User's Manual

P-708

abaco SYSTEMS

Copyrights

P-708 User's Manual (1500-060)

Software Revision: 3.30
Document Revision: 3.30
Document Date: 16 September 2019

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

avionics.support@abaco.com (email)
https://www.abaco.com/products/avionics

Additional Resources

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

www.abaco.com

Contents and Tables

Contents

Figures

Tables

CHAPTER 1

Introduction

Overview

The ARINC 708 PMC product line consists of the RoHS compliant RP-708 and legacy P-708/P-708-C two-channel ARINC 708 Weather Radar Display Data Bus interface products, supporting variable frame size and definition using standard 1MHz Manchester II Bi-Phase encoding. Each module is supported standalone, or adapted to the PCIe, PCI and cPCI busses through the appropriate PMC carrier. Conductive-cooling and rugged configurations are available with the RP-708 product, having I/O on the PMC P14 connector.

ARINC 708 PMC Specifications

All ARINC 708 PMC products are built to the PMC draft standard IEEE-P1386.1. Additionally, the P-708-C and RP-708 conform to Conductive Cooled PMC Standard ANSI/VITA 20-2001 (R2005).

PMC/PCI Interface

- Standard single width CMC module per IEEE P1386 draft standard.
- +5V and +3.3V PCI bus signaling compatibility and universal keying.
- 66 MHz, 32 bit PCI bus operation.
- Shared Memory consisting of 2 megabytes of SRAM allocated for ARINC 708 frame Data.

Typical Power Consumption

Table 1. Power Consumption

Board	+3.3V	+5V
P-708/P-708-C	325mA	275 mA
RP-708	-	784 mA

The RP-708 power was measured at ambient with both channels transmitting at 97% bus duty cycle into 78 ohm loads and the host accessing Frame Data buffer. The 5V supply was measured at 5.12V so the total power usage was 4.0W. Externally, each load was dissipating 1W. Thus, 2W were being dissipated on-board.

Calculated Mean Time between Failure (MTBF)

The reliability calculation model for the P-708/P-708-C was MIL-HDBK-217F and MIL-HDBK-217FN2 was used for the RP-708.

Table 2. Mean Time between Failure (MTBF)

Board	Ambient Temperature	Operating Environment	MTBF
P-708/P-708-C	+25° C	ground benign	582,000 hrs.
RP-708	+25° C	ground benign, controlled	2,220,766 hrs.

Operating Temperature

-40° to +85° C.

Weight

Table 3. Weight

Board	Weight
P-708	3.1 ounces
P-708-C	1.9 ounces
RP-708	1.8 ounces (excluding front I/O connector)

PCI Memory Map

Table 4 documents the PCI memory map interface definition for the ARINC 708 products.

Table 4. PCI Memory Map

Region	Type	P-708-C P-708 Size	RP-708 Size	Description
Configuration	Configuration	64 bytes	64 bytes	PCI configuration space
PCI BAR0	Memory	128 bytes	512 bytes	PCI interface local configuration registers
PCI BAR1	I/O	0	256 bytes	not used
PCI BAR2	Memory	4 Mbytes	4 Mbytes	ARINC 708 host interface
PCI BAR3	n/a	0	0	not used
PCI BAR4	n/a	0	0	not used
PCI BAR5	n/a	0	0	not used

Board Layout

The following illustrations show the component layout of the P-708, P-708-C, and RP-708 products in standalone configurations.

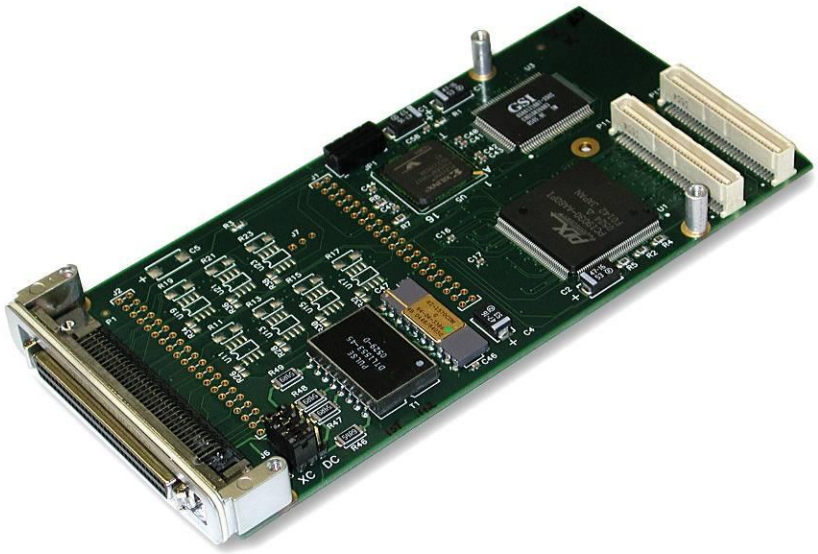


Figure 1. P-708

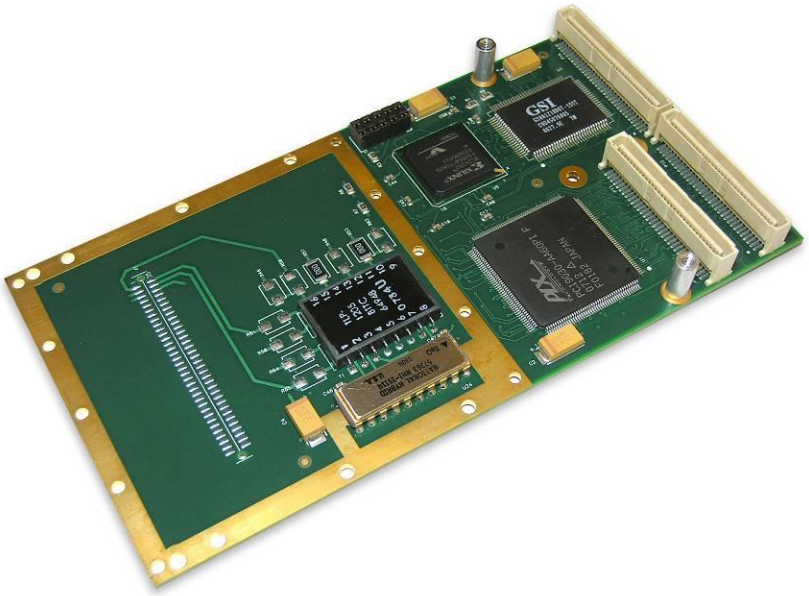


Figure 2. P-708-C

Figure 3. RP-708

I/0 Connections

Input/Output Connectors

The RP-708 supports I/O via the front panel or rear I/O via the PMC connector P14. The P-708 only supports front I/O while the P-708-C only supports rear I/O.

For the front I/O ARINC 708 boards RP-708 and P-708, the mating connector described in Table 5 is compatible with the front panel 68-pin SCSI connector. The adapter cable, CONPMC-708 for the P-708 or RCONPMC-708 for the RP-708 is supplied to use with this connection (see Figure 5).

The P-708-C rear I/O is routed through the PMC connector P14, with the pin-out described in Table 8. The RP-708 rear I/O pin-out is described in Table 9.

Table 5. Input/Output Connector

Part No.	Description	Manufacturer
750913-7 or 1-750913-7	Front Panel 68 pin SCSI-3	AMP/Tyco

Front I/O Connector Pin-out

The RP-708 68-pin front panel connector pin-out definition is described in Table 7. The P-708 68-pin front panel connector pin-out definition is described in Table 6. Figure 4 shows the view facing the receptacles of the 68-pin Front-Panel Receptacle Connector (SCSI-3-compatible with Rails and Latch Blocks).

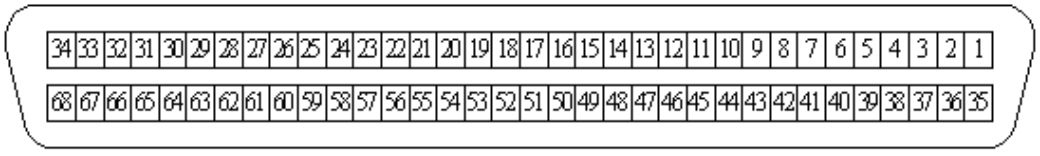


Figure 4. 68-pin Front-Panel Receptacle Connector

Table 6. P-708 Front I/O Connections

Signal	Pin	Signal	Pin
Channel 0 +	1	Channel 0 -	35
Channel 0 Shield *	2	Channel 1 Shield *	36
Channel 1 +	3	Channel 1 -	37
Ground **	34	Ground **	68
All other pins are designated as Not Used			

Notes:

* Channel 0 Shield and Channel 1 Shield are tied to their respective transformer secondary center taps.

** Ground can be used as a conductor shield if desired.

Table 7. RP-708 Front I/O Connections

Signal	Pin	Signal	Pin
Channel 0 +	1	Channel 0 -	35
Chassis *	2	Chassis *	36
Channel 1 +	3	Channel 1 -	37
Reserved	4	Reserved	38
Reserved	5	Reserved	39
Reserved	6	Reserved	40
Reserved	7	Reserved	41
Reserved	8	Reserved	42
Reserved	9	Reserved	43
Reserved	10	Reserved	44
Reserved	11	Reserved	45
Reserved	12	Reserved	46
Reserved	13	Reserved	47
Reserved	14	Reserved	48
Reserved	15	Reserved	49
Reserved	16	Not used	50
Not used	17	Not used	51
Not used	18	Not used	52
Not used	19	Not used	53
Not used	20	Not used	54
Reserved	21	Reserved	55
Reserved	22	Reserved	56
Reserved	23	Reserved	57
Reserved	24	Reserved	58
Reserved	25	Reserved	59
Reserved	26	Reserved	60
Reserved	27	Reserved	61
Reserved	28	Reserved	62
Reserved	29	Reserved	63

Signal	Pin	Signal	Pin
Reserved	30	Reserved	64
Reserved	31	Reserved	65
Reserved	32	Reserved	66
Reserved	33	Reserved	67
Ground **	34	Ground **	68

Notes:

* Chassis is tied to the mounting features of the card including the front panel and is isolated from Ground. Chassis is the preferred shield connection for Channel 0 and Channel 1 cables. Note that unlike the P-708, the transformer secondary center taps are left floating.

** Ground can be used as a conductor shield if desired.

Table 8. P-708-C P14 Rear I/O Connections

Signal	Pin	Signal	Pin
Channel 0 +	1	Channel 0 -	2
Channel 0 Shield	3	Channel 1 Shield	4
Channel 1 +	5	Channel 1 -	6
All other pins are designated as Not Used			

Table 9. RP-708 P14 Rear I/O Connections

Signal	Pin	Signal	Pin
Channel 0 +	1	Channel 0 -	2
Chassis *	3	Chassis *	4
Channel 1 +	5	Channel 1 -	6
Reserved	7	Reserved	8
Reserved	9	Reserved	10
Reserved	11	Reserved	12
Not used	13	Not used	14
Reserved	15	Reserved	16
Reserved	17	Reserved	18
Reserved	19	Reserved	20
Reserved	21	Reserved	22
Reserved	23	Reserved	24
Reserved	25	Reserved	26
Reserved	27	Reserved	28
Reserved	29	Reserved	30
Reserved	31	Reserved	32
Reserved	33	Reserved	34
Reserved	35	Reserved	36
Reserved	37	Reserved	38

Signal	Pin	Signal	Pin
Reserved	39	Reserved	40
Not used	41	Not used	42
Reserved	43	Not used	44
Not used	45	Not used	46
Reserved	47	Reserved	48
Reserved	49	Reserved	50
Reserved	51	Reserved	52
Reserved	53	Reserved	54
Reserved	55	Reserved	56
Not used	57	Not used	58
Reserved	59	Not used	60
Not used	61	Not used	62
Not used	63	Not used	64

Note: * Chassis is tied to the mounting features of the card including the primary and secondary thermal interfaces and is isolated from Ground. Chassis is the preferred shield connection for Channel 0 and Channel 1. Note that unlike the P-708, the transformer secondary center taps are left floating.

Adapter Cable

Figure 5 shows the CONPMC-708 Adapter Cable provided with the P-708 product, regardless if it is purchased standalone or adapted to a PCIe, PCI, or cPCI form-factor via the respective PMC Carrier. A RoHS compliant RCONPMC-708 is supplied with the front I/O version of the RP-708. Within this document, references to CONPMC-708 also refer to the RCONPMC-708. The twin-ax adapters are identified as *J1 708 CH 1* for the channel referenced as 0 via the API and pin-out tables, and *J2 708 CH 2* for the channel referenced as 1.

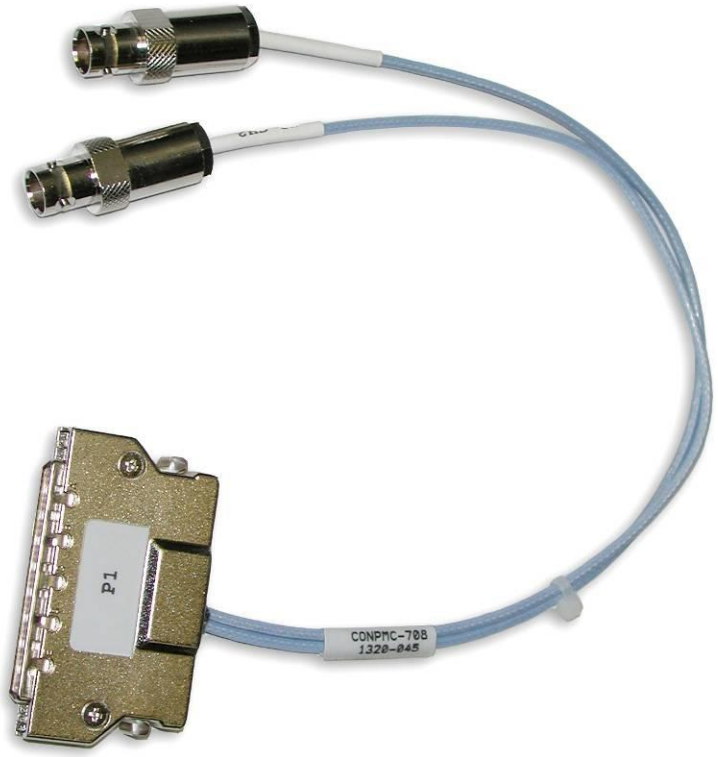


Figure 5. The CONPMC-708 Adapter Cable

ARINC 708 Bus Coupling

The ARINC 708 transmission media, or data bus, is defined as a twisted shielded pair transmission line consisting of the main bus and a number of stubs. There is one stub for each terminal connected to the bus. The CONPMC-708 cable connected to the P-708/RP-708 is designed to fulfill the requirements as one of these stubs. The main data bus must be terminated at each end with a resistance equal to the cable's characteristic impedance (plus or minus two percent). This termination makes the data bus behave electrically like an infinite transmission line.

Stubs, which are added to the main bus to connect the terminals, provide *local* loads and produce impedance mismatch where added. This mismatch, if not properly controlled, produces electrical reflections and degrades the performance of the main bus. Therefore, the characteristics of the main bus and the stubs are specified within the standard. Table 10 summarizes the transmission media characteristics.

Table 10. Transmission Media Characteristics

Cable Type	Twisted Shielded Pair
Capacitance	30.0 pF/ft. max, wire to wire
Characteristic Impedance	70.0 to 85.0 Ohms at 1 MHz
Cable Attenuation	1.5 dB/100 ft. max, at 1MHz
Cable Twists	4 Twists per ft., minimum
Shield Coverage	90% min (Notice 2)
Cable Termination	Cable impedance (+/- 2%)
Direct-coupled Stub Length	Maximum of 1 foot
Transformer-coupled Stub Length	Maximum of 20 feet

There is no maximum length of the main data bus. If you follow transmission line design practices, and are careful when placing the stubs, you can achieve working systems with the main bus length of several hundred meters. It is highly recommended that the bus be modeled to insure its operation and performance characteristics.

There are two methods for a terminal to be connected to the main bus: direct-coupled and transformer-coupled.

Transformer-coupled Method

The transformer-coupled method uses an isolation transformer for connecting the stub cable to the main bus cable. In the transformer-coupled method, the resistors are typically located with the coupling transformer in boxes called data bus couplers. The stub length can be up to a maximum of twenty feet long. Thus, the data bus may be up to twenty feet away from each terminal.

Direct-coupled Method

In the direct-coupled method, the resistors typically are located within the terminal. The stub length is limited to a maximum of one foot. Therefore, for direct-coupled systems, the data bus must be routed in close proximity to each of the terminals. With the P-708 and RP-708 products, you are limited to the length of the supplied CONPMC-708 cable.

Buses using direct-coupled stubs must be designed to withstand the impedance mismatches caused by the placement of the stubs. Thus you should perform a careful tradeoff analysis to determine where stubs are placed. You should build and test designs using the direct-coupled method to determine if the waveform distortion causes receiver problems.

ARINC 708 PMC Bus Coupling Selection

The ARINC 708 stub on which the CONPMC-708 cable connects can be defined for either transformer-coupling or direct-coupling. For the RP-708 and P-708 boards, a shunt set provides selection of the bus coupling method for each the two channels. On P-708 boards the shunts are located just behind the front panel connector, while on the RP-708 they are located behind the front-I/O adapter card connector J1. Four shunts are provided, J3 for Channel 0+ and J4 for Channel 0-; J5 for Channel 1+ and J6 for Channel 1-. Figure 6 shows the RP-708 mini-shunt and P-708 shunt settings with both channels set for transformer-coupled (center row of pins shorted to the "XC" row of pins).

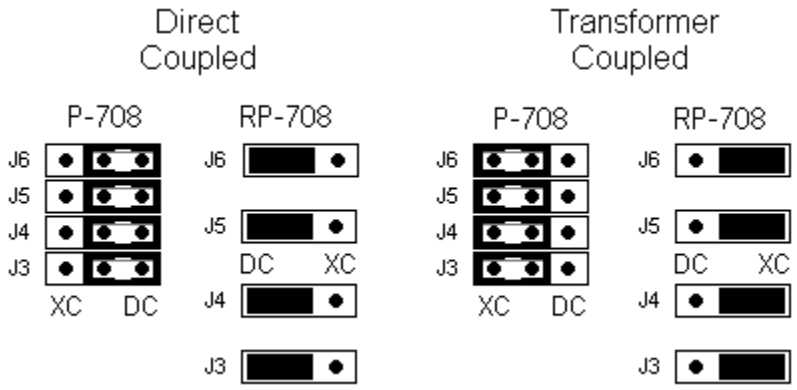


Figure 6. RP-708/P-708 Coupling Shunt Selection

A channel can be set for direct-coupled by shorting the center row of pins to the "DC" row of pins. For the P-708-C conduction cooled and rugged versions of the P-708 card, coupling selection is provided as a fixed order option from the factory.

Bus Cables and Connectors

The cables used for an ARINC 708 bus and stub connections are two-conductor twisted pair wires with twin-axial connectors.

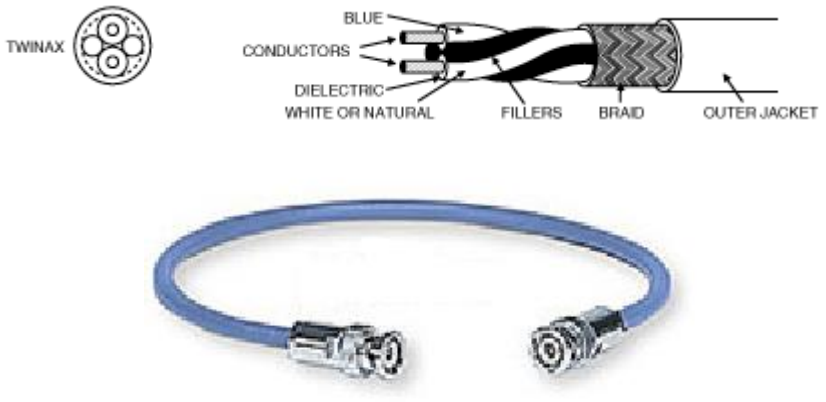


Figure 7. ARINC 708 Bus Cable

On the twin-axial connector, the center pin is the POSITIVE signal (blue conductor), and the ring is the NEGATIVE signal (white conductor). The shield is connected to ground.

Bus Terminators

The ARINC 708 bus *must* be terminated at *both* ends. Use a 78 Ohm, 2W, 1% resistor for termination. A typical ARINC 708 bus terminator is shown in the following illustration:

Figure 8. ARINC 708 Bus Terminator

Again, terminators are *not* optional. A common mistake is to leave off one or both terminators on the bus.

Bus Couplers and T Connectors

A *transformer-coupled* bus uses a bus coupler like the one shown in the following illustration:

Figure 9. ARINC 708 Bus Coupler (Transformer-coupled)

The bus coupler shown in Figure 9 is a *two-stub* coupler. It has two bus connections on either end and two stub connections on the side. Bus couplers are available with one to eight (or more) stub connections. Be careful not to confuse the bus connections with the stub connections – a common mistake is to connect stubs to bus connections or vice-versa.

A *direct-coupled* bus uses T connectors rather than bus couplers to connect stubs to terminals on the bus. A typical T connector is shown in the following illustration:

Figure 10. ARINC 708 Bus Coupler (Direct-coupled)

Remember that direct-coupled stubs should not be longer than one foot. Another common mistake is to use long stubs with direct coupling.

RFI Caps

In many situations you can leave unused stub connections open. However, in some environments it is good to use an RFI cap to limit radio frequency interference and to keep out dust, dirt, etc. An RFI cap is a metal cap that closes the shield over the unused stub connection, as shown in the following illustration:

Figure 11. RFI Cap

Note: Do *not* try to use terminators as caps on unused stubs. A terminator on a stub puts a 78 Ohm load on the stub, which is a *much* lower impedance than a normal RT would add. If you do not have RFI caps, leave the unused stub connections open.

External Wrap Connection

The following diagram demonstrates how to connect channels 1 and 2 of an ARINC 708 PMC with front I/O using the CONPMC-708 adapter to a transformer-coupled bus coupler for an external wrap configuration:

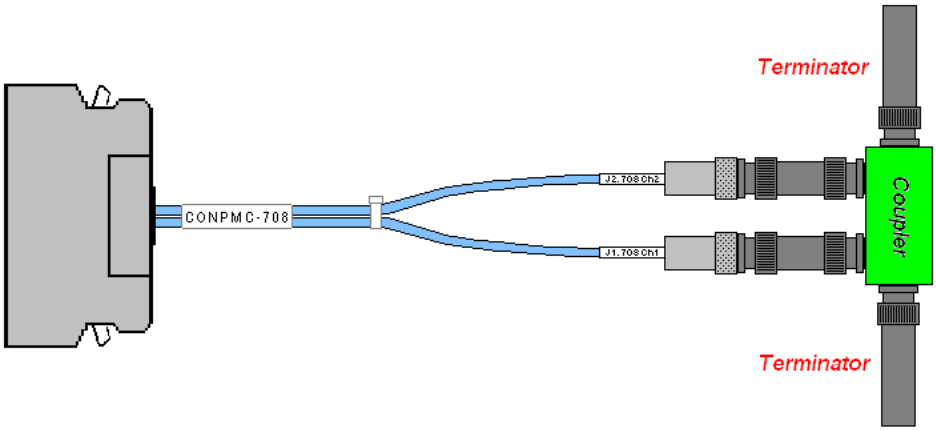


Figure 12. External Wrap Connection

Weather Radar and Radar Display Connection

The following diagram demonstrates how to connect an ARINC 708 PMC to a weather radar or weather radar display unit. In this diagram the ARINC 708 PMC front I/O connector is wired using the CONPMC-708 Adapter Cable, with the channel 1 twin-axial adapter connected to the ARINC 708 bus using transformer coupling (the most frequently encountered connection).

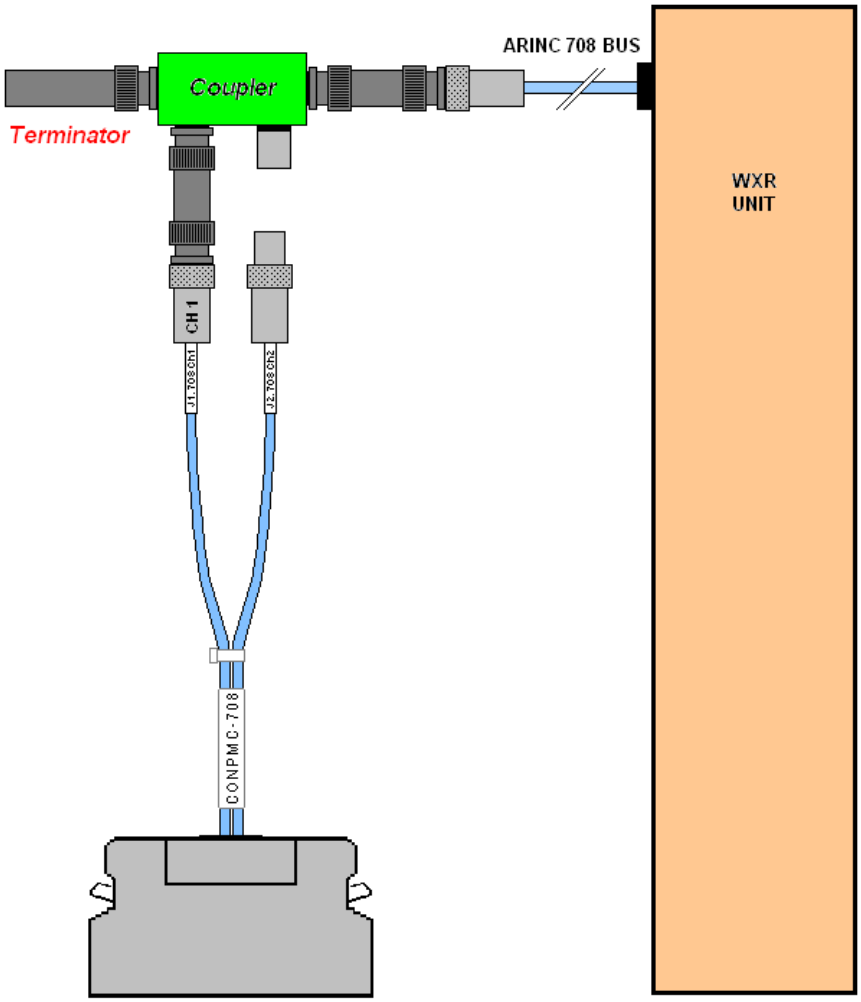


Figure 13. P-708 WXR Radar or Display Connection

CHAPTER 2

P-708-SW Windows Installation

Software Installation under Windows

Although system resources may limit the number of boards installed on a system, the P-708-SW distribution supports up to 128 devices when installed under 32-bit and 64-bit versions of the Windows 10, 8.1, 8, 7, Vista, XP and Server 2008/2012 operating systems.

Prior to physically installing the ARINC 708 board, the P-708-SW software distribution should be installed on your PC. Failure to properly install the software prior to the hardware may result in corruption of the Windows Device Manager settings and unnecessary complications.

To install the software, follow these steps:

1. Exit all programs.
2. Insert the P-708-SW CD into your CD drive.
3. If the installation does not automatically start after 10 seconds:
   - Click Start from the Window 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 and properly select the board type being installed as an RP-708 or P-708/P-708-C.
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 P-708 product into the any available PMC, cPCI, PCI Express or PCI slot.

Device Driver Installation under Windows

To install the hardware, follow these steps:

1. Power-up the PC.

   Windows 10, 8.1, 8, 7 and Server 2008/2012, device installation occurs automatically; for other Windows versions, the Windows Plug and Play hardware manager should detect the newly installed device, and the *Found New Hardware* dialog box should automatically startup. Decline any request to query the web 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**.

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 Hardware.

3. If the device drivers are not automatically detected and you are prompted for a path to the driver location, enter the full path to the distribution driver folder located beneath

   C:\Program Files\Condor Engineering\P-708-SW\Drivers

   and click **OK**.

4. To check for proper driver installation, open the Device Manager as follows:

   - Under Windows 7, Vista, XP and Server 2008, use the “Start -> Run” command prompt, enter “devmgmt.msc” to open the Windows Device Manager.
   - Under Windows 10, 8.1, 8 and Server 2012, open the “Run” application and enter “devmgmt.msc” to open the Windows Device Manager.

- Expand the **Abaco Systems Avionics Devices** folder.

5. Verify the device entry *Product Name* (P-708 or RP-708) 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.

Installation Verification

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

1. Under Windows 7, Vista, XP and Server 2008
    1. Click Start, then Programs.
    2. Find and expand the **Abaco P-708-SW** program group
    3. Invoke the **P-708 Test Installation** shortcut located therein.
2. Under Windows 8, 8.1 and Server 2012
    a. Display “Apps by name”.
    b. Invoke the **P-708 Test Installation** shortcut.
3. Under Windows 10 and Server 2016
    a. Expand “All apps”.
    b. Scroll down and expand the **Abaco P-708-SW** application group.
    c. Invoke the **P-708 Test Installation** shortcut beneath.

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.

CHAPTER 3

Linux Installation

Overview

The P-708-SW distribution provides support for the ARINC 708 products under most Linux Kernel 2.4, 2.6, 3.x and 4.x revisions. The install process builds the API as a shared library and installs the driver as a module. Application programs must 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 archive for the latest information on installing and building the common driver along with the ARINC 708 API and example program.

Software Installation

The Linux installation process requires your ARINC 708 hardware be installed prior to execution of the installation:

1. Log on as "root" (you may use "su")
2. Copy the Linux distribution compressed tar file (linux_p708_v*nnn*.tgz) to the /root directory.
3. Uncompress and extract the installation file using the following:

   tar -zxvf linux_p708_v*nnn*.tgz

After the tarball extraction completes, the following directory structure is created:

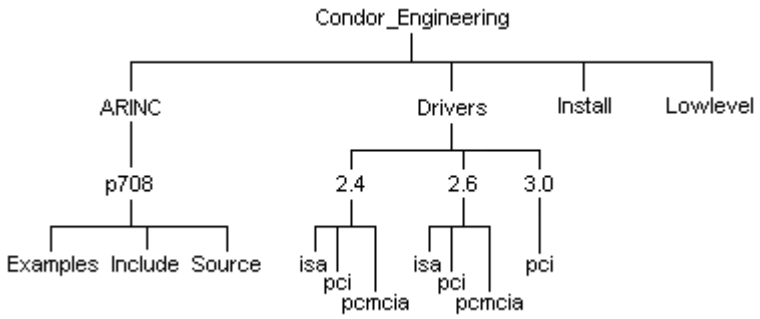


Figure 14. Linux Installation Directory Structure

Building the Distribution

Automatic installation (Builds LSP, API, and example)

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 Abaco Systems avionics PCI board in the "procfs" 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 later versions and does not agree with the "Proprietary/GPL" license, 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 are 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 are 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" does not generate. You need to follow the instructions appearing 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.

The installation is finished. Check the "install" script output and the kernel message log for any errors. If there are no errors, 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 run *tst_cnfg*.

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, 2.6 or 3.0. 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”.

Installation automatically calls these scripts. However, if you reboot the system, you need to re-run this script. You can put the script in the rc.local initialization file. The script automatically runs on power-up. The installation instructions located in the distribution file *Linux_Install.txt* explain how you manually run the script.

Troubleshooting

When installing any API distribution, you need to be logged in 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".

   If you are loading the 2.6 or 3.x PCI device driver and you receive errors indicating missing symbols with "sysfs" in the symbol name, build the distribution without support for SYSFS.

CHAPTER 4

VxWorks Installation

Overview

VxWorks is an embedded real-time operating system supporting flexible hardware configurations. The ARINC 708 API compiles and runs under most Motorola PowerPC and Intel x86 VxWorks Board Support Packages.

The ARINC 708 API can control 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 ARINC 708 API with VxWorks you must first build a VxWorks image supporting the respective ARINC 708 PMC configuration. Upon boot of this image, you may download and execute the client application. These basic steps are described in subsequent sections of this chapter.

Building a VxWorks Image

The procedure provided below for including the ARINC 708 API and common avionics driver in your VxWorks image utilizes the 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.

In addition to these general instructions are specific dependencies on the use of the VxBus Gen1 and Gen2 device drivers with your target processor and BSP, also described below.

VxBus Gen 1 Driver Support (VxWorks 6.8 and 6.9)

1. To install the generic Abaco Avionics common VxBus driver support, copy the file *40avioVxwDrv.cdf* from the *Component Installation File* folder beneath:

   /Program Files/Condor Engineering/P-708-SW/Source/VxWorks/VxBus Gen1 Driver

   To

   [*Workbench_directory_path*]\target\config\comps\vxWorks

2. Copy the following source and header files from the folder

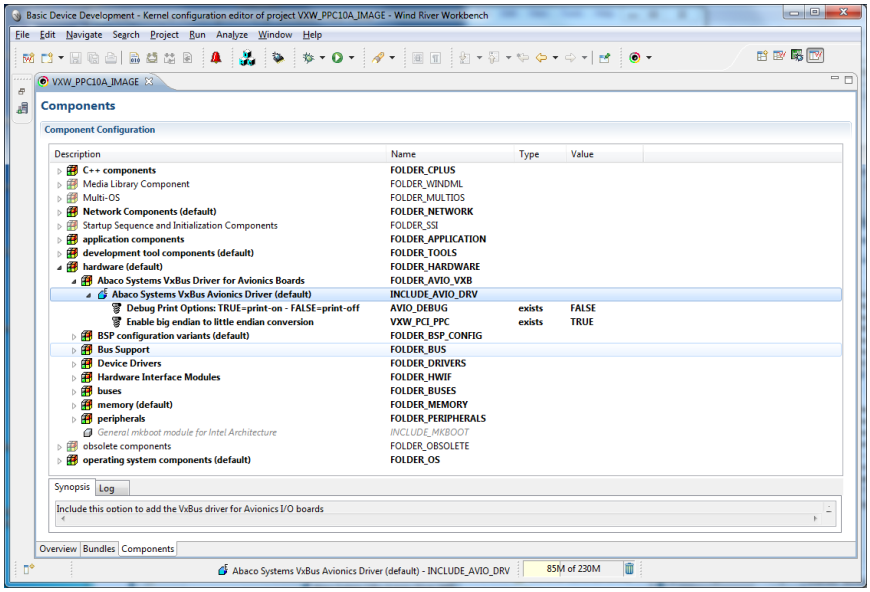
   /Program Files/Condor Engineering/P-708-SW/Source/VxWorks/VxBus Gen1 Driver

   To either folder

   [*Workbench_directory_path*]\target\config\<*BSP Folder*> or [*Workbench_directory_path*]\target\config\comps\src:

   avioVxwDrv.c
   avioVxwDrv.h

Continue with *Common Build Components* below to build the ARINC 708 API. A sample Abaco Avionics common PPC VxBus Gen 1 Kernel Configuration file setup is shown below.



VxBus Gen 2 Driver Support (VxWorks 7)

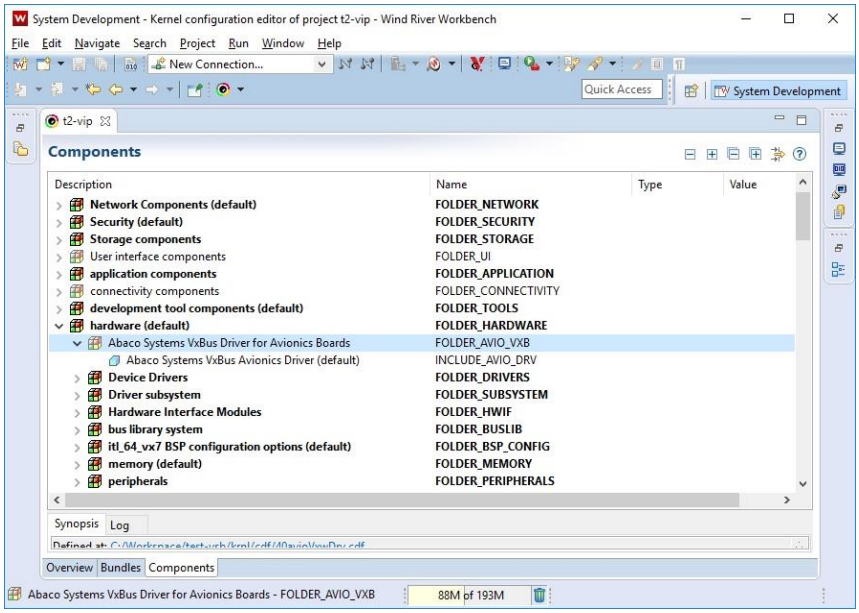
1. To install the generic Abaco Avionics common VxBus Gen2 driver support for VxWorks 7, copy the entire folder *abaco_avio* (not just the contents) from:

   /Program Files/Condor Engineering/P-708-SW/Source/VxWorks/VxBus Gen2 Driver

To

[*Workbench_directory_path*]/VxWorks-7/pkgs

2. If you plan to create a new VxWorks 7 Source Build Project, once installed the driver should be included in the build automatically; however, if you are adding the Abaco Avionics driver to an existing VxWorks 7 Source Build Project, you must perform the following steps:
   a. Double-click on your VSB Source Build Configuration entry to refresh the project.
   b. Expand the Build Targets selection
   c. Expand the Layers selection
   d. Right-click on the AVIO_1_0_0_2 layer and select Build Layer
   e. The Abaco Systems VxBus Driver for Avionics Boards should now be available for selection within any VxWorks 7 Image Project based on this VSB.
3. A sample Abaco Avionics common VxWorks 7 x86 VxBus Gen2 Kernel Component Description File setup is shown below:



4. Right-click on the *Abaco Systems VxBus Driver for Avionics Boards* entry and select *Include*.
5. Continue to *Common Build Components* to build the kernel image with the VxBus Gen2 device driver.
6. If you prefer to build an RPM installer for this driver, the spec file *abaco_driver.spec* is provided in the */VxWorks Driver/VxBus Gen2 Driver* distribution folder.

Legacy PCI Driver Support

1. To build the generic legacy (non-VxBus) Abaco Avionics common VxWorks PCI driver, copy the appropriate component installation file as specified in the following table:

Kernel Version	Platform	Component Installation File
VxWorks 5.5	x86	51_GEIP_x86_55_PCI.cdf
	PPC	51_GEIP_PPC_55_PCI.cdf
VxWorks 6.0 - 6.5	x86	51_GEIP_x86_RTP_6x_PCI.cdf
	PPC	51_GEIP_PPC_RTP_6x_PCI.cdf
VxWorks 6.6 - 6.9	x86	51_GEIP_x86_RTP_66_PCI.cdf
	PPC	51_GEIP_PPC_RTP_6x_PCI.cdf

   from the respective processor-specific folder in the *VxWorks Legacy PCI Driver/Component Installation File* folder located beneath:

   /Program Files/Condor Engineering/P-708-SW/Source/VxWorks

   To

   [*Workbench_directory_path*]/target/config/comps/vxWorks

2. Copy the following source files from the folder

   \Program Files\Condor Engineering\P-708-SW\Source\VxWorks

   And the file *cei_types.h* from the distribution Include folder, to either folder

   [*Workbench_directory_path*]\target\config\<*BSP Folder*> or [*Workbench_directory_path*]\target\config\comps\src

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

Common Build Components

1. You may choose to include the ARINC 708 API source files in the BSP kernel source folder, or create a new project folder for the ARINC 708 API and application development. In either case, copy the following source files from the folder \Program Files\Condor Engineering\P-708-SW\Source and VxWorks folder beneath, to the folder of your choice:

   p708_api.c p708_vxw.c

2. 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\P-708-SW\Include and driver source folder to that same folder, with some files dependent on the driver method used:

ar_error.h	cei_types.h
fpga_708.h	fpga_rp708.h
p708_api.h	p708_glb.h
p708_hw.h	avioVxwDrv.h

3. Open the workspace containing your VxWorks target image project and access the Kernel Configuration setup for the VxWorks image.
4. Beneath the *hardware* component, right-click the "*Abaco Systems VxBus Driver for Avionics Boards*" (or "*Abaco Systems PCI Avionics Products*"), and select *Include (quick include*).
5. Modify the default values for any definitions as required for your target system. Examples of such modifications include:
   - For VxBus x86/Pentium kernel images, change the default state of *"Enable big endian to little endian conversion"* to FALSE.
   - For Legacy PCIbus x86/Pentium kernel images, change the default state of *"Define PCI Compile for PowerPC"* to FALSE.
6. If you choose to build the API outside of the BSP source folder, you may have to manually define the directive VXW_PCI_PPC as TRUE for a PowerPC target C source compilation/build or FALSE for an x86/Pentium target C source compilation/build, (normally defined in the configuration definition file).
7. Once you have your VxWorks kernel image built and installed on your target, open a shell to the target and invoke the function *avioDeviceShow*. This routine lists all supported Abaco 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 P-708 API source code compilation for your target BSP.

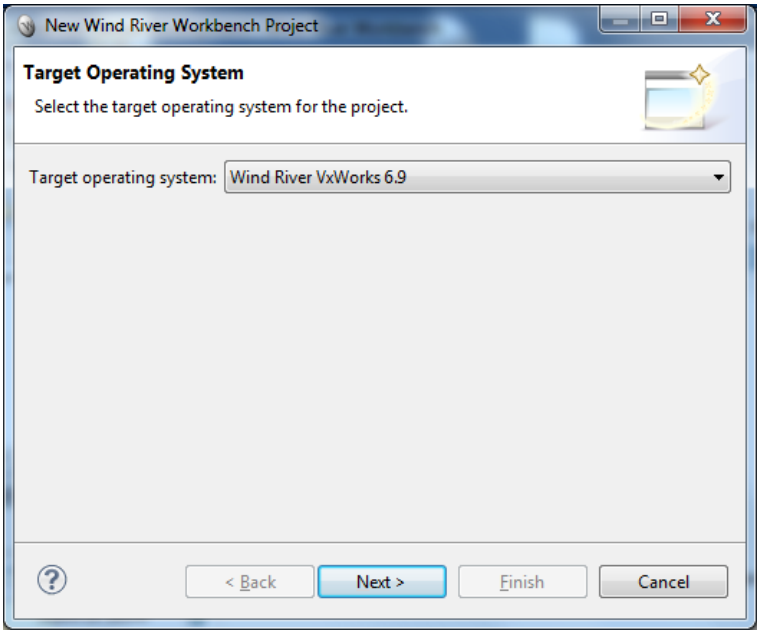
Using the Sample Program

The 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 P-708-SW\Source directory. You can use this program to test your VxWorks installation, as it simply executes an internal wrap configured as a receiver-transmitter channel pair. You can also use TST_CNFG.C as a guide for programming with the ARINC 708 API.

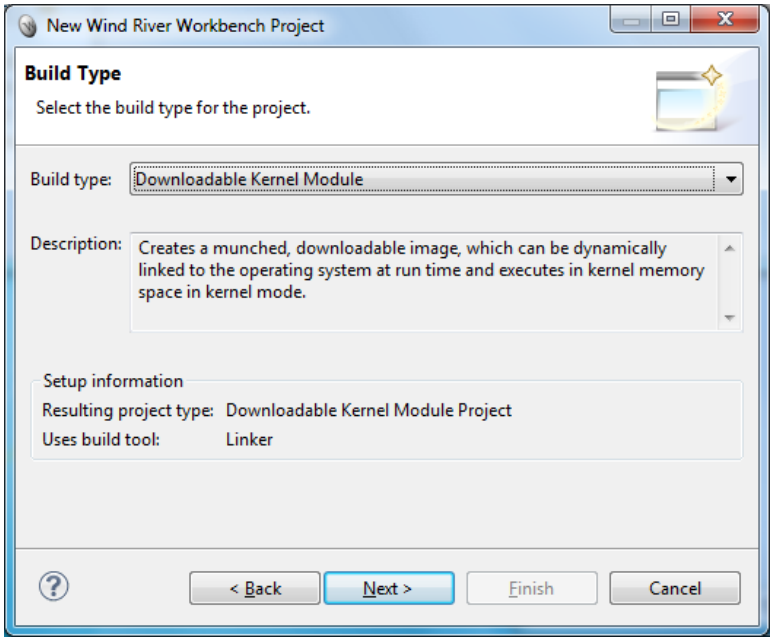
Building the API and Sample Program with Workbench

While the driver portion of the distribution must be built into the kernel image, the API and sample program can be built as downloadable objects. The following steps explain how to build the API and sample program together with Workbench for a PowerPC target supporting the VxBus driver, but can be easily adapted to Tornado, as well as other build environments.

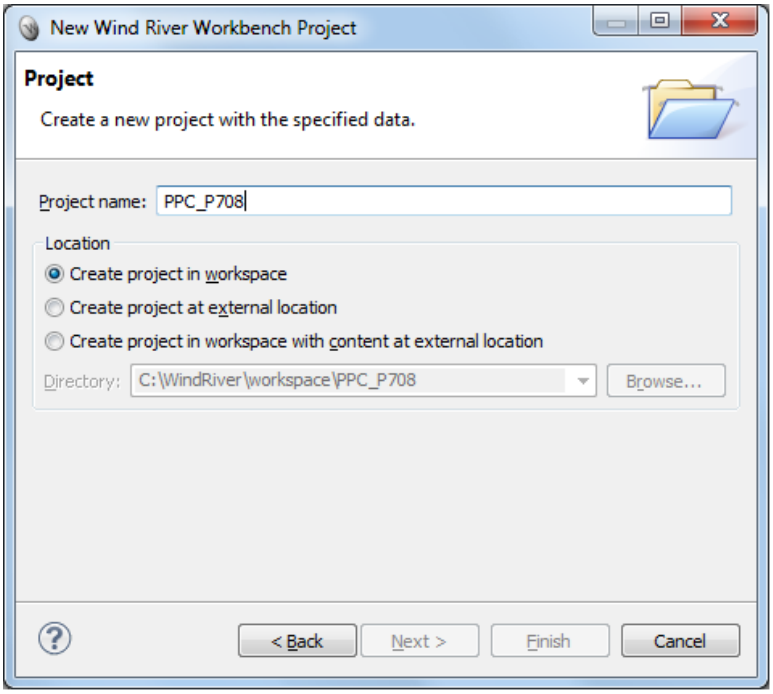
1. From the File pulldown, select New->Wind River Workbench Project… to initiate a new project for your downloadable application. Specify your target operating system, then select **Next**.



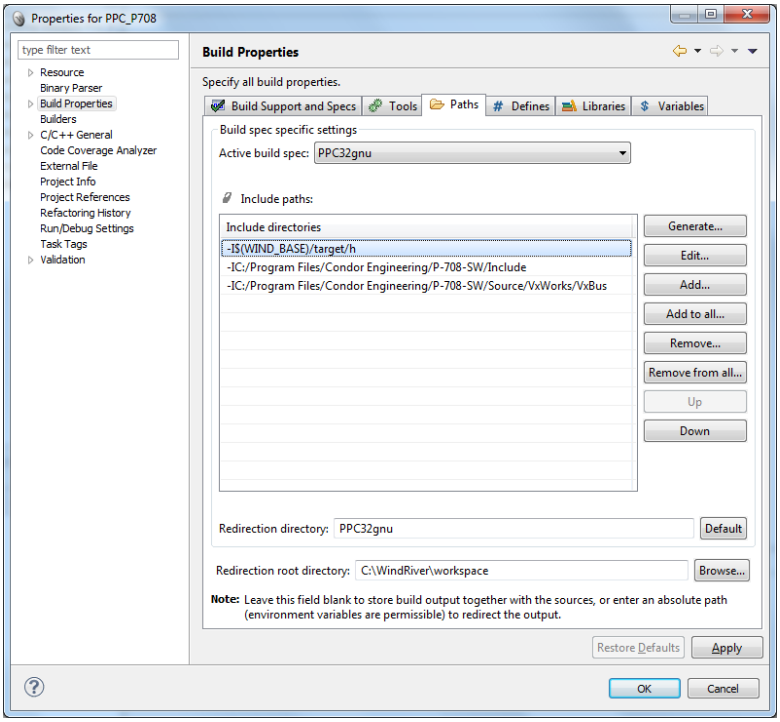
2. For initial testing of the board installation, select the build type “Downloadable Kernel Module”. Click **Next**.



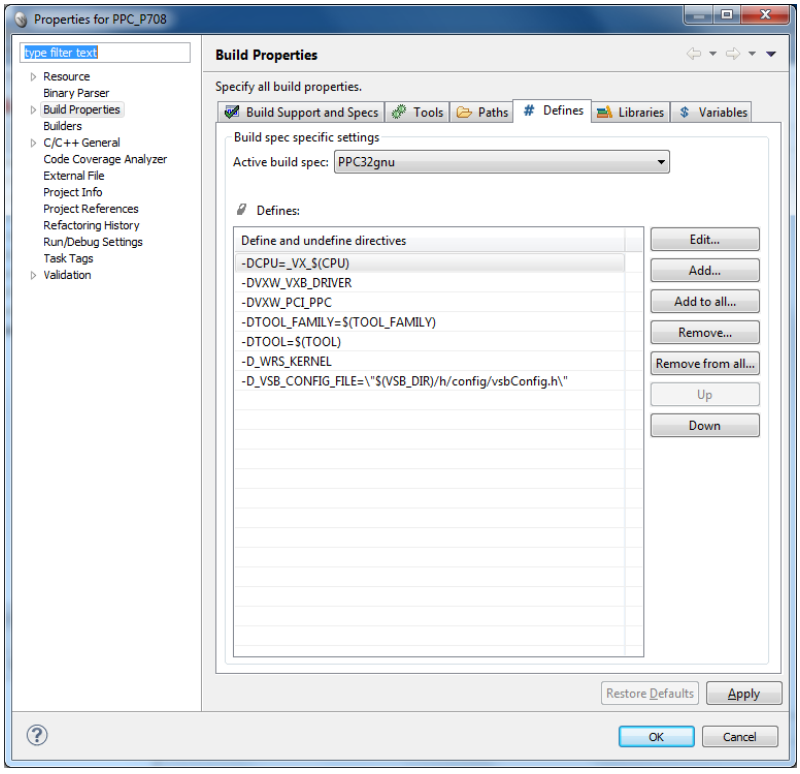
3. Enter your project name. "PPC_P708" was used for this example. Click **Finish**.



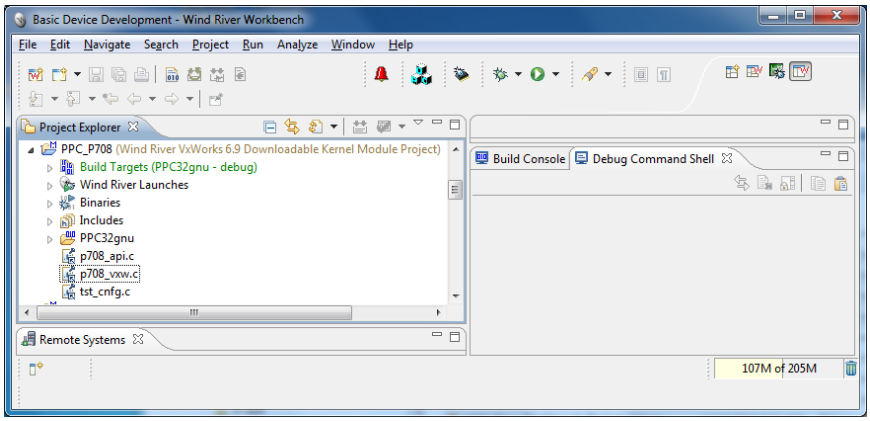
4. Right-click the new project and select *Properties*.
5. Click on *Build Properties* and select the *Paths* tab.
6. Add paths to the P-708 distribution's Include and VxBus Driver folders, then click on **Apply**.



7. Click on the Defines tab, and add defines for VXW_PCI_PPC and VXW_VXB_DRIVER. Click on **Apply**, then **OK**.



8. Open a view of the P-708-SW/Source folder the Windows Explorer. Drag the files P708_API.C, P708_VXW.C and TST_CNFG.C to the top level project, so they are included as follows:



9. Right-click the build specification and select **Build Project**.

10. Assuming you have already connected to the target via target server, right-click on the project and select **Download->VxWorks Kernel Task** to download the output file you created.

11. Open a host shell to the target, then invoke the application by typing **wrap708** at the shell prompt.

    This program executes a basic internal wrap test on the P-708/RP-708 board and notifies you of success or failure. If the program reports success, you are ready to use your new board.

Target-specific Compiler Directives

The P-708 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*. In addition to VxBus support, there are also differing methods for Legacy PCIbus driver mapping to a P-708 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 P-708 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:

Parameter	Description	Target	Options
Common Avionics API and Driver Specific Parameters and Directives			
VXW_VXB_DRIVER	When this directive is defined in the kernel image build project, it designates the use of the VxWorks VxBus device driver in place of the legacy PCI driver.	Both VxBus drivers for VxWorks 6.x and VxWorks 7	N/A
VXW_PCI_PPC	When this directive is present or defined, it designates the target processor architecture as either PowerPC or x86.	All Kernel Versions	TRUE (PPC) FALSE (x86)
VXW_PCI_X86	Used to enable x86 target processor specific interrupt processing.	Legacy PCI driver for VxWorks 5.5 and 6.x	TRUE (x86) FALSE (N/A)
AVIO_DEBUG	Used to enable console printout of debug information during driver initialization.	VxBus Gen1 driver for VxWorks 6.7 through 6.9	TRUE FALSE (default)
vxwdebug	Used to enable console printout of debug information during driver initialization.	Legacy PCI driver for VxWorks 5.5 and 6.x	TRUE FALSE (default)
VXW_X86_MAP_ADD	Adds the board's PCI memory to *sysPhysMemDesc* via invocation of *sysMmuMapAdd*. Optional for an x86 target processor	Legacy PCI driver for VxWorks 5.5 and 6.0 to 6.5	TRUE FALSE (default)
P-DIS Specific Parameters and Directives			
NON_INTEL_WORD_ORDER	This directive is intended for use when building for a Big Endian mapped target (typical for PowerPC and usually handled via target_defines.h).	All Kernel Versions	N/A
DELAY_USE_SYS_US_DELAY	This directive is intended for use when a delay should be implemented with *sysUsDelay* instead of *taskDelay* (applies to some Thales VMPC* targets).	VxWorks 5.5	N/A
DELAY_USE_SYS_MS_DELAY	This directive is intended for use when a delay should be implemented with *sysMsDelay* instead of *taskDelay* (applies to some Motorola MCP* targets).	VxWorks 5.5	N/A

CHAPTER 5

Integrity® Support

Introduction

Green Hills Integrity® is a secure high reliability real-time operating system (RTOS) intended for use in mission critical embedded systems. The ARINC 708 API supports the Integrity operating system on PowerPC and Intel processors, using the standard P-708 API C source files in conjunction with the supplied Integrity-specific driver interface and additional kernel C source file set.

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. The P-708-SW Integrity distribution supports either method. The distribution provides the Integrity PCI driver, API source code, API static library, and example program source. Link the static library with your application to obtain ARINC 708 API support. The source code allows you to modify the static library if needed.

Integrity Installation

There are two options for installing P-708-SW support for Integrity. If you are running the Multi IDE on a Windows system, you can install using the Windows installation and copy the source files from the P-708-SW distribution \Source and \Include folders. You may also copy these folders directly from the Installation CD-ROM. The installation contains the Abaco Systems Avionics Integrity PCI driver, along with the source code for the API, driver interface, and example program.

After acquiring the P-708-SW distribution source files, you need to copy the PCI device driver source into the Integrity BSP project for your target system and rebuild the kernel. The Abaco Systems Avionics Integrity driver works with most PowerPC and Intel BSPs. You can then build your own static library and example application projects.

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. The figure below 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.

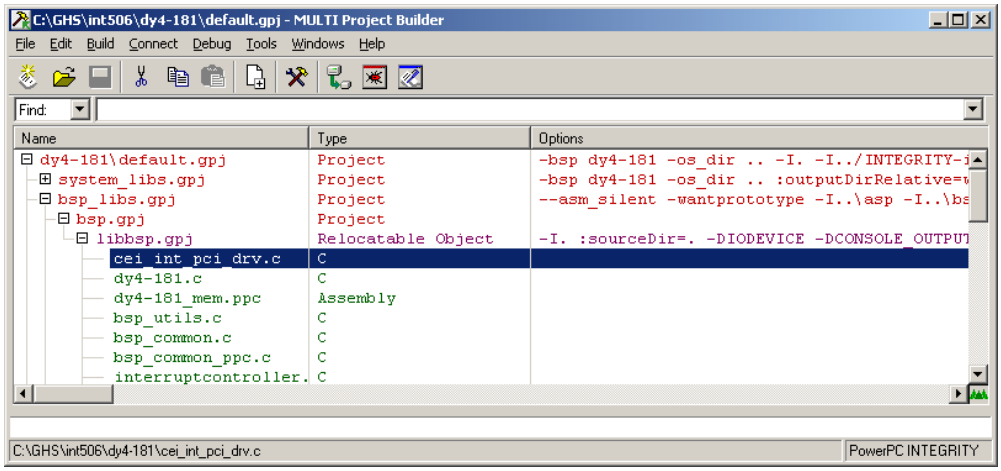


Figure 15. Integrity PCI Driver Installation

Building the ARINC 708 API with Multi

The P-708-SW Integrity Distribution contains all of the source files necessary to build a static library for a generic PowerPC target, which should suffice for most PowerPC systems. Build a static library with the supplied source and include files as shown below:

p708_api.c
p708_int.c
mem_integrity.c

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

ar_error.h	p708_api.h
p708_glb.h	p708_hw.h
fpga_708.h	fpga_rp708.h
cei_types.h	lowlevel.h
target_defines.h	

Compiler Directives

The following compiler directives should be used when building the P-708 API for your target:

- **INTEGRITY_POSIX** should be defined for any INTEGRITY project using POSIX support, which can be any build other than for an Integrity 178B host.
- **INTEGRITY_PCI_PPC** selects the INTEGRITY target compilation for a PowerPC host in the API source files.
- **INTEGRITY_PCI_X86** selects the INTEGRITY target compilation for an Intel host in the API source files.

Monolith Image versus Dynamic Download

During development, if building your application and P-708 API library as a separate **virtual AddressSpace project** to be deployed via Dynamic Download, no further directives are required. The Integrity dynamic loader will determine the virtual addresses of your P-708 board memory regions at load time and pass those addresses to the API during the open session resource acquisition process.

If you are building your application and/or P-708 API library with the kernel as a single image via **Monolith Integrity Application Project**, the symbol **GHS_KERNEL** must be defined for the compilation of the file mem_integrity.c.

The compiler define for NON_INTEL_WORD_ORDER is now provided via the file TARGET_DEFINES.H.

P-708 API Project Setup

Select a stand-alone project for your host architecture (generic PowerPC or Intel) and select a processor 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.

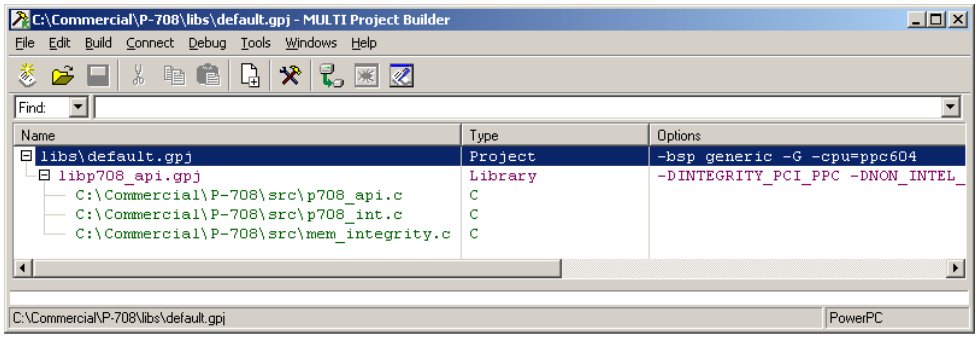


Figure 16. Example P-708 Integrity Library Project Setup

If you are building a library to run in a Monolith, you also need to define the symbol GHS_KERNEL under *Define Preprocessor Symbol*. You can name the output library to anything and use that library name to link with your application(s).

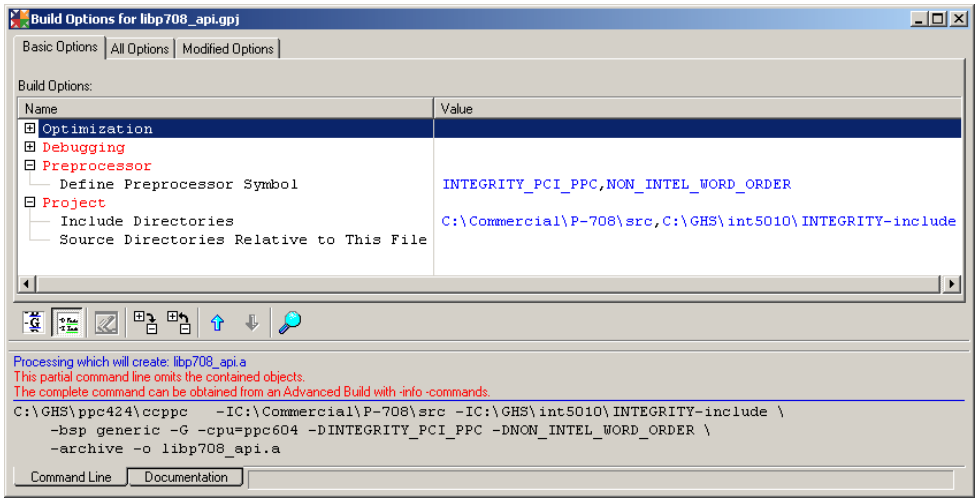


Figure 17. Example P-708 Integrity Library Project Options

Building Integrity Applications

This procedure for building your application assumes you have built a static API library for dynamic download as described previously. The figure below shows a typical Dynamic Download project using your ARINC 708 API library.

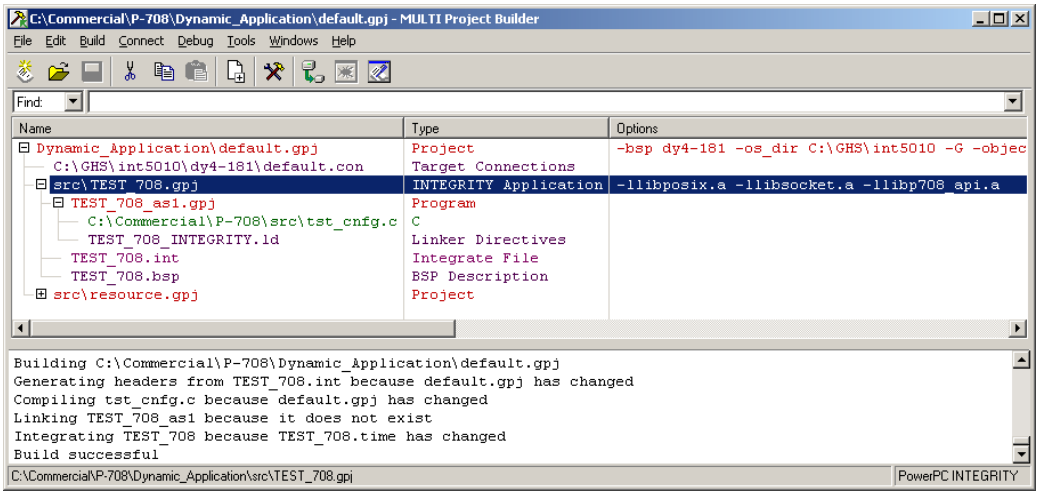


Figure 18. Example P-708 Integrity Application Project Setup

Build the application using the following steps:

12. Define the host processor-specific preprocessor symbol INTERGITY_PCI_PPC or INTERGITY_PCI_X86.
13. Link with the P-708 static library, 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.

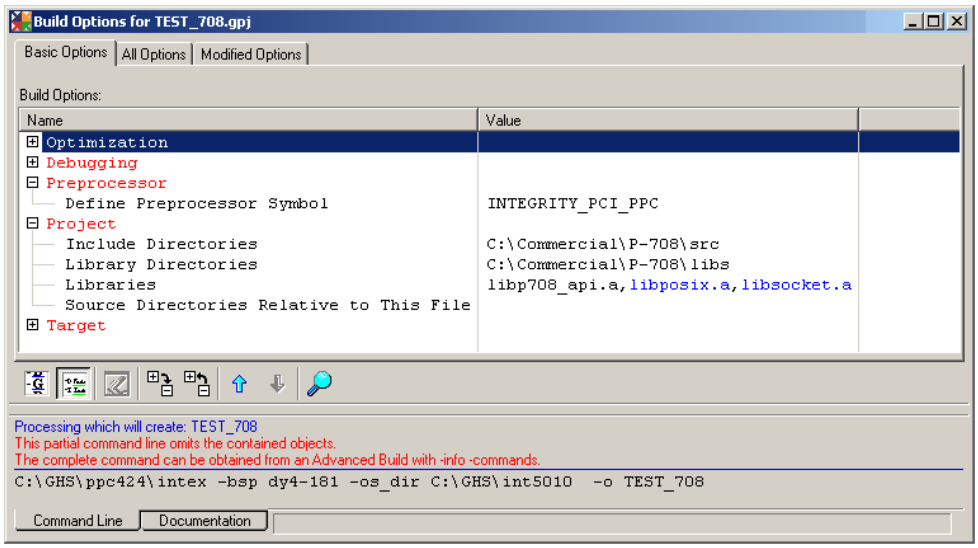


Figure 19. Example P-708 Integrity Application Project Options

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.

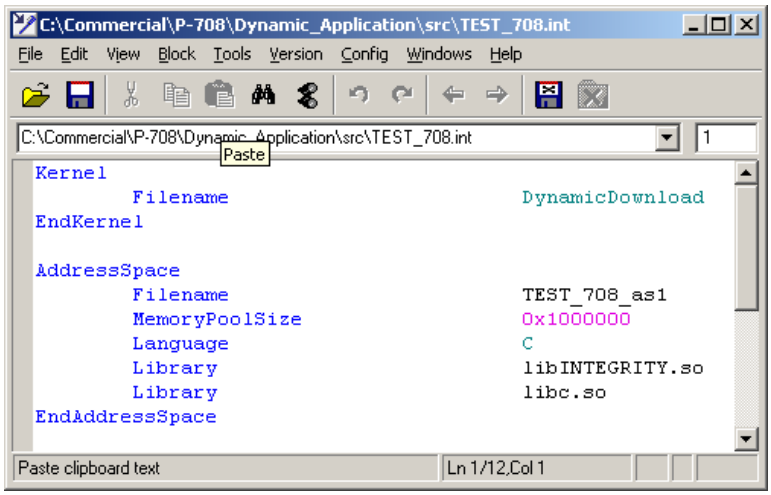


Figure 20. Adding a MemoryPoolSize Entry

14. 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:

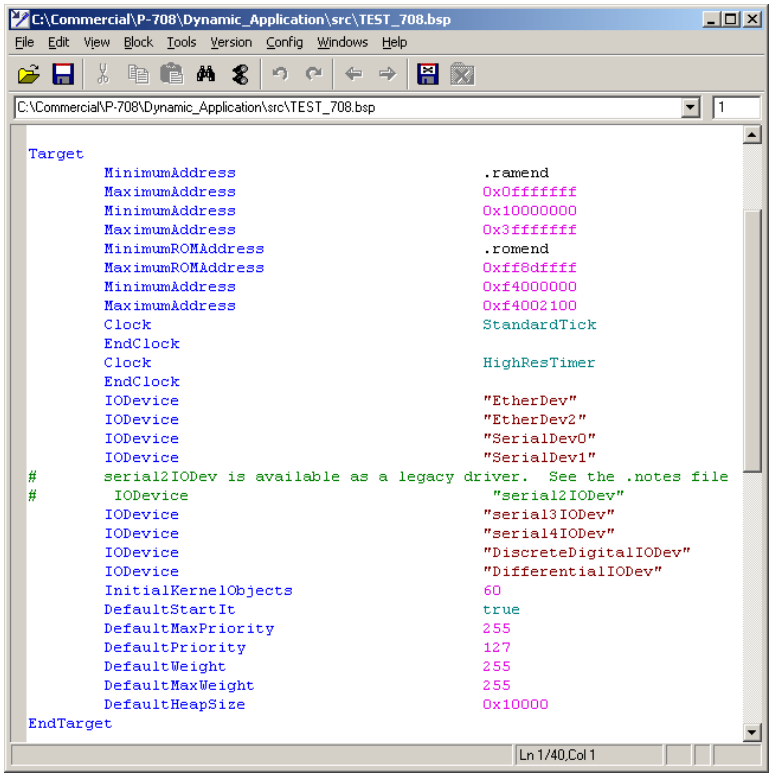


Figure 21. Modifying the Value for the DefaultStartIt Attribute

CHAPTER 6

LabVIEW Support

Introduction

The P-708-SW distribution provides a combination of the components of the ARINC 708 Application Programming Interface (API) with a set of LabVIEW™ Virtual Instruments (VIs) designed for use with either LabVIEW under most Windows operating systems or a National Instruments PXI/ETS LabVIEW Real-Time target. A basic application-level VI demonstrates how to incorporate P-708 product and API features into your ARINC 708 LabVIEW Real-Time applications.

Understanding the components of the P-708 LabVIEW support is an important first step in getting started with this product. Included in the P-708-SW distribution for the ARINC 708 PMC products are the following components, all located in folders beneath the \P-708-SW\LV folder:

- On-line VI Documentation
- Driver Files
- Example VI and Project
- The Functional VI Set
- LabWindows/CVI Real-Time Target Dynamic Link Library

Example VI and Project

An example VI and corresponding project named *P708_Wrap_Example* are provided and can be used with either LabVIEW or LabVIEW Real-Time. This example demonstrates how to setup a ARINC 708 PMC channel to continuously transmit a 720 frame sweep consisting of 3 color bands, and retrieve received frame data internally wrapped to the receive channel.

Functional VI Set

The set of VIs provided with the P-708-SW distribution support a slightly limited set of functionality for the P-708 channel configuration and buffer usage. With the exception of simplified buffer and channel configuration VIs, the VI set contains one-to-one equivalent VI to ARINC 708 API routines that interface to the ARINC 708 PMC devices. This VI set provides the fundamental VIs necessary for using most of the hardware features available with the ARINC 708 PMC product line. A help index of VI descriptions is included with access to the documentation for each VI, available in the \LV\Documentation folder and accessed via the *P-708 LabVIEW VI Help Index* shortcut.

LabVIEW Real-Time

Support for LabVIEW Real-Time is included with the P-708 LabVIEW distribution as described in the following paragraphs.

Installation in a LabVIEW Real-Time PXI/ETS System

A VISA driver INF file is included in the \LV\Driver folder, called *P-708.INF*. This file must be copied to the directory C:\ni-rt\system on your target controller using any FTP utility prior to installation of an ARINC 708 PMC board. Once this file and the ARINC 708 PMC board are installed and the target controller is running, the board should be present in the MAX device list for this target.

If you are using LabVIEW 2010 or later, a driver installation application VI called P708_Install_Driver.vi is provided in the same folder as the driver INF file. If you execute this VI and provide the IP Address for your Real-Time Target system, it will install the P-708.inf file on your target system's C:\ni-rt\system directory.

P-708 LabVIEW Real-Time API Library

A LabVIEW Real-Time only Dynamic Link Library (P708_API.DLL) is included in the \LV\Examples\Real-Time Project folder, built using LabWindows/CVI specifically for use when an example project is modified to be executed on a Real-Time PXI/ETS target. This DLL should only be used for LabVIEW Real-Time applications and will fail if referenced by any VI under any Windows operating system.

P-708 LabVIEW Projects

The LabVIEW project *P708_Wrap_Example* can be modified to run on a LabVIEW Real-Time PXI/ETS system by invoking the Add Targets and Devices dialog and adding a new Real-Time target to the project. This project will support execution on any Real-Time target system having a P-708 installed.

The LabVIEW project *Detect_Target_Devices* is designed to execute exclusively on a LabVIEW Real-Time PXI/ETS system. It queries the target system for all supported Abaco Systems Avionics boards and displays those found based on the board type.

P-708 Device Indexing

The Device Index In control used in the P-708 LabVIEW example application VI and any custom LabVIEW application referencing P-708 boards only applies to P-708 boards installed in the PXI/ETS system. The first P-708 board installed should be referenced as Device 0, and increment for each additional P-708 board installed.

This device number reference scheme differs from executing a LabVIEW application under Windows, where each Abaco Systems Avionics board installed is referenced by a specific Device Index value.

Troubleshooting

There are several issues that may arise when using the P-708 LabVIEW support with a LabVIEW Real-Time target. The following paragraphs describe some potential problems and their resolutions.

- **PROBLEM:** When deploying a LabVIEW Real-Time project or executing a LabVIEW Real-Time application executable upon target startup, an error indicating a “VI failed to load a DLL on RT target device” is encountered.

  CAUSE: The ARINC 708 API library file P708_API.DLL and/or National Instruments CVI Run-Time Engine library file CVI_LVRT.DLL are not installed on the target.

  RESOLUTION: If the file P708_API.DLL from the Projects folder is not included in your project, (see your *project* Application Properties, Source Files option), you can FTP the file P708_API.DLL to the \ni-rt\SYSTEM folder on the target system.

  FTP the version of CVI_LVRT.DLL installed by your LabVIEW distribution (usually located in the \Windows\System32 folder on your host system), to the \ni-rt\SYSTEM folder on the target system.

- **PROBLEM:** When executing any of the example or custom authored LabVIEW applications, a dialog box indicating a failure to download a VI appears, followed by the RT Target Errors dialog box appearing with the following entry in the Error List:

  > LabVIEW: Failed to load shared library p708_api.dll on RT target device.

  CAUSE: The Windows version of P708_API.DLL resides in the folder in which the execution a LabVIEW R/T application has been attempted or is referenced in the LabVIEW Real-Time application project.

  RESOLUTION: First, close LabVIEW, then overwrite the version of P708_API.DLL in the current folder or change the project library reference to the LabWindows/CVI generated version of P708_API.DLL residing the projects folder.

- **PROBLEM:** Execution of any application VI results in an Application Error Status of "Device Alloc Failure", but no RT Target Error dialog box appears.

  CAUSE: The P708.INF file for your board may not be properly installed the PXI/ETS target controller.

  RESOLUTION: Assure your board is present under the device list for your Remote System, as shown via the Measurement and Automation Explorer (MAX). If it does not appear, check to see the P708.INF file has been installed in the correct folder on your target.

- **PROBLEM:** When executing any of the example or custom authored LabVIEW applications from the host, a dialog box appears as follows:

  > LabVIEW: LabVIEW.exe – Unable To Locate DLL. The dynamic link library cvi_lvrt.dll could not be found in the specified path, (C:\<path>).

  CAUSE: LabVIEW did not detect installation of the CVI Run-Time Engine library file *cvi_lvrt.dll*.

  RESOLUTION: If you are using LabVIEW Real-Time, assure the required DLL "cvi_lvrt.dll" resides in the Windows System (\system or \system32) directory on your host computer.

CHAPTER 7

ARINC 708 PMC Product Features

Overview

The ARINC 708 products provide two channels, each of which operates as an independent transmitter or receiver with a programmable frame size and buffer storage capacity. They provide specialized receive features such as frame time-tagging and bit count error detection as well as specialized transmit features such as sweep scheduling, programmable frame gap duration, and frame bit count/sync pulse error injection. The following paragraphs document several of these features, and how they might be used in your ARINC 708 application.

ARINC 708 Protocol Support

The electrical transmission of ARINC 708 data over the bus is performed with start and stop sync pulse widths per the ARINC 708 / 453 specifications. The number of bits transmitted between the start and stop sync pulses is programmable by the host.

The API routine P708_SET_CHANNEL_CONFIG provides the method to set transmit and receive channel options for the ARINC 708 products.

Receive Frame Time-tagging

The ARINC 708 products support received frame time-tagging based on an individual internal 48-bit one-microsecond timer allocated to each channel. Each channel's timer is reset to zero and begins counting upon transition to the *run* mode. The host does not have control of, or access to, the timer source used as the receive frame time-tag reference.

The time-tag for each frame is stored in the Ancillary Buffer region allocated to the respective channel, based on detection of the frame start sync pulse. The 48-bit time-tag value is stored in three consecutive 16-bit locations, starting with the ancillary buffer offset specified in the respective channel's Ancillary Buffer Start register. Time-tag storage wraps around to the start of the Ancillary Buffer region once the end of the buffer is reached, as specified in the respective channel's Ancillary Buffer Stop register.

Receive Frame Storage

The ARINC 708 products store received frame data in the 16-bit wide Frame Data Buffer region allocated to the respective channel. Within the boundaries of that region, all 16 bits of each word are used for continuous frame data storage, (there are no unused bits). For frame sizes not divisible by a 16-bit word boundary, the frame storage ends and subsequent frame storage begins on a bit boundary within the respective memory location. Frame storage is based on the values specified in the respective channel's Frame Size register.

The starting location at which received frames are stored is specified in the respective channel's Data Buffer Start register. Frame data storage wraps around to the start of the Data Buffer region once the end of the buffer is reached, as specified in the respective channel's Data Buffer Stop register.

The number of frames actively stored in the assigned Frame Data Buffer region is indicated in the Frame Count register. This value represents the total number of frames received since this channel was last enabled. Since frame data storage is based on a variable bit size, the action of moving the end of a frame's data from the serial receive logic to the Frame Data Buffer is dependent on a full serial receive shift register. For this reason, the end of the last received frame's data will not be available from the buffer unless the receive channel is disabled through the Control Register Channel Enable bit. The remaining data in the serial receive shift register aligns and writes to the Frame Data Buffer.

Transmit Frame Storage and Transmission

The ARINC 708 products transmit frame data allocated in individual frames or grouped into single or multiple sweeps. Frame data is stored continuously in the respective channel's Data Buffer, with no unused bits inserted between the end of one frame and the beginning of the next. For frame sizes not divisible by a 16-bit word boundary, the frame data storage ends and subsequent frame data storage begins on a bit boundary within the respective memory location.

The transmit logic of the ARINC 708 products inserts a frame gap between the stop pulse of one frame and the start pulse of a subsequent frame in terms of the duration between frame start pulses, referred to as the *frame interval*. The *frame interval*, specified in microseconds, should be defined to include the duration to transmit the total number of bits for the frame, plus six bit-times for the sync pulses, and finally the number of microseconds of gap time to insert between the frames.

The calculation for frame interval is based on the standard ARINC 708 1MHz bus speed. For a standard 1600-bit frame (with 6 bits for the start and stop pulse duration), the duration required to transmit that frame is calculated as follows:

$$\text{Transmit Duration} = \frac{1}{1{,}000{,}000} \times (1600 + 6)$$

The frame interval can then be determined by the sum of the transmit duration (0.001606 seconds) and the desired frame gap. Using a frame gap of 1.6 milliseconds, the frame interval programmed for this standard frame size would be:

$$\text{Frame Interval} = 0.001606 + 0.0016 = 3206 \text{ microseconds}$$

An ARINC 708 *sweep* is considered a logical grouping of a predefined number of frames that fill a WXR display. A sweep can be programmed to a fixed number of frames for each individual channel, with the sweep size specified in terms of frames in the Transmit Sweep Size register.

The delay duration inserted between consecutive sweep transmissions is specified in microseconds through the Transmit Sweep Interval register; however, the minimum Sweep Interval between the Stop Pulse of the last frame in a sweep and the Start Pulse in the first frame of the next sweep is limited to the frame gap duration defined via the Frame Interval register.

Periodic Sweep Transmission

The Sweep Transmission feature supports periodic sweep transmission of all, or a portion of, a defined transmit Frame Data Buffer. The number of sweeps included in the repeated sweep transmission is specified through the Transmit Sweep Count register. The periodic transmission of the specified number of sweeps is enabled via the Control Register's Repeat bit.

Error Injection

The ARINC 708 products support transmit frame error injection on a frame-by-frame basis using the Ancillary Buffer assigned to the respective channel. Error injection includes transmission of an additional

bit above the specified frame size, a missing bit below the specified frame size, or inverted start and/or stop sync pulses. For a channel set in the transmit mode, each 16-bit location of the Ancillary Buffer allocated to the respective channel is assigned to the respective frame defined in the Frame Data Buffer allocated to that channel. Within each location, 4 bits are used to induce errors listed above in the transmission of the frame data, defined as follows:

15-4	3	2	1	0
Unused	Stop Sync Inversion When set the Stop Sync Pulse will be inverted	Start Sync Inversion When set the Start Sync Pulse will be inverted	Short Frame Error When set one bit will be removed from the end of the frame	Long Frame Error When set one bit will be added to the end of the frame

RP-708 Enhancements

The RoHS compliant RP-708 is a replacement for the P-708 and contains a few additional features not available on the P-708.

The revision of the firmware has been added to the RP-708 at byte offset 0x2C.

The RP-708 includes a temperature sensor to report PCB temperatures if desired. The sensor is accessed via registers at byte offset 0x30 and 0x34.

A security device is present on the RP-708 (as well as many Abaco Systems Avionics products) which involves no interaction on the part of the user. Bit 10 of Channel 0 and Channel 1 Control Registers (byte offsets 0x0 and 0x100 respectively) provides status of communication with this security device. When this bit is set, transmission and reception of ARINC 708 data is disabled. This bit should never be set. Contact Abaco Systems Avionics technical support if you are experiencing problems with the security device.

CHAPTER 8

P-708-SW Distribution and API

Overview

The Abaco Systems P-708-SW Distribution supplies an extensive software Application Programming Interface (API) for the supported ARINC 708 PMC products. API routines are supplied to setup the interface, configure channel attributes, and transmit and receive data for the most common 32-bit programming environments (Windows 7/Vista/XP, Linux, VxWorks and Integrity), and select 64-bit environments (Windows 7/Vista/XP and Linux).

API Source Files

This library of utility routines provides the ability to write your own programs to interface with the ARINC 708 product. They are written in C and delivered in a generic 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:

P708_API.C

This file contains the bulk of the API functionality. All of the utility routines that interact with the ARINC 708 hardware device reside within this file.

P708_API.H

Included in the file P708_API.C, this header file contains the majority of the ARINC 708 API constants, data types, and function prototypes for

application use, and should be included in all programs that call one or more API utility routines.

P708_GLB.H

This header file contains the majority of the local API variables and data structures.

AR_ERROR.H

This header file contains the error string constant definitions utilized by the API routine P708_Get_Error, describing each of the potential error codes returned by the ARINC 708 API utility routines.

P708_HW.H

This header file contains all of the API constants that define the hardware interface for the ARINC 708 PMC architecture.

FPGA_708.H and FPGA_RP708.H

These header files contain the firmware program downloaded to the respective ARINC 708 hardware during initialization of the board.

CEI_TYPES.H

This header file contains all of the constants that define the various data types for the respective operating system and compiler.

P708_WIN.C

This file contains the C routines that interface directly with the Windows operating system and low-level driver library. It should be included along with the compilation of P708_API.C when the compiler directive _WIN32 is defined (build for any Windows target).

P708_VXW.C

This file contains the routines that interface directly with the Abaco Systems Avionics Avionics generic VxWorks PCI/VxBus drivers and VxWorks kernel. It should be included along with the compilation of

P708_API.C when either of the compiler directives VXW_PCI_PPC or VXW_PCI_X86 is defined.

P708_INT.C

This file contains the routines that interface directly with the Abaco Systems Avionics Generic Integrity device driver file set. It should be included along with the compilation of P708_API.C when the compiler directive INTEGRITY_PCI_PPC is defined.

P708_LNX.C

This file contains the routines that interface directly with the Abaco Systems Avionics generic Linux Support Package. It should be included in your project along with the compilation of P708_API.C when either of the compiler directives _LINUX_X86_or _LINUX_PPC_ is defined.

P708_UTIL.C

This file contains several ARINC 708 Frame Record/Playback feature utilities designed for use with Windows operating systems. While these utilities are not a part of the ARINC 708 API, they are provided in generic C source to assist in your application development.

P708_SCH.C

This file contains utility routines specific to the ARINC 708 Protected Frame Update feature, supporting the ability for the application to update a specified number of frames in a scheduled sweep and avoid conflicting with the frame data actively being read by the hardware. See Appendix A, “Protected Frame Update Feature” for more information on this feature.

P708_API.DEF and P708_API64.DEF

These files contain the ARINC 708 API-specific external library references for use when building the API as a Dynamic Link Library under 32-bit and 64-bit Windows operating systems, respectively.

Windows Libraries

The P-708-SW distribution provides both 32-bit and 64-bit Windows DLLs. For Windows OS target implementation, all API function prototypes are declared "_stdcall". The ARINC 708 API library included in the installation is referenced as:

- P708_API.LIB 32-bit Microsoft VS 6.0 Library
- P708_API.DLL 32-bit Microsoft VS 6.0 DLL
- P708_API64.LIB 64-bit Microsoft VS2008 Library
- P708_API64.DLL 64-bit Microsoft VS2008 DLL

Included with the P-708-SW installation is the Abaco Systems 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

Programming with the ARINC 708 API Interface

Following the outline below, you can easily incorporate the ARINC 708 API into your application.

1. For your application to interface to any P-708-SW supported products the device must first be initialized. Invoke the P708_OPEN routine as described in the API Routines section.
2. Assign the characteristics of the channels if the default configuration is not appropriate. This is performed with multiple invocations of P708_SET_CHANNEL_CONFIG.
3. Once channel configuration is complete, invoke P708_GO to initiate data processing. In any case where the board is both transmitting and receiving for a self-test scenario, it is best to separately enable the receiver then transmitter using the OPERATIONAL_MODE option with the P708_SET_CHANNEL_CONFIG routine.
4. To transmit frame/sweep data invoke P708_WRITE_FRAME(…) and to receive frame/sweep data invoke P708_READ_FRAME(…).
5. When communication is complete, invoke P708_STOP to suspend active data processing. Subsequently, you may enable the channels again to restart data processing without reinitializing the interface.
6. On termination of the application, invoke P708_CLOSE to release all resources acquired during initialization. It is very important that all applications invoke P708_CLOSE upon termination; otherwise, the operating system will not release the memory acquired when the API was initialized.

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.

Time-tag Data Definition

The following API routines use the TIME_TAG_TYPE time-tag definition in providing the time-tag reference for received frame data:

- P708_READ_FRAME_DATA_T
- P708_READ_FRAMES

Under most operating systems, the TIME_TAG_TYPE is defined as a 64-bit integer value. For those operating systems or board support packages that do not support a 64-bit integer, the TIME_TAG_TYPE can be compiled as a 32-bit pointer. In these cases only the lower 32 bits of the received frame time-tag will be provided.

API Defined Data Types

The following data types are defined for use with the ARINC 708 API:

SINT32	integer
UINT32	unsigned integer
PINT32	unsigned integer *
SINT16	short integer
UINT16	unsigned short integer
PINT16	unsigned short integer *
SINT8	char
UINT8	unsigned char
PINT8	unsigned char *
SP_FLOAT	single-precision float

Return Status Values

The following return status values are used by the ARINC 708 API routines. They are defined in the C header file P708_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 received
ARS_NORMAL	1	Normal successful completion

C Constant	Value	Constant Definition
ARS_GOTDATA	4	Data was received
ARS_INVHARVAL	1003	Invalid configuration value
ARS_XMITOVRFLO	1004	Transmit buffer overflow
ARS_INVBOARD	1005	Invalid board argument
ARS_BADLOAD	1007	Firmware download failure
ARS_INVARG	1019	General invalid argument value
ARS_DRIVERFAIL	1022	Windows device driver error
ARS_BAD_STATIC	1027	SRAM Memory Test failure
ARS_WRAP_DATA_FAIL	1031	BIT wrap test data read-back fail
ARS_RX_OVERRUN	1037	Receive buffer overrun detected

Example Applications – Summary

Example applications for 32-bit and 64-bit Windows are available beneath the P-708-SW distribution's \Examples folder. Those examples are described in detail below.

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 *Abaco P-708-SW* and then *Test Configuration*.

The TST_CFG executable is located in the following:

\Program Files\Condor Engineering\P-708-SW\Examples\C

The default execution of TST_CFG.EXE is to internally wrap frame transmission of three standard (1600 bit) frames of ARINC 708 data with channel 1 transmitting and channel 0 receiving. Command line arguments are available with this example, as described below:

/BDx assign the ARINC 708 PMC Device ID as 'x'

/EXT disable internal wrap on both channels

/ALT transmit 3 frames of an alternate frame size (1200 bits)

/SWP bypass the wrap examples and program the board to transmit a sweep of 720 frames continuously from both channels.

/DBS transmit a 720-frame sweep using a two buffer sources in a double-buffered fashion, allowing for protected buffer updates.

Within TST_CNFG.C are application-style routines demonstrating use of the API routines for the ARINC 708 protocol:

testStandard708	an internal wrap test designed to demonstrate basic ARINC 708 API usage, using internal wrap to transmit and receive three standard 1600-bit ARINC 708 frames.
testAlternate708	an internal wrap test designed to demonstrate basic ARINC 708 API usage, using internal wrap to transmit and receive three non-standard 1200-bit frames.
sweepTransmit708	a demonstration of periodic sweep transmission of a full 180˚ sweep of 720 1600-bit frames.
doubleBufferedSweepTransmit708	a demonstration of periodic sweep transmission of a full 180˚ sweep of 720 1600-bit frames, using a double-buffered methodology.

P708ECHO.C and P708UTIL.C

The example source files P708ECHO.C and P708UTIL.C/.H are provided to demonstrate the ARINC 708 frame data recording and playback capabilities provided for Windows application usage. To access this example executable under the Windows operating system, navigate to the following folder using the Windows Explorer:

\Program Files\Condor Engineering\P-708-SW\Examples\C

The default execution of P708ECHO.EXE is to transmit on channel 0 and receive on channel 1 with device ID 0, using a frame size of 1600 bits, a single sweep recording of 720 frames, and binary data logging to a file named ‘p708echo.dat’.

Modify the batch file P708ECHO.BAT with the appropriate command line arguments, as described below:

/BDx	assign the ARINC 708 Device ID as ‘x’
/TX1	reverse channel assignments, TX 1 - RX 0
/TEXT	log received data in an ASCII text file
/PLAYBACK	transmit the contents of a BINARY file

/FILE	specify a data file to use (prompted)
/NOLOG	disable received data logging to file
/NOECHO	disable transmission of the received data
/FSnnnn	assign a frame size other than 1600 bits
/FCnnnn	assign a frame count other than 720 frames

The main routine within P708ECHO.C and the utilities provided in P708UTIL.C demonstrate typical uses of the ARINC 708 API routines for the ARINC 708 protocol:

p708_xmit_data_file	transmits the contents of the specified binary log file.
p708_bin_dump_frames	writes the specified number of frames to the specified output file in binary format.
p708_bin_read_frame	reads a single frame from the specified input file, content of which must be in a binary format.
p708_dump_frames	writes the specified number of frames to the specified output file in an ASCII text format.

SINGLE_FRAME_SWEEP.C

This file contains an example application demonstrating how to use a timer-protected frame buffer update scheme to replicate the operations of the IP-708.

The routine testSingleFrameUpdate() uses a single frame sweep, updated to coincide with the period where the ARINC 708 hardware is not reading the frame data, to support application transmission of a full sweep on a frame-by-frame basis. The mechanism to update the frame data when the hardware is idle uses a special API function set that monitors the transmit frame counter. When the transmit frame counter changes, the API has a brief period in which to update the frame data. Successful operation of this feature assumes the sweep interval provides a minimum number of milliseconds between the end of the frame's transmission and the beginning of the next transmission of that frame. In this example the sweep interval must be 2 milliseconds larger than the duration required to transmit the frame.

.NET Development Support

Support for Microsoft Visual Studio .NET programming languages is available through the use of a Visual Basic.NET Class "wrapper library" called A708ClassLib.dll, invoking routines from the standard API C library p708_api.dll. The .NET solution for this library and an example C# application are available beneath the \Examples\Net folder. Note the wrapper library A708ClassLib.dll is provided for use with 32-bit applications. If you wish to implement this API wrapper method for a 64-bit application, you must modify the DLL reference in the "a708api" class declaration in A708ClassLib.vb and modify the target DLL from "P708_API.dll" to "P708_API64.dll".

API Routines - Summary

The routines provided in the API supporting the P-708 device features are defined in the following pages, categorized and summarized:

Initialization and Control Routines

p708_board_test	Verifies the P-708 data processing capabilities via internal/external data wrap.
p708_bypass_wrap_test	Controls conditional execution of the ARINC 708 internal wrap test invoked from within *p708_initialize_device*.
p708_initialize_api	Initializes the P-708 device API.
p708_initialize_device	Initializes the P-708 device to the default state.
p708_open	Acquires the resources for, and initializes the P-708 device.

Device Control Routines

p708_go	Enables P-708 ARINC data processing.
p708_reset	Disables P-708 ARINC data processing and initializes the device to the default state.
p708_stop	Disables P-708 ARINC data processing.

Termination Routines

p708_close	Releases all resources for the specified device.

Configuration Routines

p708_get_channel_config	Retrieves the value of a bit field or register.
p708_set_channel_config	Assigns the value of a bit field or register.

Receive Data Processing Routines

p708_read_frames	Retrieves the next ARINC 708 frame from the receive buffer, used specifically for standard sized 1600-bit frames.
p708_read_frame_data	Retrieves the next ARINC 708 frame from the receive buffer, used for any size frame.
p708_read_frame_data_t	Retrieves the next ARINC 708 frame and respective time-tag from the receive buffer, used for any size frame.

Transmit Data Processing Routines

p708_write_frames	Defines and modifies standard 1600-bit ARINC 708 frame and error injection data.
p708_write_ei_data	Defines frame error injection data.
p708_write_frame_data	Defines any size ARINC 708 frame data.
p708_write_frame_data_wei	Defines any size ARINC 708 frame and error injection data.
p708_update_ei_data	Modifies existing frame error injection data.
p708_update_frame_data	Modifies existing ARINC 708 frame data already defined in the transmit buffer.
p708_update_frame_data_wei	Modifies existing ARINC 708 frame and error injection data already defined in the transmit buffers.

Information and Status Routines

p708_get_error	Retrieves a message string associated with a given error status code.

Utility Routines

p708_execute_bit	Verifies the ARINC 708 PMC's operational state through various data wrap and memory tests.
p708_read_device	Low-level device read-access utility
p708_set_multithread_protect	Enable/disable multithread access protection to all API routines accessing the hardware interface of the device
p708_wait	Delays the calling application for the specified number of seconds.
p708_write_device	Low-level device write-access utility
p708_version	Retrieves the current software version number of the ARINC 708 API.

P708_BOARD_TEST

Syntax

SINT32 p708_board_test (SINT32 board, SINT32 testType)

Description

This routine performs a two frame (1600-bit) internal or external wrap test with channel 0 configured to transmit and channel 1 configured to receive.

Note

Any invocation of this routine while either ARINC 708 channel is connected to an actively transmitting LRU may result in a false failure status.

Return Value

ARS_NORMAL	routine was successful.
ARS_WRAP_DATA_FAIL	wrap frame data pattern mismatch.
ARS_INVBOARD	uninitialized or invalid *board* value.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.
SINT32 testType	(input) type of test to execute. Valid values for this parameter are: INTERNAL_WRAP (4) EXTERNAL_WRAP (5)

P708_BYPASS_WRAP_TEST

Syntax

SINT32 p708_bypass_wrap_test (SINT32 board , SINT32 bypass)

Description

This routine defines an internal P-708 API flag used to control the invocation of the API internal utility function P708_UTL_WRAP_TEST routine during execution of P708_OPEN.

The default state of the bypass parameter is ON, indicating this internal wrap test will not be executed during the initialization sequence executed via P708_OPEN. Execution of this routine does not require an actively open API session, and should occur prior to the invocation of P708_OPEN with the bypass parameter set to OFF, if you wish the API to perform the internal wrap test as a part of the device initialization sequence. If invoked, P708_UTL_WRAP_TEST will perform a two-frame internal transmit/receive wrap test across the ARINC 708 PMC channel pair.

Note

Any invocation of P708_OPEN with the internal wrap test execution enabled, while either ARINC 708 channel is connected to an actively transmitting LRU, may result in a false failure status.

Return Value

ARS_NORMAL	routine was successful.
ARS_INVBOARD	invalid *board* value was supplied.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.
SINT32 bypass	(input) controls whether or not to bypass execution of the internal wrap test when the routine P708_OPEN is invoked. Valid values for this parameter are: AR_ON (7) bypass internal wrap test AR_OFF (8) execute internal wrap test

P708_CLOSE

Syntax

SINT32 p708_close (SINT32 board)

Description

This routine releases all resources acquired during the initialization of the specified device and closes the session. Once this routine has been executed, invoking P-708 API routines other than those that specifically indicate support without an actively open session will result the return of an error status.

Return Value

ARS_NORMAL	routine was successful.
ARS_INVBOARD	uninitialized or invalid *board* value.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.

P708_EXECUTE_BIT

Syntax

SINT32 p708_execute_bit (SINT32 board, SINT32 testType)

Description

This routine performs the functionality normally associated with board-level Initiated-BIT. Testing ranges from a full SRAM memory test to verification of an ARINC 708 frame wrap on the specified device.

Note

Any invocation of this routine with a *testType* parameter value assigned to anything other than AR_BIT_PERIODIC or AR_BIT_PARTIAL_SRAM while either ARINC 708 channel is connected to an actively transmitting LRU may result in a false failure status.

Return Value

ARS_NORMAL	routine was successful.
ARS_BAD_STATIC	device SRAM test write/read/verify failure.
ARS_WRAP_DATA_FAIL	ARINC 708 wrap test frame data pattern mismatch.
ARS_INVBOARD	uninitialized or invalid *board* value.
ARS_INVARG	invalid *testType* parameter value provided, or no free/unused SRAM was available for a non-destructive memory test execution.

Arguments

SINT32 board — (input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.

SINT32 testType — (input) type of test to execute, defined as follows:

- AR_BIT_BASIC_STARTUP (0) invokes device initialization, (including an unused SRAM memory test), via invocation of P708_INITIALIZE_DEVICE.
- AR_BIT_FULL_STARTUP (1) invokes device initialization, (including an unused SRAM memory test), via invocation of P708_INITIALIZE_DEVICE, followed by a two frame (1600-bit) internal wrap test with channel 0 configured to transmit and channel 1 configured to receive.
- AR_BIT_PERIODIC (2) invokes a non-destructive pattern test of select unused SRAM locations.
- AR_BIT_INT_LOOPBACK (3) invokes a two frame (1600-bit each) internal wrap test with channel 0 configured to transmit and channel 1 configured to receive.
- AR_BIT_EXT_LOOPBACK (4) invokes a two frame (1600-bit each) external wrap test with channel 0 configured to transmit and channel 1 configured to receive.

AR_BIT_PARTIAL_SRAM (8) invokes a short non-destructive pattern test of select unused SRAM locations. If all of SRAM is allocated to either channel's data buffers, this invocation returns a status value of ARS_INVARG.

AR_BIT_FULL_SRAM (9) invokes a destructive six pattern test of all SRAM locations.

AR_BIT_SELECT_SRAM_MIN to AR_BIT_SELECT_SRAM_MAX (100 to 611) invokes a destructive test of a select block of SRAM, parsed into 512 blocks of 512 locations each.

P708_GET_BASE_ADDR

Syntax

CEI_ULONG p708_get_base_addr (SINT32 board)

Description

This routine returns the allocated/virtual base address for the local memory region of the specified device (PCI BAR 2 region).

Return Value

Any positive value exceeding $80000 is the base address of the device.

ARS_INVBOARD	an uninitialized or invalid *board* value.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.

P708_GET_BOARDTYPE

Syntax

SINT32 p708_get_boardtype (SINT32 board)

Description

This routine returns the board type for the specified device.

Return Value

A constant value less than 1000 indicates the type of ARINC 708 board detected:

	PMC_708 (decimal 20)
	RP_708 (decimal 36)
ARS_INVBOARD	uninitialized or invalid *board* value.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.

P708_GET_CHANNEL_CONFIG

Syntax

SINT32 p708_get_channel_config (SINT32 board, SINT32 channel, UINT32 option, PINT32 data)

Description

This routine returns the value/state of the specified control register bit fields and other miscellaneous hardware registers.

Return Value

ARS_NORMAL	routine was successful.
ARS_INVBOARD	uninitialized or invalid *board* value.
ARS_INVHARVAL	Invalid *channel* parameter value
ARS_INVARG	Invalid *option* parameter value

Arguments

SINT32 board — (input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.

SINT32 channel — (input) ARINC 708 channel to access. Valid values are 0 and 1.

UINT32 option — (input) bit field/register about which to return the current state/value:

ANCILLARY_BUFFER_SIZE	ancillary buffer size
ANCILLARY_BUFFER_START	ancillary buffer starting address
ANCILLARY_BUFFER_END	ancillary buffer ending address
DATA_BUFFER_SIZE	frame data buffer size
DATA_BUFFER_START	frame data buffer starting address
DATA_BUFFER_END	frame data buffer ending address
A708_ERROR_INJECTION	error injection enable bit
FRAME_COUNT	frame count since reset
FRAME_INTERVAL	duration between start of each frame
FRAME_GAP	duration between transmitted frames
FRAME_SIZE	frame size
OPERATIONAL_MODE	data processing enable/disable bit
OPERATIONAL_DIRECTION	transmit/receive selection bit
RECEIVE_BIT_ERROR	receive bit error field state
SWEEP_SIZE	number of frames in each sweep
SWEEP_COUNT	number of sweeps buffer to repeat
SWEEP_INTERVAL	duration between sweeps
SWEEP_REPEAT	transmit buffer repeat sweep state
SELF_TEST	external operation bit

PINT32 data — (output) the current state/value of the requested bit field/register:

If the requested item is ANCILLARY_BUFFER_SIZE (4), this routine returns the computed value of the respective ARINC 708 PMC channel's ancillary data buffer size. This value is provided in terms of the storage capacity of the buffer in words, based on the defined Ancillary Data Start/Stop Address Register settings.

If the requested item is ANCILLARY_BUFFER_START (5), this routine returns the current value of the respective ARINC 708 PMC channel's Ancillary Data Start Address Register.

If the requested item is ANCILLARY_BUFFER_END (6), this routine returns the current value of the respective ARINC 708 PMC channel's Ancillary Data End Address Register.

If the requested item is DATA_BUFFER_SIZE (1), this routine returns the computed value of the respective ARINC 708 PMC channel's frame data buffer size. This value is provided in terms of the storage capacity of the buffer in frames, based on the Frame Bit Count and Frame Data Start/Stop Address Register settings.

If the requested item is DATA_BUFFER_START (2), this routine returns the current value of the respective ARINC 708 PMC channel's Frame Data Start Address Register.

If the requested item is DATA_BUFFER_END (3), this routine returns the current value of the respective ARINC 708 PMC channel's Frame Data End Address Register.

If the requested item is A708_ERROR_INJECTION (19), this routine returns the current state of the respective ARINC 708 PMC channel's Control Register Error Injection Enable Bit, indicated as either AR_ON (7) or AR_OFF (8).

If the requested item is FRAME_COUNT (18), this routine returns the current value of the respective ARINC 708 PMC channel's Frame Count Register. This value indicates how many frames have been transmitted or received on the respective channel since the channel was enabled.

If the requested item is FRAME_INTERVAL (10), this routine returns the current value of the respective ARINC 708 PMC channel's Transmit Frame Interval Register. This value indicates the duration between the Start Sync Pulse of successive frames, in microseconds.

If the requested item is FRAME_GAP (8), this routine returns the calculated frame gap value. The frame gap value is defined as the duration between the Stop Sync Pulse and Start Sync Pulse of successive frames, in microseconds. This value is derived from the value of the Frame Interval Register, less the frame size (in bits) and Start/Stop Sync Pulse duration.

If the requested item is FRAME_SIZE (9), this routine returns the current value of the respective ARINC 708 PMC channel's Frame Bit Count Register.

If the requested item is OPERATIONAL_MODE (13), this routine returns the current state of the respective ARINC 708 PMC channel's Control Register Enable Bit, indicated as either AR_RUN (1) or AR_STOP (0).

If the requested item is OPERATIONAL_DIRECTION (14), this routine returns the current state of the respective ARINC 708 PMC channel's Control Register Direction Bit, indicated as either AR_TRANSMIT (1) or AR_RECEIVE (0).

If the requested item is RECEIVE_BIT_ERROR (16), this routine returns the current state of the respective ARINC 708 PMC channel's Control Register receive error bits (Under Bit Count & Over Bit Count), indicated as:

NO_FRAME_ERROR	0	No Error Detected
LONG_FRAME_ERROR	1	Over Bit Count
SHORT_FRAME_ERROR	2	Under Bit Count
MULTIPLE_FRAME_ERROR	3	Both Over & Under

If the requested item is SWEEP_SIZE (9), this routine returns the current value of the respective ARINC 708 PMC channel's Transmit Sweep Frame Count Register. This value indicates the number of frames to be allocated to the transmission of each sweep.

If the requested item is SWEEP_INTERVAL (10), this routine returns the current value of the respective ARINC 708 PMC channel's Sweep Interval Register. This value indicates the number of microseconds inserted between the transmission of each sweep frame grouping.

If the requested item is SWEEP_REPEAT (11), this routine returns the current state of the respective ARINC 708 PMC channel's Control Register Sweep Repeat Bit, indicated as either AR_ON (7) or AR_OFF (8).

If the requested item is SWEEP_COUNT (12), this routine returns the current value of the respective ARINC 708 PMC channel's Sweep Count Register. This value indicates how many sweep frame groupings are repeatedly transmitted if the Sweep Repeat Bit is enabled.

If the requested item is SELF_TEST (17), this routine returns the current state of the respective ARINC 708 PMC channel's Control Register Internal/External Transmit Select Bit, indicated as either EXTERNAL_WRAP (0) or INTERNAL_WRAP (1).

P708_GET_ERROR

Syntax

PINT8 * p708_get_error (SINT32 status)

Description

All of the API routines that interact with an ARINC 708 PMC device return status values, a majority of which indicate an error condition. When supplied with such an error value, this routine returns a pointer to an API-supplied character string describing the error.

Review the section, “Return Status Values”, for the current list of possible error codes and their explanations.

Return Value

A pointer to the error message character string.

Arguments

SINT32 status	(input) a status value returned by any of the ARINC 708 API routines.

P708_GO

Syntax

SINT32 p708_go (SINT32 board)

Description

This routine sets both of the ARINC 708 PMC channel's Control Register Enable bits to "enabled". If set, it also resets both channel's Control Register Halt bits to zero. All message processing is activated on execution of this routine, and all internal frame and address counters reset for any channel defined for reception.

Return Value

ARS_NORMAL	routine was successful.
ARS_INVBOARD	uninitialized board or invalid *board* value.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.

P708_INITIALIZE_API

Syntax

SINT32 p708_initialize_api (SINT32 board)

Description

This routine opens a session with the specified ARINC 708 PMC device and downloads the FPGA firmware, resetting the device to an initial power-up state.

Return Value

ARS_NORMAL	routine was successful.
ARS_INVBOARD	an invalid *board* value.
ARS_BADLOAD	device firmware download failed.
ARS_DRIVERFAIL	device driver failed to open a session or acquire the necessary resources (Windows and Linux operating systems, only).
ARS_FAILURE	device driver failed to open a session or acquire the necessary resources (VxWorks and Integrity operating systems, only).

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.

P708_INITIALIZE_DEVICE

Syntax

SINT32 p708_initialize_device (SINT32 board)

Description

This routine configures the ARINC 708 PMC to a default state, with both channels defined as follows:

- Data processing disabled.
- Defined as receivers and set for external operation.
- Frame and Ancillary Data Buffer storage allocation set for a single receive frame.
- All data buffer pointers reset to buffer start.
- Frame Size defined as 1600 bits.
- Frame Interval defined for four bit gap duration between frames.
- Sweep Size defined as a single frame.
- Sweep Interval defined for 500 milliseconds re-transmission rate.
- Sweep Count defined as a single sweep.
- Sweep Repeat disabled.

Subsequent to setup of these channel configurations, this routine will execute a brief memory test of the onboard SRAM.

Return Value

ARS_NORMAL	routine was successful.
ARS_INVBOARD	uninitialized or invalid *board* value.
ARS_BAD_STATIC	unused SRAM test write/read/verify failure.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.

P708_OPEN

Syntax

SINT32 p708_open (SINT32 board)

Description

This routine acquires the address space/resources allocated to the device, opens a session with the device, and invokes an initialization/reset procedure. Following API and device initialization, optional invocation of the internal API routine P708_UTL_WRAP_TEST can provide verification of internal frame wrap operation, (execution of this test is controlled via invocation of P708_BYPASS_WRAP_TEST, disable by default).

See the routine descriptions under P708_INITIALIZE_API and P708_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 was successful.
ARS_INVBOARD	invalid *board* value.
ARS_DRIVERFAIL	device driver failed to open a session or acquire the necessary resources (Windows and Linux operating systems, only).
ARS_FAILURE	device driver failed to open a session or acquire the necessary resources (VxWorks and Integrity operating systems, only).
ARS_BADLOAD	device firmware download failed.
ARS_BAD_STATIC	unused SRAM test write/read/verify failure.
ARS_WRAP_DATA_FAIL	ARINC 708 internal wrap test frame data pattern mismatch.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.

P708_READ_FRAMES

Syntax

SINT32 p708_read_frames (SINT32 board, SINT32 channel, PINT32 frameCount, PINT32 bitErrorStatus, pRECEIVE_FRAME_TYPE data)

Description

This routine retrieves a specified number of available ARINC 708 frames and time-tag information from the requested receive channel buffer and copies them to the desired destination. If the return value for the *frameCount* parameter equals the value supplied, additional unread frame data may be available in the receive buffer. This routine can be used only with standard 1600-bit ARINC 708 frame reception. Since the validity of the last word of the last received frame is not guaranteed, the last received frame will not be included in the set of frames returned from this routine.

Return Value

ARS_GOTDATA	one or more ARINC 708 frames were retrieved from the buffer.
ARS_NODATA	no unread ARINC 708 frames were available in the buffer.
ARS_RX_OVERRUN	a receive buffer overflow condition was detected and the location of the most stale frame data cannot be determined.
ARS_INVBOARD	uninitialized board or invalid *board* value.
ARS_INVARG	invalid *channel* or *frameCount* parameter value.
ARS_INVHARVAL	the specified *channel* is invalid or the frame size isn't defined to be 1600 bits.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.
SINT32 channel	(input) ARINC 708 channel to access. Valid values are 0 and 1.
PINT32 frameCount	(input/output) As an input, this parameter specifies the number of frames to read; as an output this parameter indicates the actual number of frames read from the buffer.

PINT32 bitErrorStatus (output) A representation of the combined state of the Receive Error Bits from the Control Register as defined upon entering the routine.

pRECEIVE_FRAME_TYPE data (output) The frame and time-tag data retrieved from the P-708 receiver buffers; formatted as follows:

```
struct { TIME_TAG_TYPE timeTag;
         UINT32        controlWords[4];
         UINT32        binData[512];
}
```

P708_READ_FRAME_DATA

Syntax

SINT32 p708_read_frame_data (SINT32 board, SINT32 channel, PINT32 frameCount, PINT32 data)

Description

This routine retrieves a specified number of available ARINC 708 frames from the requested receive channel buffer and copies them to the desired destination. If the return value for the *frameCount* parameter equals the value supplied, additional unread frame data may be available in the receive buffer. This routine can be used with any size frame. Since the validity of the last word of the last received frame is not guaranteed, the last received frame will not be included in the set of frames returned from this routine.

Return Value

ARS_GOTDATA	one or more ARINC 708 frames were retrieved from the buffer.
ARS_NODATA	no unread ARINC 708 frames were available in the buffer.
ARS_RX_OVERRUN	a receive buffer overflow condition was detected and the location of the most stale frame data cannot be determined.
ARS_INVBOARD	uninitialized board or invalid *board* value.
ARS_INVARG	invalid *channel* or *frameCount* parameter value.
ARS_INVHARVAL	the specified *channel* is invalid.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.
SINT32 channel	(input) ARINC 708 channel to access. Valid values are 0 and 1.
PINT32 frameCount	(input/output) As an input, this parameter specifies the number of frames to read; as an output this parameter indicates the actual number of frames read from the buffer.
PINT32 data	(output) Array to store frame data.

P708_READ_FRAME_DATA_T

Syntax

SINT32 p708_read_frame_data (SINT32 board, SINT32 channel, PINT32 frameCount, PINT32 data)

Description

This routine retrieves a specified number of available ARINC 708 frames with their respective time-tags from the requested receive channel buffer and copies them to the desired destinations. If the return value for the *frameCount* parameter equals the value supplied, additional unread frame data may be available in the receive buffer. This routine can be used with any size frame. Since the validity of the last word of the last received frame is not guaranteed, the last received frame will not be included in the set of frames returned from this routine.

Return Value

ARS_GOTDATA	one or more ARINC 708 frames were retrieved from the buffer.
ARS_NODATA	no unread ARINC 708 frames were available in the buffer.
ARS_RX_OVERRUN	a receive buffer overflow condition was detected and the location of the most stale frame data cannot be determined.
ARS_INVBOARD	uninitialized board or invalid *board* value.
ARS_INVARG	invalid *channel* or *frameCount* parameter value.
ARS_INVHARVAL	the specified *channel* is invalid.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.
SINT32 channel	(input) ARINC 708 channel to access. Valid values are 0 and 1.
PINT32 frameCount	(input/output) As an input, this parameter specifies the number of frames to read; as an output this parameter indicates the actual number of frames read from the buffer.
PINT32 data	(output) Array to store frame data.

pTIME_TAG_TYPE timeTag (output) Array to store time-tag data.

P708_RESET

Syntax

SINT32 p708_reset (SINT32board)

Description

This routine invokes the p708_initialize_device routine. As a result of executing this routine, all data buffers are reset and any data contained therein will be lost.

Return Value

ARS_NORMAL	routine was successful.
ARS_INVBOARD	uninitialized or invalid *board* value.
ARS_BAD_STATIC	device SRAM test write/read/verify failure.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.

P708_SET_CHANNEL_CONFIG

Syntax

SINT32 p708_set_channel_config (SINT32 board, SINT32 channel, UINT32 option, UINT32 data)

Description

This routine assigns the value/state of the specified control register bit fields or other miscellaneous hardware register.

Return Value

ARS_NORMAL	routine was successful.
ARS_INVBOARD	uninitialized or invalid *board* value.
ARS_INVHARVAL	Invalid *channel* parameter value
ARS_INVARG	Invalid *option* or *data* parameter value

Arguments

SINT32 board — (input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.

SINT32 channel — (input) ARINC 708 channel to access. Valid values are 0 and 1.

UINT32 option — (input) bit field/register to assign the state/value:

ANCILLARY_BUFFER_SIZE	ancillary buffer size
ANCILLARY_BUFFER_START	ancillary buffer starting address
ANCILLARY_BUFFER_END	ancillary buffer ending address
DATA_BUFFER_SIZE	frame data buffer size
DATA_BUFFER_START	frame data buffer starting address
DATA_BUFFER_END	frame data buffer ending address
A708_ERROR_INJECTION	error injection enable bit
FRAME_INTERVAL	duration between start of each frame
FRAME_GAP	duration between transmitted frames
FRAME_SIZE	frame size
OPERATIONAL_MODE	data processing enable/disable bit
OPERATIONAL_DIRECTION	transmit/receive selection bit
SWEEP_SIZE	number of frames in each sweep
SWEEP_COUNT	number of sweeps buffer to repeat
SWEEP_INTERVAL	duration between sweeps
SWEEP_REPEAT	transmit buffer repeat sweep state
SELF_TEST	external operation bit
BUFFER_RESET	reset data and ancillary buffers

UINT32 data — (input) the state/value to assign to the requested bit field/register:

If the specified item is ANCILLARY_BUFFER_SIZE (4), this routine determines and assigns the respective ARINC 708 PMC channel's ancillary data buffer start and stop address register values based on the size value provided. The value provided should be specified in terms of the desired storage capacity of the buffer in words, with a valid range from 1-2FFEh.

If the specified item is ANCILLARY_BUFFER_START (5), this routine assigns the supplied word offset to the respective ARINC 708 PMC channel's Ancillary Data Start Address Register. The valid range is 0-2FFEh.

If the specified item is ANCILLARY_BUFFER_END (6), this routine assigns the supplied word offset to the respective ARINC 708 PMC channel's Ancillary Data Stop Address Register. The valid range is 1-2FFFh.

If the requested item is DATA_BUFFER_SIZE (1), this routine assigns the respective ARINC 708 PMC channel's frame data buffer start and stop address register values based on the number of frames requested. The value provided will determine the resulting storage capacity of the buffer, in frames, based on the defined frame size and the amount of unused data buffer RAM available. The existing allocation for the alternate channel may limit the actual data buffer size allocation with this invocation. The valid range is 1-7FFFEh.

If the requested item is DATA_BUFFER_START (2), this routine assigns the supplied word offset to the respective ARINC 708 PMC channel's Frame Data Start Address Register. The valid range is 0-7FFFEh.

If the requested item is DATA_BUFFER_END (3), this routine assigns the supplied word offset to the respective ARINC 708 PMC channel's Frame Data Stop Address Register. The valid range is 1-7FFFFh.

If the requested item is A708_ERROR_INJECTION (19), this routine assigns the state of the respective ARINC 708 PMC channel's Control Register Error Injection Enable Bit to either enabled (AR_ON - 7) or disabled (AR_OFF - 8).

If the requested item is FRAME_INTERVAL (7), this routine assigns the supplied value to the respective ARINC 708 PMC channel's Transmit Frame Interval Register. This value specifies the duration between the Stop Sync Pulses of successive frames, in microseconds.

If the requested item is FRAME_GAP (8), this routine will determine the appropriate Frame Interval value and assign the supplied value to the respective ARINC 708 PMC channel's Transmit Frame Interval Register. The frame gap value specifies the duration between the Stop Sync Pulse and Start Sync Pulse of successive frames, in microseconds. This value is added to the frame size (in bits) and Start/Stop Sync Pulse

duration, with the resulting value applied to the respective Transmit Frame Interval Register.

If the requested item is OPERATIONAL_MODE (13), this routine assigns the current state of the respective ARINC 708 PMC channel's Control Register Enable Bit to either enabled (AR_RUN – 1) or disabled (AR_STOP – 0 or AR_TX_HALT – 2). For a data value of AR_TX_HALT, the respective channel's Control Register Transmit Terminate bit will be additionally set, immediately terminating transmission regardless of the state of the current frame transmission. If a data value of AR_RUN is specified for any channel setup for reception, all of the internal frame counters and buffer address indices for that channel will be reset to an initial state.

If the requested item is OPERATIONAL_DIRECTION (14), this routine assigns the current state of the respective ARINC 708 PMC channel's Control Register Direction Bit to either transmit (AR_TRANSMIT – 1) or receive (AR_RECEIVE – 0). In either case, all of the internal frame counters and buffer address indices for that channel will be reset to an initial state.

If the requested item is SWEEP_SIZE (9), this routine assigns the supplied value to the respective ARINC 708 PMC channel's Transmit Sweep Frame Count Register. This value indicates the number of frames to be allocated to the transmission of each sweep. The valid range for this value is dependent on the frame size and data buffer allocation for the respective channel.

If the requested item is SWEEP_INTERVAL (10), this routine assigns the supplied value of the respective ARINC 708 PMC channel's Sweep Interval Register. This value indicates the number of microseconds inserted between the transmission of each sweep frame grouping.

If the requested item is SWEEP_REPEAT (11), this routine assigns the state of the respective ARINC 708 PMC channel's Control Register Sweep Repeat Bit to either enabled (AR_ON - 7) or disabled (AR_OFF - 8).

If the requested item is SWEEP_COUNT (12), this routine assigns the supplied value of the respective ARINC 708 PMC channel's Sweep Count Register. This value indicates how many sweep frame groupings are repeatedly transmitted if the Sweep Repeat Bit is enabled. The valid range for this value is dependent on the number of sweeps defined in the respective channel's data buffer.

If the requested item is SELF_TEST (17), this routine assigns the state of the respective ARINC 708 PMC channel's Control Register External Operation Bit to either external operation via EXTERNAL_WRAP (0) or internal operation via INTERNAL_WRAP (1). The ability to configure the ARINC 708 PMC to internally wrap transmitted data requires one channel to be defined as Receive, the other channel to be defined as

Transmit, and both channels External Operation Bit to be reset using the INTERNAL_WRAP option. In this state, no external data is received or transmitted through either channel's transceiver.

P708_SET_MULTITHREAD_PROTECT

Syntax

SINT32 p708_set_multithread_protect (SINT32 board, SINT32 state)

Description

This routine controls the use of mutex/semaphore protection around all device and 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.

Execution of this routine does not require an actively open API session, but should occur prior to or immediately after the invocation of P708_OPEN for any multi-threaded application.

Return Value

ARS_NORMAL	routine was successful.
ARS_INVBOARD	invalid *board* value.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.
SINT32 state	(input) Multi-thread protection setting; AR_ON enables mutex/semaphore protection for all devices. AR_OFF disables mutex/ semaphore protection for all devices.

P708_STOP

Syntax

short p708_stop (short board)

Description

This routine assigns the global enable register Global Enable bit to be *disabled* for the specified device. Upon execution of this routine, all active message processing is immediately terminated and the internal frame counters and data buffer indices are reset.

Return Value

ARS_NORMAL	routine was successful.
ARS_INVBOARD	an uninitialized board or invalid *board* value.

Arguments

short board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.

P708_UPDATE_FRAME_DATA

Syntax

SINT32 p708_update_frame_data (SINT32 board, SINT32 channel, SINT32 startFrame, SINT32 frameCount, PINT32 data)

Description

This routine overwrites the specified number of frames in the Frame Data Buffer for the specified channel.

Return Value

ARS_NORMAL	routine was successful.
ARS_INVBOARD	uninitialized board or invalid *board* value.
ARS_INVARG	invalid *channel* parameter value or the specified *frameCount* and/or *startFrame* aren't currently allocated in the transmit frame data buffer.
ARS_INVHARVAL	the specified *channel* is not configured to transmit.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.
SINT32 channel	(input) ARINC 708 channel to access. Valid values are 0 and 1.
SINT32 startFrame	(input) This parameter specifies the frame offset at which to begin overwriting frames in the transmit Frame Data Buffer.
SINT32 frameCount	(input) This parameter specifies the number of frames to overwrite in the transmit Frame Data Buffer.
PINT32 data	(input) This parameter references the data to be placed into the respective P-708 Frame Data Buffer.

P708_UPDATE_FRAME_DATA_WEI

Syntax

SINT32 p708_update_frame_data_wei (SINT32 board, SINT32 channel, SINT32 startFrame, SINT32 frameCount, PINT32 data, PINT32 eiData)

Description

This routine overwrites the specified number of frames and error injection words in the respective data buffers for the specified channel.

Return Value

ARS_NORMAL	routine was successful.
ARS_INVBOARD	uninitialized board or invalid *board* value.
ARS_INVARG	invalid *channel* parameter value or the specified locations referenced by *frameCount* and/or *startFrame* aren't currently allocated in the respective transmit and/or ancillary data buffers.
ARS_INVHARVAL	the specified *channel* is not configured to transmit.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.
SINT32 channel	(input) ARINC 708 channel to access. Valid values are 0 and 1.
SINT32 startFrame	(input) This parameter specifies the frame offset at which to begin overwriting data in the transmit data buffers.
SINT32 frameCount	(input) This parameter specifies the number of frames and error injection words to overwrite in the transmit data buffers.
PINT32 data	(input) This parameter references the frame data to be placed into the respective P-708 Frame Data Buffer.
PINT32 eiData	(input) This parameter references the error injection data to be placed into the respective P-708 Ancillary Data Buffer.

P708_ UPDATE_EI_DATA

Syntax

SINT32 p708_update_frame_data (SINT32 board, SINT32 channel, SINT32 startWord, SINT32 wordCount, PINT32 eiData)

Description

This routine overwrites the specified number of error injection words in the Ancillary Data Buffer for the specified channel.

Return Value

ARS_NORMAL	routine was successful.
ARS_INVBOARD	uninitialized board or invalid *board* value.
ARS_INVARG	invalid *channel* parameter value or the specified locations referenced by *startWord* and/or *wordCount* aren't currently allocated in the Ancillary Data Buffer.
ARS_INVHARVAL	the specified *channel* is not configured to transmit.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.
SINT32 channel	(input) ARINC 708 channel to access. Valid values are 0 and 1.
SINT32 startWord	(input) This parameter specifies the word offset at which to begin overwriting error injection words in the transmit Ancillary Data Buffer.
SINT32 wordCount	(input) This parameter specifies the number of error injection words to overwrite in the transmit Ancillary Data Buffer.
PINT32 data	(input) This parameter references the error injection data to be placed into the respective P-708 Ancillary Data Buffer.

P708_VERSION

Syntax

void p708_version (char *verstr)

Description

This routine will insert an ASCII representation of the current software version number of the ARINC 708 API into the location specified via the *verstr* parameter.

Arguments

char *verstr	(output) string representation of the API Version number consisting of up to 5 characters.

P708_WAIT

Syntax

void p708_wait (float nsecs)

Description

This routine blocks execution of the calling application by the specified number of seconds. The delay is based on the respective O/S delay utility, (such as “Sleep” or “taskDelay”).

Arguments

float nsecs		(input) number of seconds to delay.

P708_WRITE_FRAMES

Syntax

SINT32 p708_write_frames (SINT32 board, SINT32 channel, PINT32 frameCount, pTRANSMIT_FRAME_TYPE data)

Description

This routine is designed to create a new or update an existing set of standard 1600-bit frames in a transmit channel's buffer. When the frame index in the data structure is set to zero, the respective frame is appended to the last frame previously defined for this channel. On adding this frame to the transmit buffer, the frame index structure member is updated with a frame index value representing this frame in the buffer. This index can then be used to subsequently update the respective frame data content in the buffer.

The transmit frame data and ancillary buffer size is automatically adjusted to fit the number of defined frames provided in the frame data structure. By default the ARINC 708 device SRAM is evenly allocated across both channels, with storage for up to 2621 frames provided for each channel. Care are should be taken to manually adjust the frame data and ancillary buffer start and/or stop address for the alternate channel when storage of frame data in excess of this is required for the other channel. It is important to note this routine does not automatically reduce either channel’s data buffer allocation to accommodate a write-frames request for the other channel.

Return Value

ARS_NORMAL	routine was successful.
ARS_INVBOARD	uninitialized board or invalid *board* value.
ARS_XMITOVRFLO	transmit buffer overflow - indicates that there was not enough room in the frame data or ancillary buffer to store the requested number of frames.
ARS_INVARG	invalid *channel* parameter value.
ARS_INVHARVAL	the specified *channel* is not configured to transmit or its frame size isn't defined to be 1600 bits.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.
SINT32 channel	(input) ARINC 708 channel to access. Valid values are 0 and 1.

PINT32 frameCount (input/output) As an input, this parameter specifies the number of frames to transmit; as an output this parameter indicates the actual number of frames placed into the buffer.

pTRANSMIT_FRAME_TYPE data (input/output) This array of structures contains the frame and error injection data placed into the P-708 data buffers as well as the API-assigned frame index provided for updating previously defined frame data; formatted as follows:

```
struct { UINT32    controlWords[4];
         UINT32    binData[512];
         UINT32    errorInjectionWord;
         UINT32    frameIndex;
}
```

P708_WRITE_FRAME_DATA

Syntax

SINT32 p708_write_frame_data (SINT32 board, SINT32 channel, PINT32 frameCount, PINT32 data)

Description

This routine writes the specified number of frames into the Frame Data Buffer for the specified channel. This frame data is appended to any frame data already defined in the transmit buffer, with the exception of an invocation following execution of any API routine in which the transmit buffer has been reset (P708_INITIALIZE_DEVICE, P708_GO, P708_RESET, P708_STOP, or an invocation of P708_SET_DEVICE_CONFIG with the OPERATIONAL_MODE or OPERATIONAL_DIRECTION options).

This routine will accept any size frame data, based on the current frame size programmed for the respective transmit channel. The data must be presented in an array referenced by the data parameter with no unused bits. This routine assumes error injection is either disabled or the ancillary buffer for the respective transmit channel is defined using P708_WRITE_EI_DATA. If the specified number of frames (*frameCount*) exceeds the available buffer storage capacity, this routine returns an overflow indication.

Return Value

ARS_NORMAL	routine was successful.
ARS_INVBOARD	uninitialized board or invalid *board* value.
ARS_XMITOVRFLO	transmit buffer overflow - indicates that there was not enough room in the frame data buffer to store the requested number of frames.
ARS_INVARG	invalid *channel* parameter value.
ARS_INVHARVAL	the specified *channel* isn't configured to transmit.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.
SINT32 channel	(input) ARINC 708 channel to access. Valid values are 0 and 1.

PINT32 frameCount	(input/output) As an input, this parameter specifies the number of frames to store in the transmit frame data buffer; as an output this parameter indicates the actual number of frames placed into the buffer.
PINT32 data	(input) References the data to be placed into the respective P-708 frame data buffer.

P708_WRITE_FRAME_DATA_WEI

Syntax

SINT32 p708_write_frame_data (SINT32 board, SINT32 channel, PINT32 frameCount, PINT32 data, PINT32, eiData)

Description

This routine writes the specified number of frames into the frame data buffer and a respective number of error injection words into the ancillary buffer, for the specified channel. The frame and error injection data is appended to any frame and error injection data already defined in the respective buffers, with the exception of any API routine in which the transmit buffer has been reset (P708_INITIALIZE_DEVICE, P708_GO, P708_RESET, P708_STOP, or an invocation of P708_SET_DEVICE_CONFIG with the OPERATIONAL_MODE or OPERATIONAL_DIRECTION options).

This routine accepts any size frame data, based on the current frame size programmed for the respective transmit channel. The data must be presented in an array referenced by the data parameter with no unused bits. Each word of the error injection array corresponds to the respective ancillary buffer word for the assigned frame of data. Only the lower 4 bits of each error injection array element are used.

Invocation of this routine does not enable error injection for the respective channel; instead, error injection must be enabled via invocation of AR_SET_CHANNEL_CONFIG. If the specified number of frames (*frameCount*) exceeds the available buffer storage capacity, this routine returns an overflow indication.

Return Value

ARS_NORMAL	routine was successful.
ARS_INVBOARD	uninitialized board or invalid *board* value.
ARS_XMITOVRFLO	transmit buffer overflow - indicates that there was not enough room in the frame data or ancillary buffer to store the requested number of frames.
ARS_INVARG	invalid *channel* parameter value.
ARS_INVHARVAL	the specified *channel* isn't configured to transmit.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.
SINT32 channel	(input) ARINC 708 channel to access. Valid values are 0 and 1.
PINT32 frameCount	(input/output) As an input, this parameter specifies the number of frames to store in the transmit frame data buffer; as an output this parameter indicates the actual number of frames placed into the buffer.
PINT32 data	(input) This array references the data to be placed into the respective P-708 frame data buffer.
PINT32 eiData	(input) This array references the error injection data to be placed into the respective P-708 ancillary data buffer, with one word allocated for each frame specified. Reference the Error Injection description in Chapter 4 for a description of the error injection options.

P708_WRITE_EI_DATA

Syntax

SINT32 p708_write_ei_data (SINT32 board, SINT32 channel, PINT32 wordCount, PINT32, eiData)

Description

This routine writes the specified number of error injection words into the ancillary buffer for the specified channel. These error injection words is appended to any error injection words already defined in the transmitter Ancillary Buffer, with the exception of any API routine in which the transmit buffer has been reset (P708_INITIALIZE_DEVICE, P708_GO, P708_RESET, P708_STOP, or an invocation of P708_SET_DEVICE_CONFIG with the OPERATIONAL_MODE or OPERATIONAL_DIRECTION options).

Each word of the error injection array corresponds to the respective ancillary buffer word for the assigned frame of data. Only the lower 4 bits of each error injection array element are used.

Invocation of this routine does not enable error injection for the respective channel; instead, error injection must be enabled via invocation of AR_SET_CHANNEL_CONFIG. If the value of the *wordCount* parameter exceeds the available buffer storage capacity, this routine returns an overflow indication.

Return Value

ARS_NORMAL	routine was successful.
ARS_INVBOARD	uninitialized board or invalid *board* value.
ARS_XMITOVRFLO	transmit buffer overflow - indicates that there was not enough room in the ancillary buffer to store the requested number of error injection words.
ARS_INVARG	invalid *channel* parameter value.
ARS_INVHARVAL	the specified *channel* isn't configured to transmit.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.
SINT32 channel	(input) ARINC 708 channel to access. Valid values are 0 and 1.
PINT32 wordCount	(input/output) As an input, this parameter specifies the number of words to store in the transmit ancillary data buffer; as an output this parameter indicates the actual number of words placed into the buffer.
PINT32 eiData	(input) This array references the error injection data to be placed into the respective P-708 ancillary data buffer, with one word allocated for each frame specified. See “Error Injection” in chapter 4, “ARINC 708 PMC Product Features” for a description of the error injection options.

P708_READ_DEVICE

Syntax

SINT32 p708_read_device (SINT32 board, UINT32 offset, PINT32 data)

Description

This routine reads a 32-bit value from the device based on the specified 32-bit (long word) offset to the board's base virtual address. An example of the use of this routine is to read the programmed firmware version from the RP-708 Firmware Version Register:

```
p708_read_device(board, RP708_FIRMWARE_VERSION,
&fwVersion);
```

Return Value

ARS_NORMAL	routine was successful.
ARS_FAILURE	invalid *board* or *offset* parameter value.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.
UINT32 offset	(input) The long word (32-bit) offset from the base of the device host interface to read.
PINT32 data	(output) data value read from the device. If the offset resides in the Ancillary or Frame Data Buffer regions, this value contains the contents of two successive device locations in the lower and upper 16 bits.

P708_WRITE_DEVICE

Syntax

SINT32 p708_write_device (SINT32 board, UINT32 offset, SINT32 data)

Description

This routine writes a 32-bit value to the device based on the specified 32-bit (long word) offset to the board's base virtual address.

Return Value

ARS_NORMAL	routine was successful.
ARS_FAILURE	invalid *board* or *offset* parameter value.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.
UINT32 offset	(input) The long word (32-bit) offset from the base of the device host interface to write.
PINT32 data	(input) data value written to the device. If the offset resides in the Ancillary or Frame Data Buffer regions, this value overwrites the contents of two successive device locations using the lower and upper 16 bits of this value.

CHAPTER 9

ARINC 708 PMC Hardware Interface

Overview

This chapter describes the low level programming of the ARINC 708 PMC products.

Note: The information in this chapter is provided for those who intend to author their own software interface and device driver.

Control of an ARINC 708 PMC device is performed by reading and writing FPGA registers and SRAM, mapped into host memory space through the PCI BAR2 memory region. To program the device, you must first know where this memory region was mapped in host memory space. The following sections describe configuration registers and memory layout. Each one has its own unique address given as an offset from the beginning of the BAR2 memory region for the device. All SRAM and registers can be read and write via 32-bit access.

PCI Configuration Space

Table 11 describes the PCI Configuration Space definition for all ARINC 708 PMC products.

Table 11. ARINC 708 Product PCI Configuration Space

31	23	15	7	Offset
Device ID 0708h or 708Ah		Vendor ID 13C6h		00h
Status		Command		04h
Base class 06h	Sub-class 80h	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 local configuration registers (128 or 512 bytes)				10h
Base Address Register 1 for I/O-mapped local configuration registers				14h
Base Address Register 2 (4 Mbytes) 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 0708h or 708Ah		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 00h	3Ch

Host Memory Map

Table 12 summarizes the BAR2 memory-mapped "host interface" for all ARINC 708 PMC products. It is described in detail in the following sections.

Table 12. ARINC 708 Product Host Memory Map

Byte Offset	Read/Write	Description
0x00000000	Read/Write	Channel 0 Control Register
0x00000004	Read/Write	Channel 0 Frame Data Start Address Register
0x00000008	Read/Write	Channel 0 Frame Data Stop Address Register
0x0000000C	Read/Write	Channel 0 Ancillary Data Start Address Register
0x00000010	Read/Write	Channel 0 Ancillary Data Stop Address Register
0x00000014	Read/Write	Channel 0 Frame Bit Count Register
0x00000018	Read only	Channel 0 Frame Count Register
0x0000001C	Read/Write	Channel 0 Transmit Frame Interval Register
0x00000020	Read/Write	Channel 0 Transmit Sweep Frame Count Register
0x00000024	Read/Write	Channel 0 Transmit Sweep Interval Register
0x00000028	Read/Write	Channel 0 Transmit Sweep Count Register
0x0000002C	Read Only	Firmware Version (RP-708 only)
0x00000030	Read/Write	Temp Sensor Read Command (RP-708 only)
0x00000034	Write Only	Temp Sensor Write Command (RP-708 only)
0x00000038 – 0x000000FF	n/a	Reserved
0x00000100	Read/Write	Channel 1 Control Register
0x00000104	Read/Write	Channel 1 Frame Data Start Address Register
0x00000108	Read/Write	Channel 1 Frame Data Stop Address Register
0x0000010C	Read/Write	Channel 1 Ancillary Data Start Address Register
0x00000110	Read/Write	Channel 1 Ancillary Data Stop Address Register
0x00000114	Read/Write	Channel 1 Frame Bit Count Register
0x00000118	Read only	Channel 1 Frame Count Register
0x0000011C	Read/Write	Channel 1 Transmit Frame Interval Register
0x00000120	Read/Write	Channel 1 Transmit Sweep Frame Count Register
0x00000124	Read/Write	Channel 1 Transmit Sweep Interval Register
0x00000128	Read/Write	Channel 1 Transmit Sweep Count Register
0x0000012C – 0x0000FFFF	n/a	Reserved

Byte Offset	Read/Write	Description
0x00010000 – 0x00015FFF	Read/Write	Ancillary Data Buffer (FPGA Dual-Port Memory)
0x00016000 – 0x000FFFFF	n/a	Reserved
0x00100000 – 0x001FFFFF	Read/Write	Frame Data Buffer (SRAM)
0x00200000 – 0x003FFFFF	n/a	Reserved

Hardware Registers and Memory

Control Register

31-11	10	9	8	7	6	5	4	3	2	1	0
Unused	SF	EI	EXT	LFE	SFE	RBE	RSA	RPT	HALT	ENA	MODE

The fields in the Control Register are described in Table 13.

Table 13. Control Register Fields

Field	Description	Values
MODE (read/write)	This bit is used to configure the channel to receive or transmit operation.	0 = receive (reset condition) 1 = transmit
ENA CHANNEL ENABLE (read/write)	This bit is used to enable and disable data processing on the channel. When disabled, all data buffer actions are halted, and the internal tracking of data buffer accesses are cleared; however, active transmission and/or reception of a currently incomplete frame of data will proceed to completion before data buffer processing stops.	0 = channel disabled 1 = channel enabled
HALT (write only)	This bit is used to halt frame transmission or reception immediately. This bit should be toggled during board/channel initialization to assure all transmit/receive activity from a previous session has ceased.	0 = disabled 1 = terminate immediately
RPT SWEEP REPEAT (read/write)	This bit enables periodic transmission of the defined number of sweeps at the defined sweep interval. When disabled, the entire contents of the respective frame data buffer is transmitted one time following channel enable.	0 = repeat disabled 1 = repeat enabled

Field	Description	Values
RSA RESET START ADDRESS (write only)	This bit causes the firmware buffer reference to reset to the start of the buffer at the end of the current transmit sweep or receive frame.	0 = disabled 1 = reset requested
RBE RECEIVE BIT ERROR (read only)	This bit will be set if the receive frame decoder encounters any bit error within any received frame. It is self-cleared when this register is read.	0 = no error encountered 1 = a bit error was detected
SFE SHORT FRAME BIT ERROR (read only)	This bit will be set if the receive frame decoder encounters any frame containing fewer bits than what is defined in the respective channel Frame Bit Count register. It is self-cleared when this register is read.	0 = no error encountered 1 = a short frame error was detected
LFE LONG FRAME BIT ERROR (read only)	This bit will be set if the receive frame decoder encounters any frame containing more bits than what is defined in the respective channel Frame Bit Count register. It is self-cleared when this register is read.	0 = no error encountered 1 = a long frame error was detected
EXT EXTERNAL OPERATION (read/write)	This bit enables external operation on this channel. When one channel is configured to receive and the other to transmit, and this bit is cleared on both channels, internal wrap of transmitted data is enabled and external transmission/reception is disabled on both channels.	0 = external operation is disabled 1 = external operation is enabled
EI ERROR INJECTION (read/write)	This bit enables the use of the defined Ancillary Data Buffer as the error injection definition for all transmit frames defined in the respective Frame Data Buffer.	0 = error injection disabled 1 = error injection enabled
SF	This read-only bit provides status of communication with an on-board security device. When this bit is set, transmitting and receiving of 708 data is disabled. This bit should never be set. Contact Abaco Systems Avionics technical support if experiencing problems.	0 = Security OK 1 = Security Failure

Frame Data Start Address Register

31 – 20	19 - 0
Unused	Frame Data Buffer Start Offset

The Frame Data Start Address Register defines the lower boundary of the Frame Data Buffer region assigned to the respective channel. This offset must be assigned as a word offset on a word boundary basis, with a valid range from 0 to $7FFFE. For proper operation, this assigned

offset must not equal to or exceed the value of the respective channel's Frame Data Stop Address Register. When both channels are enabled, the value assigned to this register for one channel must not reside in the Frame Data Buffer region assigned to the other channel.

For any channel configured to transmit, this register contains the offset into the Frame Data Buffer where the transmit control function begins reading frame data. For any channel configured to receive, this register contains the offset into the Frame Data Buffer where the receive control function begins storing received frame data.

Frame Data Stop Address Register

31 – 20	19 - 0
Unused	Frame Data Buffer Stop Offset

The Frame Data Stop Address Register defines the upper boundary of the Frame Data Buffer region assigned to the respective channel. This offset must be assigned as a word offset on a word boundary basis, with a valid range from 1 to $7FFFF. For proper operation, this assigned offset must not be less than or equal to the value of the respective channel's Frame Data Start Address Register. When both channels are enabled, the value assigned to this register for one channel must not reside in the Frame Data Buffer region assigned to the other channel.

For any channel configured to transmit, this register contains the last offset location in the Frame Data Buffer where the transmit control function reads frame data.

For any channel configured to receive, this register contains the last offset location in the Frame Data Buffer where the receive control function stores received frame data. When this offset location is encountered, the receive function resets its internal buffer pointers and continues storing data in the buffer based on the respective Frame Data Start Address Register offset value.

Ancillary Data Start Address Register

31 – 20	19 - 0
Unused	Frame Data Buffer Start Offset

The Ancillary Data Start Address Register defines the lower boundary of the Ancillary Data Buffer region assigned to the respective channel. This offset must be assigned as a word offset on a word boundary basis, with a valid range from 0 to $2FFE. For proper operation, this assigned offset must not equal or exceed the value of the respective channel's Ancillary

Data Stop Address Register. When both channels are enabled, the value assigned to this register for one channel must not reside in the Ancillary Data Buffer region assigned to the other channel.

For any channel configured to transmit with error injection enabled, this register contains the offset into the Ancillary Data Buffer where the transmit control function begins reading error injection data. For any channel configured to receive, this register contains the offset into the Ancillary Data Buffer where the receive control function begins storing received frame time-tags.

Ancillary Data Stop Address Register

31 – 20	19 - 0
Unused	Frame Data Buffer Stop Offset

The Ancillary Data Stop Address Register defines the upper boundary of the Ancillary Data Buffer region assigned to the respective channel. This offset must be assigned as a word offset on a word boundary basis, with a valid range from 1 to $2FFF. For proper operation, this assigned offset must not be equal or less than the value of the respective channel's Ancillary Data Start Address Register. When both channels are enabled, the value assigned to this register for one channel must not reside in the Ancillary Data Buffer region assigned to the other channel.

For any channel configured to transmit with error injection enabled, this register contains the offset into the Ancillary Data Buffer where the transmit control function ceases reading error injection data.

For any channel configured to receive, this register contains the last offset location in the Ancillary Data Buffer where the receive control function will store received frame time-tags. When this offset location is encountered, the receive function resets its internal buffer pointers and continues storing time-tags in the buffer based on the respective Ancillary Data Start Address Register offset value.

Frame Bit Count Register

31 – 24	23 - 0
Unused	Frame Size (in bits)

The Frame Bit Count Register defines the bit size of the respective channel's frame. For transmission, the Frame Size informs the transmit control function how many bits to read from the Frame Data Buffer and insert between the Start Sync and Stop Sync Pulses. For reception, the Frame Size informs the receive control function of how many bits are

expected to be received in each frame. The receive control function compares the number of bits actually present between the incoming Start Sync and Stop Sync Pulses and indicates any deviation via the respective Control Register receive error bit(s).

The Frame Size is also used within the API frame transmit and receive functions to determine the exact word and bit offsets in the Frame Data Buffer when accessing frame data.

Frame Count Register

31 – 0
Frame Count

The Frame Count Register indicates how many frames have been transmitted or received since the respective channel was last enabled. The Frame Count is reset to zero when the respective channel is disabled.

Transmit Frame Interval Register

31 – 16	15 - 0
Unused	Frame Interval Count

The Transmit Frame Interval Register defines the duration, in microseconds, from the leading edge of the Start Sync Pulse between consecutive frames. The Frame Interval Count must always be assigned a value greater than the duration required to transmit the programmed frames. Specifically, the Frame Interval Count should exceed the value assigned in the respective channel's Frame Bit Count Register plus six (six microseconds is the duration of the start and stop sync pulse widths). The actual frame interval implemented by the transmit control function is one greater than the Frame Interval Count.

Transmit Sweep Frame Count Register

31 – 24	23 - 0
Unused	Sweep Frame Count

The Transmit Sweep Frame Count Register defines how many frames are assigned to each sweep "frame grouping". This provides a frame count to the transmit control at which the Sweep Interval is inserted.

Note: The Sweep Frame Count must never be 0.

Transmit Sweep Interval Register

31 – 24	23 - 0
Unused	Sweep Interval Count

The Transmit Sweep Interval Register defines how many microseconds of delay is inserted between the transmissions of each defined sweep in the Frame Data Buffer. The delay begins at the end of the Stop Sync Pulse and ends with the beginning of the next frame's Start Sync Pulse. The actual duration of the delay is one microsecond greater than the Sweep Interval Count. If it is less than the value of the Frame Interval Count, the Sweep Interval Count is superseded by the minimum duration between transmitted frames as specified via the Frame Interval Count.

Transmit Sweep Count Register

31 – 8	7 – 0
Unused	Sweep Count

The Transmit Sweep Count Register defines the number of sweeps from the Frame Data Buffer to transmit. This includes the number of sweeps to repeat when the Control Register Sweep Repeat bit is enabled, or the number of sweeps to transmit on a one-shot basis when the Control Register Sweep Repeat bit is disabled.

Firmware Revision Register (RP-708 Only)

31 – 16	15 - 0
Unused	Firmware Revision

This register provides the FPGA firmware revision of the RP-708.

Temp Sensor Read Command Register (RP-708 Only)

Write

31 – 8	7 – 0
Unused	cmd

Read

31-9	8	7 – 0
Unused	otn	cmd

The RP-708 uses a Maxim MAX6658 temperature sensor at location U31 to give an indication of the PCB board temperature. The Temp Sensor Read Command Register allows the reading of various registers within the MAX6658 using the Read Byte format. To read a particular register, you write its' register address as the command. After waiting 700 us, you then read this register.

Bit 8 of the read register is the programmable internal over temperature limit and comes directly from the MAX6658 as the OVERT# (low true) signal. The default limit is 85C but can be changed via the RWOL device register.

As an example, if you wish to read the MAX6658 Manufacture ID, you would first write a 0xFE, the Read Manufacturer ID address. After 700us, you would read this register to get 0x4D, the devices Read Manufacturer ID in the lower 8 bits. The OTN bit may or may not be set. To read the board temperature, you write a '0', the address to read the MAX6658 internal temperature. After 700us, you read this register where the lower 8 bits is the board two's complement temperature in degrees Celsius. A returned value of 0x19 would convert to 25C whereas a value of 0xE7 corresponds to -25C (0xFF – 0xE7 + 1).Refer to the device datasheet for more detailed information.

Temp Sensor Write Command Register (RP-708 Only)

Write

31 – 16	15 – 8	7 – 0
Unused	data	cmd

The Temp Sensor Write Command Register allows the setting of programmable registers within the MAX6658 using the Write Byte format.

As an example, if you wish to change the OVERT# limit (ROWL register address 0x20) to 70C, you would convert 70 decimal to 0x46 and write 0x4620 to this register. If you want to verify what you wrote, you would then write 0x20 to the Temp Sensor Read Command Register, wait 700 us and then read the Temp Sensor Read Command Register where you should read 0x46.

Ancillary Data Buffer

The Ancillary Data Buffer consists of FPGA dual-port memory allocated to two different functions depending on whether a channel is configured to transmit or receive data.

Receive Time-tagging

When a channel is configured to receive data, the Ancillary Data Buffer is used to store an internal 48-bit, one microsecond resolution time-tag value at which a frame's Start Sync Pulse is detected. The timer value is stored in three consecutive words with the least significant 16 bits stored in the first word and the most significant 16 bits stored in the last word, as shown in the following table:

Ancillary Byte Offset	Data
0	First Frame Time-tag Low Word
2	First Frame Time-tag Mid Word
4	First Frame Time-tag High Word
6	Second Frame Time-tag Low Word
8	Second Frame Time-tag Mid Word
10	Second Frame Time-tag High Word
…	…

The timer resets and begins counting from zero when the receiver transitions from disabled to enabled.

Transmit Error Injection

When a channel is configured to transmit data, the Ancillary Data Buffer is used to define error injection data. The data is referenced sequentially by the transmit control function on a frame-by-frame basis, with each location containing the error injection data for the respective frame defined in the Frame Data Buffer. Transmit Error Injection data is ignored if the EI bit in the respective Control Register (bit 9) is defined to be "disabled".

The definition of an error injection data word is described below:

15 – 4	3	2	1	0
Unused	SPSPI	STSPI	SFE	LFE

The fields in an Error Injection Data Word are described in Table 14.

Table 14. Error Injection Data Word Fields

Field	Description	Values

Field	Description	Values
LFE LONG FRAME ERROR	When this error injection bit is set to enabled, the respective frame is transmitted with an additional bit over the value specified in the Frame Bit Count Register, inserted at the end of the frame.	0 = disabled 1 = enabled
SFE SHORT FRAME ERROR	When this error injection bit is set to enabled, the respective frame is transmitted with one bit less than the value specified in the Frame Bit Count Register, removed from the end of the frame.	0 = disabled 1 = enabled
STSPI START SYNC PULSE INVERSION	When this error injection bit is set to enabled, the respective frame is transmitted with an inverted Start Sync Pulse.	0 = disabled 1 = enabled
SPSPI STOP SYNC PULSE INVERSION	When this error injection bit is set to enabled, the respective frame is transmitted with an inverted Stop Sync Pulse.	0 = disabled 1 = enabled

APPENDIX A

Protected Frame Update Feature

Overview

To assist with integration of ARINC 708 PMC products into Windows-based applications currently designed around the operation of the IP-708, a feature referenced as the "Protected Frame Update" is provided with the ARINC 708 API.

Most ARINC 708 Windows applications based on the IP-708 and IP-AVIONICS API utilize the frame transmission method using repeated invocations of the IP708_PUT_FRAME routine. This method provides for active storage of up to 307 ARINC 708 frames in the IP-708 transmit buffer, allowing for simultaneous application frame definition and IP-708 frame transmission.

Since the ARINC 708 PMC product sweep/frame buffering method does not provide a guarantee that application updates to an active transmit buffer won't occur on frames that are simultaneously being read by the P-708 hardware, the Protected Frame Update feature was created. This feature provides a method to define a sweep using a fixed frame count, then update the contents of those frames between the time the last word of the last frame transmission completes and the time at which the subsequent transmission of the first word of the first frame begins.

In support of this feature, three routines were added to the ARINC 708 API, p708_frame_transmit_start, p708_frame_transmit_stop, and p708_request_frame_transmission, described in the following pages of this document. To use these routines the application should setup the P-708 hardware in a fashion similar to what is presented in the example C source file, SINGLE_FRAME_SWEEP.C. While this example uses a single frame sweep, a sweep containing any number of frames can be used with the Protected Frame Update feature.

p708_frame_transmit_start

Syntax

SINT32 p708_frame_transmit_start (SINT32 board, UINT32 polling_interval)

Description

This routine initiates a Windows periodic Multimedia Timer interrupt to invoke the internal ARINC 708 API frame update request process at the rate specified via *polling_interval*. The minimum rate at which the frame update request process executes is one millisecond; however, the rate value may impact application throughput at that value. For optimal performance, the rate value supplied should be set to one-half the duration between the time required to transmit the defined sweep and the programmed sweep interval.

Return Value

ARS_NORMAL	routine was successful.
ARS_SYNCTIMEOUT	the multimedia timer resource could be acquired.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.
UINT32 polling_interval	(input) the multimedia timer interval at which the frame update request process will be invoked. Only one interval, the last one requested, is used.

p708_frame_transmit_stop

Syntax

void p708_frame_transmit_start (void)

Description

This routine terminates a Windows periodic Multimedia Timer interrupt process previously generated to invoke the internal ARINC 708 API frame update request.

Return Value

None.

Arguments

None.

p708_request_frame_transmission

Syntax

SINT32 p708_request_frame_transmission (SINT32 board, SINT32 channel, SINT32 type, UINT32 frameCount, void * data)

Description

This routine provides the method for the application to request an update to the current transmit channel frame buffer with the supplied frame data, to occur at the next available break in the P-708 sweep scheduler.

Return Value

ARS_NORMAL	routine was successful.
ARS_INVBOARD	uninitialized board or invalid *board* value.
ARS_XMITOVRFLO	indicates a previous transmit request has yet to be processed.
ARS_INVARG	invalid *channel* parameter value or frame *type* selection.
ARS_INVHARVAL	the specified channel is not defined for transmission.

Arguments

SINT32 board	(input) Abaco Systems Avionics Avionics device to access. Valid range is 0-127.
SINT32 channel	(input) Transmit channel to use. Valid values are 0 or 1.
SINT32 type	(input) Format of the transmit frame data to use:

FRAME_STRUCT_DATA_FORMAT (0) – the *data* parameter references a data structure of the type TRANSMIT_FRAME_TYPE.

FRAME_RAW_DATA_FORMAT (1) – the *data* parameter references an array of unsigned 32-bit integer values containing frame data packed with no unused bits.

UINT32 frameCount	(input) This parameter specifies the number of frames to write to the transmit buffer.
void * data	(input/output) The frame data source for the update, referenced as either a structure array or raw data array based on the value of the *type* parameter. If the *type* parameter is set to FRAME_STRUCT_DATA_FORMAT, this parameter should be defined as an array of structures contains the frame and error

injection data placed into the P-708 data buffers as well as the API-assigned frame index provided for updating previously defined frame data; formatted as follows:

```
struct { UINT32 controlWords[4];
         UINT32 binData[512];
         UINT32 errorInjectionWord;
         UINT32 frameIndex;
       }
```

See the description for P708_WRITE_FRAMES for more details regarding how the API uses this data structure.

If the *type* parameter is set to FRAME_RAW_DATA_FORMAT, this array is referenced as unsigned 32-bit integer values containing frame data packed with no unused bits.