of the board.

In addition to IRIG reception, CEI-x30 products can be configured to generate IRIG time using on-board IRIG circuitry. The transmitted IRIG time value is initialized to the host calendar time by the API, and can be modified by the host application via `AR_SET_TIME`.

Receive Message Time-tagging and Timer Usage

The CEI-x30 products time-stamp ARINC 429 received messages in the respective receive buffer and in the snapshot buffer (with label-only snapshot storage enabled), with a 64-bit one-microsecond time-tag. This time-tag value is based on the on-board timer, recorded when the last bit of the 32-bit message is detected. The CEI-x30 API supports multiple time-tag reference methods based on this one-microsecond timer. The active timer reference mode may be assigned by the host application by invoking the API routine `AR_SET_CONFIG`, using the `ARU_RX_TIMETAG_MODE` option and the selections discussed below. This assignment determines the format of the timer/time-tag value returned from all API invocations providing time-related information.

The following receive message time-tag and timer-read reference modes are available for selection:

IRIG 64-Bit Time Reference

This time reference is based on the CEI-x30 IRIG receiver, with the one-second resolution extrapolated by the one-microsecond internal timer to provide an estimated one-microsecond IRIG reference value. When the IRIG time reference is selected, all legacy receive data API routines based on a 32-bit time-tag parameter return a time-tag value with a resolution of one millisecond; while all receive data API routines supporting a 64-bit time-tag will return an IRIG timer-based time-tag.

The CEI-x30 device records the internal timer value when the IRIG signal is received and decoded, (referred to as IRIG Reference Time). The API then calculates the offset between the IRIG Reference Time and the received ARINC data time-tag. Finally, the API applies that offset to the IRIG signal time value to produce an IRIG-reference message time stamp for the received data, extrapolated to provide the 1 microsecond resolution.

Internal 64-Bit One Microsecond Time Reference

This time reference is based on the CEI-x30 device one-microsecond timer. When this mode is active, all legacy receive data API routines based on a 32-bit time-tag and all routines based on a 64-bit time-tag return a time-tag value with a resolution of one microsecond. The 32-bit time-tag is returned as the lower 32-bits of the 64-bit time-tag. This internal timer can be reset by the host application to any value desired, (see `AR_SET_TIME`).

Internal 32-Bit Twenty Microsecond Time Reference

This time reference is provided for backward compatibility to applications designed around the CEI-710 or IP-429HD-based products. When this mode is active, all legacy receive data API routines based on a 32-bit time-tag return a time-tag value with a resolution of twenty microseconds; all receive data API routines based on a single 64-bit time-tag value return a 32-bit value with a resolution of twenty microseconds (the upper 32-bits of the time-tag is zero).

Internal 32-Bit One Millisecond Time Reference

This time reference is based on a scaled version of the CEI-x30 device one-microsecond timer. When this mode is active, all legacy receive data API routines based on a 32-bit time-tag/timer value return a 32-bit value with a resolution of one millisecond; all receive data API routines based on a 64-bit time-tag/timer value return a 64-bit value with a resolution of one millisecond.

CEI-x20 Compatible Time Reference

This time-tag and timer option is not available as a selection via `AR_SET_CONFIG/ ARU_RX_TIMETAG_MODE`; instead, this time reference mode is selected when the CEI-x20 legacy API routine `AR_SET_TIMERRATE` is invoked. In this mode, time references are based on a programmable time-tag resolution specified through `AR_SET_TIMERRATE`. When this mode is active, all receive data API routines return a time-tag/timer value based on either a 32-bit or 64-bit

value scaled using the application-defined resolution. The message rate and start offset attributes assigned to scheduled message table entries are also scaled to the application-defined resolution.

Receive Message Buffering Methods

The CEI-x30 products provide three methods of message storage for receiving ARINC 429 data.

Individual Circular Buffer Storage

The first storage method is the individual circular buffer, (sometimes referred to as a sequential or FIFO buffer), in which ARINC 429 messages are continuously saved in the order in which they are received. Each receiver can store up to 2048 messages in its individual circular buffer before overflowing.

Merged Circular Buffer Storage

The second storage method is the merged circular buffer. This buffer provides for time-based sequentially ordered receive message buffering for multiple receive channels in a single buffer. Each receiver can be individually enabled for storage in the merged circular buffer; however, message storage in the merged buffer is exclusive of the individual receive buffers. The merged circular buffer can store up to 16384 messages before overflowing.

It is important to note that once a buffer overflows, all messages previously contained therein are lost. For this reason, when using either of these circular buffer storage methods for data retrieval, the host application should periodically flush the buffers. The API routines `AR_GETWORD*`, `AR_GETNEXT*`, `AR_GET_DATA*`, and `AR_GET_BLOCK*` are provided to support various data retrieval methods for receive circular buffer storage.

Snapshot Buffer Storage

The third storage method is a snapshot buffer, sometimes referred to as dedicated storage. Independent of either circular buffer storage method, snapshot storage records the latest received message for each ARINC 429 Label and optional SDI field value combination on a particular channel. The API routine `AR_SET_DEVICE_CONFIG` should be invoked using the receive channel attribute `ARU_ACCESS_SNAPSHOT_BUFFER` to

assign the snapshot storage mode based on the label field value alone or the combined label/SDI field values.

When the snapshot storage mode is set to store messages based on the label field value alone, a 64-bit one microsecond time-tag is also recorded in the buffer with each message.

Prior to receiving a message with a specific label/SDI combination, the table is initialized to zero.

The API routines `AR_GET_LATEST` (message only) and `AR_GET_LATEST_T` (message with time-tag) should be invoked to retrieve ARINC 429 data from the snapshot buffer when the snapshot storage mode is set to *Label Only*; use `AR_GET_SNAP_DATA` when the storage mode is set to *Label and SDI*. Any zero value data retrieved from this buffer using either routine is an indication that the respective message has not been received on that channel.

Interrupts and Triggers

There are two interrupt and trigger sources available with the CEI-x30 products. Any event that can cause a trigger can also invoke a hardware interrupt on the PCI bus. For simplicity of discussion, the term “interrupt event” applies to both interrupts and triggers for the remainder of this section regardless of whether or not an actual hardware interrupt is generated as a result.

The mechanism by which an interrupt event is logged is the interrupt queue. The interrupt queue is a 2048 entry circular buffer, consisting of a single 32-bit value for each generated interrupt event. The most recent entry written to the interrupt queue by the CEI-x30 firmware is indicated by the interrupt queue head pointer, accessed by reading the Interrupt Queue Register. The precise numeric definition for the individual interrupt queue entries is described in the section “Interrupt Queue”, in Chapter 14.

The first type of interrupt event is based on the Receive Label Filter functionality, described in detail in the next section of this chapter. The second type of interrupt event is based on host interaction with the device. Typically used in verification of a custom interrupt service routine, this type of interrupt is triggered by a host write to the Interrupt Queue Register in the firmware interface. When a write-access to this register is detected, an entry value of 255 is written to the next entry in the interrupt queue and the interrupt queue head pointer value is incremented. You can invoke this feature using the API routine `AR_SET_DEVICE_CONFIG` with the *item* parameter option `ARU_INSERT_INT_Q_ENTRY`.

Hardware interrupts passed on to the PCI bus can be enabled or disabled, independent of the interrupt queue operation. If the `INTERRUPT`

ENABLE bit is set to Enabled in the Global Enable Register, an interrupt is also generated on the PCI bus for each event written to the interrupt queue. You can set this bit by invoking the API routine, `AR_SET_DEVICE_CONFIG`, using the *item* parameter option `ARU_HW_INTERRUPT_ENABLE`.

The CEI-x30 API provides a general interrupt service routine (ISR) for all hardware interrupt processing. When PCI interrupts are enabled, the default ISR logs the interrupt entry in an internal API interrupt buffer, for recall by the API routine, `AR_HW_INTERRUPT_BUFFER_READ`. The host application can replace the invocation of the default ISR with an invocation of a custom host-supplied ISR via the API routine `AR_SET_ISR_FUNCTION`. The host may also defer generation of PCI interrupts and monitor the CEI-x30 device interrupt queue activity via invocation of the API routine `AR_INTERRUPT_QUEUE_READ`.

ARINC 429 Receive Label Filtering and Interrupt Event

The CEI-x30 products provide the capability to both filter received ARINC 429 messages from storage in the circular and snapshot buffers, and generate a PCI interrupt based on matching receive message bit-field values. The trigger definition for ARINC 429 message filtering and interrupt generation is based on the combination of matching 8-bit Label value, 2-bit SDI field value, and 3-bit ESSM field value, with these fields defined within a 32-bit ARINC 429 message as follows:

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

Each receiver contains a separate label filter table section in which the trigger definition is applied. This table is used by the CEI-x30 firmware to control storage of received labels to both the circular and snapshot buffers and generate a PCI interrupt, with each table entry defined via the CEI-x30 API as follows:

Note:

These definitions are CEI-x20 API-compatible and do not match the actual bit definition defined in the CEI-x30 device Label Filter Table.

<code>FILTER_SEQUENTIAL</code>	0x10	If SET filter label from the circular receive buffer
<code>FILTER_SNAPSHOT</code>	0x20	If SET filter label from the snapshot receive buffer
<code>FILTER_INTERRUPT</code>	0x40	If SET insert channel # in the interrupt queue and if enabled, generate a PCI interrupt

When buffer filtering has been enabled for a specified Label/SDI/ESSM combination, messages received with matching bit field values are discarded for the respective receive buffer until buffer filtering for that specified message has been disabled. Selection of individual and merged circular buffer storage is independent of the filter definition, with the `FILTER_SEQUENTIAL` option being applied to the buffer based on the respective receive channel's active circular buffer storage mode. All label filtering is disabled for each Label/SDI/ESSM combination by default.

When the `FILTER_INTERRUPT` bit is set for a specified label/SDI/ESSM combination, any message received containing that combined field value triggers an entry in the Interrupt Queue. This type of interrupt event is referred to as a *receive interrupt event*, with entry values for receivers 1 through 32 ranging from 64 to 95, respectively.

ARINC 429 Periodic Message Scheduling

The CEI-x30 message scheduling feature supports periodic message transmission of ARINC 429 label data, with a total of 1024 message entries. It is programmed by writing message and rate information to the Message Scheduler table, supported by the API routines `AR_DEFINE_MSG` and `AR_DEFINE_MSG_BLOCK`. As a part of the API initialization of the device, the Message Scheduler table is reset to an empty state. Once entries are defined by the host application, message scheduling is enabled by invoking the `AR_GO` routine.

When enabled, the Message Scheduler queries each table entry on a one millisecond basis, checking for all messages required for transmission at that particular millisecond value. The entire table is processed each millisecond, with the lowest table entry being processed first and highest table entry last. When invoking `AR_STOP` or `AR_RESET`, it is important to note the scheduler processing responds only to being disabled at the beginning of a one millisecond epoch. If the scheduler is requested to disable in the middle of creating scheduled traffic, all of the ARINC words previously scheduled for that millisecond are loaded into the various transmit buffers before the scheduler transitions to idle.

The efficiency of the Message Scheduler is based on the number of defined messages and the frequency at which those messages are transmitted. While the Message Scheduler feature is designed to be very accurate, there are ways in which the host application definition of channel-specific message transmission scenarios may cause deviations in the periodic transmission of the messages defined therein. The most common deviation is referred to as message rate skew.

Message Rate Skew

Message rate skew is defined as the characteristic of a scheduled message appearing on the bus at a rate that is either above or below the defined rate by a significant percentage. Message rate skew typically occurs when several different message rates are defined simultaneously on the same channel, and a majority of these message rates are multiples of the other message rates. The example below illustrates this situation.

Assuming there are three groups of messages being transmitted at rates of 100 msec (referenced as block A), 200 msec (block B) and 300 msec

(block C). If messages from each of the different rate groups were defined in the order of rate priority using same initial starting point of reference, the transmission of data would be defined as follows:

Time	Group
100	A
200	A,B
300	A,C
400	A,B
500	A
600	A,B,C
repeat...	

If we assume that each group of messages requires 10 msec to transmit, we can expand the timeline in more detail as follows:

Time	Group
100	A
200	A
210	B
300	A
310	C
400	A
410	B
500	A
600	A
610	B
620	C
700	A
800	A
810	B
900	A
910	C
etc.	

The time between the first two occurrences of the 300 msec message group (block C) is 310 msec. The time between the second and third occurrences of this group is 290 msec. Message skew like this is unpredictable as the number of different message rates increases.

The solution to this problem is to use the *start offset* feature of the message scheduler, (see the description for AR_DEFINE_MSG). In the next alternative example, the 300 msec message group was defined with a start offset of 20 msec (see note below). In this scenario, no message rate skew would occur, (as shown in the following timeline).

Note:

The 20 msec offset was derived as the sum of the duration required to transmit the groups that precede this group in the scheduling order.

Time	Group
100	A
200	A
210	B
300	A
320	C
400	A
410	B
500	A
600	A
610	B
620	C
700	A
800	A
810	B
900	A
920	C
etc.	

In this transmission example, any start offset from 20 msec to 80 msec would suffice for the 300msec (block C) message group. A start offset greater than 90 msec would cause this message group to overlap into the next scheduled frame for the block A message group and would subsequently induce skewing for those messages.

If the three message rate groups were all even multiples of each other (e.g. 100, 200, and 400 msec) then rate skew would never occur. The good news is that, although the slowest rate messages are most susceptible to rate skew, they are also typically the most tolerant to variation in transmission time.

Since the message scheduler processes all messages in the order of their location in the schedule table, rate skew may also appear on faster rate messages defined further into the table following slower rate messages. This skew can be easily eliminated by defining faster rate messages first and slower rate messages last.

In conclusion, if the minimum rate skew is desired on all transmitted messages, you must make an a priority determination of the message loading on each channel and insure messages are scheduled not only with the fastest rates first, but taking full advantage of the start offset feature.



BusTools/ARINC™ Data Bus Analyzer

General Information

BusTools/ARINC™ is an optional ARINC 429 analysis and simulation utility which runs under 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.ge-ip.com/products/1068'. The demo software operates over a simulated ARINC 429 interface board, but is otherwise identical to the full version.



Program Interface Library

Overview

GE Intelligent Platforms Embedded Systems supplies an extensive software Application Programming Interface (API) for the CEI-x30 supported products. API routines are supplied to setup the interface, configure channel attributes, and transmit and receive data for the most common desktop and embedded programming environments (Windows, Linux, Integrity, and VxWorks).

API Source Files

This library of utility routines provides the ability to write your own programs to interface with a CEI-x30 product. They are written in C and delivered in a generic ANSI C compiler-compatible format. They can be called from other languages by adhering to the procedures defined in the applicable documentation. The API consists of the following C source files:

CDEV_API.C

This file contains the bulk of the API functionality. Most of the routines that interact directly with the hardware device reside within this file.

CDEV_API.H

This header file contains the majority of the API constants, data types, and function prototypes, and should be included in all C programs that reference one or more CEI-x30 API utility routines.

CDEV_GLB.H

This header file contains the majority of the API global variables, internal definitions, and data structures.

AR_ERROR.H

This header file contains the error string constant definitions utilized by the API routine `AR_Get_Error`, describing each of the potential error codes returned by the CEI-x30 API utility routines.

CDEV_HW.H

This header file contains all of the API constants that define the hardware interface for the CEI-x30 architecture; included in `CDEV_API.H`.

CEI_TYPES.H

This header file contains all of the type defines for the various data types used with the respective operating system and compiler; included in `CDEV_API.H`.

CDEV_WIN.C

This file contains the C routines that interface directly with the ARINC common low-level driver interface library, `CEI_LL.LIB/DLL`, supporting all Windows operating systems.

CDEV_VXW.C

This file contains the C routines that interface directly with the VxWorks kernel.

CDEV_LNX.C

This file contains the routines that interface directly with the Linux kernel driver provided with the CEI-x30 Linux distribution archive.

CDEV_INT.C

This file contains the routines that interface directly with the Integrity operating system.

CDEV_LRT.C

This file contains the routines that interface directly with the LabVIEW Real-Time operating environment.

CDEV_FW.H - Firmware Load Files

The header file CDEV_FW.H is included in the source file CDEV_API.C, containing the array declarations for all CEI-x30 board firmware. The default compilation of CDEV_API.C includes the firmware load modules for the entire CEI-x30 product line. When the compiler directive LABVIEW_RT is defined, only the firmware for the CEI-830, R830RX, RCEI-530, and RAR-CPCI boards are included in the build, as these are the products currently supported with LabVIEW Real-Time. When the compiler directive INTEGRITY_PCI_PPC is defined, only the CEI-830 firmware is included in the build. The RAR-PCIE firmware is loaded from Flash Memory, and is not subject to a required firmware header file.

If for any reason you wish to reduce the API library or object module size by omitting the firmware load modules for extraneous boards, you may replace the header file reference in the respective include statement(s) with the header file FPGAX30N.H. For example, to omit the RAR-CPCI firmware load module you would modify line 86 of the file CDEV_FW.H as follows:

```
static CEI_UINT32 const fpga_630[]={
    #include "fpgax30n.h"
};
```

The firmware header files are referenced as follows:

FPGA830.H	CEI-830 Firmware
FPGA830RX.H	R830RX Firmware
FPGA430.H	CEI-430 Firmware
FPGA430A.H	CEI-430A Firmware
FPGA530.H	CEI-530 Firmware
FPGA630.H	RAR-CPCI Firmware
FPGAA30.H	AMC-A30 Firmware
FPGA_EC.H	RAR-EC Firmware
FPGAX30N.H	Two element array (empty f/w allocation)

Windows Libraries

For the CEI-x30-SW supported products, separate 32-bit and 64-bit Windows API Libraries are provided. For Windows OS target implementation, all API function prototypes are declared “_stdcall”. The CEI-x30 API library included in the installation is referenced as:

- CDEV_API.LIB 32-bit Microsoft VS6.0 Library
- CDEV_API.DLL 32-bit Microsoft VS6.0 DLL
- CDEV_API64.LIB 64-bit Microsoft VS2008 Library
- CDEV_API64.DLL 64-bit Microsoft VS2008 DLL

Included with the installation are the GE Common Low-level driver interface and installation verification libraries (not required for linking application programs):

- CEI_Install.DLL 32-bit Microsoft VS6.0 DLL
- CEI_Install64.DLL 64-bit Microsoft VS2008 DLL

All DLLs are installed in the Windows “System” folder. The exact folder name depends on the host version of Windows operating system. The 32-bit versions of these DLLs are typically installed in either ‘c:\winnt\system32’ or ‘c:\windows\system32’ under 32-bit Windows or ‘c:\windows\syswow64’ under 64-bit Windows. The 64-bit versions of these DLLs will be installed in the 64-bit Windows system folder (typically ‘c:\windows\system32’ under 64-bit Windows).

Time-tag Structure Definition

The following API routines use the AR_TIMETAG_TYPE data structure definition in providing the timer/time-tag reference or as an initial value for the reset of the internal timer:

- AR_GET_TIME
- AR_SET_TIME
- AR_GETNEXT_XT
- AR_GETWORD_XT
- AR_GET_DATA_XT
- AR_CONVERT_TIME_TO_STRING

Under the Windows and VxWorks operating systems, the AR_TIMETAG_TYPE data structure and pAR_TIMETAG_TYPE pointer types used by these routines are defined to use 64-bit integer values, as follows:

timeTagFormat	__int64 or long long
	The format of the corresponding <i>timeTag</i> structure member. Valid values for this element are: AR_TIMETAG_EXT_IRIG_64BIT AR_TIMETAG_INT_USEC_64BIT AR_TIMETAG_HOST_USEC_64BIT ¹ AR_TIMETAG_INT_20USEC_32BIT AR_TIMETAG_INT_MSEC_32BIT AR_TIMER_X20_COMPAT_32BIT
timeTag	__int64 or long long
	The timer-referenced time-tag, formatted as specified in the <i>timeTagFormat</i> structure member.
referenceTimeTag	__int64 or long long
	The 64-bit, one microsecond timer value corresponding to the time value supplied in the timeTag member.

Setting the Device Time

When assigning an initial time reference, the host application may choose to set either the device 1 microsecond timer or the IRIG generator timer via invocation of AR_SET_TIME.

When AR_SET_TIME is invoked with an AR_TIMETAG_TYPE data structure parameter *timeTagFormat* member defined to be AR_TIMETAG_EXT_IRIG_64BIT, the format of the *timeTag* member is defined as a 30-bit entity of BCD-like values using the following format:

29-28	27-24	23-20	19-18	17-14	13-11	10-7	6-4	3-0
hundreds of days	tens of days	days	tens of hours	hours	tens of minutes	minutes	tens of seconds	seconds

When AR_SET_TIME is invoked with a *timeTagFormat* member defined to be AR_TIMETAG_INT_USEC_64BIT, the *timeTag* member is referenced as a 64-bit 1 microsecond timer value.

¹ This format returns the host OS system time value converted to have a 1 microsecond resolution, supported only by the API routine ar_get_time().

Return Status Values

The following return status values are used by the CEI-x30 API routines. They are defined in the C header file CDEV_API.H and are used in the following context:

C Constant	Value	Constant Definition
ARS_FAILURE	-1	Requested operation failed
ARS_NODATA	0	No data was detected or received
ARS_NORMAL	1	Normal successful completion
ARS_GOTDATA	4	Data was received
ARS_BAD_MESSAGE	5	Receipt of an invalid ARINC 429 message was detected
ARS_INVHARVAL	1003	Invalid configuration value
ARS_XMITOVRFLO	1004	Transmit buffer overflow
ARS_INVBOARD	1005	Invalid board argument
ARS_NOSYNC	1006	Transmit buffer flush failed
ARS_MEMWRERR	1013	SRAM memory test error
ARS_INVARG	1019	General invalid argument value
ARS_DRIVERFAIL	1021	Driver failed to install or uninstall the ISR
ARS_WINRTFAIL	1022	Device driver open failure
ARS_CHAN_TIMEOUT	1023	Channel timeout in receive function
ARS_NO_HW_SUPRT	1024	Function not supported by specified hardware
ARS_HW_CONSISTENCY	1029	Device is not programmed for Enhanced Firmware operations
ARS_WRAP_DATA_FAIL	1031	BIT wrap test data read-back fail
ARS_WRAP_FLUSH_FAIL	1035	BIT cannot execute external wrap test due to unknown external data reception
ARS_WRAP_DROP_FAIL	1036	BIT wrap test data not received
ARS_INT_ISR	1037	Driver failed to install or uninstall API interrupt support
ARS_BOARD_MUTEX	1038	API routine failed to acquire or release a board lock mechanism
ARS_NO_OS_SUPPORT	1041	There is no operating system support for the requested feature
ARS_ERR_SH_MEM_OBJ	1050	API failed to allocate a shared object (semaphore or mutex)

C Constant	Value	Constant Definition
ARS_ERR_SH_MEM_MAP	1051	API failed to allocate a shared memory region (multi-process)

Programming with the CEI-x30 API Interface

Following the outline below, you can easily incorporate the CEI-x30 API into your application.

1. For your application to interface to any CEI-x30-SW supported products the device must first be initialized. Invoke the AR_LOADSLV routine with parameters as described in the API Routines section.
2. Assign the characteristics of the transmit and receive channels if the default configuration is not appropriate. This is performed with multiple invocations of either AR_SET_DEVICE_CONFIG or AR_SET_CONFIG.
3. Perform receiver buffer mode selection based on individual channel/protocol usage via the routine AR_SET_DEVICE_CONFIG. Using buffered mode for multiple channels of the same protocol provides channel-specific access to received data, where merged mode provides for single receive channel access to selected channel data.
4. Once channel configuration is complete, invoke AR_GO to initiate data processing.
5. Then invoke AR_PUT_429_MESSAGE and AR_GET_429_MESSAGE to send and receive single ARINC 429 messages, respectively.
6. When communication is complete, invoke AR_STOP to suspend active data processing.

Subsequently, you could invoke AR_GO again to restart the interface.

7. On termination of the application, invoke AR_CLOSE to release all resources acquired during initialization. It is very important that all applications invoke AR_CLOSE upon termination; otherwise, the operating system does not release the memory acquired when the API was initialized.

The example wrap program source code, contained in TST_CNFG.C, is supplied with your installation. This program demonstrates the use of the API for the ARINC 429 and equivalent protocols.

When calling the utility routines that return a status value, it is important to verify the returned status indicates success; otherwise, the application may not be aware that an important function may have failed to fulfill a requested operation.

Example Routines – Summary

Example applications demonstrating various CEI-x30 API features are provided in C source format, as described in the following paragraphs.

Tst_cnfg.c

The example source file TST_CNFG.C is included with your installation. To access this example executable under the Windows operating system:

1. Click Start, and then Programs.
2. Select GE CEI-x30-SW and then Test Configuration.

Within TST_CNFG.C are application-style routines demonstrating use of the API routines for the ARINC 429 protocol:

test_basic_arinc_429	an internal wrap test designed to demonstrate ARINC 429 API usage. This routine enables internal wrap on all 429 receive channels. It also assigns a bus speed of 12.5kbps and ODD parity to both transmit and receive channels. Ten ARINC 429 messages are sent on each transmit channel and proper reception verified on the respective receive channel.
demo_advanced_arinc_429	a demonstration of the following advanced features available with CEI-x30 products: <ul style="list-style-type: none"> Transmit Message Scheduling Enhanced Label/SDI/ESSM Data Filtering Snapshot Message Data Acquisition Enhanced Time-tag Reset and Conversion IRIG Time-tag Selection (if installed on hardware)
demo_discrete_io_features	a demonstration on the use of the Discrete I/O Channels and the respective API routines
demo_irig_features	a demonstration of IRIG features requiring an external IRIG connection: <ul style="list-style-type: none"> IRIG DAC Threshold Adjustment IRIG Bias (Offset) Time Assignment IRIG Validity Determination IRIG Time Conversion and Display
test_arinc_717	an internal wrap test designed to demonstrate ARINC 573/717 API usage. This routine enables internal wrap on the ARINC 573/717 receive channel. It also assigns a

	bus speed 768bps, a sub-frame size of 64 words, and a BPRZ selection to the ARINC 573/717 transmit and receive channels. A frame consisting of a data pattern incrementing from \$01 to \$FF and sync words of \$123, \$224, \$325, and \$426 is transmitted and proper reception verified.
demo_pci_interrupts	This routine demonstrates how to setup a custom interrupt service routine, setup Label Filter Table triggers generating interrupt events, enabling interrupts, and retrieving interrupt event data from the queue.
custom_interrupt_handler	This routine is the custom interrupt service routine assigned within the example routine demo_pci_interrupts.

Multiprocess_test.c

Provisions for simultaneous multiple process access to a CEI-x30 board are supported under the Windows and Linux operating systems. This feature is implemented in the standard API with minimal requirements on the application developer. The example program contained in the source file multiprocess_test.c describes how one method of multi-process application may be implemented.

This example application is based on separate processes, one for ARINC 429 scheduled message transmission on multiple transmit channels, and others for ARINC 429 message reception on individual receive channels. Regardless of the process invocation type, setup for the multiple process application is performed via invocation of the API routine AR_SET_PRELOAD_CONFIG, prior to invocation of AR_LOADSLV.

A single "transmit" process should be launched first, with subsequent invocation of one or more "receive" processes. The "transmit" process loads the board using AR_LOADSLV, configure board and transmit channel specific parameters using AR_SET_DEVICE_CONFIG, and activate data processing using AR_GO. After the "transmit" process invokes AR_GO, it is permissible to launch one or more "receive" processes. A "receive" process first attaches to the board using AR_LOADSLV, then execute operations strictly confined to the particular channel to which that process is associated (in this case AR_SET_DEVICE_CONFIG and AR_GETWORDT). When finished, each "receive" process invokes AR_CLOSE and terminates. The "transmit" process should remain running until all "receive" processes have terminated; upon termination the "transmit" process invokes AR_STOP and AR_CLOSE.

While the previously discussed application API invocation order is recommended, it is not strictly required. You may actually invoke a "receive" process first and terminate it last; however, multi-process applications should rely on a primary board-control process as the focus for board initialization, BIT functionality, and control of h/w message processing.

Visual Basic

A text file, CDEV_API_VB.TXT, is provided to aid the Visual Basic programmer in using the CDEV_API DLL in the Examples\VB folder of the software distribution. This text file contains the Function Declaration and Global Constant statements required to interface to CDEV_API.DLL. You can manually copy and paste text from this file to your project, or you can use the Microsoft API Text Viewer utility included with Visual Basic. For more information on the API Text Viewer, consult Microsoft Visual Basic documentation.

The CDEV_API_VB.TXT is designed for use with any 32-bit version of Visual Basic; however, the VB example and API interface are recommended for use with Visual Basic version 6.0 or later.

Working with Unsigned Integers in Visual Basic

Visual Basic doesn't support unsigned integers. Since the CEI-x30 API library uses unsigned integers for some function parameters, problems can arise when attempting to set values in the upper half of the range.

Example

When a CEI-x30 API function uses an argument of C type *unsigned short* the equivalent type is *integer* in Visual Basic. Both are 16-bit values but the Visual Basic variable has a range of -32768 to 32767. The C argument has a range of 0 to 65535. A Visual Basic error is generated if an *integer* type is set to value greater than 32767.

Solutions

There are two solutions to this problem. The easiest is to set the value of variables directly in Hex. To set an integer variable to 65535 use: `myVariable = &HFFFF`. (note the &H syntax) . For setting long variables use the ending "&" as in `myvar = &H12&`.

The second solution is to convert the desired unsigned value to the signed equivalent. This can be accomplished in a small utility function:

```

Function u_conv (unsigned as Long) as Integer
    Dim signed as Integer
    If unsigned > 32767 then
        signed = unsigned - 65536
    Else
        signed = unsigned
    End If
    u_conv = signed
End Function

```

When using returned values from CEI-x30 API functions the opposite conversion can be made. Often, the returned values from CEI-x30 API functions simply need to be compared to the predefined values.

For more information on using Visual Basic with unsigned integers, consult the Microsoft Support Knowledgebase Article ID: “112673 *How to Pass & Return Unsigned Integers to DLLs from VB*”.

API Routines - Summary

The routines provided in the API supporting the CEI-x30 device features are defined in the following pages, categorized and summarized:

Initialization and Control Routines

ar_loadslv	The main initialization routine, ar_loadslv() acquires the resources for the PCI memory regions and initializes the CEI-x30 device.
ar_board_test	Verifies the CEI-x30 data processing capabilities via internal/external data wrap.
ar_bypass_wrap_test	Controls conditional execution of the ARINC 429 internal wrap test within <i>ar_initialize_device</i> .
ar_initialize_api	Initializes the CEI-x30 device API.
ar_initialize_device	Initializes the CEI-x30 device to the default state.

Device Control Routines

ar_go	Enables CEI-x30 ARINC data processing.
ar_reset	Disables CEI-x30 ARINC data processing and initializes the device to the default state.
ar_stop	Disables CEI-x30 ARINC data processing.

Termination Routines

ar_close	Releases all resources for the specified device.
----------	--

Receive/Transmit Channel-level Configuration Routines

ar_set_device_config	As the main channel configuration routine, it assigns ARINC 429-specific transmitter and receiver channel configuration information.
ar_get_device_config	Retrieves the value of a bit field for I/O and ARINC 429 transmitter or receiver channel configuration registers.
ar_enh_label_filter	Assigns the enhanced label filter table definition for each CEI-x30 device receiver.
ar_get_config	Retrieves board-level configuration and API local attribute values.
ar_get_573_config	Retrieves the value of a bit field for an ARINC 573/717 transmitter or receiver channel configuration register.
ar_get_filter	Retrieves the specified label filter buffer entry from the enhanced label filter table.
ar_get_label_filter	Retrieves the active state of label filtering for a single label on all receivers.
ar_label_filter	Assigns ARINC 429 label values to be filtered by the specified receive channel.
ar_putfilter	Places the specified label filter buffer entry in the enhanced label filter table.
ar_set_config	Assigns board-level configuration and API local attribute values.
ar_set_573_config	Assigns ARINC 573/717 transmitter and receiver channel configuration information.

Device-level Configuration Routines

<code>ar_get_storage_mode</code>	Retrieves the API state of a device-generic receive data storage mode.
<code>ar_set_raw_mode</code>	Assigns both transmitter and receiver parity state on the specified ARINC 429 channel.
<code>ar_set_storage_mode</code>	Assigns the device-level receive data storage mode.

Receive Data Processing Routines

<code>ar_get_429_message</code>	Retrieves the next ARINC 429 message from a receive buffer. Optionally, it waits up to ½ second for data to become available.
<code>ar_get_573_frame</code>	Retrieves a specified number of ARINC 573 data words from the receive buffer.
<code>ar_getnext</code>	Retrieves the next message from the specified receive buffer. It waits up to ½ second for data to become available.
<code>ar_getnexttt</code>	Retrieves the next message and a 32-bit, 20 µsec time-tag from the specified receive buffer. It waits up to ½ second for data to become available.
<code>ar_getnext_xt</code>	Retrieves the next message with a 64-bit, user-programmable time-tag from the specified receive buffer. It waits up to ½ second for data to become available.
<code>ar_getword</code>	Retrieves the next message from the specified receive buffer.
<code>ar_getwordt</code>	Retrieves the next message and a 32-bit, 20 µsec time-tag from the specified receive buffer.
<code>ar_getwordt_xt</code>	Retrieves the next message from the specified receive buffer with a 64-bit, user-programmable time-tag.
<code>ar_get_data</code>	Retrieves the next available data and the 64-bit, 1 µsec time-tag from a receive buffer.

ar_get_data_xt	Retrieves the next available data from a receive buffer with a 64-bit, user-programmable time-tag.
ar_getblock	Retrieves all of the available ARINC 429 messages from the requested receive buffer with 32-bit time-tags.
ar_getblock_t	Retrieves all of the available ARINC 429 messages from the requested receive buffer, with 64-bit time-tags.
ar_get_latest	Retrieves the latest message from the snapshot buffer for the specified channel/label combination.
ar_get_latest_t	Retrieves the latest message and time-tag from the snapshot buffer for the specified channel/label combination.
ar_get_snap_data	Retrieves the latest message from the snapshot buffer for the specified channel/label/sdi combination.

Transmit Data Processing Routines

ar_define_msg	Defines a scheduled ARINC 429 messages.
ar_define_msg_block	Defines a block of scheduled ARINC 429 messages.
ar_modify_msg	Modifies an existing ARINC 429 message already defined for periodic transmission.
ar_modify_msg_block	Modifies a block of ARINC 429 messages already defined for periodic transmission.
ar_put_429_message	Places a single message in the specified ARINC 429 transmit buffer.
ar_put_573_frame	Places a specified number of ARINC 573 data words in the transmit buffer.
ar_putword	Places a single message in the specified ARINC 429 transmit buffer.
ar_putblock	Places multiple messages in a single ARINC 429 transmit buffer.
ar_putblock_multi_chan	Places multiple messages in multiple ARINC 429 transmit buffers.

Timer-related Routines

<code>ar_get_time</code>	Retrieves the current hardware reference time based on the selected timer mode.
<code>ar_get_timercntl</code>	Retrieves the current value of the OS timer.
<code>ar_reset_timercnt</code>	Resets the internal timer/time-tag reference to zero.
<code>ar_set_timerrate</code>	Assigns the CEI-x30 compatible timer resolution for use with <code>ar_get_time()</code> and any ARINC 429 receive data routines returning 32-bit time-tag values.
<code>ar_set_time</code>	Sets the internal clock/timer or IRIG time generator to an application supplied value.

Information and Status Routines

<code>ar_get_base_addr</code>	Retrieves a pointer to the base address of the address space allocated to the specified device.
<code>ar_get_boardname</code>	Returns a string description of the specified device.
<code>ar_get_boardtype</code>	Retrieves the target device configuration.
<code>ar_get_error</code>	Retrieves a message string associated with a given error status code.
<code>ar_get_status</code>	Retrieves the combined state of each receive FIFO status register Data Available bit.
<code>ar_num_rchans</code>	Retrieves the number of receive channels supplied by the CEI-x30 device.
<code>ar_num_xchans</code>	Retrieves the number of transmit channels supplied by the CEI-x30 device.

Utility Routines

<code>ar_execute_bit</code>	Verifies the CEI-x30 operational state through various data wrap and timer tests.
<code>ar_hw_interrupt_buffer_read</code>	Returns the contents of the API maintained interrupt buffer entries.
<code>ar_interrupt_queue_read</code>	Provides host access to read the device interrupt queue.

<code>ar_set_isr_function</code>	Provides the method for the host application to define a custom interrupt service routine.
<code>ar_set_multithread_protect</code>	Enable/disable multithread access protection to all API routines accessing the hardware interface of the device.
<code>ar_set_preload_config</code>	Defines the process and thread setup for the calling application.
<code>ar_sleep</code>	Suspends the calling thread for a specified number of milliseconds.
<code>ar_wait</code>	Delays the calling application for the specified number of seconds.
<code>ar_version</code>	Retrieves the current software version number of the CEI-x30 API.

AR_BOARD_TEST

Syntax	CEI_INT16 ar_board_test (CEI_INT16 board, CEI_INT16 testType)	
Description	This routine performs a single message internal or external wrap test on each matched ARINC 429 transmit/receive channel pair. On successful completion of the wrap test, the board is initialized to the default state via invocation of AR_INITIALIZE_DEVICE.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVBOARD	An uninitialized or invalid <i>board</i> value was provided.
	ARS_MEMWRERR	The SRAM test write/read/verify failed.
	ARS_WRAP_DROP_FAIL	ARINC 429 wrap test data missing.
	ARS_WRAP_DATA_FAIL	ARINC 429 wrap test data pattern mismatch.
	ARS_WRAP_FLUSH_FAIL	Unexpected data from an external source was received during wrap test execution.
	ARS_XMITOVRFLO	A transmit buffer overrun occurred.
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.
	CEI_INT16 testType	(input) Type of test to execute. Valid values for this parameter are:
		INTERNAL_WRAP (4) EXTERNAL_WRAP (5)

AR_BYPASS_WRAP_TEST

Syntax CEI_INT16 ar_bypass_wrap_test (CEI_INT16 board, CEI_INT16 bypass)

Description

This routine defines an internal flag used to control the invocation of the AR_BOARD_TEST routine during execution of AR_LOADSLV. The routine AR_BOARD_TEST will perform a single-message internal wrap test for each matching ARINC 429 transmit/receive channel pair. The default state of this internal flag is ON, indicating no internal wrap test is executed during the board/API initialization process.

This routine should be invoked with the bypass parameter set to AR_OFF if you wish the API to perform an internal wrap test as part of the board/API initialization process. Note that if AR_LOADSLV is invoked with any ARINC 429 receive channel connected to an actively transmitting LRU, execution of AR_BOARD_TEST may return a false failure status indication.

Return Value

ARS_NORMAL Routine execution was successful.

ARS_INVBOARD An invalid *board* value was supplied.

Arguments

CEI_INT16 board (input) Device to access. Valid range is 0-15.

CEI_INT16 bypass (input) Bypass flag indicating whether or not to bypass the internal wrap test when the routine AR_LOADSLV is invoked. Valid values for this parameter are:

AR_ON (7) bypass internal test

AR_OFF (8) execute internal test

AR_CLR_RX_COUNT

Syntax	CEI_VOID ar_clr_rx_count (CEI_INT16 board, CEI_INT16 channel)	
Description	This routine resets the API-tracked count of ARINC data words received by a particular channel to zero. The CEI-x30 device maintains a count of ARINC data words received over the interface for each channel since the device was initialized. When this routine is invoked, the API saves the current count, to be used as the most recent reset reference and subtracted from the device count when the count value is requested by AR_GET_RX_COUNT.	
Return Value	None	
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.
	CEI_UINT32 channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.

AR_CLOSE

Syntax	CEI_INT16 ar_close (CEI_INT16 board)	
Description	This routine releases all resources acquired during the initialization of the specified device. Once this routine has been executed, invocation of other API routines results in the return of an invalid status.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_WINRTFAIL	Windows device driver failed to close the session with the device.
	ARS_FAILURE	Generic failure to close the driver session or terminate an active ISR.
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.

AR_CONVERT_TIME_TO_STRING

Syntax

```
void ar_convert_time_to_string (CEI_INT16 board, CEI_INT16
displayFormat, pAR_TIMETAG_TYPE timeIn, pCEI_CHAR timeString)
```

Description

This routine converts the time value provided in the timeIn structure to a character string representation of date/time, format based on what is specified via the displayFormat parm. The supplied time format (LSB resolution) must be specified in the timeIn structure member **timeTagFormat**, representing the resolution of the respective **timeTag** member data.

Return Value

none.

Arguments

CEI_INT16 board	(input) Device to access. Valid range is 0-15.
CEI_INT16 displayFormat	(input) Format for returned string:
AR_TD_REL_MIDNIGHT	Relative to Midnight Format and Full IRIG Format, defined as "(DDD)hh:mm:ss.uuuuuu"
AR_TD_IRIG	
AR_TD_DATE	Date Format defined as "(MM/DD)hh:mm:ss.uuuuuu"
pAR_TIMETAG_TYPE timeIn	(input) Source 64-bit time structure
pCEI_CHAR	timeString (output) Pointer to destination text string

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 32-bit ARINC 429 message for periodic retransmission at the specified rate. Once defined, the message rate, content, or assigned channel may be altered through AR_MODIFY_MSG.	
Return Value	Any positive value between 0 and 1023 is the unique message scheduler table entry index assigned to this message.	
	ARS_FAILURE	Indicates the routine encountered an uninitialized board, an invalid board/channel parameter value, or a full message scheduler table.
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.
	CEI_INT16 channel	(input) Channel message scheduler table this routine is to access. The valid range is 0 to one less than the number of installed transmit channels.
	CEI_INT16 rate	(input) Periodic transmission rate, defined in milliseconds by default. For backward compatibility to the CEI-x20 tick-timer message rate method, when AR_SET_TIMERRATE has been executed to simulate the CEI-x20 tick-timer resolution assignment within the CEI-x30 API, the rate and start parameters is scaled to the specified tick-timer resolution.
	CEI_UINT16 start	(input) Offset, (in milliseconds), from the start of CEI-x30 device message processing at which this message will begin its initial periodic transmission.
	CEI_INT32 data	(input) The 32-bit ARINC 429 message to transmit.

AR_DEFINE_MSG_BLOCK

Syntax	CEI_INT16 ar_define_msg_block (CEI_INT32 numberOfEntries, pAR_SCHEDULED_MSG_ENTRY_TYPE messageEntry)	
Description	This routine defines a series of 32-bit ARINC 429 messages for periodic retransmission at the specified rate. Once defined, the message rate, content, or assigned channel for any individual message scheduler table entry within this same structure may be altered via invocation of AR_MODIFY_MSG_BLOCK.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	Invalid <i>channel</i> parameter value.
	ARS_INVHARVAL	Message scheduling is not supported on the specified channel.
	ARS_FAILURE	Message scheduler table full indication.
Arguments	CEI_INT32 numberOfEntries (input) The number of entries to define from the subsequent structure pointer parameter, messageEntry.	
	pAR_SCHEDULED_MSG_ENTRY_TYPE messageEntry	(input)
	Array of structures of message definition content, defined as follows:	
	unsigned long messageIndex	The unique message scheduler table entry index assigned to this message. Upon completion of this routine, the messageIndex structure member will have been updated to reflect the message scheduler table index assigned to the respective message.
	unsigned long board	Device to access. Valid range is 0-15.
	unsigned long channel	Which channel portion of the message scheduler table this routine is to access. The valid range is 0 to one less than the number of installed transmit channels.

unsigned long rate	Periodic transmission rate, defined in milliseconds by default. For backward compatibility to the CEI-x20 tick-timer message rate method, when AR_SET_TIMERRATE has been executed to simulate the CEI-x20 tick-timer resolution assignment within the CEI-x30 API, the rate and start parameters will be scaled to the specified tick-timer resolution.
unsigned long start	Offset, (in milliseconds), from the start of CEI-x30 device message processing at which this message will begin its initial periodic transmission.
unsigned long txCount	The total number of times this message will be transmitted. The constant value ARU_SCHED_MSG_INFINITE (0xFFFFFFFF) indicates infinite transmission of this message is requested.
unsigned long data	The 32-bit ARINC 429 message to transmit.

AR_ENH_LABEL_FILTER

Syntax	CEI_INT32 ar_enh_label_filter (CEI_INT16 board, CEI_INT16 channel, CEI_UINT16 label, CEI_UINT16 sdi, CEI_UINT16 essm, CEI_INT16 action)
Description	This routine supports the assignment of both a single entry in the enhanced label filter table for the specified receive channel and channel-wide field definitions for the entire channel filter table. The CEI-x30 device enhanced label filtering feature supports the ability to both filter ARINC 429 messages and generate a hardware interrupt based on reception of a message matching the combined 8-bit label value, 2-bit SDI value, and 3-bit ESSM value.
Note:	<hr/> This routine should be used exclusive of the use of the legacy API routine AR_LABEL_FILTER, as any filter table value assigned with one routine supersedes a previous assignment with another. <hr/>

Label Filtering

Once message reception filtering has been enabled for a specified channel/label/sdi/essm combination, data received with matching bit field values will be discarded until label filtering for that specified message has been disabled.

Interrupt Generation

Once interrupt generation filtering has been enabled for a specified channel/label/sdi/essm combination, data received with matching bit field values induce an entry in the CEI-x30 device interrupt queue. If the device hardware interrupt has been enabled by invoking AR_SET_DEVICE_CONFIG with the option ARU_HW_INTERRUPT_ENABLE set to AR_ON, the device generates a PCI Interrupt to be serviced by the default interrupt service routine provided with the API or a custom ISR assigned by the host application.

The label filtering feature is disabled for all labels/sdi/essm combinations by default. Label filtering changes are effective immediately on completion of this routine.

Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.

Arguments	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided .
	ARS_INVHARVAL	Invalid <i>channel</i> parameter value.
	ARS_INVARG	Invalid <i>label</i> , <i>sdi</i> , <i>essm</i> , or <i>action</i> parameter value.
	CEI_INT16 board	(input) Device to access. Valid range is 0-15.
	CEI_INT16 channel	(input) Channel label filter table this routine is to access. The valid range is 0 to one less than the installed receive channel count.
	CEI_UINT16 label	(input) The <i>label</i> of interest. Valid range is 0-255. Also valid is ARU_ALL_LABELS (511), which invokes the action for all labels.
	CEI_UINT16 sdi	(input) The <i>SDI</i> field value of interest. Valid range is 0-3. Also valid is ARU_ALL_SDI (4), which invokes the action for all SDI entries for the specified label.
	CEI_UINT16 essm	(input) The <i>ESSM</i> field value of interest. Valid range is 0-7. Also valid is ARU_ALL_ESSM (8), which invokes the action for all ESSM entries for the specified label.
	CEI_INT16 action	(input) Enable or disable filtering action for this table entry. Valid values are:
	FILTER_SEQUENTIAL 0x10	If CLEAR add the respective message to the sequential receive buffer; if SET filter the respective message from the sequential receive buffer.
	FILTER_SNAPSHOT 0x20	If CLEAR add the respective message to the snapshot buffer; if SET filter the respective message from the snapshot buffer.
	FILTER_INTERRUPT 0x40	If CLEAR does nothing; if SET, receipt of the respective message creates a receive channel index entry in the device's interrupt queue and if enabled, generates a PCI interrupt.

AR_EXECUTE_BIT

Syntax

CEI_INT16 ar_execute_bit (CEI_INT16 board, CEI_INT16 testType)

Description

This routine performs hardware test functionality normally associated with board-level Built-In-Test (BIT). Testing ranges from a full SRAM memory test to verification of ARINC 429 message wrap on transmit/receive channel pair on the specified device.

Note:

This routine bypasses execution and returns a failure status if you invoke it when multi-process execution is enabled by AR_SET_PRELOAD_CONFIG and multiple processes are attached to a single board.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_MEMWRERR	Device SRAM test write/read/verify failure.
ARS_WRAP_DROP_FAIL	ARINC 429 wrap test data missing.
ARS_WRAP_DATA_FAIL	ARINC 429 wrap test data pattern mismatch.
ARS_WRAP_FLUSH_FAIL	Unknown external data received during wrap test execution.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided .
ARS_INVARG	Invalid <i>testType</i> parameter value.
ARS_FAILURE	Timer-deviation test failed or multi-process execution is enabled.

Arguments

CEI_INT16 board	(input) Device to access. Valid range is 0-15.
CEI_INT16 testType	(input) Type of test to execute, defined as follows:

AR_BIT_BASIC_STARTUP (0) invokes a basic device initialization to a reset state (all buffers flushed and channel configurations reset); however, the device firmware is not reloaded or restarted.

- AR_BIT_FULL_STARTUP (1) invokes device initialization, a full, destructive SRAM memory test, and an internal wrap test of all matched transmit/ receive channels, (regardless of prior invocation of the API routine AR_BYPASS_WRAP_TEST). The duration of this test is approximately 17 seconds.
- AR_BIT_PERIODIC (2) invokes a short, non-destructive SRAM memory test and a timer-deviation test, providing verification of the basic health status of the device.
- AR_BIT_INT_LOOPBACK (3) invokes an internal wrap test of all matched transmit/receive channels.
- AR_BIT_EXT_LOOPBACK (4) invokes an external wrap test of all matched transmit/receive channels.
- AR_BIT_PARTIAL_SRAM (8) invokes a short destructive test of select, unused SRAM locations
- AR_BIT_FULL_SRAM (9) invokes a destructive test of all SRAM locations
- AR_BIT_SELECT_SRAM_MIN to
AR_BIT_SELECT_SRAM_MAX (100 to 1123)
invokes a destructive test of a select block of SRAM, parsed into 1024 blocks of 512 locations each.

AR_GET_573_FRAME

Syntax	CEI_INT16 ar_get_573_frame (CEI_INT16 board, pCEI_UINT32 numberWords, pCEI_UINT16 arincData)	
Description	<p>This function retrieves <i>numberWords</i> of ARINC 573/717 data from the ARINC 573/717 receive channel. If any data is available, the actual number of words received is indicated in the return value of <i>numberWords</i>. If auto-synchronization is configured for the ARINC 573/717 channel, this function will search the receive buffer for any occurrence of the first sub-frame sync word (defined via invocation of AR_SET_573_CONFIG with the item set to ARU_573_SYNC_WORD1) and return the specified number of words of frame data following the instance of that sync word. With automatic synchronization selected and the full frame size specified in <i>numberWords</i>, this function will wait until the full frame is received and copied to the destination array. The acquisition of an entire ARINC 573/717 frame may require up to four seconds to complete.</p>	
Return Value	ARS_NODATA	No frame data was available.
	ARS_GOTDATA	At least one ARINC 573/717 data word has been retrieved.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVHARVAL	ARINC 573 support is not available on device.
	ARS_INVARG	Invalid <i>numberWords</i> or <i>arincData</i> parameter.
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.
	pCEI_UINT32 numberWords	(input/output) As an input this specifies the number of words to retrieve from the receive buffer. As an output this indicates how many words were retrieved from the receive buffer, less than or equal to the input value of <i>numberWords</i> .
	pCEI_UINT16 arincData	(output) The address that is to receive the frame data. The format of each data word in the ARINC 573/717 frame is defined as follows:

15	14	13 – 12	11 - 0
sync word	RESERVED	subframe	data

sync word: indicates this word was detected as a sync word, where a value of 1 indicates sync word and 0 indicates data word.

subframe: identifies the sub-frame assignment for this word, where 1 indicates sub-frame 1, 2 indicates sub-frame 2, 3 indicates sub-frame 3, and 0 indicates sub-frame 4.

data: the 12-bit ARINC 573/717 data.

AR_GET_429_MESSAGE

Syntax	CEI_INT16 ar_get_429_message (CEI_INT16 board, CEI_INT16 channel, CEI_INT16 waitState, pCEI_VOID data, pCEI_VOID timetag)	
Description	This routine retrieves the most recent ARINC 429 data and 32-bit time-tag from the specified channel. If no data is present in the receiver buffer, this routine attempts to retrieve data for up to one-half second. If no data is present after one-half second, a time-out status is returned. If no wait is specified and no data is available, the return status is so indicated.	
Return Value	ARS_GOTDATA	An ARINC 429 message and its time-tag have been retrieved.
	ARS_CHAN_TIMEOUT	No data available (if waitState is AR_ON).
	ARS_NODATA	No data available (if waitState is AR_OFF).
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided .
	ARS_INVHARVAL	Channel is not available on the device.
	ARS_INVARG	A NULL <i>data</i> parameter value was provided.
Arguments	CEI_INT16 board	(input) Device this routine is to access. Valid range is 0-15.
	CEI_INT16 channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
	CEI_INT16 waitState	(input) Whether or not to wait for data. A value of AR_ON specifies to wait ½ second for data; a value of AR_OFF specifies to return if no data is immediately available.

pCEI_VOID data	(output) The address that is to receive the data. The returned ARINC 429 data is always in normal ARINC format.
pCEI_VOID timetag	(output) The address that is to receive the 32-bit time-tag associated with the data, (resolution is programmable). If the merged receive mode is active for the specified channel, the upper five bits of the 32-bit time-tag word will contain the receive channel number on which the data was received. If the <i>timetag</i> parameter is NULL, time-tag information will not be provided.

AR_GET_BASE_ADDR

Syntax	pCEI_UINT32 ar_get_base_addr (CEI_INT16 board)	
Description	This routine returns the driver-acquired virtual base address for the PCI memory region for the host interface of the specified device. This routine should be invoked only after successfully invoking AR_LOADLSV.	
Return Value	Any positive value exceeding \$2000 is the driver-acquired virtual base address for the host interface PCI memory region of the specified device.	
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.

AR_GETBLOCK

Syntax

CEI_INT16 ar_getblock (CEI_UINT32 board, CEI_UINT32 channel, CEI_INT32 maxMessages, CEI_INT32 offset, pCEI_INT32 actualCount, pCEI_INT32 data, pCEI_INT32 timeTags);

Description

This routine retrieves all of the available ARINC messages from the requested receive channel buffer and copies them to the desired destination. If the *timeTags* parameter is not NULL, the 32-bit time-tag data associated with each retrieved message is also copied.

Return Value

ARS_GOTDATA	Message(s) and time-tag(s) were retrieved.
ARS_NODATA	No data was available.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided .
ARS_INVHARVAL	Channel is not available on device.
ARS_INVARG	Invalid <i>maxMessages</i> , <i>actualCount</i> , or <i>data</i> parameter was encountered.

Arguments

CEI_UINT32 board	(input) Device to access. Valid range is 0-15.
CEI_UINT32 channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
CEI_INT32 maxMessages	(input) The number of messages to retrieve.
CEI_INT32 offset	unused parameter, retained for legacy API support
pCEI_INT32 actualCount	(output) The number of messages retrieved.
pCEI_INT32 data	(output) Array to store 32-bit ARINC data.
pCEI_INT32 timeTags	(output) Array to store 32-bit time-tag data.

AR_GETBLOCK_T

Syntax

CEI_INT16 ar_getblock_t (CEI_UINT32 board, CEI_UINT32 channel, CEI_INT32 maxMessages, pCEI_INT32 actualCount, pCEI_UINT32 msgChan, pCEI_INT32 data, pCEI_INT32 timeTagMsw, pCEI_INT32 timeTagLsw)

Description

This routine retrieves the available ARINC 429 messages from the requested receive channel buffer and copies them to the desired destination. If the *msgChan*, *timeTagMsw*, and *timeTagLsw* parameters are not NULL, the receiver channel and 64-bit time-tag data associated with each retrieved message are also copied.

Return Value

ARS_GOTDATA	Message(s) and time-tag(s) were retrieved.
ARS_NODATA	No data was available.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVHARVAL	Channel is not available on device.
ARS_INVARG	Invalid <i>maxMessages</i> , <i>actualCount</i> , or <i>data</i> parameter was encountered.

Arguments

CEI_UINT32 board	(input) Device this routine is to access. Valid range is 0-15.
CEI_UINT32 channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
CEI_INT32 maxMessages	(input) The number of messages to retrieve.
pCEI_INT32 actualCount	(output) The number of messages retrieved.
pCEI_UINT32 msgChan	(output) Array to store the receiver channel indication, necessary for actual receive channel determination when using the Merged Receive Mode

pCEI_INT32 data	(output) Array to store 32-bit ARINC 429 data.
pCEI_INT32 timeTagMsw	(output) Array to store the most significant 32-bits of the 64-bit time-tag data.
pCEI_INT32 timeTagLsw	(output) Array to store the least significant 32-bits of the 64-bit time-tag data.

AR_GET_BOARDNAME

Syntax	pCEI_CHAR ar_get_boardname (CEI_INT16 board, pCEI_CHAR boardName)	
Description	This routine returns a character string describing the board name for the specified device. It should only be invoked after successful invocation of AR_LOADLSV.	
Return Value	NULL	An uninitialized board or invalid <i>board</i> value was provided.
	For any valid detected board, the return value is a character string description of board associated with the supplied <i>board</i> value:	
	“AMC-A30” “CEI-430” “CEI-430A” “CEI-530” “CEI-830” “R830RX” “RAR-CPCI” “RAR-EC”“RAR-PCIE”	
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.
	pCEI_CHAR boardName	(output) If a valid board is detected and this parameter is not NULL, the character description of that board is copied to the location referenced by this parameter. A minimum of 10 bytes of allocation is required for the destination array.

AR_GET_BOARDTYPE

Syntax	CEI_INT16 ar_get_boardtype (CEI_INT16 board)	
Description	This routine returns the API/device type for the specified device. It should only be invoked after successful invocation of AR_LOADLSV.	
Return Value	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	For any value less than ARS_INVBOARD, the return value indicates the type of board associated with the supplied <i>board</i> value:	
		CEI-830 (19)
		CEI-430 (21)
		AMC-A30 (22)
		CEI-530 (26)
		R830RX (27)
		RAR-CPCI (28)
		RAR-EC (29)
		RAR-PCIE (30)
		CEI-430A (31)
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.

AR_GET_CONFIG

Syntax

CEI_INT32 ar_get_config (CEI_INT16 board, CEI_INT16 item)

Description

This routine returns the active state of API information, board level settings, and limited ARINC 429 channel configuration register bit fields. It is provided for backward compatibility to CEI-x20 based applications. The routine AR_GET_DEVICE_CONFIG is the desired routine for acquiring information regarding channel and board-level configuration.

See the ARU_* definitions in the file CDEV_API.H for the most current list of parameter options supported by this routine and the values associated with those definitions.

Return Value

If the requested item is ARU_RX_CH nn _BIT_RATE (500-531), where nn is the receiver channel (01 - 32), this routine returns the current value of the channel configuration register baud rate field:

AR_HIGH	(0) high rate (100Kbs)
AR_LOW	(1) low rate (12.5Kbs)
Any other value is returned as a frequency value in Hertz.	

If the requested item is ARU_TX_CH nn _BIT_RATE (700-731), where nn is the transmitter channel (01 - 32), this routine returns the current value of the channel configuration register baud rate field:

AR_HIGH	(0) high rate (100Kbs)
AR_LOW	(1) low rate (12.5Kbs)
Any other value is returned as a frequency value in Hertz.	

If the requested item is ARU_RX_CH nn _PARITY (900-931), where nn is the receiver channel (01 - 32), this routine returns the current state of the specified receiver channel configuration register parity field:

AR_ODD	(0) receiver parity check enabled
AR_OFF	(8) receiver parity check disabled

If the requested item is ARU_TX_CH nn _PARITY (1100-1131), where nn is the transmitter channel (01 - 32), this routine returns the current state of the specified transmitter channel configuration register parity field:

AR_ODD	(0) odd transmitter parity
AR_EVEN	(1) even transmitter parity

If the requested item is ARU_TX_CH nn _SHUT_OFF (1700-1731), where nn is the transmitter channel (01 - 32), this routine returns the current state of the specified transmitter channel configuration register transmit disable field:

AR_ON	(7) external transmission is disabled
AR_OFF	(8) external transmission is enabled

If the requested item is ARU_TX_CH nn _HB_INJ (3300-3331), where nn is the transmitter channel (01 - 32), this routine returns the current state of the specified transmitter channel configuration register high-bit error injection field:

AR_ON (7) 33-bit transmission is enabled
 AR_OFF (8) standard 32-bit transmission is enabled

If the requested item is ARU_TX_CH nn _LB_INJ (3500-3531), where nn is the transmitter channel (01 - 32), this routine returns the current state of the specified transmitter channel configuration register low-bit error injection field:

AR_ON (7) 31-bit transmission is enabled
 AR_OFF (8) standard 32-bit transmission is enabled

If the requested item is ARU_TX_CH nn _GAP_INJ (3700-3731), where nn is the transmitter channel (01 - 32), this routine returns the current state of the specified transmitter channel configuration register message gap error injection field:

AR_ON (7) 3-bit message gap is used
 AR_OFF (8) standard 4-bit message gap is used

If the requested item is ARU_CONFIGURATION (21), this routine returns the value of the CEI-x30 Board Configuration, defined as follows:

CEIDEV_CONFIG_CEI830	(7)	CEI-830
CEIDEV_CONFIG_CEI430	(8)	CEI-430
CEIDEV_CONFIG_AMCA30	(9)	AMC-A30
CEIDEV_CONFIG_CEI530	(10)	CEI-530
CEIDEV_CONFIG_R830RX	(11)	R830RX
CEIDEV_CONFIG_RAR_CPCI	(12)	RAR-CPCI
CEIDEV_CONFIG_RAR_EC	(13)	RAR-EC
CEIDEV_CONFIG_RAR_PCIE	(14)	RAR-PCIE

If the requested item is ARU_RX_TIMETAG_MODE (440), this routine returns a value representing the currently selected timer/time-tag source and resolution. This value indicates the resolution of any timer-read or receive data time-tag value obtained via the API, and is defined as follows:

AR_TIMETAG_EXT_IRIG_64BIT	(0)
AR_TIMETAG_INT_USEC_64BIT	(1)
AR_TIMETAG_INT_20USEC_32BIT	(3)
AR_TIMETAG_INT_MSEC_32BIT	(4)
AR_TIMER_X20_COMPAT_32BIT	(5)

A value of AR_TIMETAG_EXT_IRIG_64BIT indicates the source is the external IRIG receiver, if connected; otherwise, if the IRIG signal is not internally wrapped this selection would be invalid. All other values represent various timer/time-tag LSB resolution values based on the internal CEI-x30 device timer.

If the requested item is ARU_ACCESS_SNAPSHOT_BUFFER (38), this routine returns the currently selected Snapshot Buffer storage mode:

ARU_LABEL_ONLY	(0) messages stored based on label
ARU_LABEL_WITH_SDI	(1) messages stored based on the combined label and SDI field values

If the requested item is ARU_IRIG_WRAP_ENABLE (441), this routine returns the current state of the IRIG Receiver internal wrap feature:

AR_ON	(7) IRIG Receiver is patched into the IRIG Generator
AR_OFF	(8) IRIG Receiver is configured for external IRIG source

If the requested item is ARU_IRIG_AVAILALBE (445), this routine returns TRUE if IRIG-B support is available, FALSE if it is not.

If the requested item is ARU_IRIG_CALIBRATED (447), this routine verifies the ability to capture consecutive IRIG time samples at a one second interval. If this results in a return status of FALSE (0), the IRIG signal is not consistent; otherwise, a return value of TRUE (1) indicates the signal is valid, or an error status (any value greater than 1) indicates a failure occurred.

If the requested item is not on this list or in the list of valid items for AR_GET_DEVICE_CONFIG, this routine will return a value of ARS_INVARG.

If the requested item is not valid for the specified device, this routine returns a value of ARS_INVHARCMD.

If the specified board is invalid or has not been initialized, this routine returns ARS_INVBOARD.

If access to the Board Lock timed-out or failed, this routine returns ARS_BOARD_MUTEX.

Arguments

CEI_INT16 board	(input) Device to access. Valid range is 0-15.
-----------------	--

CEI_INT16 item	(input) Control function about which to return information:
----------------	---

ARU_RX_CH01_BIT_RATE –	
ARU_RX_CH32_BIT_RATE	receiver 1 – 32 bit rate selection.

ARU_TX_CH01_BIT_RATE –	
ARU_TX_CH32_BIT_RATE	transmitter 1 – 32 bit rate selection.

ARU_RX_CH01_PARITY –	
ARU_RX_CH32_PARITY	receiver 1 – 32 parity state.

ARU_TX_CH01_PARITY –	
ARU_TX_CH32_PARITY	transmitter 1 – 32 parity state.

ARU_TX_CH01_SHUT_OFF – ARU_TX_CH32_SHUT_OFF	transmitter 1 – 32 enable state.
ARU_TX_CH01_LB_INJ – ARU_TX_CH32_LB_INJ	transmitter 1 – 32 low bit error enable state.
ARU_TX_CH01_HB_INJ – ARU_TX_CH32_HB_INJ	transmitter 1 – 32 high bit error enable state.
ARU_TX_CH01_GAP_INJ – ARU_TX_CH32_GAP_INJ	transmitter 1 – 32 message gap error enable state.
ARU_IRIG_AVAILABLE	IRIG Receiver installed state.
ARU_IRIG_WRAP_ENABLE	IRIG Receiver internal wrap state.
ARU_IRIG_CALIBRATED	IRIG signal validity.
ARU_ACCESS_SNAPSHOT_BUFFER	snapshot storage mode.
ARU_FW_VERSION	Hardware Version reg. value.
ARU_CONFIGURATION	configuration of the device.
ARU_RX_TIMETAG_MODE	active timer/time-tagging mode.

AR_GET_DATA

Syntax

CEI_INT16 ar_get_data (CEI_INT16 board, pCEI_INT16 channel,
pCEI_UINT32 data, pCEI_UINT32 timeTagLo, pCEI_UINT32
timeTagHi)

Description

This routine retrieves the next unread message and 64-bit time-tag from the specified receive channel. If it successfully returns data, there may or may not be more data in the buffer. It means only that there was at least one message in the buffer. Subsequent calls are required to determine if more data words are available in the buffer. If this routine returns a status value of ARS_NODATA, the buffer is empty.

If the specified channel was configured for merged mode operation along with other receive channels, this routine returns the next unread message from the merged receive buffer and indicate on which channel the message was received via the *channel* parameter.

Return Value

ARS_GOTDATA	A message and time-tag have been received.
ARS_NODATA	No data available.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided .
ARS_INVHARVAL	The channel is not available on the device.

Arguments

CEI_INT16 board	(input) Device this routine is to access. Valid range is 0-15.
pCEI_INT16 channel	(input/output) As an input, specifies which hardware receive channel this routine is to access, (see the description of the Receive Channel Select Register – 0x0040 for the list of valid hardware receive channel values) . As an output, indicates the receive channel number on which the data was received (for merged-mode channel reporting).
pCEI_UINT32 data	(output) Address that is to receive the data.

pCEI_UINT32 timeTagLo	(output) Address that is to receive the least-significant 32 bits of the 64-bit time-tag associated with the data, (resolution of the combined time-tag words is 1 μ sec).
pCEI_UINT32 timeTagHi	(output) Address that is to receive the most-significant 32 bits of the 64-bit time-tag associated with the data, (resolution of the combined time-tag words is 1 μ sec).

AR_GET_DATA_XT

Syntax	CEI_INT16 ar_get_data (CEI_INT16 board, pCEI_INT16 channel, pCEI_INT32 data, pAR_TIMETAG_TYPE timeTagRef)	
Description	<p>This routine retrieves the next unread message and the associated time-tag structure from the specified receive channel. If it successfully returns data, there may or may not be more data in the buffer. It means only that there was at least one message in the buffer. Subsequent calls are required to determine if more data words are available in the buffer. If this routine returns a status value of ARS_NODATA, the buffer is empty.</p> <p>If the specified channel was configured for merged mode operation along with other receive channels, this routine returns the next unread message from the merged receive buffer and indicate on which channel the message was received via the <i>channel</i> parameter.</p>	
Return Value	ARS_GOTDATA	A message and time-tag have been received.
	ARS_NODATA	No data available.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVHARVAL	The channel is not available on the device.
Arguments	CEI_INT16 board	(input) Device this routine is to access. Valid range is 0-15.
	pCEI_INT16 channel	(input/output) As an input, specifies which hardware receive channel this routine is to access, (see the description of the Hardware Channel Assignments for the list of valid hardware receive channel values) . As an output, indicates the receive channel number on which the data was received (for merged-mode channel reporting).
	pCEI_INT32 data	(output) Address that is to receive the data.
	pAR_TIMETAG_TYPE timeTagRef	(output)
		The address that is to receive the time-tag data structure associated with the data.

AR_GET_DEVICE_CONFIG

Syntax	CEI_INT16 ar_get_device_config (CEI_INT16 board, CEI_INT16 channel, CEI_INT16 item, pCEI_INT16 value)	
Description	This routine returns the state of the device configuration register attribute based on the combined item/value parameters. It is designed to support all ARINC 429 channel configuration register bit fields available to the device.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided .
	ARS_INVARG	The item argument value is not supported by this API routine.
	ARS_INVHARVAL	The item argument value is not supported by this device configuration.
	ARS_HW_CONSISTENCY	Indicates the board is compatible with the CEI-x30 Enhanced Operations (Version 2.00 API or later).
	ARS_INVALID	Indicates an invalid return value.
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.
	CEI_INT16 channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed channel count for the specified channel type. For board-level configuration items, this parameter is not used.
	CEI_INT16 item	(input) configuration register attribute for which to return the current state:
	ARU_RX_PARITY	receive channel parity enable.
	ARU_RX_BITRATE	receive channel bit rate.
	ARU_RX_FIFO_ENABLE	receive channel enable.
	ARU_RX_DISABLE	receive channel enable.
	ARU_RECV_MODE	receiver internal wrap.
	ARU_RX_MERGED_MODE	receiver merged mode enable.
	ARU_TX_BITRATE	transmit channel bit rate.

ARU_TX_PARITY	transmit channel parity select.
ARU_TX_FIFO_ENABLE	transmit channel enable.
ARU_TX_DISABLE	transmit channel transceiver disable.
ARU_TX_BIT_ERROR	transmit channel bit error enable.
ARU_TX_GAP_ERROR	transmit channel gap error enable.
ARU_FAST_SLEW_RATE	transmit channel slew rate select.
ARU_ACCESS_SNAPSHOT_BUFFER	snapshot storage mode.
ARU_IRIG_WRAP_ENABLE	IRIG receiver internal wrap state.
ARU_IRIG_AVAILALBE	IRIG receiver installed state.
ARU_IRIG_OUTPUT_ENABLE	R830RX IRIG Tx state.
ARU_IRIG_INPUT_TIME	IRIG received sample value.
ARU_IRIG_CALIBRATED	IRIG signal validity.
ARU_DEVICE_DISABLE	CEI-430 device disabled state.
ARU_DISCRETE_IN	discrete input state.
ARU_DIFFERENTIAL_IN	differential input state.
ARU_DIFFERENTIAL_OUT	differential output enable state.
ARU_RX_TIMETAG_MODE	active timer/time-tagging mode.
ARU_CHAN_COUNT_429	ARINC 429 Tx channel count.
ARU_CHAN_COUNT_573	ARINC 573/717 channel count.
ARU_CHAN_COUNT_DISC	discrete I/O channel count.
ARU_CHAN_COUNT_DIFF	differential I/O channel count.
ARU_RX_FIFO_COUNT	receive FIFO buffer fill count.
ARU_TX_FIFO_COUNT	transmit FIFO buffer fill count.
ARU_RX_MSG_COUNT	receive message count.
ARU_TX_MSG_COUNT	transmit message count.
ARU_FW_VERSION	current programmed firmware.
ARU_HW_ENHANCE_CHECK	current device BAR2 size.
ARU_HW_INTERRUPT_ENABLE	current PCI interrupt state

pCEI_INT16 value (output) state of the configuration register attribute:

If the requested item is ARU_RX_FIFO_ENABLE (16) , ARU_RX_DISABLE (9), or ARU_TX_FIFO_ENABLE (17), this routine will return the current value of the specified channel configuration register FIFO Enable field:

AR_ON	(7) FIFO operation enabled
AR_OFF	(8) FIFO operation disabled

If the requested item is ARU_RX_BITRATE (1) or ARU_TX_BITRATE (2), this routine will return the current value of the specified channel configuration register baud rate field:

ARU_SPEED_HIGH	(0) high rate (100Kbs)
ARU_SPEED_LOW	(1) low rate (12.5Kbs)

Any other value is translated as a non-standard bus speed value divisor for the 16MHz device clock reference. This value and the respective baud rate may be interpreted using the following formula:

$$\text{Baud Rate} = 16,000,000 / (\text{Value} + 2)$$

If the requested item is ARU_RX_PARITY (3), this routine returns the current state of the specified receive channel configuration register parity field:

AR_ON (7) receiver parity check enabled
 AR_OFF (8) receiver parity check disabled

If the requested item is ARU_TX_PARITY (4), this routine returns the current state of the specified transmitter channel configuration register parity field:

ARU_PARITY_ODD (0) odd transmitter parity
 ARU_PARITY_EVEN (1) even transmitter parity
 ARU_PARITY_NONE (2) transmitter parity disabled

If the requested item is ARU_RECV_MODE (5), this routine returns the current state of the specified receive channel configuration register Internal Wrap Enable field:

AR_WRAP_ON (0) internal wrap reception enabled
 AR_WRAP_OFF (1) internal wrap reception disabled

If the requested item is ARU_RX_MERGED_MODE (18), this routine returns the current state of the specified receive channel configuration register Merge Mode enable field:

AR_ON (7) Merged Mode enabled
 AR_OFF (8) Merged Mode disabled

If the requested item is ARU_TX_DISABLE (10), this routine returns the current state of the specified receive channel configuration register Transmit Disable field:

AR_ON (7) external transmission disabled
 AR_OFF (8) external transmission enabled

If the requested item is ARU_TX_BIT_ERROR (6), this routine returns the current state of the specified transmitter channel configuration register Bit Count Hi and Low fields:

AR_LO (0) Bit Count Low enabled
 AR_HI (1) Bit Count High enabled
 AR_OFF (8) both Bit Count Low and High disabled

If the requested item is ARU_TX_GAP_ERROR (8), this routine returns the current state of the specified transmitter channel configuration register Gap Error field:

AR_ON (7) Gap Error enabled
 AR_OFF (8) Gap Error disabled

If the requested item is ARU_FAST_SLEW_RATE (323), this routine returns the current state of the specified transmitter channel configuration register Slew Rate field:

AR_ON (7) Fast Slew Rate selected (1.5 μ sec rise time)

AR_OFF (8) Slow Slew Rate selected (10 μ sec rise time)

If the requested item is ARU_ACCESS_SNAPSHOT_BUFFER (38), this routine returns the current state of the device snapshot storage mode:

ARU_LABEL_ONLY (0) message storage on a label basis
 ARU_LABEL_WITH_SDI (1) message storage on a combined label/SDI basis.

If the requested item is ARU_IRIG_AVAILALBE (445), this routine returns TRUE if IRIG-B support is available, FALSE if it is not.

If the requested item is ARU_IRIG_OUTPUT_ENABLE (26), this routine returns the enable state of the R830RX IRIG Generator Enable:

AR_ON (7) IRIG output is enabled
 AR_OFF (8) IRIG output is disabled

If the requested item is ARU_IRIG_INPUT_TIME (27), this routine returns the most recent received IRIG received sample value.

If the requested item is ARU_IRIG_WRAP_ENABLE (441), this routine returns the current state of the IRIG Receiver internal wrap feature:

AR_ON (7) IRIG Receiver is patched into the IRIG Generator
 AR_OFF (8) IRIG Receiver is configured for external IRIG source

If the requested item is ARU_IRIG_CALIBRATED (447), this routine verifies the ability to capture consecutive IRIG time samples at a one second interval. If this results in a return status of FALSE (0), the IRIG signal is not consistent; otherwise, a return value of TRUE (1) indicates the signal is valid, or an error status (any value greater than 1) indicates a failure occurred.

If the requested item is ARU_DISCRETE_IN (14), this routine returns the current state of the specified discrete I/O channel:

AR_HI (1) the discrete is High
 AR_LO (0) the discrete is Low

If the requested item is ARU_DIFFERENTIAL_IN (22), this routine returns the current state of the specified differential I/O channel:

AR_HI (1) the differential input is High
 AR_LO (0) the differential input is Low

If the requested item is ARU_DIFFERENTIAL_OUT (23), this routine returns the enable state of the specified differential I/O channel:

AR_ON (7) the differential output enabled
 AR_OFF (8) the differential output disabled

If the requested item is ARU_RX_TIMETAG_MODE (440), this routine returns a value representing the currently selected timer/time-tag source and resolution. This value indicates the resolution of any timer-read or receive data time-tag value obtained via the API, and is defined as follows:

AR_TIMETAG_EXT_IRIG_64BIT	(0)
AR_TIMETAG_INT_USEC_64BIT	(1)
AR_TIMETAG_INT_20USEC_32BIT	(3)
AR_TIMETAG_INT_MSEC_32BIT	(4)
AR_TIMER_X20_COMPAT_32BIT	(5)

A value of AR_TIMETAG_EXT_IRIG_64BIT indicates the source is the external IRIG receiver, if connected; otherwise, if the IRIG signal is not internally wrapped this selection would be invalid. All other values represent various timer/time-tag LSB resolution values based on the internal CEI-x30 device timer.

If the requested item is ARU_CONFIGURATION (21), this routine returns the value of the CEI-x30 Board Configuration, defined as follows:

CEIDEV_CONFIG_CEI830	(7)	CEI-830
CEIDEV_CONFIG_CEI430	(8)	CEI-430
CEIDEV_CONFIG_AMCA30	(9)	AMC-A30
CEIDEV_CONFIG_CEI530	(10)	CEI-530
CEIDEV_CONFIG_R830RX	(11)	R830RX
CEIDEV_CONFIG_RAR_CPCI	(12)	RAR-CPCI
CEIDEV_CONFIG_RAR_EC	(13)	RAR-EC
CEIDEV_CONFIG_RAR_PCIE	(14)	RAR-PCIE
CEIDEV_CONFIG_CEI430A	(15)	CEI-430A

If the requested item is ARU_DEVICE_DISABLE (39), this routine returns the current value of the CEI-430 Global Enable Register – Device Disabled bit.

If the requested item is ARU_CHAN_COUNT_429 (448), this routine returns the ARINC 429 transmit channel count detected on the board.

If the requested item is ARU_CHAN_COUNT_573 (449), this routine returns the ARINC 573/717 transmit channel count detected on the board.

If the requested item is ARU_CHAN_COUNT_DISC (450), this routine returns the discrete output channel count detected on the board.

If the requested item is ARU_CHAN_COUNT_DIFF (451), this routine returns the differential output channel count detected on the board.

If the requested item is ARU_TX_FIFO_COUNT (19), this routine returns the current buffer count of messages in the specified ARINC 429 transmit FIFO awaiting transmission.

If the requested item is ARU_RX_FIFO_COUNT (28), this routine returns the current buffer count of messages in the ARINC 429 receive FIFO available to be read by the host application.

If the requested item is ARU_RX_MSG_COUNT (35), this routine returns the number of messages received on this channel since the board was last initialized.

If the requested item is ARU_TX_MSG_COUNT (36), this routine returns the number of messages transmitted on this channel since the board was last initialized.

If the requested item is ARU_FW_VERSION (20), this routine returns the current programmed firmware.

If the requested item is ARU_HW_ENHANCE_CHECK (30), this routine returns either ARS_NORMAL to indicate the board is compatible with the CEI-x30 Enhanced Operations (Version 2.00 API), or ARS_HW_CONSISTENCY to indicate it is not.

If the requested item is ARU_HW_INTERRUPT_ENABLE (29), this routine returns the current state of the PCI Interrupt Enable bit:

AR_ON	(7) PCI Interrupts are enabled
AR_OFF	(8) PCI Interrupts are disabled

AR_GET_573_CONFIG

Syntax	CEI_INT16 ar_get_573_config (CEI_INT16 board, CEI_INT16 item, pCEI_INT32 value)	
Description	This routine returns the state of the device configuration register attribute based on the combined item/value. It is designed to support the ARINC 573/717 configuration register attributes available to the device.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided .
	ARS_INVARG	The item argument value is not supported by this API routine.
	ARS_INVHARVAL	The item argument value is not supported by this device configuration.
Arguments	CEI_INT16 board	(input) Device this routine is to access. Valid range is 0-15.
	CEI_INT16 item	(input) Configuration item about which to return information:
	ARU_RECV_MODE	receiver internal wrap.
	ARU_RX_BITRATE	receive channel bit rate.
	ARU_RX_FIFO_ENABLE	receive channel enable.
	ARU_RX_MERGED_MODE	receiver merged mode enable
	ARU_573_RX_AUTO_DETECT	receiver frame auto-detect enable.
	ARU_573_RX_BPRZ_SELECT	receiver BPRZ/HBP selection.
	ARU_TX_BITRATE	transmit channel bit rate.
	ARU_TX_FIFO_ENABLE	transmit channel enable.
	ARU_573_TX_BPRZ_SELECT	transmitter BPRZ encoder enable.
	ARU_573_TX_HBP_SELECT	transmitter HBP encoder enable.
	ARU_573_TX_SLEW_RATE	transmitter slew rate select.
	ARU_TX_FIFO_COUNT	transmit FIFO buffer fill count.
	ARU_573_SYNC_WORD1	receiver auto-detect sync word 1.
	ARU_573_SYNC_WORD2	receiver auto-detect sync word 2.
	ARU_573_SYNC_WORD3	receiver auto-detect sync word 3.
	ARU_573_SYNC_WORD4	receiver auto-detect sync word 4.
	pCEI_INT32 value	(output) The address that receives the state of the item requested:

If the requested item is ARU_RECV_MODE (5), this routine returns the current state of the ARINC 573/717 receive channel Internal Wrap Enable:

AR_WRAP_ON	(0) internal wrap reception enabled
AR_WRAP_OFF	(1) internal wrap reception disabled

If the requested item is ARU_RX_FIFO_ENABLE (16) or ARU_TX_FIFO_ENABLE (17), then this routine will return the current value of the ARINC 573/717 channel FIFO Enable:

AR_ON	(7) FIFO operation enabled
AR_OFF	(8) FIFO operation disabled

If the requested item is ARU_RX_MERGED_MODE (18), this routine returns the current state of the ARINC 573/717 receive channel Merge Mode Enable:

AR_ON	(7) Merge mode enabled
AR_OFF	(8) Merge mode disabled

If the requested item is ARU_573_RX_AUTO_DETECT (301), this routine returns the current state of the ARINC 573/717 receive channel Auto-synchronization Enable:

AR_ON	(7) ARINC 573/717 frame auto-detection enabled
AR_OFF	(8) ARINC 573/717 frame auto-detection disabled

If the requested item is ARU_573_RX_BPRZ_SELECT (302), this routine returns the current state of the ARINC 573/717 receive channel Encoding Enable:

AR_ON	(7) ARINC 573/717 BPRZ encoding enabled
AR_OFF	(8) ARINC 573/717 HBP encoding enabled

If the requested item is ARU_RX_BITRATE (1) or ARU_TX_BITRATE (2), this routine returns the current state of the ARINC 573/717 channel Baud Rate/Subframe Size selection (ranging from 0 to 7):

ARU_573_RATE_SIZE_384_32	384 bps, 32 word sub-frame
ARU_573_RATE_SIZE_768_64	768 bps, 64 word sub-frame
ARU_573_RATE_SIZE_1536_128	1536 bps, 128 word sub-frame
ARU_573_RATE_SIZE_3072_256	3072 bps, 256 word sub-frame
ARU_573_RATE_SIZE_6144_512	6144 bps, 512 word sub-frame
ARU_573_RATE_SIZE_12288_1024	12288 bps, 1024 word sub-frame
ARU_573_RATE_SIZE_24576_2048	24576 bps, 2048 word sub-frame
ARU_573_RATE_SIZE_49152_4096	49152 bps, 4096 word sub-frame

If the requested item is ARU_573_TX_BPRZ_SELECT (313), this routine returns the current state of the ARINC 573/717 transmit channel Encoding Enable:

AR_ON	(7) ARINC 573/717 BPRZ encoding enabled
-------	---

AR_OFF (8) ARINC 573/717 BPRZ encoding disabled

If the requested item is ARU_573_TX_HBP_SELECT (314), this routine returns the current state of the ARINC 573/717 transmit channel Encoding Enable:

AR_ON (7) ARINC 573/717 HBP encoding enabled

AR_OFF (8) ARINC 573/717 HBP encoding disabled

If the requested item is ARU_573_TX_SLEW_RATE (305) this routine returns the current state of the ARINC 573/717 transmit channel Slew Rate selection:

ARU_573_TX_SLEW_1PT5 (1) 1.5 μ sec rise time

ARU_573_TX_SLEW_10PT0 (0) 10.0 μ sec rise time

If the requested item is ARU_TX_FIFO_COUNT (19), this routine returns the current buffer count of messages in the ARINC 717 transmit FIFO awaiting transmission.

If the requested item is ARU_573_SYNC_WORD1 (307), ARU_573_SYNC_WORD2 (308), ARU_573_SYNC_WORD3 (309), or ARU_573_SYNC_WORD4 (310), this routine returns the 12-bit value for the respective receiver sub-frame sync word.

AR_GET_ERROR

Syntax	pCEI_CHAR ar_get_error (CEI_INT16 status)	
Description	<p>Most of the API routines return status values, a majority of which indicate an error condition. When supplied with such an error value, this routine returns a pointer to a message string describing the error.</p> <p>Review the section, “Return Status Values”, for the current list of possible error codes and their explanations.</p>	
Return Value	A pointer to the error message character string.	
Arguments	CEI_INT16 status	(input) a status value returned by any of the API utilities.

AR_GETFILTER

Syntax

CEI_UINT32 ar_getfilter (CEI_UINT32 board, CEI_UINT32 channel, pCEI_CHAR filterTable)

Description

This routine returns the contents of a single channel label filter table from the device. Each receive channel has a separate section within the label filter table, is used by the firmware to control FIFO storage of received labels and generate hardware interrupts. Each element of the filter buffer consists of a bit field defined for compatibility with the CEI-x20 product line 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 on reception insert the respective receive channel tag (ranging from 64-95) in the interrupt queue and if enabled generate a PCI interrupt.

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

filterTable[MAX_ESSM][MAX_SDI][MAX_LABEL]

and accessed as:

filterTable[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

To write an entry to the label interrupt and filter table, refer to the API routines AR_ENH_LABEL_FILTER, AR_PUTFILTER, and AR_LABEL_FILTER.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided .
ARS_INVARG	Invalid <i>channel</i> or <i>filterTable</i> parameter.

Arguments

CEI_UINT32 board	(input) Device to access. Valid range is 0-15.
CEI_UINT32 channel	(input) Specifies which receive channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
pCEI_CHAR filterTable	(input) Array to receive the contents of the specified channel's label filter table. This array must have a minimum allocation of 8Kbytes.

AR_GET_LABEL_FILTER

Syntax	CEI_INT16 ar_get_label_filter (CEI_INT16 board, CEI_UINT16 label)	
Description	This routine returns the active state of label filtering for the specified label on each of the first sixteen installed receive channels.	
Return Value	<p>Given the routine is supplied with a valid board and label value, the return value indicates the active state of the specified label on each receive channel through the respective bit state, where the label filter state on receive channel zero (zero-referenced) is indicated via b0 as “1” to indicate the label is filtered and “0” to indicate either the label is not filtered or the receive channel is not installed. Subsequent bits in the value indicate the label filter state for the respective receive channel.</p> <p>A label is indicated to be “filtered” if the respective entry in the Label Filter Table is defined to filter the label from either the Sequential (FIFO) or Snapshot buffers.</p>	
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.
	CEI_UINT16 label	(input) Specifies which label to query. Valid range is 0 to 255.

AR_GET_LATEST

Syntax

CEI_VOID ar_get_latest (CEI_INT16 board, CEI_INT16 channel,
CEI_UINT16 label, pCEI_VOID data, pCEI_CHAR seq_num)

Description

In support of backward compatibility to previous ARINC product APIs, this routine returns the latest ARINC 429 message received for the specified channel/label combination from the snapshot buffer.

If the *label* parameter value requested is either 256 or the value ARU_ALL_LABELS (511), this routine treats the *data* parameter as an array reference and returns the most recent received ARINC message for all 256 valid ARINC labels for the specified channel, in successive *data* array elements. This function assumes that the caller has allocated at least 1024 bytes for *data* when used in this mode.

When using this routine, the host application should set the snapshot storage mode to **label field only**, (see the documentation on the routine AR_SET_DEVICE_CONFIG, for the configuration option ARU_ACCESS_SNAPSHOT_BUFFER). This sets up the x30 device to store snapshot data based on the label field value only, ignoring the SDI bit field value.

If no message has been received for the specified channel/label since the last initialization of the device, a data value of zero will be returned.

Arguments

CEI_INT16 board	(input) Device this routine is to access. Valid range is 0-15.
CEI_INT16 channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
CEI_UINT16 label	(input) The label value of interest.
pCEI_VOID data	(output) Location to store 32-bit ARINC data.
pCEI_CHAR seq_num	(output) Unsupported legacy parameter.

AR_GET_LATEST_T

Syntax	CEI_INT32 ar_get_latest_t (CEI_INT16 board, CEI_INT16 channel, CEI_UINT16 label, pCEI_UINT32 data, TIME_TAG_TYPE * timeTag)	
Description	<p>This routine returns the latest ARINC 429 message and time-stamp received for the specified channel/label combination from the snapshot buffer.</p> <p>When using this routine, the host application should set the snapshot storage mode to label field only, (see the documentation on the routine AR_SET_DEVICE_CONFIG, for the configuration option ARU_ACCESS_SNAPSHOT_BUFFER). This sets up the x30 device to store the ARINC message and time-stamp in the snapshot buffer based on the label field value only, ignoring the SDI bit field value.</p> <p>If no message has been received for the specified channel/label since the last initialization of the device, a data value of zero is returned for the message and time-stamp. If the timeTag parameter is NULL, no time-stamp information is returned.</p>	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	Invalid <i>label</i> or null <i>data</i> parameter.
	ARS_INVHARVAL	Channel is not available on device.
Arguments	CEI_INT16 board	(input) Device this routine is to access. Valid range is 0-15.
	CEI_INT16 channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
	CEI_UINT16 label	(input) The label value of interest.
	pCEI_UINT32 data	(output) Location to store ARINC message.
	TIME_TAG_TYPE * timeTag	(output) The address that is to receive the 64-bit message time-stamp, the format of which is determined by the current API time-tag format.

AR_GETNEXT

Syntax

CEI_INT16 ar_getnext (CEI_INT16 board, CEI_INT16 channel, pCEI_VOID destination)

Description

This routine retrieves the next unread message from the specified receive channel. If no message is present in the receiver FIFO buffer upon invocation, this routine polls the buffer waiting for the presence of a received message for up to one-half second. If no message is present after one-half second, a time-out status is returned.

Return Value

ARS_GOTDATA	A message has been retrieved.
ARS_CHAN_TIMEOUT	No message was available or received.
ARS_BAD_MESSAGE	An invalid length ARINC 429 message was received on the specified channel.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided .
ARS_INVARG	Invalid <i>destination</i> parameter.
ARS_INVHARVAL	Channel is not available on device.

Arguments

CEI_INT16 board	(input) Device this routine is to access. Valid range is 0-15.
CEI_INT16 channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
pCEI_VOID destination	(output) The address that is to receive the message.

AR_GETNEXTT

Syntax

CEI_INT16 ar_getnextt (CEI_INT16 board, CEI_INT16 channel, pCEI_VOID destination, pCEI_VOID timetag)

Description

This routine retrieves the next unread message and scaled 32-bit time-stamp from the specified receive channel. If no message is present in the receiver FIFO buffer when invoked this routine polls the buffer waiting for the presence of a received message for up to one-half second. If no message is present after one-half second, a time-out status is returned.

If the *timetag* parameter is not NULL, the 32-bit translation of the 64-bit message time-stamp will be returned, scaled to the active legacy 32-bit time-tag mode, (1 millisecond resolution by default).

Return Value

ARS_GOTDATA	A message has been retrieved.
ARS_CHAN_TIMEOUT	No message was available or received.
ARS_BAD_MESSAGE	An invalid length ARINC 429 message was received.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVARG	Invalid <i>destination</i> parameter.
ARS_INVHARVAL	Channel is not available on device.

Arguments

CEI_INT16 board	(input) Device this routine is to access. Valid range is 0-15.
CEI_INT16 channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
pCEI_VOID destination	(output) The address that is to receive the message.
pCEI_VOID timetag	(output) The address that is to receive the 32-bit time-tag associated with the data, (resolution is programmable). If the merged receive mode is active for the specified channel, the upper five bits of the 32-bit time-tag word contain the receive channel number on which the data was received.

AR_GETNEXT_XT

Syntax

CEI_INT16 ar_getnext_xt (CEI_INT16 board, CEI_INT16 channel, pCEI_UINT32 data, pAR_TIMETAG_TYPE timeTagRef)

Description

This routine retrieves the next unread message and the associated time-tag structure from the specified receive channel. If no message is present in the receiver FIFO buffer when invoked, this routine polls the buffer waiting for the presence of a received message for up to one-half second. If no message is present after one-half second, a time-out status is returned.

If the *timeTagRef* parameter is not NULL, the time-tag structure containing the message time-stamp will be returned.

Return Value

ARS_GOTDATA	A message has been retrieved.
ARS_CHAN_TIMEOUT	No message was available or received.
ARS_BAD_MESSAGE	An invalid length ARINC 429 message was received.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVARG	Invalid <i>data</i> parameter.
ARS_INVHARVAL	Channel is not available on device.

Arguments

CEI_INT16 board	(input) Device this routine is to access. Valid range is 0-15.
CEI_INT16 channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
pCEI_UINT32 data	(output) The address that is to receive the message.
pAR_TIMETAG_TYPE timeTagRef	(output) The address that is to receive the time-tag data structure associated with the message.

AR_GET_RX_COUNT

Syntax	CEI_UINT32 ar_get_rx_count (CEI_INT16 board, CEI_INT16 channel)	
Description	<p>The device maintains a count of the number of ARINC data messages received over the interface for each channel since the device was last initialized (see AR_LOADSLV). This routine returns that number.</p> <p>If the API routine AR_CLR_RX_COUNT has been invoked by the host application prior to this routine's invocation, the API logs the current value of the message count and returns the difference between that value and the value read from the device upon invocation.</p>	
Return Value	Current count of ARINC messages received on the specified channel.	
Arguments	CEI_INT16 board	(input) Device this routine is to access. Valid range is 0-15.
	CEI_INT16 channel	(input) Specifies which receive channel this routine is to access.

AR_GET_SNAP_DATA

Syntax	CEI_INT32 ar_get_snap_data (CEI_INT16 board, CEI_INT16 channel, CEI_UINT16 label, CEI_UINT16 sdi, pCEI_UINT32 data)	
Description	<p>This routine returns the latest ARINC 429 message received for the specified channel/label combination from the snapshot buffer.</p> <p>When using this routine, the host application should set the snapshot storage mode to label/sdi storage, (see the documentation on the routine AR_SET_DEVICE_CONFIG, for the configuration option ARU_ACCESS_SNAPSHOT_BUFFER). This sets up the x30 device to store snapshot data based on the label field value in combination with the SDI bit field value.</p> <p>If no message has been received for the specified channel/label since the last initialization of the device, a data value of zero is returned for the message.</p>	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	Invalid <i>label</i> , <i>sdi</i> , or null <i>data</i> parameter.
	ARS_INVHARVAL	Channel is not available on device.
Arguments	CEI_INT16 board	(input) Device this routine is to access. Valid range is 0-15.
	CEI_INT16 channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
	CEI_UINT16 label	(input) The label value of interest.
	CEI_UINT16 sdi	(input) The SDI value of interest.
	pCEI_UINT32 data	(output) Location to store 32-bit ARINC data.

AR_GET_STATUS

Syntax	CEI_UINT32 ar_get_status (CEI_INT16 board, pCEI_UINT16 state)	
Description	This routine returns the state of the FIFO Data Available bit for up to 16 receivers in a bitwise 16-bit value.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided .
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.
	pCEI_UINT16 state	(output) Location to store the receiver FIFO status. The Status Register Bit Assignments are defined as follows, ("1" indicates Data Available, "0" indicates No Data Available): b0 - ARINC 429 Receiver 1 b1 - ARINC 429 Receiver 2 ... b14 - ARINC 429 Receiver 15 b15 - ARINC 429 Receiver 16

AR_GET_STORAGE_MODE

Syntax CEI_INT16 ar_get_storage_mode (CEI_INT16 board, pCEI_INT16 mode)

Description This routine is designed to provide compatibility with the CEI-x20 ARINC device API. It returns the current state of the API receive storage mode. When the API receive storage mode is *buffered*, each receiver is assigned an independent circular buffer for data storage (merged mode is disabled). When the storage mode is *merged*, all receivers are set to enable merged receive mode and data received on each is stored in the merged FIFO buffer. Each receive data API routine processes the active storage mode internally, acquiring data from the appropriate buffer. Since each receive channel can be independently programmed to store data in *buffered* or *merged* mode through AR_SET_DEVICE_CONFIG, this routine should only be used in conjunction with the AR_SET_STORAGE_MODE routine.

Return Value ARS_NORMAL Routine execution was successful.

ARS_INVBOARD An uninitialized board or invalid *board* value was provided.

Arguments CEI_INT16 board (input) Device to access. Valid range is 0-15.

pCEI_INT16 mode (output) The address that is to receive the state of the current API storage mode. Valid return values for this parameter are:

ARU_BUFFERED (0) buffered receive mode

ARU_MERGED (2) merged receive mode

AR_GET_TIME

Syntax

CEI_INT16 ar_get_time (CEI_INT16 board, CEI_INT16 format,
pAR_TIMETAG_TYPE timeTag)

Description

This routine returns the current time reference value scaled from either the CEI-x30 device internal 64-bit timer or the most recently received IRIG timer reference, as specified via the *format* parameter.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVARG	An invalid format parameter value was provided.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.

Arguments

CEI_INT16 board (input) Device to access. Valid range is 0-15.

CEI_INT16 format (input) Time format requested. Valid options are:

AR_TIMETAG_EXT_IRIG_64BIT	0
AR_TIMETAG_INT_USEC_64BIT	1
AR_TIMETAG_HOST_USEC_64BIT	2
AR_TIMETAG_INT_20USEC_32BIT	3
AR_TIMETAG_INT_MSEC_32BIT	4
AR_TIMER_X20_COMPAT_32BIT	6

pAR_TIMETAG_TYPE timeTag

(output) Current device timer or translated IRIG sample value. The *timeTag.timeTag* structure member will be defined as follows based on the supplied *format* parameter value:

AR_TIMETAG_EXT_IRIG_64BIT - 64-bit IRIG sample time in microseconds since beginning of current year. The returned *timeTag.R.referenceTimeTag* structure member will contain the board internal timer-referenced time-stamp assigned when the last bit of the IRIG sample was processed by the CEI-x30 IRIG receiver.

AR_TIMETAG_INT_USEC_64BIT - 64-bit internal board timer in microseconds.

AR_TIMETAG_HOST_USEC_64BIT - 64-bit host operating system time scaled to have a 1 microsecond resolution.

AR_TIMETAG_INT_20USEC_32BIT - 32-bit internal board timer in microseconds.

AR_TIMETAG_INT_MSEC_32BIT - 32-bit internal board timer scaled to have a 20 microsecond resolution.

AR_TIMER_X20_COMPAT_32BIT - 32-bit internal board timer scaled to have a 1 millisecond resolution.

AR_GET_TIMERCNTL

Syntax

CEI_UINT32 ar_get_timerctl (CEI_INT16 board)

Description

This routine is provided for legacy support of the CEI-x20 ARINC API, returning the current 32-bit, 1 millisecond resolution time reference value based on the current application-specified timer mode, (specified through AR_SET_CONFIG using the attribute ARU_RX_TIMETAG_MODE). If the current timer mode is assigned to any 64-bit timer, the least-significant 32-bits of the internal device timer will be returned (this applies to IRIG, host, or internal timer). If the current timer mode is assigned to either of the 32-bit CEI-x20 API compatibility or 20 microsecond (IP-AVIONICS) resolution modes, the respective 32-bit adjusted timer value will be returned.

Return Value

The 32-bit timer value.

Arguments

CEI_INT16 board

(input) Device to access. Valid range is 0-15.

AR_GETWORD

Syntax

CEI_INT16 ar_getword (CEI_INT16 board, CEI_INT16 channel, pCEI_VOID destination)

Description

This routine retrieves the next unread message from the specified receive channel. If it successfully returns data message, there may or may not be more messages in the buffer. It only means there was at least one message in the buffer. Subsequent calls would be required to determine if more messages are available in the buffer. If this routine returns a status value of ARS_NODATA, the buffer is empty.

The channel value passed to this routine corresponds to the ARINC 429 receive channel index, starting with zero. If that value exceeds the 429 receive channel count and an ARINC 573/717 receiver exists, it is used as the designated receive channel buffer.

Return Value

ARS_GOTDATA	A message has been retrieved.
ARS_NODATA	No data available.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided .
ARS_INVARG	A null data parameter value was provided.
ARS_INVHARVAL	Channel is not available on device.
ARS_BAD_MESSAGE	An invalid length ARINC 429 message was received on the specified channel.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.

Arguments

CEI_INT16 board	(input) Device this routine is to access. Valid range is 0-15.
CEI_INT16 channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
pCEI_VOID destination	(output) The address that is to receive the message. The format of the 32-bit value is dependent on the protocol assigned to the respective receive channel. See Chapter 17, “CEI-x30 Hardware Interface” for more details on the receive buffer message formats.

AR_GETWORDT

Syntax

CEI_INT16 ar_getwordt (CEI_INT16 board, CEI_INT16 channel,
pCEI_VOID destination, pCEI_VOID timetag)

Description

This routine retrieves the next unread message from the specified receive channel. If it successfully returns data message, there may or may not be more messages data in the buffer. It means only that there was at least one message in the buffer. Subsequent calls are required to determine if more messages are available in the buffer. If this routine returns a status value of ARS_NODATA, the buffer is empty.

Return Value

ARS_GOTDATA	A message has been retrieved.
ARS_NODATA	No data available.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVARG	A null <i>data</i> parameter value was provided.
ARS_INVHARVAL	Channel is not available on device.
ARS_BAD_MESSAGE	An invalid length ARINC 429 message was received on the specified channel.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.

Arguments

CEI_INT16 board	(input) Device this routine is to access. Valid range is 0-15.
CEI_INT16 channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
pCEI_VOID destination	(output) The address that is to receive the message. The format of the 32-bit value is dependent on the protocol assigned to the respective receive channel. See Chapter 17, “CEI-x30 Hardware Interface” for more details on the receive buffer message formats.

pCEI_VOID timetag	(output) The address that is to receive the 32-bit time-tag associated with the data, (resolution is programmable). If the merged receive mode is active for the specified channel, the upper five bits of the 32-bit time-tag word contain the receive channel number on which the data was received.
-------------------	--

AR_GETWORD_XT

Syntax	CEI_INT16 ar_getword_xt (CEI_INT16 board, CEI_INT16 channel, pCEI_VOID data, pAR_TIMETAG_TYPE timeTagRef)	
Description	This routine retrieves the next unread message and the associated time-tag structure from the specified receive channel. If it successfully returns a message, there may or may not be more messages in the buffer. It means only that there was at least one message in the buffer. Subsequent calls are required to determine if more messages are available in the buffer. If this routine returns a status value of ARS_NODATA, the buffer is empty.	
Return Value	ARS_GOTDATA	A message has been retrieved.
	ARS_NODATA	No data available.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	A null <i>data</i> parameter value was provided.
	ARS_INVHARVAL	Channel is not available on device.
	ARS_BAD_MESSAGE	An invalid length ARINC 429 message was received on the specified channel.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
Arguments	CEI_INT16 board	(input) Device this routine is to access. Valid range is 0-15.
	CEI_INT16 channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
	pCEI_VOID data	(output) The address that is to receive the 32-bit message.
	pAR_TIMETAG_TYPE timeTagRef	(output) The address that is to receive the time-tag data structure associated with the message.

AR_GO

Syntax

CEI_INT16 ar_go (CEI_INT16 board)

Description

This routine assigns the global enable register Global Enable bit to be *enabled* for the specified device. All message processing on the device is activated when this routine executes.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.

Arguments

CEI_INT16 board	(input) Device to access. Valid range is 0-15.
-----------------	--

AR_HW_INTERRUPT_BUFFER_READ

Syntax	CEI_UINT32 ar_hw_interrupt_buffer_read (CEI_INT16 board, pCEI_UINT32 numberOfWords, pCEI_UINT32 data)	
Description	<p>This routine provides read access to the local API copy of the CEI-x30 device interrupt queue. The local API copy is filled by hardware interrupt processing within the default API ISR. If the host application replaces the default API ISR with a custom ISR, this routine is not usable.</p> <p>Each time this routine is invoked, the specified number of queue entries is read from the buffer region starting at the location last referenced by the API in a previous invocation and ending at the location written by the most recent execution of the default API interrupt service routine.</p>	
Return Value	ARS_GOTDATA	Routine execution was successful and one or more interrupt buffer entries were returned.
	ARS_NODATA	No unread interrupt buffer entries were available or returned.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	A NULL <i>data</i> buffer pointer was supplied.
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.
	pCEI_UINT32 numberOfWords	(input/output) As an input, this argument specified the number of interrupt buffer entries to read and return. As an output, this argument indicates the number of interrupt buffer entries actually read, if there were fewer unread entries available than what was requested.
	pCEI_UINT32 data	(output) The location to store the interrupt buffer entries read.

AR_INTERRUPT_QUEUE_READ

Syntax	CEI_UINT32 ar_interrupt_queue_read (CEI_INT16 board, pCEI_UINT32 numberOfWords, pCEI_UINT32 data)	
Description	This routine provides read access directly to the CEI-x30 device hardware interrupt queue. Each time this routine is invoked, the specified number of queue entries will be read from the interrupt queue starting at the location referenced by last invocation of this routine, and ending at the location indicated by the device interrupt queue pointer.	
Return Value	ARS_GOTDATA	Routine execution was successful and one or more interrupt queue entries were returned.
	ARS_NODATA	No unread interrupt queue entries were available or returned.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	A NULL <i>data</i> buffer pointer was supplied.
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.
	pCEI_UINT32 numberOfWords	(input/output) As an input, this argument specified the number of interrupt buffer entries to read and return. As an output, this argument indicates the number of interrupt buffer entries actually read, if there were fewer unread entries available in the interrupt queue than what was requested.
	pCEI_UINT32 data	(output) The location to store the interrupt queue entries read.

AR_INITIALIZE_API

Syntax	CEI_INT16 ar_initialize_api (CEI_INT16 board)	
Description	This routine acquires the resources for the device and initializes API local variables. With the exception of the RAR-PCIE, it also downloads the CEI-x30 firmware program, resetting the device to an initial power-up state.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_WINRTFAIL	The device driver failed to open a session with the device, either because the device is not properly installed in the host system or a resource conflict is inhibiting device driver initialization.
	ARS_BADLOAD	The device driver session was opened successfully but the device firmware download failed.
	ARS_HW_DETECT	The device driver session was opened but the detected device is not recognized as a CEI-x30 product.
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.

AR_INITIALIZE_DEVICE

Syntax

CEI_INT16 ar_initialize_device (CEI_INT16 board)

Description

This routine performs a non-destructive SRAM memory test, flushes the receiver FIFO buffers, and assigns the default state of all channel configuration registers. It also initializes the label filtering and message scheduling features. The default state of the CEI-x30 board is defined as follows:

- ARINC 429 Transmitter FIFOs enabled, speed set for 100Kbps and ODD parity enabled.
- ARINC 429 Receiver FIFOs enabled, speed set for 100Kbps, ODD parity, Merged Mode disabled, and Internal Wrap disabled.
- ARINC 717 Transmitter and Receiver FIFOs disabled, set for BPRZ encoding at 768BPS (64-word subframe), auto-detect enabled, and Internal Wrap disabled.
- Message Scheduler enabled, no messages defined.
- All receive label filtering disabled

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_MEMWRERR	Device SRAM test write/read/verify failure.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.

Arguments

CEI_INT16 board	(input) Device to access. Valid range is 0-15.
-----------------	--

AR_HW_INTERRUPT_BUFFER_READ

Syntax	CEI_INT32 ar_hw_interrupt_buffer_read (CEI_INT16 board, pCEI_UINT32 numberOfWords, pCEI_UINT32 data)	
Description	<p>This routine provides read access to the local API copy of the CEI-x30 device interrupt queue. The local API copy of the current device interrupt queue is maintained by hardware interrupt processing within the default API interrupt service routine (ISR). If the host application replaces the default API ISR with a custom ISR, this routine is not usable.</p> <p>Each time this routine is invoked, the specified number of queue entries will be read from the buffer region starting at the location last referenced by the API and ending at the location referenced by the interrupt queue pointer. If fewer than the requested number of entries are found, only those entries available will be returned.</p>	
Return Value	ARS_GOTDATA	At least one interrupt queue entry has been retrieved.
	ARS_NODATA	No unread interrupt queue entries are available.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	Invalid or null <i>data</i> buffer parameter.
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.
	pCEI_UINT32 numberOfWords	(input/output) As an input this parameter specifies the number of interrupt queue entries to read; as an output this parameter indicates how many interrupt queue entries were actually copied to the <i>data</i> array.
	pCEI_UINT32 data	(output) An array referencing the location to store the requested 32-bit interrupt queue entry/entries. Valid Interrupt Queue entry values range from 64 to 95, and 255. The value 64 indicates receive channel 0 label filter triggered, where 95 indicates receive channel 31 label filter triggered, and 255 is reserved for host-triggered interrupt.

AR_INTERRUPT_QUEUE_READ

Syntax	CEI_UINT32 ar_interrupt_queue_read (CEI_INT16 board, pCEI_UINT32 numberOfWords, pCEI_UINT32 data)	
Description	This routine provides read access directly to the CEI-x30 device interrupt queue. Each time this routine is invoked, the specified number of queue entries will be read from the buffer region starting at the location last referenced by the host/API from this routine. If fewer than the requested number of entries are found, only those entries available are returned.	
Return Value	ARS_GOTDATA	At least one interrupt queue entry has been retrieved.
	ARS_NODATA	No unread interrupt queue entries are available.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	Invalid or null <i>data</i> buffer parameter.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.
	pCEI_UINT32 numberOfWords	(input/output) As an input this parameter specifies the number of interrupt queue entries to read; as an output this parameter indicates how many interrupt queue entries were actually copied to the <i>data</i> array.
	pCEI_UINT32 data	(output) An array referencing the location to store the requested 32-bit interrupt queue entry/entries. Valid Interrupt Queue entry values range from 64 to 95, and 255. The value 64 indicates receive channel 0 label filter triggered, where 95 indicates receive channel 31 label filter triggered, and 255 is reserved for host-triggered interrupt.

AR_LABEL_FILTER

Syntax	CEI_INT16 ar_label_filter (CEI_INT16 board, CEI_INT16 channel, CEI_UINT16 label, CEI_INT16 action)	
Description	CEI-x30 devices support the ability to filter ARINC 429 messages by the 8-bit label value. Once filtering has been enabled for a specified channel/label combination, data received with that label value would be discarded until label filtering for the specified label has been disabled. Label filtering is disabled for all labels by default. Label filtering changes are effective immediately on completion of this routine.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	An invalid <i>label</i> or <i>action</i> value was provided.
	ARS_INHARVAL	The specified <i>channel</i> does not support label filtering.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.
	CEI_INT16 channel	(input) channel label filter table this routine is to access. The valid range is 0 to one less than the installed receive channel count.
	CEI_UINT16 label	(input) The label of interest. Valid range is 0-255. Also valid is ARU_ALL_LABELS (511), which invokes the action for all labels on the specified channel.
	CEI_INT16 action	(input) Enable or disable filtering for this combination of board/channel/label. Valid values are: ARU_FILTER_ON (1) enable filtering ARU_FILTER_OFF (0) 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 opens a session and acquires the memory resources allocated to the device, downloads the firmware to the FPGA, and invokes an initialization/reset procedure. Following API and device initialization, optional invocation of AR_BOARD_TEST may provide verification of internal message wrap operation, (execution controlled via invocation of AR_BYPASS_WRAP_TEST).

See the routine descriptions under AP_INITIALIZE_API and AR_INITIALIZE_DEVICE for details regarding the default setup of the API and the device following execution of this routine.

If any portion of the initialization fails or the board is not detected, a status other than ARS_NORMAL is returned.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_WINRTFAIL	The device driver failed to open a session with the device, either because the device is not properly installed in the host system or a resource conflict is inhibiting device driver initialization.
ARS_BADLOAD	The device driver session was opened successfully but the device firmware download failed.
ARS_HW_DETECT	The device driver session was opened but the detected device is not recognized as a CEI-x30 product.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_WRAP_DROP_FAIL	ARINC 429 wrap test data missing.
ARS_WRAP_DATA_FAIL	ARINC 429 wrap test data pattern mismatch.
ARS_WRAP_FLUSH_FAIL	Unknown external messages were received during the internal wrap test execution.
ARS_MEMWRERR	Device SRAM test write/read/verify failure.

Arguments

CEI_INT16 board	(input) Device to access. Valid range is 0-15.
CEI_UINT32 base_seg	(input) This parameter is ignored, (supplied for ARINC API compatibility only).
CEI_INT32 base_port	(input) This parameter is ignored, (supplied for ARINC API compatibility only).
CEI_UINT16 ram_size	(input) This parameter is ignored, (supplied for ARINC API compatibility only).

AR_MODIFY_MSG

Syntax	CEI_INT16 ar_modify_msg (CEI_INT16 board, CEI_INT16 channel, CEI_INT16 msgNumber, CEI_INT16 rate, CEI_INT32 data)	
Description	This routine modifies an existing 32-bit ARINC message for periodic retransmission, originally created through use of the AR_DEFINE_MSG or AR_DEFINE_MSG_BLOCK API routines.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	An invalid <i>channel</i> value was provided.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVHARVAL	Message scheduling is not supported on the specified channel.
	ARS_FAILURE	The supplied message table index exceeds the available number of table entries.
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.
	CEI_INT16 channel	(input) Channel message scheduler table this routine is to access. The valid range is 0 to one less than the number of installed transmit channels.
	CEI_INT16 msgNumber	(input) The unique message scheduler table entry index assigned to this message, as returned for the respective message from the routine AR_DEFINE_MSG or AR_DEFINE_MSG_BLOCK.
	CEI_INT16 rate	(input) Periodic transmission rate, defined in milliseconds. A rate value of zero will disable message transmission for this message scheduler table entry and make this entry available for reuse on the next invocation of AR_DEFINE_MSG or AR_DEFINE_MSG_BLOCK.
	CEI_INT32 data	(input) The updated 32-bit ARINC message to transmit.

AR_MODIFY_MSG_BLOCK

Syntax	CEI_INT16 ar_modify_msg_block (CEI_INT32 numberOfEntries, pAR_SCHEDULED_MSG_ENTRY_TYPE messageEntry)	
Description	This routine provides a method to modify the channel assignment or rate and data values on a series of 32-bit ARINC messages previously defined for periodic retransmission via AR_DEFINE_MSG_BLOCK.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	An invalid <i>channel</i> structure member value was provided.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVHARVAL	Message scheduling is not supported on the specified channel.
	ARS_FAILURE	A supplied message table index exceeds the available number of table entries.
Arguments	CEI_INT32 numberOfEntries (input) The number of entries to modify using the subsequent structure pointer parameter, messageEntry.	
	pAR_SCHEDULED_MSG_ENTRY_TYPE messageEntry	(input)
	array of structures of message definition content, defined as follows:	
	unsigned long messageIndex	The unique message scheduler table entry index assigned to this message. This messageIndex structure member will have been defined in a previous invocation of AR_DEFINE_MSG_BLOCK.
	unsigned long board	Device to access. Valid range is 0-15.

unsigned long channel	Which channel portion of the message scheduler table this routine is to access. The valid range is 0 to one less than the number of installed transmit channels.
unsigned long rate	Periodic transmission rate, in milliseconds. A rate value of zero will disable message transmission.
unsigned long start	Not supported during message modification.
unsigned long txCount	Not supported during message modification.
unsigned long data	The 32-bit ARINC message to transmit.

AR_NUM_RCHANS

Syntax	CEI_INT16 ar_num_rchans (CEI_INT16 board)	
Description	This routine retrieves the number of receive channels installed on the specified device.	
Return Value	Any value less than 40	number of installed receive channels.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.

AR_NUM_XCHANS

Syntax	CEI_INT16 ar_num_xchans (CEI_INT16 board)	
Description	This routine retrieves the number of transmit channels installed on the specified device.	
Return Value	Any value less than 40	number of installed transmit channels.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.

AR_PUT_429_MESSAGE

Syntax

CEI_INT16 ar_put_429_message (CEI_INT16 board, CEI_INT16 channel, CEI_INT32 data)

Description

This routine places the provided ARINC 429 message data in the specified channel transmit buffer. If the specified transmit buffer is full, an overflow status is returned.

Note:

Since ARINC 429 transmit data rates are relatively slow, almost any host can generate transmit data at a much faster rate than data is transmitted.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INHARVAL	The specified <i>channel</i> is invalid or does not support the ARINC 429 protocol.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_XMITOVRFLO	A transmit buffer overrun occurred.

Arguments

CEI_INT16 board	(input) Device to access. Valid range is 0-15.
CEI_INT16 channel	(input) ARINC 429 transmit channel this routine is to access. The valid range is 0 to one less than the number of installed transmit channels.
CEI_INT32 data	(input) ARINC 429 message to transmit in standard ARINC 429 format.

AR_PUT_573_FRAME

Syntax

CEI_INT16 ar_put_573_frame (CEI_INT16 board, CEI_UINT32 numberWords, pCEI_UINT32 transmitCount, pCEI_INT16 arincData)

Description

This routine attempts to transfer *numberWords* of ARINC 573/717 data from the *arincData* source to the device ARINC 573/717 transmit buffer. The amount of data transferred to the transmitter is based on what is available in the buffer, with the actual number of words transferred indicated in the return value of *transmitCount*.

Note:

Since ARINC 573/717 transmit data rates are relatively slow, almost any host can generate transmit frame data at a much faster rate than frame data is actually transmitted.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INHARVAL	The specified <i>board</i> does not support the ARINC 573/717 protocol.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.

Arguments

CEI_INT16 board	(input) Device to access. Valid range is 0-15.
CEI_UINT32 numberWords	(input) Number of words to copy from the source 573 frame to the transmit buffer.
pCEI_UINT32 transmitCount	(output) Indicates how many words were copied from the source 573 frame to the transmit buffer, either less than or equal to the value of <i>numberWords</i> .
pCEI_INT16 arincData	(output) Pointer to the array of ARINC 573/717 frame data. The format of each data word in the source ARINC 573/717 frame is defined as follows:

15 – 12	11 - 0
RESERVED	data

data: the 12-bit ARINC 573/717 data.

AR_PUTBLOCK

Syntax

CEI_INT32 ar_putblock (CEI_UINT32 board, CEI_UINT32 channel, CEI_INT32 maxMessages, CEI_INT32 offset, pCEI_INT32 data, pCEI_INT32 actualCount)

Description

This routine transfers the array of ARINC 429 messages to the specified transmit channel buffer. When this routine returns, the data has not been transmitted, it has only been placed in the transmit buffer. If other data is in the transmit buffer ahead of it, this data is transmitted in turn.

Note:

Since ARINC 429 transmit data rates are relatively slow, almost any host can generate transmit data at a much faster rate than data is transmitted.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INHARVAL	The specified <i>channel</i> is invalid or does not support the ARINC 429 protocol.
ARS_INVARG	An invalid or null <i>maxMessages</i> , <i>data</i> , or <i>actualCount</i> parameter was provided.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_XMITOVRFLO	A transmit buffer overrun occurred.

Arguments

CEI_UINT32 board	(input) Device to access. Valid range is 0-15.
CEI_UINT32 channel	(input) ARINC 429 transmit channel this routine is to access. The valid range is 0 to one less than the number of installed transmit channels.
CEI_INT32 maxMessages	(input) The number of messages to transmit.
CEI_INT32 offset	Unused legacy API parameter .
pCEI_INT32 data	(input) Array supplying 32-bit ARINC data values.
pCEI_INT32 actualCount	(output) The number of messages copied to the transmit buffer.

AR_PUTBLOCK_MULTI_CHAN

Syntax

CEI_INT32 ar_putblock_multi_chan (CEI_UINT32 board, CEI_INT32 maxMessages, pCEI_UINT32 channels, pCEI_INT32 data, pCEI_INT32 actualCount)

Description

This routine transfers messages from the *data* array source to the channel transmit buffer corresponding to the respective transmit channel element of the *channels* array. When this routine returns, the data has not necessarily been transmitted, it has only been placed in the respective transmit buffer(s). If other data is in the transmit buffer ahead of it, this data will be transmitted in turn.

Note:

Since ARINC 429 transmit data rates are relatively slow, almost any host can generate transmit data at a much faster rate than data is transmitted.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INHARVAL	One of the specified <i>channel</i> array elements is invalid or does not support the ARINC 429 protocol.
ARS_INVARG	An invalid or null <i>maxMessages</i> , <i>data</i> , or NULL <i>actualCount</i> parameter was provided.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_XMITOVRFLO	A transmit buffer overrun occurred.

Arguments

CEI_UINT32 board	(input) Device to access. Valid range is 0-15.
CEI_INT32 maxMessages	(input) The number of messages to transmit.
pCEI_UINT32 channels	(input) Array supplying the ARINC 429 transmit channel on which this routine is to transmit the respective ARINC 429 data. The transmit channel index in each element of this array corresponds directly to the ARINC 429 message defined in the respective element of the <i>data</i> array. The valid range for each element of this array is 0 to one less than the number of installed transmit channels.

pCEI_INT32 data	(input) Array supplying 32-bit ARINC data values.
pCEI_INT32 actualCount	(input) The number of messages transmitted.

AR_PUTFILTER

Syntax

CEI_UINT32 ar_putfilter (CEI_UINT32 board, CEI_UINT32 channel, pCEI_CHAR filterTable)

Description

This routine assigns an entire channel portion of the label filter table for the specified receive channel. Each receive channel has a separate area in the device label filter table, which is used by the firmware to control storage of received labels. Each element of the filter table consists of a three bit field defined for compatibility with the CEI-x20 product line as follows:

FILTER_SEQUENTIAL	0x10	If CLEAR add label to circular receive buffer
FILTER_SNAPSHOT	0x20	If CLEAR add label to snapshot receiver buffer
FILTER_INTERRUPT	0x40	If SET on reception insert the respective receive channel tag (ranging from 64-95) in the interrupt queue and if enabled generate a PCI interrupt.

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

filterTable[MAX_ESSM][MAX_SDI][MAX_LABEL]

and accessed as follows in the array referenced by *filterTable*:

filterTable[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

Note:

For FILTER_INTERRUPT processing, an entry is made into the interrupt queue if specified through receiver interaction with the label filter table definition, even if hardware interrupts are not enabled.

To write individual label filter table elements, refer to the API routines AR_ENH_LABEL_FILTER and AR_LABEL_FILTER.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVARG	An invalid <i>channel</i> or <i>filterTable</i> value was provided.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.

Arguments

CEI_UINT32 board	(input) Device to access. Valid range is 0-15.
------------------	--

CEI_UINT32 channel	(input) Specifies which receive channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
pCEI_CHAR filterTable	(input) Array containing the contents of the specified channel's label filter table. This array must have an allocation of 8Kbytes.

AR_PUTWORD

Syntax

CEI_INT16 ar_putword (CEI_INT16 board, CEI_INT16 channel, CEI_INT32 arincdata)

Description

This routine places the provided message data in the specified channel transmit buffer. When this routine returns, the data has not necessarily been sent, it has only been placed in the transmit buffer. If other data is in the transmit buffer ahead of it, this data will be transmitted in turn. If the specified transmit buffer is full, an overflow status is returned.

The channel value passed to this routine corresponds to the ARINC 429 transmit channel index, starting with zero. If that value exceeds the 429 transmit channel count and an ARINC 573/717 transmitter exists, it is used as the designated transmit channel buffer.

Note:

Since ARINC transmit data rates are relatively slow, almost any host can generate transmit data at a much faster rate than data is transmitted.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INHARVAL	The specified <i>channel</i> is invalid or does not support the ARINC 429 (or 717) protocol.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_XMITOVRFLO	A transmit buffer overrun occurred.

Arguments

CEI_INT16 board	(input) Device to access. Valid range is 0-15.
CEI_INT16 channel	(input) Transmit channel this routine is to access. The valid range is 0 to one less than the installed transmit channel count.
CEI_INT32 arincdata	(input) 32-bit ARINC 429 message to transmit.

AR_RESET

Syntax

CEI_INT16 ar_reset (CEI_INT16 board)

Description

This routine assigns the global enable register Global Enable bit to be *disabled* and reinitializes the device to the same channel configuration as that following an invocation of AR_LOADSLV. See the description for the routine AR_INITIALIZE_DEVICE for more details.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_MEMWRERR	Device SRAM test write/read/verify failure.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.

Arguments

CEI_INT16 board	(input) Device to access. Valid range is 0-15.
-----------------	--

AR_RESET_TIMERCNT

Syntax	CEI_VOID ar_reset_timercnt (CEI_INT16 board)	
Description	This routine is designed to provide compatibility with the CEI-x20 ARINC API. It resets the CEI-x30 device internal one-microsecond timer to zero.	
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.

AR_SET_CONFIG

Syntax	CEI_INT16 ar_set_config (CEI_INT16 board, CEI_INT16 item, CEI_UINT32 value)	
Description	This routine provides a means to define general device configuration attributes, as well as limited individual channel configuration attributes. It is provided for backward compatibility to CEI-x20 based applications. The routine AR_SET_DEVICE_CONFIG is the desired routine for defining channel and board-level configuration items.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVARG	The <i>item</i> argument value is not supported by this API routine.
	ARS_INVHARVAL	The <i>item</i> argument value is not supported by this device configuration.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed .
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.
	CEI_INT16 item	(input) Attribute about which to set information:
	ARU_XMIT_RATE	transmit rate for all transmitters.
	ARU_RECV_RATE	receive rate for all receivers.
	ARU_PARITY	parity for all transmitters and receivers.
	ARU_INTERNAL_WRAP	enables internal wrap mode for all receivers.
	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_TX_CH01_SHUT_OFF –
 ARU_TX_CH32_SHUT_OFF transmitter 1 - 32 disable.

 ARU_TX_CH01_LB_INJ – transmitter 1 - 32 low bit
 ARU_TX_CH32_LB_INJ error enable.

 ARU_TX_CH01_HB_INJ – transmitter 1 - 32 high bit
 ARU_TX_CH32_HB_INJ error enable.

 ARU_TX_CH01_GAP_INJ – transmitter 1 - 32
 ARU_TX_CH32_GAP_INJ message gap error enable.

 ARU_RX_TIMETAG_MODE the timer/time-tag source and
 resolution

 ARU_ACCESS_SNAPSHOT_BUFFER snapshot storage mode

 ARU_IRIG_WRAP_ENABLE enables IRIG receiver internal wrap

 ARU_IRIG_INPUT_THRESHOLD sets the IRIG DAC threshold

 ARU_IRIG_ADJUST_THRESHOLD invokes a more precise IRIG
 DAC auto-adjustment procedure

 ARU_IRIG_QUICK_ADJUSTMENT invokes a quick IRIG DAC
 auto-adjustment procedure

 ARU_IRIG_SET_BIAS assigns an offset to the board IRIG
 time value

CEI_UINT32 value (input) the value to set the item.

If the specified item is ARU_XMIT_RATE (1) or ARU_RECV_RATE (2), valid value parameter selections are:

AR_HIGH (0) high rate (100Kbs)
 AR_LOW (1) low rate (12.5Kbs)
 Any other value specifies a frequency value in Hertz.

If the specified item is ARU_RX_CH nn _BIT_RATE (500-531), where nn is the receiver channel (01 - 32), valid value parameter selections are:

AR_HIGH (0) high rate (100Kbs)
 AR_LOW (1) low rate (12.5Kbs)
 Any other value specifies a frequency value in Hertz.

If the specified item is ARU_TX_CH nn _BIT_RATE (700-731), where nn is the transmitter channel (01 - 32), valid value parameter selections are:

AR_HIGH (0) high rate (100Kbs)
 AR_LOW (1) low rate (12.5Kbs)

Any other value specifies a frequency value in Hertz.

Note

Any specified transmit bus frequency below 15KHz will be assigned to a slow slew rate. Any specified transmit bus frequency above 15KHz will be assigned to a fast slew rate.

If the specified item is ARU_PARITY (3), the value parameter specifies the parity selection for all transmit and receive channels.

AR_ODD (0) odd transmit parity and receive parity detect enabled
 AR_EVEN (1) even transmit parity and rx parity detect enabled
 AR_OFF (8) transmit parity and receive parity detect disabled
 AR_RAW (0x2000) transmit parity and rx parity detect disabled

If the specified item is ARU_RX_CH nn _PARITY (900-931), where nn is the receiver channel (01 - 32), valid value parameter selections are:

AR_ODD (0) receiver parity detection enabled
 AR_OFF (8) receiver parity detection disabled
 AR_RAW (0x2000) receiver parity detection disabled

If the specified item is ARU_TX_CH nn _PARITY (1100-1131), where nn is the transmitter channel (01 - 32), valid value parameter selections are:

AR_ODD (0) odd transmitter parity
 AR_EVEN (1) even transmitter parity
 AR_OFF (8) transmitter parity disabled
 AR_RAW (0x2000) transmitter parity disabled

If the requested item is ARU_TX_CH nn _SHUT_OFF (1700-1731), where nn is the transmitter channel (01 - 32), valid value parameter selections are:

AR_ON (7) external transmission is disabled
 AR_OFF (8) external transmission is enabled

For the RAR-PCIE board, disabling external transmission also causes the transmit pins to switch to a tri-state condition; for all other boards the transmit pins will switch to a null condition.

If the requested item is ARU_TX_CH nn _HB_INJ (3300-3331), where nn is the transmitter channel (01 - 32), valid value parameter selections are:

AR_ON (7) 33-bit transmission is enabled
 AR_OFF (8) standard 32-bit transmission is enabled

If the requested item is ARU_TX_CH nn _LB_INJ (3500-3531), where nn is the transmitter channel (01 - 32), valid value parameter selections are:

AR_ON (7) 31-bit transmission is enabled
 AR_OFF (8) standard 32-bit transmission is enabled

If the requested item is ARU_TX_CH nn _GAP_INJ (3700-3731), where nn is the transmitter channel (01 - 32), valid value parameter selections are:

AR_ON (7) 3-bit message gap is used
 AR_OFF (8) standard 4-bit message gap is used

If the specified item is ARU_INTERNAL_WRAP (4), valid value parameter selections are:

AR_WRAP_ON	(0) internal wrap enabled
AR_WRAP_OFF	(1) internal wrap disabled

If the specified item is ARU_RX_TIMETAG_MODE (440), valid value parameter selections represent the timer/time-tag source and resolution. This item specifies the resolution of any timer-read or receive data time-tag value obtained via the API, with value selections defined as follows:

AR_TIMETAG_EXT_IRIG_64BIT	(0)
AR_TIMETAG_INT_USEC_64BIT	(1)
AR_TIMETAG_INT_20USEC_32BIT	(3)
AR_TIMETAG_INT_MSEC_32BIT	(4)

A value of AR_TIMETAG_EXT_IRIG_64BIT selects the source as the external IRIG receiver, if connected; otherwise, if the IRIG signal is not internally wrapped this selection would be invalid. All other values represent various timer/time-tag LSB resolution values based on the internal CEI-x30 device timer.

If the specified item is ARU_ACCESS_SNAPSHOT_BUFFER (38), a valid value parameter for selecting the active Snapshot Buffer storage mode is:

ARU_LABEL_ONLY	(0) messages stored based on label
ARU_LABEL_WITH_SDI	(1) messages stored based on the combined label and SDI field values

If the specified item is ARU_IRIG_WRAP_ENABLE (441), valid value parameter selections are:

AR_ON	(7) IRIG receiver internal wrap enabled
AR_OFF	(8) IRIG receiver internal wrap disabled

If the specified item is ARU_IRIG_INPUT_THRESHOLD (442), the value parameter specifies the IRIG receiver threshold voltage in millivolts.

The item ARU_IRIG_ADJUST_THRESHOLD (443) invokes the IRIG DAC auto-adjustment procedure. This procedure will determine the low and high threshold values at which the incoming IRIG signal is present. It then determines the best threshold level for the IRIG DAC, and returns the value to the application in place of a returned status (failures are indicated via return value of ARS_FAILURE). This procedure may execute for up to, and in some cases in excess of, one minute before finding the best-case threshold value for the incoming IRIG signal. If a printed status of the execution progress within this procedure is desired, assign the *value* parameter to any non-zero value.

If the specified item is ARU_IRIG_QUICK_ADJUSTMENT (444), the API will perform a quick adjustment of the IRIG DAC for an external input IRIG signal using signal edge detection for verification of signal presence. This execution of this adjustment should require less than one second.

If the specified item is ARU_IRIG_SET_BIAS (446), a valid value parameter consists of an offset to the board-supplied IRIG time specified in milliseconds. The bias time range is +/-32.768 seconds.

AR_SET_DEVICE_CONFIG

Syntax	CEI_INT16 ar_set_device_config (CEI_INT16 board, CEI_INT16 channel, CEI_INT16 item, CEI_INT16 value)	
Description	This is the recommended routine to define the general device and ARINC 429 channel configuration attributes.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVARG	The item argument value is not supported by this API routine.
	ARS_INVHARVAL	The item argument value is not supported by this device configuration.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.
	CEI_INT16 channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed channel count for the respective channel type.
	CEI_INT16 item	(input) Specifies the configuration attribute to define:
	ARU_RX_BITRATE	receive rate for specified channel.
	ARU_TX_BITRATE	transmit rate for specified channel.
	ARU_RX_PARITY	receive parity for specified channel.
	ARU_TX_PARITY	transmit parity for specified channel.
	ARU_RX_FIFO_ENABLE	receive channel FIFO enable.
	ARU_TX_FIFO_ENABLE	transmit channel FIFO enable.
	ARU_TX_DISABLE	transmit channel transceiver disable.
	ARU_TX_GAP_ERROR	transmit message gap error enable.
	ARU_TX_BIT_ERROR	transmit message size error enable.
	ARU_FAST_SLEW_RATE	transmit channel slew rate select.
	ARU_RECV_MODE	receive channel internal wrap mode.
	ARU_RX_MERGED_MODE	receive channel merge mode enable.
	ARU_ACCESS_SNAPSHOT_BUFFER	snapshot storage mode
	ARU_BYPASS_INIT_WRAP_TEST	bypass initialization wrap test
	ARU_MULTITHREAD_PROTECT	control use of thread protection
	ARU_RX_TIMETAG_MODE	timer/time-tag source and resolution
	ARU_DIFFERENTIAL_OUT	differential output enable and state

ARU_DISCRETE_OUT sets a discrete output state
 ARU_IRIG_WRAP_ENABLE enables IRIG receiver internal wrap
 ARU_IRIG_INPUT_THRESHOLD sets the IRIG DAC threshold
 ARU_IRIG_ADJUST_THRESHOLD both invoke IRIG DAC
 ARU_IRIG_QUICK_ADJUSTMENT auto-adjustment procedures
 ARU_IRIG_SET_BIAS assigns an offset to IRIG time
 ARU_IRIG_OUTPUT_ENABLE R830RX IRIG Tx state
 ARU_HW_ENHANCE_UPDATE update board for enhanced f/w
 ARU_HW_INTERRUPT_ENABLE enable/disable PCI interrupts
 ARU_INSERT_INT_Q_ENTRY insert entry in interrupt queue

CEI_INT16 value (input) the value to set the specified item.

If the requested item is ARU_RX_BITRATE (1) or ARU_TX_BITRATE (2), valid value parameter selections are:

ARU_SPEED_HIGH (0) high rate (100Kbs)
 ARU_SPEED_LOW (1) low rate (12.5Kbs)

Any other value assigns a non-standard bus speed, and is translated as a divisor for the 16MHz device clock reference. This value and the respective baud rate may be interpreted using the following formulas:

$$\text{Baud Rate} = 16,000,000 / (\text{Value} + 2)$$

$$\text{Value} = (16,000,000 / \text{Desired Baud Rate}) - 2$$

Note

Any non-standard transmit bus speed value resulting in a baud rate below 15KHz will be assigned to a slow slew rate. Any non-standard transmit bus speed value resulting in a baud rate at or above 15KHz will be assigned to a fast slew rate.

If the requested item is ARU_RX_PARITY (3), valid value parameter selections are:

AR_ON (7) receiver parity enabled
 AR_OFF (8) receiver parity disabled

If the requested item is ARU_TX_PARITY (4), valid value parameter selections are:

ARU_PARITY_ODD (0) odd transmitter parity
 ARU_PARITY_EVEN (1) even transmitter parity
 ARU_PARITY_NONE (2) transmitter parity disabled

If the requested item is ARU_RECV_MODE (5), valid value parameter selections are:

AR_WRAP_ON (0) internal wrap enabled
 AR_WRAP_OFF (1) internal wrap disabled

If the requested item is ARU_RX_FIFO_ENABLE (16) or ARU_TX_FIFO_ENABLE (17), valid value parameter selections are:

- AR_ON (7) Rx/Tx FIFO operation enabled
- AR_OFF (8) Rx/Tx FIFO operation disabled

If the requested item is ARU_TX_DISABLE (10), valid value parameter selections are:

- AR_ON (7) external transmission disabled
- AR_OFF (8) external transmission enabled

For the RAR-PCIE board, disabling external transmission also causes the transmit pins to switch to a tri-state condition; for all other boards the transmit pins will switch to a null condition.

If the requested item is ARU_TX_GAP_ERROR (8), valid value parameter selections are:

- AR_ON (7) transmit message gap error enabled
- AR_OFF (8) transmit message gap error disabled

If the requested item is ARU_TX_BIT_ERROR (6), valid value parameter selections are:

- AR_LO (0) Low Bit Error operation is enabled
- AR_HI (1) High Bit Error operation is enabled
- AR_OFF (8) bit errors are disabled on this transmitter

If the requested item is ARU_FAST_SLEW_RATE (323), valid value parameter selections are:

- AR_ON (7) Fast Slew Rate selected (1.5 μ sec rise time)
- AR_OFF (8) Slow Slew Rate selected (10 μ sec rise time)

If the requested item is ARU_RX_MERGED_MODE (18), valid value parameter selections are:

- AR_ON (7) receiver merged mode operation enabled
- AR_OFF (8) receiver merged mode operation disabled

If the requested item is ARU_ACCESS_SNAPSHOT_BUFFER (38), valid value parameter selections are:

- ARU_LABEL_ONLY (0) message storage on a label basis
- ARU_LABEL_WITH_SDI (1) message storage on a label/sdi basis

If the specified item is ARU_BYPASS_INIT_WRAP_TEST (320), valid value parameter selections are:

- AR_ON (7) bypass internal wrap test invocation during init
- AR_OFF (8) execute internal wrap test invocation during init

If the specified item is ARU_MULTITHREAD_PROTECT (321), valid value parameter selections are:

- AR_ON (7) enables mutex/semaphore thread protection
- AR_OFF (8) disables mutex/semaphore thread protection

If the specified item is ARU_RX_TIMETAG_MODE (440), valid value parameter selections represent the timer/time-tag source and resolution. This item specifies the resolution of any timer-read or receive data time-tag value obtained via the API, with value selections defined as follows:

AR_TIMETAG_EXT_IRIG_64BIT	(0)
AR_TIMETAG_INT_USEC_64BIT	(1)
AR_TIMETAG_INT_20USEC_32BIT	(3)
AR_TIMETAG_INT_MSEC_32BIT	(4)

A value of AR_TIMETAG_EXT_IRIG_64BIT selects the source as the external IRIG receiver, if connected; otherwise, if the IRIG signal is not internally wrapped this selection would be invalid. All other values represent various timer/time-tag LSB resolution values based on the internal CEI-x30 device timer.

If the specified item is ARU_DISCRETE_OUT (12), valid value parameter selections are:

AR_HI	(1) = the discrete is set High
AR_LO	(0) = the discrete is set Low

If the specified item is ARU_DIFFERENTIAL_OUT (23), valid value parameter selections assign both the enable state and output state of the differential channel:

AR_HI	(1) the differential output is set high
AR_LO	(0) the differential output is set low
AR_ON	(7) the differential output is enabled
AR_OFF	(8) the differential output is disabled

If the specified item is ARU_IRIG_WRAP_ENABLE (441), valid value parameter selections are:

AR_WRAP_ON	(0)	IRIG receiver internal wrap enabled
AR_WRAP_OFF	(1)	IRIG receiver internal wrap disabled

If the specified item is ARU_IRIG_INPUT_THRESHOLD (442), the value parameter specifies the IRIG receiver threshold voltage in millivolts.

The item ARU_IRIG_ADJUST_THRESHOLD (443) invokes the IRIG DAC auto-adjustment procedure. This procedure determines the low and high threshold values at which the incoming IRIG signal is present. It then determines the best threshold level for the IRIG DAC, and returns the value to the application in place of a returned status (failures are indicated through return value of ARS_FAILURE). This procedure may execute for up to and in some cases in excess of one minute before finding the best-case threshold value for the incoming IRIG signal. If a printed status of the execution progress within this procedure is desired, assign the *value* parameter to any non-zero value.

If the specified item is ARU_IRIG_QUICK_ADJUSTMENT (444), the API performs a quick adjustment of the IRIG DAC for an external input

IRIG signal, using signal edge detection for verification of signal presence. This execution of this adjustment should require less than one second.

If the specified item is ARU_IRIG_SET_BIAS (446), the API assigns an offset to the board IRIG time value calculation for any time and time-tag retrieval.

If the specified item is ARU_IRIG_OUTPUT_ENABLE (26), valid value parameter selections to control the state of the R830RX IRIG Generator Enable are:

AR_ON	(7) IRIG output is enabled
AR_OFF	(8) IRIG output is disabled

If the specified item is ARU_HW_ENHANCE_UPDATE (31), valid value parameter selections to control the board's PCI BAR2 Size allocation residing in an on-board EEPROM are:

AR_ON	(7) support for the CEI-x30 Enhanced Firmware Interface is enabled
AR_OFF	(8) support for the CEI-x30 Enhanced Firmware Interface is disabled

Based on the value parameter selection, the board may be reprogrammed to support the 512Kb CEI-x30 Enhanced Firmware Interface (exclusively supported with CEI-x30 API Version 2.00 and later); or it may be reprogrammed to support only the standard 4Kb CEI-x30 interface, (exclusively supported with CEI-x30 API Versions 1.00 through 1.70).

Note: Any modification to the current PCI BAR2 Size allocation requires a host restart for those changes to take effect in the system.

If the specified item is ARU_HW_INTERRUPT_ENABLE (29), valid value parameter selections to control the state of the PCI Interrupt Enable state are:

AR_ON	(7) PCI Interrupts are enabled
AR_OFF	(8) PCI Interrupts are disabled

If the specified item is ARU_INSERT_INT_Q_ENTRY (37), the value parameter is ignored and the value 255 is inserted as the next entry in the device interrupt queue.

AR_SET_573_CONFIG

Syntax	CEI_INT16 ar_set_573_config (CEI_INT16 board, CEI_INT16 item, CEI_INT32 value)	
Description	This routine provides the method for manipulating the ARINC 573/717 channel configuration attributes.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVHARVAL	the item argument value is not supported by the device configuration or this API routine.
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.
	CEI_INT16 item	(input) Specifies the configuration attribute to define:
	ARU_RECV_MODE	receiver internal wrap.
	ARU_RX_MERGED_MODE	receiver merge mode enable.
	ARU_RX_BITRATE	receive channel bit rate.
	ARU_RX_FIFO_ENABLE	receive channel FIFO enable.
	ARU_TX_BITRATE	transmit channel bit rate.
	ARU_TX_FIFO_ENABLE	transmit channel FIFO enable.
	ARU_573_RX_AUTO_DETECT	data frame auto-detect enable.
	ARU_573_RX_BPRZ_SELECT	receiver BPRZ/HBP selection.
	ARU_573_TX_BPRZ_SELECT	transmit BPRZ encoding enable.
	ARU_573_TX_HBP_SELECT	transmit HBP encoding enable.
	ARU_573_TX_SLEW_RATE	transmit slew rate select.
	ARU_573_SYNC_WORD1	receiver auto-detect sync word 1.
	ARU_573_SYNC_WORD2	receiver auto-detect sync word 2.
	ARU_573_SYNC_WORD3	receiver auto-detect sync word 3.
	ARU_573_SYNC_WORD4	receiver auto-detect sync word 4.
	CEI_INT32 value	(input) The state to assign to the specified configuration item:

If the requested item is ARU_RECV_MODE (5), valid value parameter selections are:

AR_WRAP_ON (0) = internal wrap enabled
 AR_WRAP_OFF (1) = internal wrap disabled

If the requested item is ARU_RX_MERGED_MODE (18), valid value parameter selections are:

AR_ON (7) receiver merged mode operation enabled
 AR_OFF (8) receiver merged mode operation disabled

If the requested item is ARU_RX_FIFO_ENABLE (16) or ARU_TX_FIFO_ENABLE (17), valid value parameter selections are:

AR_ON (7) FIFO operation enabled
 AR_OFF (8) FIFO operation disabled

If the configuration item is ARU_RX_BITRATE (1) or ARU_TX_BITRATE (2), valid item values are one of the following (0-7):

ARU_573_RATE_SIZE_384_32	384 bps, 32 word sub-frame
ARU_573_RATE_SIZE_768_64	768 bps, 64 word sub-frame
ARU_573_RATE_SIZE_1536_128	1536 bps, 128 word sub-frame
ARU_573_RATE_SIZE_3072_256	3072 bps, 256 word sub-frame
ARU_573_RATE_SIZE_6144_512	6144 bps, 512 word sub-frame
ARU_573_RATE_SIZE_12288_1024	12288 bps, 1024 word sub-frame
ARU_573_RATE_SIZE_24576_2048	24576 bps, 2048 word sub-frame
ARU_573_RATE_SIZE_49152_4096	49152 bps, 4096 word sub-frame

If the configuration item is ARU_573_RX_AUTO_DETECT (301), valid item values are one of the following:

AR_ON (7) ARINC 573/717 frame auto-detection enabled
 AR_OFF (8) ARINC 573/717 frame auto-detection disabled

If the configuration item is ARU_573_RX_BPRZ_SELECT (302), valid item values are one of the following:

AR_OFF (7) ARINC 573/717 HBP reception enabled
 AR_ON (8) ARINC 573/717 BPRZ reception enabled

If the configuration item is ARU_573_TX_BPRZ_SELECT (313), valid item values are one of the following:

AR_OFF (7) ARINC 573/717 BPRZ transmission disabled
 AR_ON (8) ARINC 573/717 BPRZ transmission enabled

If the configuration item is ARU_573_TX_HBP_SELECT (314), valid item values are one of the following:

AR_OFF (7) ARINC 573/717 HBP transmission disabled
AR_ON (8) ARINC 573/717 HBP transmission enabled

If the configuration item is ARU_573_TX_SLEW_RATE (315), valid item values are one of the following:

ARU_573_TX_SLEW_1PT5 (1) = fast (1.5µsec rise time)
ARU_573_TX_SLEW_10PT0 (0) = slow (10.0µsec rise time)

If the configuration item is ARU_573_SYNC_WORD1 (307), ARU_573_SYNC_WORD2 (308), ARU_573_SYNC_WORD3 (309), or ARU_573_SYNC_WORD4 (310), a valid item value is any 12-bit non-zero value.

AR_SET_MULTITHREAD_PROTECT

Syntax CEI_INT16 ar_set_multithread_protect (CEI_INT16 board, CEI_INT16 state)

Description This routine controls the use of mutex/semaphore protection around all device channel-specific accesses performed within the API routines. This type of thread protection should be enabled for any multi-threaded application or reentrant API usage.

Return Value	ARS_NORMAL	routine was successful.
	ARS_INVARG	An invalid <i>state</i> value was provided.
	ARS_INVBOARD	An invalid <i>board</i> value was provided.

Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.
	CEI_INT16 state	(input) Multi-thread protection setting, valid values are defined as follows:

AR_ON (7) enables mutex/semaphore protection.

AR_OFF (8) disables mutex/ semaphore protection.

AR_SET_ISR_FUNCTION

Syntax

CEI_INT32 ar_set_isr_function (CEI_INT32 board, pCEI_VOID function)

Description

This routine allows the host application to define a custom interrupt service routine to be referenced by the operating system-specific hardware interrupt initialization. It assigns the host-supplied function pointer to an array of pointers indexed by the *board* parameter value. The function referenced by this pointer is invoked from the internal routine `cei_utl_interrupt_handler()` instead of executing the default API-supplied "flush the h/w interrupt queue" processing.

The function declaration for the supplied interrupt service routine should be defined as follows for the respective operating system:

Any Windows:

```
void _stdcall host_interrupt_handler ( CEI_INT32 deviceIndex );
```

Any Linux distribution:

```
void host_interrupt_handler ( CEI_INT32 deviceIndex );
```

Any VxWorks or Integrity distribution:

```
void host_interrupt_handler ( CEI_INT32 deviceIndex ,  
                             pCEI_UINT32 data );
```

Note:

The data parameter should not be used by the ISR.

This interrupt service routine is executed as a separate process or task from the actual host low-level h/w interrupt processing, with an execution priority based on default process/task priority settings for the respective host operating system.

Return Value

ARS_NORMAL	routine was successful.
ARS_INVARG	An null <i>function</i> value was provided.
ARS_INVBOARD	An invalid <i>board</i> value was provided.

Arguments

CEI_INT16 board	(input) Device index for storing the function pointer. Valid range is 0-15.
pCEI_VOID function	(input) Function pointer to the host specified interrupt service routine.

AR_SET_PRELOAD_CONFIG

Syntax

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

Description

This routine is designed to provide protection when executing multi-threaded or multi-process applications with your CEI-x30 device. Call this routine before calling AR_LOADSLV to update the value of a particular pre-load API operational configuration setting. This routine should not be called subsequent to any invocation of AR_LOADSLV.

If *item* is ARU_CONCURRENCY_MODE, the *value* parameter specifies the API 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 operating systems.

The default concurrency mode, AR_CONC_NONE, provides no multi-thread protection to the device and no multi-process API support. The user application must ensure that only 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 for each device access 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. This mode is supported on all operating systems supported by the CEI-x30 software distribution.

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. This mode is only supported under Windows operating systems and Linux Kernel 2.6 distributions specifically supporting System V features.

Note:

The use of hardware interrupts is prohibited when multi-process operations are enabled under the Windows operating system.

In this mode, all processes must invoke AR_LOADSLV during initialization of the process and AR_CLOSE upon termination. Failure to follow this strict requirement could result in irrecoverable errors. 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 all process invocations of AR_LOADSLV and AR_CLOSE. 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 interfaces with each board.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An invalid <i>board</i> value was provided.
ARS_INVARG	An invalid <i>item</i> or <i>value</i> parameter was provided.
ARS_BOARD_MUTEX	Creation of the Board Lock mechanism failed.
ARS_NO_OS_SUPPORT	The item selection not supported with the host operating system.

Arguments

CEI_INT16 board	(input) Device to access. Valid range is 0-15.
CEI_INT16 item	(input) Attribute about which to set information, currently limited to a single option, ARU_CONCURRENCY_MODE.
CEI_UINT32 value	(input) the value to set the specified item.
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).

AR_SET_RAW_MODE

Syntax	CEI_INT16 ar_set_raw_mode (CEI_INT16 board, CEI_INT16 direction, CEI_INT16 channel, CEI_INT16 control)	
Description	<p>This routine is designed to provide compatibility with the CEI-x20 ARINC APIs. The routine AR_SET_DEVICE_CONFIG is the recommended routine for manipulating the channel parity attribute.</p> <p>Each transmit and receive channel can be configured to run in <i>raw</i> mode, where parity assignment and detection is disabled. When raw mode is selected, every 32-bit ARINC word is transmitted or received with the parity bit (msb) unchanged. This differs from a standard ARINC 429 data transfer in which the message parity is always calculated. Raw mode is typically used for older ARINC specifications such as ARINC 575.</p>	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVHARVAL	An invalid <i>channel</i> parameter was provided or the specified <i>channel</i> doesn't support parity selection.
	ARS_INVARG	An invalid <i>direction</i> or <i>control</i> parameter was provided.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.
	CEI_INT16 direction	<p>(input) The type of channel specified in the channel argument (transmit or receive). Valid values to select transmit channels are:</p> <p>TRANSMIT_CHANNEL (0) ARU_XMIT (34)</p> <p>Valid values to select receive channels are:</p> <p>RECEIVE_CHANNEL (1) ARU_RECV (35)</p>

CEI_INT16 channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed channel count for the respective channel type.
CEI_INT16 control	(input) Enables or disables raw mode. AR_ON (7) enable "raw" mode, parity is disabled AR_OFF (8) disable "raw" mode, parity assignment and/or checking is enabled

AR_SET_STORAGE_MODE

Syntax	CEI_INT16 ar_set_storage_mode (CEI_INT16 board, CEI_INT16 mode)	
Description	<p>This routine is designed to provide compatibility for data storage mode selection with the CEI-x20 ARINC API. The routine AR_SET_DEVICE_CONFIG is the recommended routine for manipulating the receive channel data storage mode of operation.</p> <p>CEI-x30 devices are capable of storing received data in either individual FIFO buffers or in a single merged FIFO buffer on a channel-by-channel basis. This routine allows you to perform a single invocation to select the universal receive mode for all receive channels on the device, as either BUFFERED or MERGED.</p> <p>When a channel storage mode is <i>buffered</i>, each receiver is assigned an independent circular FIFO buffer for data storage. When a channel storage mode is <i>merged</i>, data received on than channel is stored in the merged receive FIFO. Each receive data API routine handles the respective channel active storage mode internally, acquiring data from the appropriate buffer as necessary.</p>	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVARG	An invalid <i>mode</i> value was provided.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.
	CEI_INT16 mode	(input) The type of receive data storage mode to assign. Valid values are:
	ARU_BUFFERED ARU_MERGED	(0) use individual FIFO buffers (2) use the merged FIFO buffer

AR_SET_TIME

Syntax

CEI_INT16 ar_set_time (CEI_INT16 board, pAR_TIMETAG_TYPE timeTag)

Description

This routine assigns a value to the specified CEI-x30 device internal timer or IRIG time generator based on an application-supplied time format and value.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVARG	An invalid timeTag structure member, timeTagFormat selection was provided.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed .

Arguments

CEI_INT16 board (input) Device to access. Valid range is 0-15.

pAR_TIMETAG_TYPE timeTag

(input) The 64-bit device timer or 32-bit IRIG time generator value to assign to the respective hardware. Valid options for the *timeTagFormat* structure member are:

AR_TIMETAG_EXT_IRIG_64BIT (0)

AR_TIMETAG_INT_USEC_64BIT (1)

To assign a 32-bit IRIG Day/Time value, the *timeTag* structure member should be defined as a 30-bit value of the following bit format:

29-28	27-24	23-20	19-18	17-14	13-11	10-7	6-4	3-0
hundreds of days	tens of days	days	tens of hours	hours	tens of minutes	minutes	tens of seconds	seconds

To assign a 64-bit internal timer value, the *timeTag* structure member should be defined as a 64-bit 1 microsecond resolution time value.

The *timeTagRef* structure member is not used by this routine.

See the section titled *Time-tag Structure Definition* for more information on the AR_TIMETAG_TYPE data structure.

AR_SLEEP

Syntax

CEI_VOID ar_sleep (CEI_UINT32 sleep_ms)

Description

This routine suspends execution of the calling thread for the specified number of milliseconds. Platform-dependent thread delay methods are used to implement this operation, defined below for the supported operating system. The accuracy of this operation is dependent upon the accuracy of the underlying operating system call.

Return Value

None

Arguments

CEI_INT32 sleep_ms (input) Sleep duration, in milliseconds.

AR_SET_TIMERRATE

Syntax

CEI_VOID ar_set_timerrate (CEI_INT16 board, CEI_INT16 rate)

Description

This routine assigns the API internal timer reference resolution for compatibility with applications based on the CEI-x20 product family device timer and time-tag operation. When you invoke this routine, the CEI-x30 API sets the current timer usage and time-tag reporting mode to the “CEI-x20 compatibility mode”. In this mode, all scheduled message rate and start offset values and receive message time-stamp values are referenced in terms of the resolution value assigned in the “rate” parameter instead of the standard one millisecond (for scheduled message rate/offset) or one microsecond (for receive message time-stamps).

The actual CEI-x30 hardware device time-tag reference timer resolution is not programmable; rather, it is a fixed one microsecond resolution.

The CEI-x30 message scheduler minimum rate resolution is fixed at a one millisecond resolution. As a result, any timer rate assignment having a resolution that is not divisible by, or is less than, one millisecond, coupled with an attempt to define a message scheduler entry rate or start offset value that is not divisible by one millisecond results in that value being assigned to the nearest 1 millisecond value below the assigned value.

Arguments

CEI_INT16 board	(input) Device to access. Valid range is 0-15.
CEI_INT16 rate	(input) Resolution of the CEI-x30-emulated timer operation, specified as a tick-timer value having a resolution of 250 nanoseconds.

AR_STOP

Syntax	CEI_INT16 ar_stop (CEI_INT16 board)	
Description	This routine assigns the global enable register Global Enable bit to be <i>disabled</i> for the specified device. All active message processing is terminated upon execution of this routine.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
Arguments	CEI_INT16 board	(input) Device to access. Valid range is 0-15.

AR_VERSION

Syntax	CEI_VOID ar_version (pCEI_CHAR verstr)	
Description	This routine retrieves the current software version number of the device API.	
Arguments	pCEI_CHAR verstr	(output) String representation of the API Version number consisting of up to 10 characters.

AR_WAIT

Syntax

CEI_VOID ar_wait (CEI_FLOAT nsecs)

Description

This routine delays the calling application by the specified number of seconds. The delay is based on the respective OS system time utility.

Arguments

CEI_FLOAT nsecs (input) Number of seconds to delay.



CEI-x30 Hardware Interface

Overview

This chapter describes the low level programming of the CEI-x30 product.

Note: The information in this chapter is provided if you intend to author your own software interface and device driver.

Control of a CEI-x30 device is performed by reading and writing the board registers, mapped into the host memory space via the BAR0 and/or BAR2 memory regions. PCI devices use the BAR0 region for PCI target interface configuration registers, and the BAR2 region for all other registers and memory. Native PCI-Express devices do not have a PCI target interface component and use the BAR0 region for all access to registers and memory. To program the device, you must first know where these memory regions are mapped in host memory space. In the following sections, the PCI Configuration region and BAR2 (BAR0 for native PCI-Express boards) device configuration registers and buffers are described. All BAR2/BAR0 registers and buffers are 32 bits wide for both read and write access.

PCI Configuration Space

The following table describes the CEI-x30 PCI Configuration Space definition.

Table 59. CEI-x30 PCI Configuration Space

31	23	15	7	Offset
Device ID NNNNh (0830 for example)		Vendor ID 13C6h		00h
Status		Command		04h
Base class FFh	Sub-class 00h	Interface 00h	Revision ID 00h	08h
BIST Reserved 00h	Header type 00h	Latency Timer Reserved 00h	Cache Line Size Reserved 00h	0Ch
Base Address Register 0 for memory-mapped PCI local configuration registers (or RAR-PCIE FPGA access).				10h
Base Address Register 1 for I/O-mapped PCI local configuration registers.				14h
Base Address Register 2 PCI local bus (FPGA access).				18h
Base Address Register 3 (reserved)				1Ch
Base Address Register 4 (reserved)				20h
Base Address Register 5 (reserved)				24h
Cardbus CIS Pointer (reserved) 00000000h				28h
Subsystem ID NNNNh		Subsystem Vendor ID 13C6h		2Ch
Expansion ROM Base Address (reserved)				30h
Reserved 0x00000000h				34h
Reserved 0x00000000h				38h
MAX_LAT 00h	MIN_GNT 00h	Interrupt pin 00h	Interrupt line	3Ch

PCI Device Identifiers and Resources

The following table provides the PCI Device ID and Subsystem ID values for the CEI-x30 products, along with the BAR memory size allocations for the memory regions utilized with the enhanced firmware configuration:

Product	PCI Device ID	BAR0 (bytes)	BAR1 (bytes)	BAR2 (bytes)
CEI-430	0430h	128	0	512K
CEI-430A	430Ah	128	0	512K
CEI-830	0830h	512	256	512K
R830RX	0831h	512	256	512K
RCEI-530	0530h	512	256	512K
RAR-CPCI	0630h	512	256	512K
RAR-EC	100Ah	128	0	512K
RAR-PCIE	100Bh	512K	0	0
AMC-A30	1009h	512	256	512K

Host Memory Map

The following table summarizes the memory-mapped host interface for the CEI-x30 device firmware, described in detail in the following sections.

Table 60. CEI-x30 Host Memory Map

Byte Address	Read/Write	Device Interface Register Description
0x00000	Read/Write	Global Enable Register
0x00004	Write only	DAC Control Register
0x00008	Read/Write	Timer Register- least significant 32 bits
0x0000C	Read/Write	Timer Register - most significant 32 bits
0x00010	Write only	Update IRIG Generator Time Register
0x00014	Read only	IRIG Sample Time Register
0x00018	Read only	IRIG Sample Timestamp Register (least significant 32 bits)
0x0001C	Read only	IRIG Sample Timestamp Register (most significant 32 bits)
0x00020	Write only	SRAM Address Register
0x00024	Read/Write	SRAM Data Register
0x00028 – 0x0037	Read only	General Input Registers 1-4
0x00038	Read/Write	Interrupt Queue Register
0x0003C – 0x0007F		Unused
0x00080	Read only	Board temperature (RAR-PCIE and CEI-430A only)
0x00084	Read only	Voltage Monitor (+1.0V) (RAR-PCIE only)
0x00088	Read only	Voltage Monitor (+2.5V) (RAR-PCIE only)
0x0008C – 0x003FF		Unused
0x00400 – 0x007FF	Read only	Channel Statistics Table
0x00800 – 0x03FFF		Unused
0x04000 – 0x07FFF		Channel Register Set (64 byte sections defined as follows for each of the 256 allocated channels)
0x00	Read only	Channel Status Register
0x04	Read/Write	Channel Configuration Register 1
0x08	Read/Write	Channel Configuration Register 2
0x0C	Read/Write	Channel Configuration Register 3
0x10	Read or Write	Channel Buffer Word 1 (Read for Receive, Write for Transmit)
0x14	Read or Write	Channel Buffer Word 2
0x18	Read or Write	Channel Buffer Word 3
0x1C	Read or Write	Channel Buffer Word 4
0x20 – 0x3F		Spare
0x08000 – 0x09FFF	Read Only	Interrupt Queue (4 bytes/entry for each of 2048 entries)
0x0A000 – 0x20000		Unused
0x20000 – 0x27FFF	Read/Write	Message Scheduler Table
0x28000 – 0x3FFFF		Unused

Byte Address	Read/Write	Device Interface Register Description
0x40000 – 0x7FFFF	Read Only	Snapshot Buffer (4096 bytes for each of channels 0 - 63)

Device Interface Register Set (Common Memory)

Global Enable Register

31	30-24	23 - 16	15 - 8	7	6	5	4	3	2	1	0
Device Disable	Not Used	F/W Vers.	Board Config	IRIG Output Enable	Snap-shot Mode	Clear Int.	Interrupt Enable	IRIG Edge Received	IRIG Present	IRIG Internal Wrap	Global Enable

The fields in the Global Enable register are described as follows:

Field	Description	Values
GLOBAL ENABLE	This bit is used to enable and disable transmit and receive operation of the device. When disabled, data transfer between the transmit and receive FIFOs in SRAM are cleared. Individual channel data (de)serialization processes is disabled. This bit also enables and disables message scheduler operation.	0 = disabled (reset condition) 1 = enabled
IRIG INTERNAL WRAP	This bit is used to enable and disable internal connection of the on-board IRIG generator to the IRIG receiver.	0 = wrap disabled 1 = wrap enabled
IRIG PRESENT (read only)	This bit indicates whether or not IRIG hardware is installed.	0 = IRIG not installed 1 = IRIG installed
IRIG EDGE RECEIVED (read only)	This bit indicates an IRIG signal edge was detected on the IRIG receiver input pins. This bit is cleared when this register is read. This bit is used to help determine the appropriate IRIG receive threshold level.	0 = IRIG edge not detected 1 = IRIG edge detected
CLEAR INTERRUPT (write only)	This bit clears a pending interrupt in the firmware interface	0 = undefined 1 = clear the pending interrupt
INTERRUPT ENABLE	This bit enables generation of PCI interrupts from the label filter table trigger mechanism.	0 = PCI interrupt disabled 1 = PCI interrupt enabled
SNAPSHOT MODE	This bit defines how received messages are stored in the snapshot buffer.	0 = storage using label + SDI 1 = storage using label only
IRIG OUTPUT ENABLE (R830RX only)	This bit enables IRIG Generator signal output on P1 pins 33 and 66, P14 pins 63 and 64, when jumper shunt sets J4 and J5 are shorted	0 = IRIG Output disabled 1 = IRIG Output enabled

Field	Description	Values
BOARD CONFIGURATION (read only)	This bit field indicates the programmed configuration of the device.	7 = CEI-830 11 = R830RX 8 = CEI-430 12 = RAR-CPCI 9 = AMC-A30 13 = RAR-EC 10 = CEI-530 14 = RAR-PCIE 15 = CEI-430A
FIRMWARE VERSION (read only)	This bit field indicates the version of firmware programmed on the board	2 digit value, 4 bits/digit
DEVICE DISABLE (read only)	This bit indicates the on-board security feature has disabled the board. This bit is not supported by the CEI-830.	0 = device enabled 1 = device disabled

DAC Control Register

The DAC Control Register determines the IRIG Receiver voltage threshold and the AMC-A30 Discrete Input voltage threshold. The optimal threshold for a DC level IRIG signal is the midpoint between the upper and lower voltage levels of the IRIG signal. An appropriate level for an AM encoded IRIG signal is at the 80% point between the upper and lower voltage levels of the IRIG signal. The 80% value is acceptable for a DC signal, and should be used if the host does not know which encoding (DC or AM) is used.

For the AMC-A30, the Discrete Input voltage threshold is set when bit 8 of the DAC control register is set high, otherwise, the IRIG Receiver voltage threshold will be selected. The default Discrete Input voltage threshold level is 10.2 volts.

31 - 9	8	7 - 0
N/A	AMC-A30 DAC Select	DAC Value $\text{IRIG DAC value} = (128 + ((256/3.3) * (4.99/22.1) * V_{\text{IRIG_Threshold}}))$ Where $V_{\text{IRIG_Threshold}}$ is the value in Volts for the IRIG receive threshold relative to the input pins. $\text{Discrete DAC value} = 0.3V + (256/3.3) * V_{\text{Discrete_In_Threshold}}$
	0 = IRIG 1 = Discrete	

Timer Registers

These two registers contain the current 64-bit 1μsec device timer value. When reading these registers, the Time-Tag High Word is latched when the Time-Tag Low Word is read. When writing these registers, the value written to the Time-Tag Low Word is latched and stored and the entire 64-bit timer is updated when the Time-Tag High Word is written. For this

reason, the Time-Tag Low Word must always be read or written before the Time-Tag High Word or the combined contents will be invalid.

31 – 0
Time-Tag Low Word

The least significant 32-bits of the 64-bit timer, 1µsec resolution.

31 – 0
Time-Tag High Word

The most significant 32-bits of the 64-bit timer, resolution is approximately 71.58 minutes.

Update IRIG Generator Time Register

This register provides the means to assign the current IRIG time-of-year for the IRIG generator circuit. The format follows the standard IRIG 30-bit encoded time-of-year. The upper two bits of this register are unused. This register is only valid if the Global Enable “IRIG Installed” bit is set.

29-28	27-24	23-20	19-18	17-14	13-11	10-7	6-4	3-0
hundreds of days	tens of days	days	tens of hours	hours	tens of minutes	minutes	tens of seconds	seconds

IRIG Sample Time Register

This register contains the last received IRIG bit-encoded time value. This register is only valid if the Global Enable “IRIG Installed” bit is set.

29-28	27-24	23-20	19-18	17-14	13-11	10-7	6-4	3-0
hundreds of days	tens of days	days	tens of hours	hours	tens of minutes	minutes	tens of seconds	seconds

IRIG Sample Timestamp Registers

These two registers contain the device-referenced timer timestamp recorded when the last IRIG time one-second sample was received. The device will read the current 64-bit 1µsec device timer value when the first bit of the IRIG time is detected and store it in these registers when the Last IRIG Time register is updated. When reading these registers the Time-Tag High Word is latched when the Time-Tag Low Word is read. For this reason, the Time-Tag Low Word must always be read before the Time-Tag High Word or the combined contents will be invalid.

31 – 0
Time-Tag Low Word

The least significant 32-bits of the 64-bit time-tag, resolution is 1µsec.

31 – 0
Time-Tag High Word

The most significant 32-bits of the 64-bit time-tag, resolution is approximately 71.58 minutes.

SRAM Access Address Register

This register provides a means to access the CEI-x30 on-board SRAM through the FPGA interface. The value assigned to this register is an offset into the SRAM memory device, from 0 to 0x7FFFF, selecting the SRAM address at which a read or write access will occur based on a subsequent operation with the SRAM Access Data Register.

SRAM Access Data Register

This register defines the type of operation to perform on the SRAM location specified in the SRAM Access Address Register. A read from this register will result in a read and return the current 32-bit value in that SRAM location, where a write to this register will result in a write of the 32-bit value to the respective SRAM location. Writes to this register should be done carefully, since every SRAM location, including locations used for label filtering tables, receive and transmit buffering, and other internal functions, can be altered by this mechanism.

General Input Registers

General Input Register 1 - Discrete Inputs

For each Discrete Input on the device, a single bit is allocated in General Input Register 1. The Discrete Input bit assignments start with the LSB, b0 as Discrete Input 1, b1 as Discrete Input 2, and increment by Discrete Input Channel value through b31. Unpopulated inputs are always zero.

General Input Register 2 - Differential Inputs

For each Differential Input on the CEI-430, a single bit is allocated in General Input Register 2. The Differential Input bit assignments are b0 for

Differential Input 1, b1 for Differential Input 2, b2 for Differential Input 3, and b3 for Differential Input 4. Unpopulated inputs are always zero.

General Input Registers 2 and 3 are currently unused.

Interrupt Queue Register

The Interrupt Queue Register has different definitions based on the access method used. When written, the Interrupt Queue Register is used to generate a single host-defined entry in the Interrupt Queue, having a value of 255. When read, the Interrupt Queue Register provides the current Interrupt Queue Head Pointer, pointing to the most recent interrupt entry in the queue and defined as follows:

31 – 1	10 – 0
Unused	Current offset into the Interrupt Queue for the most recent entry, with a valid range of 0 – 2047.

Channel Statistics Table

The Channel Statistics Table contains one entry for each of the 256 channels allocated on the device. Each entry contains a 32-bit counter whose definition is based on whether the channel is a receive or transmit channel. For a receive channel, the respective entry indicates the number of messages received on that channel since the board was last initialized. For a transmit channel, the respective entry indicates the number of messages transmitted on that channel since the board was last initialized.

Channel Register Set

The Channel Register Set contains the status, configuration, and buffer access registers for all 256 channels allocated on the device. Each channel register sub-set is defined as shown below with the respective offset:

Channel Status Register	0x00
Channel Configuration Register 1	0x04
Channel Configuration Register 2	0x08
Channel Configuration Register 3	0x0C
Channel Buffer Word 1	0x10
Channel Buffer Word 2	0x14
Channel Buffer Word 3	0x18
Channel Buffer Word 4	0x1C

These components of the Channel Register Set are defined in the next several paragraphs of this chapter.

Channel Status Register

The Channel Status Register has a common bit field definition across all channels, but other than the Channel Type bit field, its use only applies to those channels defined for protocol processing. The Channel Status Register is defined as follows:

31 - 16	15 - 8	7 - 2	1	0
Buffer Fill Level	Channel Type	Unused	Message Error	Buffer Status

The fields in the Channel Status Register are described as follows:

Field	Description	Values
BUFFER FILL LEVEL (read only)	For a receive channel, this field indicates the number of unread messages currently in the receive buffer. For a transmit channel, this field indicates the number of "untransmitted" messages residing in the transmit buffer.	0 to 2047
CHANNEL TYPE (read only)	This field indicates the channel type assigned to this Channel Register Set.	0 = unassigned channel 1 = Merged Mode Receiver 2 = ARINC 429 Receiver 3 = ARINC 429 Transmitter 4 = ARINC 717 Receiver 5 = ARINC 717 Transmitter 6 = ARINC 561 Receiver 7 = ARINC 561 Transmitter 8 = Avionics Discrete Input 9 = Avionics Discrete Output 10 = Digital Input 11 = Digital Output 12 = Differential Input 13 = Differential Output 14 = Serial Receiver 15 = Serial Transmitter 16 to 255 = Undefined
MESSAGE ERROR (read/write)	This bit is set to "1" when an ARINC 429 message length error is detected on the respective receive channel. This bit is cleared when a "1" is written to it by the host.	0 = no message error detected 1 = a message bit length error was detected since this bit was last cleared

Field	Description	Values
BUFFER STATUS (read only)	For a receive channel, the Buffer Status bit indicates the availability of unread messages. For a transmit channel, the Buffer Status bit indicates the fill-state of the buffer.	Receive Channel: 0 = no messages in the buffer 1 = an unread message is available in the buffer Transmit Channel: 0 = buffer is not full 1 = buffer is full

Channel Configuration Registers

The definition of the Channel Configuration Registers is dependent on the associated channel type, and their use only applies to those channels defined for protocol processing. The specific Channel Configuration Register types are described in the following paragraphs, categorized by channel type, as well as receive and transmit usage.

Channel Configuration Register 1 – ARINC 429 Receive

Channel Configuration Register 1 is defined for use as a Receive Channel Configuration Register for the ARINC 429 protocol as follows:

31-28	27 - 16	15	14	13-3	2	1	0
Not Used	Baud Rate	Channel Enable	Merge Mode Enable	Not Used	Internal Wrap Enable	Parity Enable	Not Used

The bit fields in the ARINC 429 Receive Channel Configuration Register are described as follows:

Field	Description	Values
PARITY ENABLE	This bit is used to enable and disable parity checking on the received ARINC 429 data.	0 = disabled (reset condition) 1 = enabled
INTERNAL WRAP ENABLE	This bit is used to enable and disable the ability to wrap transmitted data internal to the device. Each receive channel is internally connected to the respective transmitter channel.	0 = disabled (reset condition) 1 = enabled

Field	Description	Values
MERGE MODE ENABLE	This bit controls the path for storage of received data on the respective channel. When disabled, all received data is stored in the individual receive FIFO for the respective channel. Then enabled, all received data is stored in the merged receive FIFO, accessed via channel 0.	0 = disabled (reset condition) 1 = enabled
CHANNEL ENABLE	This bit controls the receiver's conversion of incoming ARINC 429 data for transfer to the respective FIFO buffer. When disabled, no data reception operations are performed. When enabled, data reception operations are performed and data is processed for storage in the receiver's FIFO buffer.	0 = disabled (reset condition) 1 = enabled
BAUD RATE	This bit field controls the baud rate for the ARINC 429 protocol. The value of this field is used as a divisor for the 16MHz clock reference.	Baud Rate = $16,000,000 / (N+2)$

Channel Configuration Register 1 – ARINC 573 Receive

Channel Configuration Register 1 is defined for use as a Receive Channel Configuration Register for the ARINC 573/717 protocol as follows:

31-16	15	14	13	12	11 - 8	7-3	2	1-0
Not Used	Channel Enable	Not Used	ARINC 717 Raw Mode Enable	ARINC 717 Encoding	ARINC 717 Rate & Size	Not Used	Internal Wrap Enable	Not Used

The bit fields in the ARINC 573/717 Receive Channel Configuration Register are described as follows:

Field	Description	Values
INTERNAL WRAP ENABLE	This bit is used to enable and disable the ability to wrap transmitted data internal to the device. Each receive channel is internally connected to the respective transmitter channel.	0 = disabled (reset condition) 1 = enabled

Field	Description	Values		
ARINC 717 RATE & SIZE	This bit field selects the ARINC 573/717 baud rate and respective sub-frame size for use when Auto-Sync reception is enabled.	Value	Speed (bps)	Sub-frame Size (words)
		0x00	384	32
		0x01	768	64
		0x02	1536	128
		0x03	3072	256
		0x04	6144	512
		0x05	12288	1024
		0x06	24576	2048
		0x07	49152	4096
ARINC 717 ENCODING	This bit selects the ARINC 573/717 encoding supported by the receiver. When internal wrap is enabled, the value of this bit has no effect on data reception.	0 = Harvard Bi-Phase (HBP) 1 = Bi-Polar Return-to-Zero (BPRZ)		
ARINC 717 RAW MODE ENABLE	This bit disables the receiver automatic frame detection logic on the incoming ARINC 573/717 frame data and enables "Raw Mode". When disabled, the receiver will automatically synchronize to the incoming frame using the sub-frame size selection and the sync words programmed in the configuration words 1 through 4 for the respective channel. When Raw Mode is enabled, frame data will be logged to the receive buffer beginning with the first bit encountered from the ARINC 717 receiver.	0 = disabled (reset condition) 1 = Raw Mode enabled		
MERGE MODE ENABLE	This bit controls the path for storage of received data on the respective channel. When disabled, all received data is stored in the individual receive FIFO for the respective channel. When enabled, all received data is stored in the merged receive FIFO, accessed via channel 0.	0 = disabled (reset condition) 1 = enabled		

Field	Description	Values
CHANNEL ENABLE	This bit controls the receiver's conversion of incoming ARINC 429 data for transfer to the respective FIFO buffer. When disabled, no data reception operations are performed. When enabled, data reception operations are performed and data is processed for storage in the receiver's FIFO buffer.	0 = disabled (reset condition) 1 = enabled
BAUD RATE	This bit field controls the baud rate for the ARINC 429 protocol. The value of this field is used as a divisor for the 16MHz clock reference.	Baud Rate = $16,000,000 / (N+2)$

Channel Configuration Register 2 – ARINC 573 Receive

31 - 28	27 - 16	15 - 12	11 - 0
Not used	Sync Word 2	Not used	Sync Word 1

Channel Configuration Register 3 – ARINC 573 Receive

31 - 28	27 - 16	15 - 12	11 - 0
Not used	Sync Word 4	Not used	Sync Word 3

When the Channel Register Set is assigned to an ARINC 573/717 Receive Channel, Channel Configuration Registers 2 and 3 specify the respective ARINC 573/717 sub-frame sync words 1-4. These registers support definition of the four 12-bit sub-frame sync words and are only used when ARINC 717 Auto Sync is enabled. All four fields must be defined for the Auto Sync feature to function.

Channel Configuration Register 1 – ARINC 429 Transmit

Channel Configuration Register 1 is defined for use as a Transmit Channel Configuration Register for the ARINC 429 protocol as follows:

31-28	27 - 16	15	14 - 8	7	6	5	4	3	2	1	0
Not Used	Baud Rate	Channel Enable	Unused	Parametric Mode	Transmit Disable	Gap Error	Bit Count High	Bit Count Low	Even Parity	Parity Enable	Slew Rate

The bit fields in the ARINC 429 Transmit Channel Configuration Register are described as follows:

Field	Description	Values
SLEW RATE	This bit is used to select the slew rate utilized with the ARINC 429 protocol.	0 = Slow Slew Rate (10 μ sec) 1 = Fast Slew Rate (1.5 μ sec)
PARITY ENABLE	This bit is used to enable and disable parity application to the transmitted ARINC 429 message.	0 = disabled (reset condition) 1 = enabled
EVEN PARITY	This bit is used to select between even and odd parity when ARINC 429 message parity is under device control. When enabled, even parity is applied, when disabled, odd parity is applied.	0 = odd parity (reset condition) 1 = even parity
BIT COUNT LOW ¹	This bit is used to enable a low bit count error for ARINC 429 messages, resulting in the transmission of 31 bits instead of 32.	0 = disabled (reset condition) 1 = enabled
BIT COUNT HIGH ¹	This bit is used to enable a high bit count error for ARINC 429 messages, resulting in the transmission of 33 bits instead of 32.	0 = disabled (reset condition) 1 = enabled
GAP ERROR ¹	This bit is used to induce a message gap error between ARINC 429 message transmissions, resulting in a two-bit gap instead of the standard four-bit gap.	0 = disabled (reset condition) 1 = enabled

¹ This function is only supported by channels with a CHANNEL TYPE indicating *ARINC 429 Transmitter* with PARAMETRIC MODE enabled.

Field	Description	Values
TRANSMIT DISABLE	This bit is used to disable external transmission to the respective ARINC 429 transmitter. This provides for the ability to transmit internally to the respective receiver with internal wrap enabled, without exposing the communications to a connected device. For products that have the ability to tri-state the transmit output, setting this bit will cause the output to be tri-stated. For all other products, setting this bit will cause the output to remain a "null".	0 = disabled (reset condition) 1 = enabled
PARAMETRIC MODE	This bit is used to enable parametric operation on the respective transmit channel DAC.	0 = disabled (reset condition) 1 = enabled
CHANNEL ENABLE	For ARINC 429 transmitters, this bit controls the conversion of data passed from the transmitter FIFO for transmission on the bus. When disabled, no data transmission operations are performed. When enabled, data transmission operations are performed using data provided from the specified transmitter's FIFO buffer.	0 = disabled (reset condition) 1 = enabled
BAUD RATE	This bit field controls the baud rate for the ARINC 429 protocol. The value of this field is used as a divisor for the 16MHz clock reference.	Baud Rate = $16,000,000 / (N+2)$

Channel Configuration Register 1 – ARINC 573 Transmit

Channel Configuration Register 1 is defined for use as a Transmit Channel Configuration Register for the ARINC 573/717 protocol as follows:

31-16	15	14	13	12	11 - 8	7	6	5-1	0
Not Used	Channel Enable	Unused	ARINC 717 HBP Encoding	ARINC 717 BPRZ Encoding	ARINC 717 Baud Rate	Not used	Transmit Disable	Not used	Slew Rate

The bit fields in the ARINC 573/717 Transmit Channel Configuration Register are described as follows:

Field	Description	Values
SLEW RATE	This bit is used to select the slew rate utilized with the ARINC 429 protocol.	0 = Slow Slew Rate (10 μ sec) 1 = Fast Slew Rate (1.5 μ sec)
TRANSMIT DISABLE	This bit is used to disable external transmission to the respective ARINC 573 transmitter (either HBP or BPRZ). This provides for the ability to transmit internally to the respective receiver with internal wrap enabled, without exposing the communications to a connected device. For products that have the ability to tri-state the transmit output, setting this bit will cause the output to be tri-stated. For all other products, setting this bit will cause the output to remain a "null".	0 = disabled (reset condition) 1 = enabled
ARINC 717 BAUD RATE	This bit field selects the ARINC 573/717 baud rate.	0 = 384 bps 1 = 768 bps 2 = 1536 bps 3 = 3072 bps 4 = 6144 bps 5 = 12288 bps 6 = 24576 bps 7 = 49152 bps
ARINC 717 BPRZ ENCODING	This bit enables the ARINC 573/717 Bi-Polar Return-to-Zero (BPRZ) encoding transmitter.	0 = disabled (reset condition) 1 = enabled
ARINC 717 HBP ENCODING	This bit enables the ARINC 573/717 Harvard Bi-Phase (HBP) encoding transmitter.	0 = disabled (reset condition) 1 = enabled

Field	Description	Values
CHANNEL ENABLE	For ARINC 573 transmitters, this bit controls the conversion of data passed from the transmitter FIFO for transmission on the bus. When disabled, no data transmission operations are performed. When enabled, data transmission operations are performed using data provided from the specified transmitter's FIFO buffer.	0 = disabled (reset condition) 1 = enabled

Channel Configuration Register 1 – Discrete or Digital Output

The definition of the Discrete Output State is discussed in the section, “Avionics Discrete I/O”.

31 – 1	0
Unused	Discrete Output State

Channel Configuration Register 1 – Differential Output

The definition of the Differential Output State is discussed in the section, “Differential Discrete I/O”.

31 – 2	1	0
Unused	Differential Output Enable 0 = tri-state 1 = enabled	Differential Output State

Channel Buffer Words

The definition of the Channel Buffer Words is dependent on the associated channel type. Use of the different Channel Buffer Word sets is described in the following paragraphs, categorized by receive, transmit, and I/O usage.

General Buffer FIFO Operations

Protocol-based Receive and Transmit Channel Configuration Registers control the processing related to the corresponding FIFO buffer for message storage, accessed via Channel Buffer Words. FIFO operations are completely independent of either of the Global Enable or Individual Channel Enable states. Host access to FIFO buffers and FIFO head/tail

processing is enabled immediately following initialization of the board. Regardless of the state of the “enable” bits, the host can read available data from and write data to, any applicable FIFO buffer.

Channel Buffer Word 1 - Receive

31 – 0
Time-Tag Low Word

When the Channel Status Register indicates a receiver FIFO is not empty, Channel Buffer Word 1 contains the least significant 32 bits of the 64-bit time-tag. The resolution of this time-tag low word is 1µsec.

Channel Buffer Word 2 - Receive

31 – 0
Time-Tag High Word

When the Channel Status Register indicates the FIFO is not empty, Channel Buffer Word 2 contains the most significant 32 bits of the 64-bit time-tag. The resolution of this time-tag high word is approximately 71.58 minutes.

Channel Buffer Word 3 – Receive

31 – 8	7 – 0
Not used (all zeros)	Channel

When the Channel Status Register indicates the FIFO is not empty, Channel Buffer Word 3 contains the channel on which this data was received. This information is useful when the Merged Mode channel is enabled and utilized for the respective receive channel.

Channel Buffer Word 4 – Receive

31 – 0
Data

When the Channel Status Register indicates the FIFO is not empty, Channel Buffer Word 4 contains the data that was received.

Note: When the host reads the most significant byte of the long word 3, the receiver FIFO for the selected channel is incremented to the next entry. This could result in a FIFO Not Empty reset in the respective Receive Status register if the FIFO becomes empty. This implies that Receive Buffer Word 3 must be the last register accessed when reading a FIFO buffer entry. This also implies that it is NOT necessary to read all four Receive Buffer Word registers from the FIFO if the data is not needed.

The format for the data retrieved from Receive Buffer Word 4 is dependent on the protocol assigned to the respective channel. The protocol data format is described as follows:

ARINC 429/575 Data Format

31	30 – 10	9 - 8	7 - 0
Parity Indication or ARINC Data MSB	ARINC Data	SDI bits or ARINC Data bits 0-1	ARINC Label (MSB – LSB)

If the Parity Enable bit is set to one in the respective Receive Configuration register, the Parity Indication is set by the device to indicate the parity of the message. A Parity Indication bit value of zero indicates this message was received with odd parity, where a Parity Indication value of one indicates the message was received with even parity. If the Receive Configuration register Parity Enable bit is set to zero, this bit is not manipulated by the device; instead, it reflects the value of bit 31 as transmitted from the ARINC 429 source.

ARINC 573/717 Data Format

31 – 16	15	14	13 - 12	11 – 0
Unused	Sync Indication	Unused	Sub-frame Identification	Data Word

When the Sync Indication bit is set to one, it is an indication this word was detected as a sync word. A Sync Indication bit value of zero indicates the message was treated as a data word. The Sub-frame Identification bit field identifies the sub-frame assignment for this word; where a value of one indicates sub-frame 1, two indicates sub-frame 2, three indicates sub-frame 3, and zero indicates sub-frame 4.

Channel Buffer Word 1, 2, and 3 - Transmit

Channel Buffer Words 1, 2, and 3 are not currently used for transmit operations.

Channel Buffer Word 4 - Transmit

31 – 0
Data

When the Channel Status Register indicates the FIFO is not full, Channel Buffer Word 4 can be defined for transmission on the respective channel.

Note: When the host writes the most significant byte of the long word 3, the transmitter FIFO for the selected channel is incremented to the next entry. This could result in a FIFO Not Full reset in the respective Transmit Status register if the FIFO becomes full. This implies that Transmit Buffer Word 3 must be the last register accessed when writing a transmit FIFO buffer entry. This also implies that it is NOT necessary to write all four Transmit Buffer Word registers.

The format for the data written to Channel Buffer Word 4 is dependent on the protocol assigned to the respective channel. The protocol data format is described as follows:

ARINC 429/575 Data Format

31	30 – 10	9 – 8	7 – 0
Parity Bit or ARINC Data MSB	ARINC Data	SDI bits or ARINC Data bits 0-1	ARINC Label (MSB – LSB)

If the Parity Enable bit is set to one in the respective Transmit Configuration register, bit 31 of the 32-bit data word will be overwritten by the device. The value assigned to bit 31 is based on the parity type and bit content of the remaining 31 bits. If the Parity Enable bit is set to zero or the ARINC 561 protocol is in use, bit 31 remains unchanged from the value provided.

ARINC 573/717 Data Format

31 – 12	11 – 0
Unused	Data Word

ARINC 573/717 data is assigned to the lower 12 bits of the 32-bit word.

Interrupt Queue

The CEI-x30 Interrupt Queue contains 2048 32-bit entries, each indicating the source of an interrupt trigger. The most recent Interrupt Queue entry updated by the CEI-x30 device is indicated via the Interrupt Queue Head Pointer, accessed by reading the Interrupt Queue Register.

Valid Interrupt Queue entries are defined as follows:

31 – 8	7 - 0
Unused	Interrupt Queue Entry Definition: Values in the ranges 0–63 and 128–254 are unused 64–127 = Interrupt Source/Receive Channel Number 0 to 63, (offset by 64) 255 = Host generated interrupt via write access to the Interrupt Queue Register.

Message Scheduler Table

The CEI-x30 Message Scheduler feature is programmed via the Message Scheduler Table, a table of 1024 entries each consisting of eight 32-bit elements. As a part of the initialization of the device, the message scheduler table should be initialized to a known state. This is required before the message scheduler is enabled; otherwise, inadvertent message transmission may result. The GLOBAL ENABLE bit of the Global Enable Register turns message scheduling on and off; however, it has no affect on the contents of the message schedule table.

The Message Scheduler will query each table entry on a one millisecond basis, checking for all messages required for transmission at that particular millisecond value. The entire table is processed each millisecond, with the lowest table entry being processed first and highest table entry last. The scheduler can only be enabled or disabled at the beginning of a one millisecond epoch (this means that if the scheduler is disabled in the middle of creating scheduled traffic, all of the ARINC words for that millisecond will get loaded into the various transmit buffers before the scheduler goes idle).

The elements of each Message Table entry are defined as follows:

Element	Element	Description	Host Access Limitations
0	MESSAGE RATE	This element defines the periodic message transmission rate in milliseconds, with the following valid values: 0: entry is disabled/unused 1: transmission every 1 millisecond 0xFFFFFFFF: transmission every 2^{32} milliseconds	None
1	CHANNEL	This element must contain a valid ARINC 429 transmit channel number.	Writable only when entry is disabled.

Element	Element	Description	Host Access Limitations
2	NUMBER OF MESSAGES TO TRANSMIT	This element defines the number of times to transmit this periodic message, with the following valid values: 0x00000000: message disabled, still processed 0x00000001: transmit one word 0xFFFFFFFF: transmit (2 ³²)-1 words 0xFFFFFFFF: unlimited continuous transmission	Writable only when entry is disabled.
3	OFFSET	This element is utilized in two ways. When started, the Message Scheduler assigns this element to the sum of the previous value in this element and the contents of this element to the MESSAGE RATE value. It then decrements the value in this element every millisecond, using it as a counter to track the periodic transmission schedule of this message. Every time this offset value reaches zero, a transmit word is placed in the respective buffer and this value is reset to the MESSAGE RATE value minus one. For the application this element can be used as an initial offset, providing a mechanism to incorporate a delayed transmission of one or more words. This is useful for avoiding message rate skew.	Writable only when entry is disabled.
4, 5, 6	Not Used		
7	DATA	This element defines the periodic message content to be transmitted.	None

Snapshot Buffer

The Snapshot Buffer feature provides the ability to store the latest ARINC 429 message data received on a channel for subsequent recall by the host application. It is implemented as a table of 256 entries/channel, (one entry for each valid ARINC 429 label value), allocated for each of the first sixty-four channels on the board. Each channel/label combination is allotted four 32-bit locations, for a total of 4K words/channel.

Application definition of the Snapshot Storage Mode should be based on the desired use of the Snapshot Buffer feature. Received ARINC 429 messages can be stored in each Snapshot Buffer entry based on the message label value or in one of four separate locations within the entry based on the combined values of the message label and SDI fields. The Snapshot Storage Mode is defined via the SNAPSHOT MODE bit in the Global Enable Register.

A SNAPSHOT MODE bit value of 0 configures snapshot storage based on the message label field value only, while a value of 1 configures snapshot storage based on the combined value of the message label and SDI fields.

Entries in the Snapshot Buffer are accessed using a combination of channel value, ARINC 429 label value, and SDI field value. Combinations of these field values range from channel 0 to 63, octal label value 000 to 377, and SDI field value from 0 to 3. These latter two fields are identified in the ARINC 429 word, with all three used in combination to calculate an offset from the Snapshot Buffer base address as follows:

31 - 16	15 - 10	9 - 2	1 - 0
0	Channel Number (0 - 63)	ARINC 429 Label Bits 7:0 of the ARINC Word	<i>Snapshot Mode = 0:</i> Use SDI Field Value as the offset to message data within this entry <i>Snapshot Mode = 1:</i> Location 1 = 64-bit time-tag lsw Location 2 = 64-bit time-tag msw Location 3 = reserved Location 4 = message data

SRAM Memory Organization

The CEI-x30 device uses several features mapped to the on-board SRAM device, some of which are accessible through the use of the SRAM Address and Data Registers. The memory map and descriptions for the portions of this external memory are defined in the following three paragraphs.

Lword Offset	Read/Write	Device Interface Register Description
0x00000 – 0x0FFFF	Read/Write	Label Filter Table
0x10000 – 0x1FFFF	Read/Write	Snapshot Buffer, (also mapped to Common Memory)
0x20000 – 0x7FFFF	No access	Individual Channel Buffers, allocated as follows:
0x20000 – x2FFFF	No access	Transmit Channel Buffers
0x30000 – x3FFFF	No access	Merged Receive Buffer
0x40000 – x7FFFF	No access	Receive Channel Buffers

Label Filter Table

The Label Filter Table provides a method to program buffer filtering and hardware interrupt generation based on a field-match trigger using a combination of the ARINC 429 message label, SDI, and ESSM field values. This table contains an entry for each combination of ARINC 429 label, SDI, and ESSM values, programmed as eight 4-bit ESSM fields (0 to 7) in individual 32-bit locations accessed via combined label (octal 000 to

377) and SDI (0 to 3) values. The content of each Label Filter Table entry is defined as follows:

3	2	1	0
Unused	Interrupt Enable 0: disabled 1: enabled	Snapshot Filter Enable 0 : unfiltered 1 : filtered	FIFO Filter Enable 0 : unfiltered 1 : filtered

When the Snapshot or FIFO Filter enable bit is zero for a specified label/SDI/ESSM combination, any message received containing that combined field value is recorded in the respective buffer. When the respective filter enable bit is set to one for a specified label/SDI/ESSM combination, any message received containing that combined fields value is not recorded in the respective buffer. The FIFO Filter Enable bit enables/disables filtering to either the regular or merged receive buffer, depending on how this channel is programmed.

When the respective Interrupt Enable bit is set to one for a specified label/SDI/ESSM combination, any message received containing that combined field value will trigger an entry in the Interrupt Queue. If the INTERRUPT ENABLE bit is set to Enabled in the Global Enable Register, a PCI interrupt is also generated.

Snapshot Buffer

The Snapshot Buffer is mapped to the Device Interface Register Set, located in common memory, described in a previous section of this document.

Individual Channel Buffers

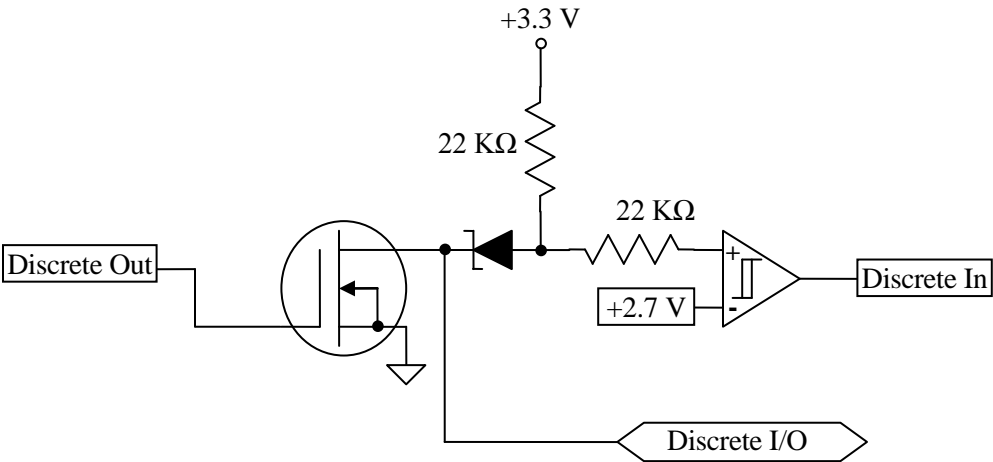
The Individual Channel Buffers provide storage for all transmit and receive FIFO buffers. All host interaction surrounding the use of the FIFO buffers is described in the respective Channel Buffer Word paragraphs, with the underlying memory inaccessible to the host interface.

ARINC 429 Receive Threshold

The ARINC 429 Receive Threshold is configured at the factory to be nominally +/- 3.0 volts, and is not user selectable.

Avionics Discrete I/O

The CEI-430 provides up to sixteen bi-directional individually configurable Avionics Discrete channels, while the CEI-830 optionally provides up to four Avionics Discrete channels. Avionics Discrete I/O is used for general avionics-level I/O interfacing. The discrete outputs are low side n-channel FET switches capable of sinking 500mA, while the inputs are single ended, protected (50V max), with a logic threshold of approximately 2.7 V. The basic circuit for a discrete I/O channel is shown below:



The discrete output channels have the following truth-table functionality:

Discrete Out	Discrete I/O pin
1	FET ON [conduct to Ground]
0	FET OFF [tri-state]

When disconnected from any external signal, Discrete In reflects the value of Discrete Out. When the FET is on, reading Discrete In should return a “0”. When the FET is off, reading Discrete In generally returns a “1” (because of the weak 22 KΩ pullup resistor) but the load attached to the discrete I/O pin must also be taken into consideration.

The discrete input channels have the following truth-table functionality:

Discrete I/O pin	Discrete In
> 2.7 VDC	1
< 2.7 VDC	0

Differential Discrete I/O

The CEI-430 provides up to four individually configurable RS-485-level differential input/output channels. RS-485 differential channels are

suitable for discrete signaling requiring long cable runs or when requiring true differential signaling.

The Differential I/O channel receive data always reflects the state of the channel's I/O pair. The receive truth table is listed below.

Differential I/O	Differential Input Data
DIFF+ \geq DIFF- by 300mV	1
DIFF+ \leq DIFF- by 300mV	0
DIFF+ and DIFF- shorted	1
DIFF+ and DIFF- floating	1

The Differential Discrete transmitter section requires the Differential Transmit Enable bit to be set in order for the transmitter to function.

Differential Enable	Differential Transmit	Differential I/O
0	X	Hi-Z
1	1	DIFF+ > DIFF-
1	0	DIFF+ < DIFF-

Hardware Channel Assignments

While the hardware channel assignments may change with firmware revisions, the API handles the actual channel indexing internally. When using the AR_Get_Data and AR_Get_Data_XT API routines to access board-level receive channel buffers, the following channel index values apply:

Channel Index Value	Channel Assignment
0 through 31	ARINC 429 Receive Channels
32	ARINC 717 Receive Channel
63	Merged Mode Receive Buffer
64 through 95	ARINC 429 Transmit Channels
96	ARINC 717 Transmit Channel
128 through 191	Discrete Input Channels
192 through 207	Discrete Output Channels