VxWorks User's Manual

BusTools/1553-API

Supporting Products:

- QPCX-1553
- QPMC-1553
- R15-MPCIE
- R15-EC
- QVME-1553
- RQVME2-1553
- QCP-1553
- R15-AMC
- Q104-1553P
- QPCI-1553
- RPCIE-1553
- RXMC-1553
- RXMC2-1553
- R15-LPCIE
- QPM-1553
- RAR15-XMC-IT/RAR15XF

abaco SYSTEMS

Copyrights

BusTools/1553-API VxWorks User's Manual

BusTools/1553-API Software Revision:	8.28
Hardware Revision:	6.17/6.11/6.09/6.08/6.03/5.18/4.6x/4.4x
Document Date:	December 30, 2019
Document Revision:	1.2

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

Introduction

VxWorks is a real-time operating system (RTOS). The BusTools/1553-API runs under VxWorks on PowerPC and Intel x86 processors. You can also port BusTools/1553-API to other BSPs. There are two VxWorks releases currently in use, VxWorks 6.x and VxWorks 7. BusTools/1553-API supports operation with VxWorks 6.x and VxWorks 7; although, the driver and API can be built under VxWorks 5.5.1. Currently the 6.x support covers versions 6.2 – 6.9.

BusTools/1553 supports both kernel modules with legacy device drivers (for PCI/PMC and VME devices) and VxBus device drivers (for PCI/PMC and PCI Express/XMC devices).

Note: This document assumes a Tornado or Workbench environment and that you are familiar with your specific system and VxWorks BSP. The most common configuration consists of a Windows PC running the Tornado or Workbench and networked to the VxWorks system. You may encounter differences with other system configurations.

Supported Products

The following Abaco Systems products are supported with VxWorks:

VxWorks 5.5 and VxWorks 6.2-6.6

- VME bus products: QVME-1553, RQVME2-1553
- PCI bus products: QPMC-1553, QPM-1553, QPCI-1553, QPCX-1553, QCP-1553, and Q104-1553P, R15-EC, RXMC-1553, RXMC2-1553, R15-PCIE, R15-LPCIE, RAR15-XMC, R15-PMC

VxWorks 6.7-6.9

- VME bus products: QVME-1553, RQVME2-1553
- PCI bus products (VxBus Gen1): QPM-1553, QCP-1553, Q104-1553P, RXMC-1553, RXMC2-1553, RAR15-XMC-IT/RAR15XF, R15-MPCIE and R15-PMC.

VxWorks 7.0 VxBus Gen2

- PCI bus products: QPM-1553, QCP-1553, RXMC-1553, RXMC2-1553, RAR15-XMC-IT/RAR15XF, R15-MPCIE and R15-PMC.

Installation Options

There are two methods for installing and configuring your VxWorks systems to interface with Abaco Systems Avionics PCI, PMC, PCIe, or XMC products. One is a Common Driver model that builds a legacy avionics PCI driver into the VxWorks kernel. You can use this method for VxWorks 5.5 and 6.x. The Common Driver is designed for PCI and PMC products. The other method is the VxBus Driver method with

VxWorks kernel versions 6.7 and greater. The VxBus Drivers provided are designed to support PCI, PMC, PCI Express, and XMC products.

Note: If you are incorporating Abaco Avionics VME 1553 products with your VxWorks-based system, you should not use either of the VxBus or Common device drivers; instead build the kernel with the VME bus support as provided in your BSP.

BusTools/1553-API Driver-Specific Compiler Directives

When building the BusTools/1553-API for use with the Common PCI Driver option, in your API library project define the directive VXW_DRIVER_OPTION. For the VxBus driver define the directive VXW_VXB_DRIVER, and for the VxBus Gen2 driver define the directive VXW_VXB_DRIVER for a DKM project and VXW_DRIVER_OPTION for a RTP project.

VxBus Driver Installation

The VxBus Driver method provides a driver compatible with Wind River's VxBus infrastructure. The VxBus driver works with VxWorks 6.7 and greater, for PowerPC and Intel x86 processors.

VxBus Gen 1 Support for VxWorks 6.7 to 6.9

Installing the VxBus Gen 1 Driver Files

The following VxBus Driver and configuration files can be found in the beneath the "\VxWorks Driver\VxBus Driver\VxBus1 Driver" folder in the BusTools-1553-API VxWorks Distribution.

VxBus Driver Files	Description
avioVxwDrv.c	Avionics Common Driver
avioVxwDrv.h	Driver declarations and definitions.
40avioVxwDrv.cdf	Component Installation File.

Manually copy the driver files as follows.

Copy avioVxwDrv.c and avioVxwDrv.h, to:

[*VxWorks_Directory*]\ ]\ vxworks-6.x\target\config\comps\src

Copy 40avioVxwDrv.cdf to:

[*VxWorks_Directory*]\ vxworks-6.x\target\config\comps\vxWorks

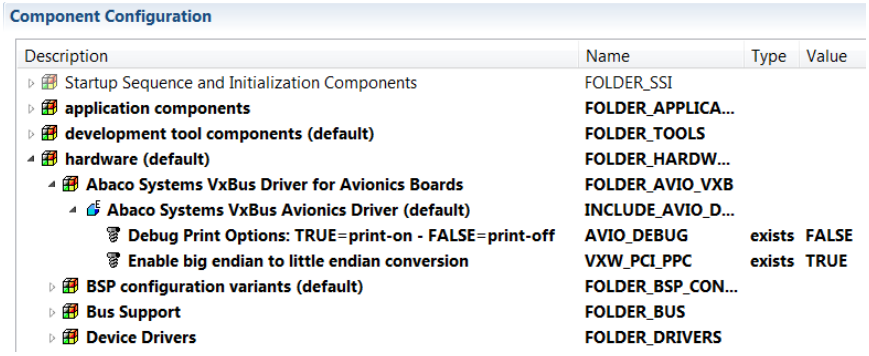
Configuring VxBus Gen 1 Driver Operation

The follow table shows the driver configuration options.

Table 1. VxBus Gen 1 Driver Configuration Options

Parameter	Description	Default
AVIO_DEBUG	Setting this TRUE provides console debug printout during boot and initialization and inhibits execution of the VxBus driver. If TRUE, you must invoke **avioDrvRegister2** from the command line to install the driver.	FALSE
VXW_PCI_PPC	Enables big endian conversion (for PPC)	FALSE

Abaco Avionics Board configuration is under the hardware section in the component configuration for the kernel. You can select whether to include Abaco Avionics Boards here. When selected, this adds the driver into the kernel with the default settings. Modify those setting by selecting options when you right-click on the *Abaco Systems VxBus Driver for Avionics Boards* entry. A sample Abaco Avionics common PPC VxBus Kernel Configuration file setup is shown below:



VxBus Gen 2 Driver Support (VxWorks 7)

To install the generic Abaco Systems' Avionics common VxBus Gen2 driver support for VxWorks 7, there are two methods:

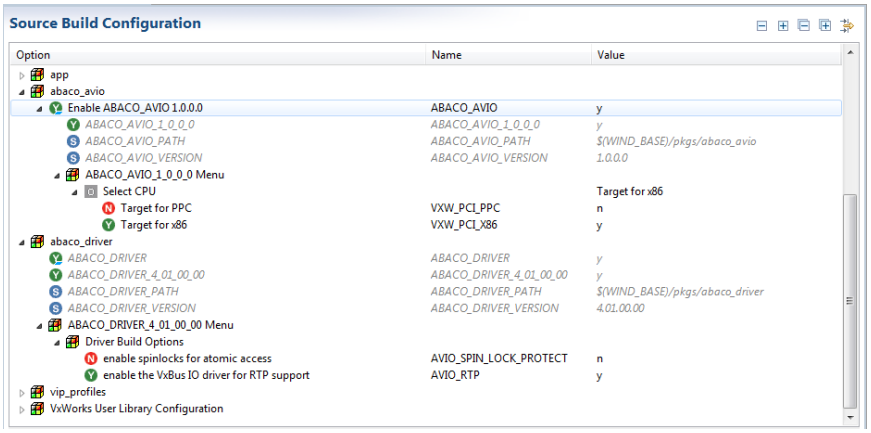
1. The first is to use the provided Abaco Systems RPM files with the VxWorks installer to install the necessary files to build/configure the driver.
2. The second method is to perform a manual installation which involves copying the two folders "abaco_avio" and

"abaco_driver" located in the "\VxWorks Driver\VxBus Driver\VxBus2 Driver" folder in the BusTools-1553-API VxWorks Distribution.

To

[*Workbench_directory_path*]/VxWorks-7/pkgs

After the installation the build options for the Abaco Systems Avionics driver(s) will be available in a VxWorks 7 Source Build project as shown:



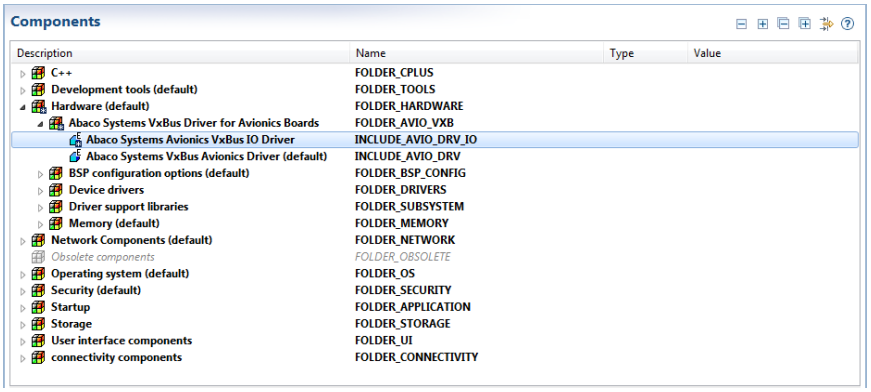
The Abaco Systems Avionics VxBus Gen 2 driver has the following build options.

1. *Select CPU*: select either PPC or x86
2. *Enable the VxBus IO Driver for RTP support*: only needed if requiring operational in an RTP environment.
3. *Enable spinlocks for atomic access*: optional protection in ISR

If you plan to create a new VxWorks 7 Source Build Project, once the driver is installed it should be included in the build automatically; however, if you are adding the Abaco Systems 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 "ABACO_DRIVER_4_01_00_00" layer and select Build Layer
e. Right-click on the "ABACO_AVIO_1_0_0_0" layer and select Build Layer

After the VxWorks 7 Source Build project has included the drivers a VxWorks 7 Build Image project that includes this VSB will have the drivers available to be included as shown:



4. Right-click on the *Abaco Systems VxBus Driver for Avionics Boards* entry and select *Include*. This will include both drivers:
    a. *Abaco Systems VxBus Avionics Driver* (primary driver)
    b. *Abaco Systems Avionics VxBus IO Driver* (secondary driver for operating within an RTP environment)
5. Continue to *Common Build Components* to build the kernel image with the VxBus Gen2 device driver.

Verify Abaco Avionics Boards Installed

Once you have your VxWorks kernel image built and running on your target with your avionics board(s) installed, open a shell to the target and invoke the function *avioDeviceShow*. This routine lists all detected Abaco Avionics products in the discovered "Device ID" order that should be referenced from your application for the respective boards

Common Driver Installation

The Common Driver method provides a driver and configuration files that support VxWorks, 5.5 – 6.6 hosted on PowerPC and Intel x86 processors. The VxWorks kernel configuration process uses the configuration files to install and configure the driver. This allows users to customize the driver operation to match the BSP without having to edit code.

Installing the Common Driver Files

The following files are the new Common Driver and configuration files.

Common Driver Files	Description
CondorVxWRTPDrv.c	Avionics Common Driver
CondorVxWRTPDrv.h	Declarations and definitions for driver
51_GEFES_PPC_RTP_6x_PCI.cdf	Component Installation File for PPC for 6.x
51_GEFES_x86_RTP_6x_PCI.cdf	Component Installation File for x86 for 6.x
51_GEFES_PPC_55_PCI.cdf	Component Installation File for PPC for 5.5
51_GEFES_x86_55_PCI.cdf	Component Installation File for x86 for 5.5
gefes_ioctl.h	ioctl defines

In addition to these files, you need to add the include files cei_types.h and lowlevel.h to the driver configuration directories.

Manually copy the driver files after you install the BusTools/1553-API software onto your system. Where you copy these files depends on the version of VxWorks you are running.

For VxWorks 5.5, copy CondorVxWRTPDrv.c, CondorVxWRTPDrv.h, gefes_ioctl.h, cei_types.h and lowlevel.h to:

[*VxWorks_Directory*]\target\config\comps\src

Copy 51_GEFES_PPC_55_PCI.cdf (PPC) or
51_GEFES_x86_55_PCI.cdf (x86) to:

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

This adds the Abaco Systems Avionics Driver component to all BSPs supported by your VxWorks Installation.

For VxWorks 6.x copy CondorVxWRTPDrv.c, CondorVxWRTPDrv.h, gefes_ioctl.h, cei_types.h, lowlevel.h,
51_GEFES_PPC_RTP_6x_PCI.cdf (PPC), or
51_GEFES_x86_RTP_6x_PCI.cdf (x86) to:

[*VxWorks_Directory*]\vxworks-6.*x*\target\config*bsp_target_dir*

For example, is you are using a pcPentium3 the directory is

[*VxWorks_Directory*]\vxworks-6.6\target\config\pcPentium

This adds the Abaco Systems Avionics Driver component to only that BSP. If you are using multiple BSPs, you need to copy the files into all appropriate directories.

Configuring Driver Operation

The follow table shows the driver configuration options.

Table 2. Driver Configuration Options

Parameter	Description	Target	Default
VXW_PCI_PPC	Define PCI Compile for PowerPC	PPC 5.x and 6.x	TRUE (PPC)
VXW_PCI_X86	Define PCI Compile for x86 and Pentium	x86 5.5 – 6.6	TRUE (X86)
MAX_BTA	Maximum number of devices supported	All	4
vxwdebug	Enables debug print option	All	0
VXW_SYS_BUS_MAP	Use sysBusToLocalAdrs to map board address. Required by some BSPs.	BSP dependent	FALSE
VXW_X86_MAP_ADD	Adds boards PCI memory to sysPhysMemDesc. Needed for all VxWorks Pentium kernel version, except VxWorks 6.6	x86 VxWorks 5.5 – 6.5	FALSE
VXW_PCI_INT_CON	Use pciIntConnect in place of intConnect.	All	FALSE
SPIN_LOCK_PROTECT	Adds spinlocks for SMP. VxWorks 6.6 only	x86 and PPC VxXorks 6.6	FALSE
IRQ_OFFSET	IRQ Offset used by some BSPs.	BSP Dependent	0

Modify these parameters to get the required driver operation.

Diagnostics Built into the Common Driver

CondorVxWRTPDrv.c has several built-in diagnostic functions. The kernel configuration for Abaco Systems Avionic boards has the option for debug print (vxwdebug). This option turns on the print macro during the installation process. You can view the discovery and mapping of each device during boot. However, for Pentium systems running on VxWorks version 6.5 or earlier this takes place early in the boot process and console print invocations are not displayed.

```
mapCondorPCIAddress in debug mode -> 0
PCI device found on bus 17
```

```
PCI device number = 1553
page_addr = 80000000
page_offset = 0
mem_base_region = 0
Found PCI memory region 0
PCI base address = 80000000
PCI region size = 800000
Mem base addr = 80000000
Complete with unit 0
PCI device found on bus 17
PCI device number = afd0
page_addr = 80800000
page_offset = 0
PCI unadjusted region size = 200
Found PCI memory region 0
PCI base address = 80800000
PCI region size = 1000
page_addr = 80900000
page_offset = 0
Found PCI memory region 2
PCI base address = 80900000
PCI region size = 100000
Mem base addr = 80800000
Complete with unit 1
mapCondorPCIAddress Complete 2

gefesInitPCI 0
gefes_pci_driver_control add driver 2
calling gefesPCIDrvCreate for device 0
gefesPCIDrvCreate for device 0
device_name = /gefesDev/0
calling gefesPCIDrvCreate for device 1
gefesPCIDrvCreate for device 1
device_name = /gefesDev/1
calling gefes_config_1553_devices for device 0
gefes_config_1553_devices 0
base addr = 80000000
Host_interface = 1902
Condor Device ID = 1553
1553 Qboard host interface = 1902
btype = 1
nchan = 4
dev_id = 110
gefes_config_pci_devices Complete for device 0
Using intConnnect on intProcess for IRQ 2a
```

Other Debug Functions

avioDeviceShow displays all Abaco boards installed in the system

```
avioDeviceShow
2 Avionic I/O Device(s) Detected:

Unit  Device ID  Base address   IRQ   Channels  UCA32 ARINC
------------------------------------------------------------
```

```
0      1553        0x80000000    0x2a       2        0      0
1      afd0        0x80800000    0x28       0        0      0
```

memDescShow shows the content of the sysPhysMemDesc table. This allows users to see if our boards are included in that table. See gefBoardShow for device addresses.

```
-> memDescShow

entry 0
virtual addr 0x0
physical addr 0x0
size in bytes 1048576

entry 1
virtual addr 0x100000
physical addr 0x100000
size in bytes 250609664

    .
    .
    .

entry 9
virtual addr 0x40000000
physical addr 0x40000000
size in bytes 8388608

entry 10
virtual addr 0x60000000
physical addr 0x60000000
size in bytes 8396800

entry 11
virtual addr 0xe2000000
physical addr 0xe2000000
size in bytes 8192
```

VxW_GetMQID Returns the interrupt message queue ID for devices using interrupts. You need this value for msgQShow.

```
VxW_GEetMQID 0
value = 250597712 = 0xeefd150
```

VxW_GetDevIRQ Returns the device IRQ value for the device.

```
-> VxW_GetDevIRQ 0
value = 25 = 0x19
```

VxW_Get1553DevID Returns the device ID of the device.

```
-> VxW_GetDevID 0
value = 5459 = 0x1553
```

VxW_GetDevChan Returns the channel count for the device.

```
-> VxW_GetDevChan 0
value = 2 = 0x2
```

VxW_Get1553HIF Returns the value of the host interface register for 1553 device.

```
-> VxW_GetHIF 0
value = 6274 = 0x1882
```

VxW_GetCardID Returns the board type.

```
-> VxW_GetCardID 0
value = 272 = 0x110
```

After boot, you can use the VxWork function devs to list all devices in the system.

```
-> devs
drv name
  0 /null
  1 /tyCo/0
  1 /tyCo/1
  2 /aioPipe
  5 toucan:
  6 /vio
  7 /tgtsvr
  8 /gefesDev/0
  8 /gefesDev/1
```

All Abaco Systems Avionic devices are named /avioVxbDrv***n***. Where ***n*** is the device number. That number is based on the host's PCIbus discovery order.

Installing the QVME-1553 or RQVME2-1553

The QVME-1553 and RQVME2-1553 are native VME bus 1553 interface boards with 1, 2, or 4 channels. To program a VME board you will need to address both A16 and A32 (or A24) address space. Set the A16 base address through on-board jumpers described in the "MIL-STD-1553 Hardware Installation and Reference Manual" chapter 12 "QVME-1553, VME-1553, QVXI-1553X, and VXI-1553 Installation". The factory default A16 address is 0xC3C0. The QVME -1553 and RQVME2-1553 require 8 megabytes of memory and can use either A24 or A32 addressing. The A24/32 address is programmable.

You must program the A24 address to start at either 0x0 or 0x800000. You must program the A32 address on an even 8-megabyte (0x800000) boundary. Furthermore, the A32 address must fall within the defined address-range for the BSP's VME A32 address space. See the memory map for your processor for information about the A32 address range.

Set the A24/A32 addressing mode and program the A24/32 address through the board initialization routine, BusTools_API_InitExtended(). Use the following lines to initialize the QVME-1553 or RQVME2-1553 under VxWorks.

```
Status = BusTools_API_InitExtended(cardnum,
                                   0x10000000,// A32 Addr
                                   0xc3c0,    // A16 Addr
                                   &pwFlag,   // mode flag
                                   PLATFORM_PC
                                   QVME1553,  // Board type
                                   NATIVE_xx,
                                   Channel_n,
                                   CARRIER_MAP_Axx);
```

Where n = the Channel number 1, 2, 3, or 4 and xx = the map address bits 24 or 32.

The BusTools/1553-API installation contains the object modules for the API in formats for the PowerPC G3 and G4 processor. You can download these object modules to any PowerPC.

PCI/PMC Board Mapping

BusTools/1553-API can control up to 16 1553 interface channels on up to 16 1553 boards. During boot up, mapCondorPCIAddress() determines the number and type of Abaco Systems 1553 boards in the system.

If you only have one 1553 board installed, it is always device 0. If multiple boards are installed, the device number starts at zero and increments by one for each board installed. The device number assigned to a board depends on how the system enumerates the boards. Use BusTools_API_OpenChannel() to initialize 1553 channels on PCI or ISA boards. BusTools_API_OpenChannel() takes the device, channel and operating mode, returns status and sets the card number.

UINT VxW_GetCardID(UINT cardnum) returns the board type. This is the board type (PCI1553, QPCI1553 for example) by card number. Returns 0xf if cardnum is not present.

UINT VxW_GetDevChan(UINT device) return the number of channels for each device. Returns 0xf if device is not present

UINT VxW_GetDevID(UINT device) returns the board type by device number. Return 0 if device not present

UINT get_device(UINT cardnum) returns the device number for each cardnum. Returns 0xf if device is not present.

VxWorks Interrupts

VxWorks supports interrupt processing. Applications should use BusTools_RegisterFunction() to process interrupts (the same method as Windows and UNIX), referred to as high-level interrupts. The advantage of using BusTools_RegisterFunction() is that it provides common interrupt processing across all operating systems and it pre-

processes the interrupt data before calling the user function. The disadvantage in using BusTools_RegisterFunction() is that it can increases the latency of the interrupt. There is an option EVENT_IMMEDIATE to reduce latency in some cases.

Interrupt handling can vary among the PPC BSPs. Some BSPs may not be compatible with this interrupt configuration. If not, you will need to set up interrupts by editing rtp_int_setup.c. Each processor has a specific method for connecting and disconnecting interrupts to the interrupt service routine. Review the PCI interrupt section of the BSP reference to find out the how to connect and disconnect interrupts.

The BusTools_RegisterFunction() interrupt method can use both pthreads or VxWorks native thread as options. The default is native VxWorks threads. This allows compatibility to VxWorks versions 5.4 and earlier. If you want to use POSIX threads you can define _POSIX_ (#define _POSIX_) in your target block in target_defines.h and recompile the API library. If you select the POSIX option, your VxWorks image must support both POSIX threads and POSIX timers.

BusTools_RegisterFunction() passes the user callback function in the API_INT_FIFO structure. Refer to the BusTools-1553/API Reference manual for information on BusTools_RegisterFunction() and the API_INT_FIFO structure.

PCI Interrupts

The VxWorks-specific portion of the API provides PCI interrupt processing in int_setup.c. This file contains the setup and processing for interrupts based on the PowerPC and x86 architectures.

Int_setup.c handles connecting the interrupt to the interrupt service routine (ISR). This method varies for different PowerPC and x86 processors. You need to select the version of the API matching your processor. If you have a processor not supported by this file you will need to modify the code to correctly connect the interrupt to the ISR.

There is a single interrupt for each 1553 device no matter how many channels are on that device. The interrupt service routine determines which channel(s) on the device is interrupting, clears the interrupt and starts the user interrupt processing function.

VME Interrupts

The VxWorks-specific portion of the API provides VME interrupt processing in the file vme_int_setup.c. This file also uses binary semaphores to control the deferred processing. With VME interrupts, you must select the interrupt level and interrupt vector. There is a single interrupt level for a VME board, but each channel can use a unique

interrupt vector. The vme_int_setup.c has default setting for these values for four 4-channel QVME-1553 and RQVME2-1553 boards. You can set your own values by calling BusTools_SetIntVector(int cardnum, int vector) and BusTools_SetIRQ_LVL(int cardnum, int irq). Use the same level for all channels on a QVME-1553/RQVME2-1553 board.

Building BusTools/1553-API

Build BusTools/1553-API as a downloadable application or a static library by creating the respective application project in Tornado or Workbench and adding the following source files:

- bc.c
- bm.c
- bit.c
- btdrv.c
- discrete.c
- ei.c
- flash_config.c
- hwsetup.c
- init.c
- rt.c
- time.c
- notify.c
- CEI_VXW_INTERRUPT_FUNCTIONS.c
- mem_vxWorks.c
- vxw_int_setup.c (for PCI/PMC/XMC boards only)
- vxwBoardSetup.c (for PCI/PMC/XMC boards only)
- vme_int_setup_vme.c (for VME/VXI boards only)
- vxwBoardSetupVME.c (for VME/VXI boards only)

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

- apiint.h
- busapi.h[1]
- target_defines.h (included by busapi.h)
- cei_types.h (included by busapi.h)
- btdrv.h
- avioVxwDrv.h (for a VxBus Driver based project only)
- globals.h
- lowlevel.h

[1] busapi.h includes target_defines.h which has target specific definitions. You define a macro to select a target definition block in target_defines.h that matches your processor. There is a set of pre-define blocks for supported processors. See below.

You can find these files in the BusTools-1553-API/Source folder in the VxWorks Distribution.

When building the API using Tornado, Workbench, or command line, you must define a macro that tells the API the target processor type. There are pre-defined macros for the processors supported under VxWorks.

VXW_PCI_PPC is for default PowerPC devices. There are two BSP requirements to use this macro. One is that the base of address of the board does not require any BSP specific mapping function (sysPciMemToLocalAdrs for example) and the other is to use the interrupt line value is unmodified. For RTP projects, all PowerPC BSPs use the same mapping, so you need to consider only the interrupt line.

The following list shows the pre-defined macro definitions.

VXW_PCI_X86	any PCI, PMC, or ISA board on x86 BSP
VXW_PCI_PPC	PCI (PMC) boards on PowerPC
VXW_VME_PPC	VME boards on PowerPC
VXW_VME_X86	VME boards on Intel x86
VXW_DRIVER_OPTION	Define when using the Common Driver.
VXW_VXB_DRIVER	Define when using the VxBus driver

These macros select pre-defined “target_defines.h” blocks. Configure the API to include or exclude features by editing the selected block. You can also customize that block to fit your processor and BSP definitions.

If you are running Tornado, define the macro by adding it to the Workspace, project build tab, C/C++ compiler window. **Remove the –ansi (GNU) or Xansi (DIAB) directive before compiling the BusTools/1553-API.** You can also add paths to source and include files in this window.

If you are building from the command line, define the macro in your API Makefile. Add the paths to the include files and source files to the Makefile. Do not define the –ansi (GNU) or Xansi (DIAB) compiler directive.

Target_defines.h

Each processor/bus combination in the list above has a block of defines in the target_defines.h file. These blocks configure the API for the target processor and operating environment. You can also customize the API using define blocks. The code example below shows the define block for the native PCI board on a PowerPC (VXW_PCI_PPC). Modify the

configuration of the API by changing the statements in this block. For example, if your VxWorks target has a file system, define FILE_SYSTEM to enable the API to dump data to a file. Defining DO_BUS_LOADING configures the API to gather bus-loading statistics. See the following list for a description of all target define symbols.

```
/*************************************************************
* Target Defines for VxWorks PowerPC or PCI boards
*************************************************************/
#if defined(VXW_PCI_PPC)                                      */
  #include <string.h>     /* for mset                        */
  #define MAX_BTA   16    /* Defines board and channels      */
  #define _UNIX_          /* UNIX definitions                */
  #define _GCC_           /* Using GCC compiler              */
  #define PPC_SYNC        /* enforce sync                    */
  #define VXWORKS         /*                                 */
  #undef USE_BM_DMA       /*                                 */
  #undef _Windows         /*                                 */
  #undef __WIN32__        /*                                 */
  #undef INCLUDE_VME_VXI_1553 /*                             */
  #undef ERROR_INJECTION  /* Error injection Disabled        */
  #undef FILE_SYSTEM      /* Dump output functions           */
  #define  NON_INTEL_BIT_FIELDS  /* Intel Bit Ordering       */
  #define  NON_INTEL_WORD_ORDER  /* Intel Word Ordering      */
  #define  WORD_SWAP      /* define the flipw macro          */
  #undef  ADD_TRACE       /* Exclude trace code              */
  #undef  DO_BUS_LOADING  /* Exclude bus loading code        */
  #define NO_HOST_TIME    /* no host time available          */
  #define POSIX_MSG_QUEUE /* use POSIX message queues        */
  /* the following are the O/S specific definitions          */
 /* mutexes and events                                       */
  #define CEI_MALLOC(a) malloc(a) /* memory alloc            */
  #define CEI_FREE(a)   free(a)   /* memory free             */
  #define CEI_REALLOC   realloc   /*                         */
  #undef _POSIX_          /* POSIX or native VxWorks         */
  #ifdef _POSIX_          /*
   #include <pthread.h>   /* Use this for VxWorks 5.5        */
   #define CEI_MUTEX pthread_mutex_t /* define mutex type    */
   #define CEI_THREAD pthread_t      /* define thread type   */
   #define CEI_EVENT pthread_cond_t  /* define event         */
   #define CEI_HANDLE int            /* define handle        */
  #else /* Native VxWorks Threads                            */
   #include <msgQLib.h> /* Using Message Queue              */
   #include <semLib.h>  /* Use mutual exclusion semaphore   */
   #include <timers.h>  /* Timer data type include          */
   #include <taskLib.h> /*                                  */
   #define CEI_MUTEX  SEM_ID    /* define mutex             */
   #define CEI_THREAD int       /* define thread            */
   #define CEI_EVENT  MSG_Q_ID  /* define event             */
   #define CEI_HANDLE int       /* define handle            */
  #endif /* POSIX_                                          */
  #define TIMER_THREAD /* use timer thread                  */
  void MSDELAY(int msec); /*                                */
  #define VXW_TASK_OPTION 0 /* VX_FP_TASK, VX_SPE_TASK      */
#endif   /* end VXW-PCI-PPC                                 */
/*************************************************************/
```

Building BusTools/1553-API as a Downloadable Application or Static Library in Tornado

The installation contains the BusTools/1553-API source code. This allows you to modify API source to customize execution. If you make changes, you need to rebuild the API. You also need to re-compile if you make any changes in target_defines.h. If you are running Tornado, use the following steps.

1. Create a new download application project.

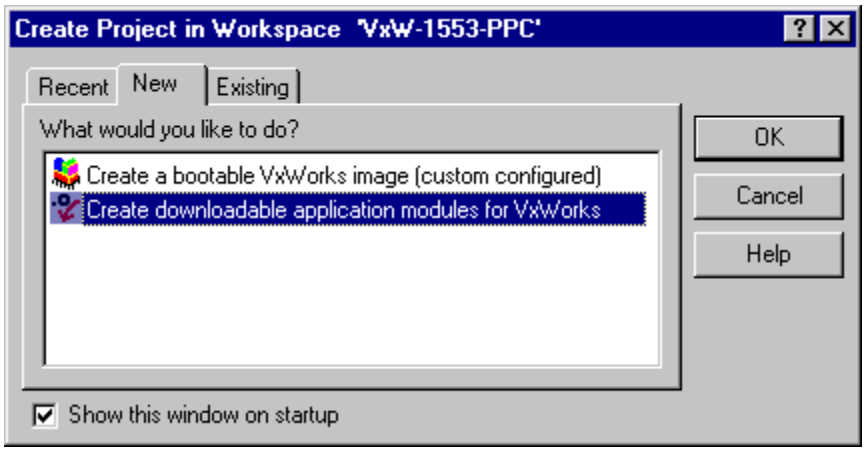


Figure 1. Creating a Download Application Project in Tornado Workspace

2. Add the API source files to the project.

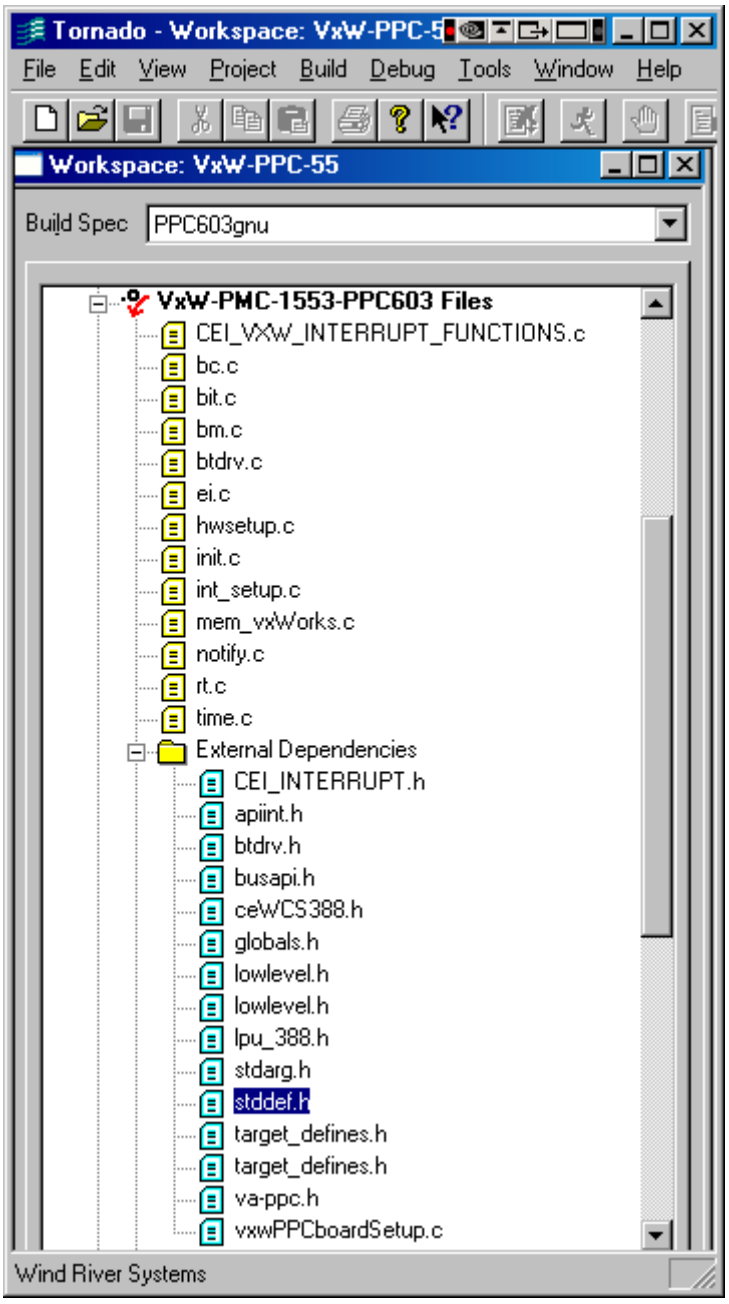


Figure 2. Adding API Source files to Tornado Project

3. Add the symbol definition for your processor type.
4. Add the path to the include files. You need the path to BusTools-1553-API/Include and BusTools-1553-API/Source.
5. Edit the project properties window to define symbols and add the include paths.

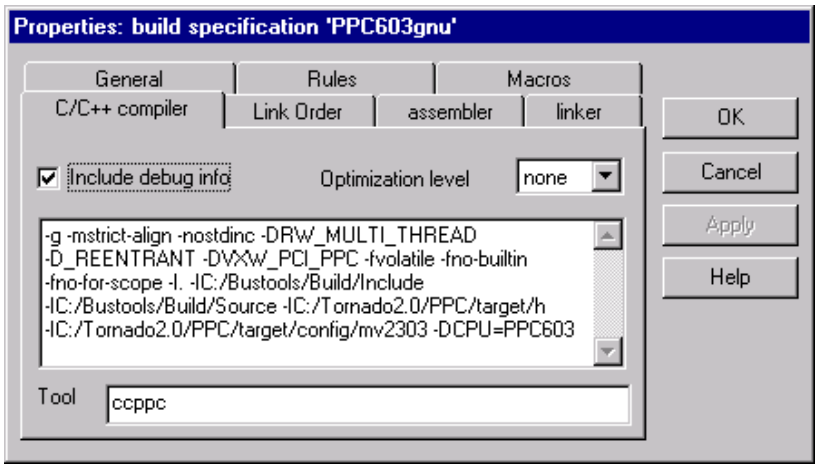
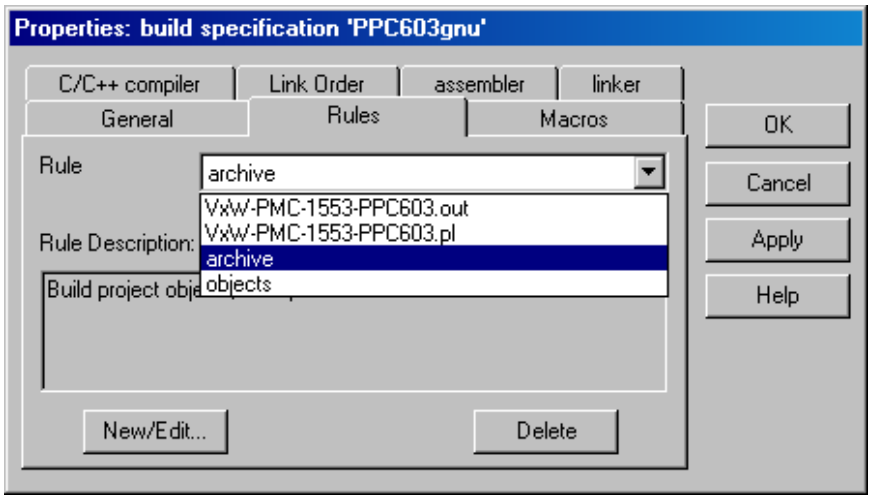


Figure 3. Editing the Tornado Project Properties

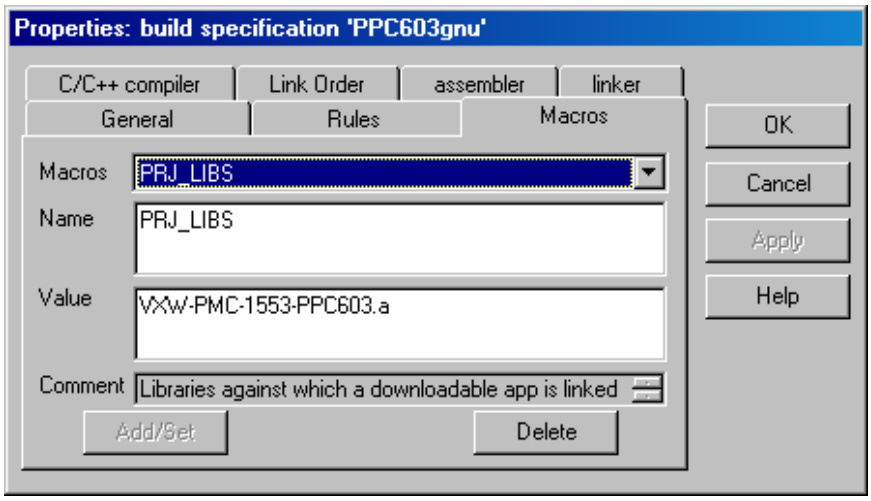
6. Build the downloadable application.
7. Download the API.

You can also use these steps to create your own applications calling API functions. Add your file in place of the API source files.

If you prefer to compile the API into a static library (archive file), you can select the Rules tab on the Properties Window and select the archive option.



The API compiles as a static library (.a) file. You can then link your application to this archive by going to the Macros tab for your application properties window and add the library to PRJ_LIBS macro.



VxW_Demo.c Test Program

The API distribution includes an example program named VxW_Demo.c. Use this program to test your installation. VxW_Demo.c sets up a Bus Controller, Bus Monitor and four Remote Terminals and provides the option to view the data from each of these operations. You can also use VxW_Demo.c as a guide for programming with the BusTools/1553-API.

Download VxW_Demo.out after you boot up and load the API.

Enter:

demo *device, channel*

in the command shell to start the application running. Where *device* is the device number and *channel* is the channel number (0 - 3, depending on board type). For VME boards enter the A16 address of the board in device parameter and A32 address in the channel parameter.

If VxW_Demo initializes without error, you see the following:

```
-> Using BusTools_RegisterFunction to process interrupts
interrupt mode is Software 1
Calling Open Channel with mode = 1
Initializing channel 0 on device 0
Initialization status = 0
BusTools_BM_Init status = 0
BusTools_SetBroadcast status = 0

VxW_Demo:
Start the BM (1), BC (2) and RT (3).  To display data
type:
c - for Bus Montroller
r - for Remote Terminal
m - for Bus Monitor
Select options from Menu below...
```

```
Hit q to Exit
Type 1 to start the Bus Monitor
Type 2 to start the Bus Controller
Type 3 to start the Remote Terminal
Type 4 to stop the Bus Monitor
Type 5 to stop the Bus Controller
Type 6 to stop the Remote Terminal
Type C for Bus Controller output - 'x' to quit
Type M for Bus Monitor output - 'x' to quit
Type R for Remote Terminal output - 'x' to quit
Type E for External Bus
Type I for Internal Bus
Type D for Direct Couple
Type T for Transformer Couple
Type V for Version information
Type W for Cable Wrap Test
Type A for Aperiodic Message
Type B for Internal BIT test
Type H for lists of commands
```

If you have a multi-function card, you can run the BC, RT, and BM functions together. VxW_Demo displays message traffic to the screen. Type C, R, or M to display the messages to the screen. The message scrolls on the screen until you type an 'x'. Below is a display of this information for the Bus Controller.

```
BC Time-tag = (3257)18:18:01.743362
BC->RT (0-1)
BC**RT-2
BC**SA-2
BC**WC-2
BC**status-1 0x1000
BC**int_stat-0x00010000
BC**2000 BC**2001

BC Time-tag = (3257)18:18:01.743362
BC**RT->BC (1-2)
BC**RT-3
BC**SA-3
BC**WC-3
BC**status-1 0x1800
BC**int_stat-0x00010000
BC**f003 BC**f003 BC**f003

BC Time-tag = (3257)18:18:01.743362
BC->RT (2-3)
BC**RT-4
BC**SA-4
BC**WC-4
BC**status-1 0x2000
BC**int_stat-0x00010000
BC**4000 BC**4001 BC**4002 BC**4003

BC Time-tag = (3257)18:18:01.743362
BC**RT->BC (3-4)
BC**RT-5
BC**SA-5
BC**WC-5
BC**status-1 0x2800
```

```
BC**int_stat-0x00010000
BC**f005 BC**f005 BC**f005 BC**f005 BC**f005
```

If you have a single function board, you can only run one function at a time per channel. Select BC, RT, or BM and stop one function before starting another. Running the BC function will allow you to display BC messages. If you run the BM or RT function, you need to connect to an external 1553 bus to display data.

If you want to check out interrupts with VxW_Demo then pass 2 for the "int_mode". This will set up interrupts on RT messages. The interrupt callback function prints data to the console screen (not the shell). You should start VxW_demo and select the 1, 2, and 3 options to start the BC, RT, and BM. The RT interrupt data appears on the console. If you have a single function board you can only run a single function at a time, so start the RT and connect to a MIL-STD-1553 bus with messages sent to RT 2, 3, 4, and 5.

For further information about programming with the BusTools/1553-API, refer to the "BusTool/1553-API Reference Manual".

Initializing the API

VxWorks API requires the use of either BusTools_API_OpenChannel() or BusTools_API_InitExtended() to initialize the board. PCI boards including PMC on PowerPC can use either initialization function. VME boards must use BusTools_API_InitExtended(). Here are examples of this API call for various 1553 boards.

RQVME2-1553 or QVME-1553 on a PowerPC or x86

```
status = BusTools_API_InitExtended(
      cardnum,          // Handle to 1553 channel.
                        // Used with all other API
                        // calls
      0x10000000,       // base memory address.
      0xc3c0,           // base A16 register
                        // address.  default=0xc3c0
      &flag,            // pointer to "int", set=1
      PLATFORM_PC,      // Fixed platform value
      VME1553,          // Card type
      NATIVE_32         // carrier type use NATIVE_32
      or NATIVE_24
      CHANNEL_x,        // Channel number x=1,2,3,4
      CARRIER_MAP_A32); // Mapping A24 or A32
```

VME devices are identified by their A16 and A32 addresses. You must have unique addresses for each VME device in the chassis.

All Plug-n-Play boards on a PowerPC or Intel x86

```
Status = BusTools_API_OpenChannel(
   &cardnum,       // A pointer to cardnum
   mode,           // Same as pwFlag, not a pointer
   device,         // The device number
   CHANNEL_x);     // Channel number x=1,2,3, or 4
```

The device is always 0 if you have only a single board installed. If you have multiple boards, the device number is determined by how the PCI bus enumerated the boards. You need to find out the device number for each 1553 board. Device numbers start at 0 and increment for each device. Refer to the BusTools1553-API Reference Manual for more information and examples on Initialization.

Notes

ANSI option

VxWorks normally includes the –ansi compiler directive. This option does not recognize C++ style comments (//). You need to delete this directive or convert C++ style comments to standard C style comments (/* …*/) for correct compiles.

#pragma alignment

VxWorks uses the keyword "__attribute__" to get the correct alignment. This allows you to specify special attributes of variables or structure fields. You must use the keyword with every variable and structure element requiring the attribute.

__attribute__ ((aligned(n))) specifies the minimum alignment for the variable or structure field in bytes, where n = byte alignment.

__attribute__ ((packed)) packs the data leaving no gaps.

The following example shows alignment and packing of a structure to ensure that it is six bytes. Without adding the "__attribute__" keyword, this structure is eight bytes long and padded with zeros when compiled with the GNU CC compiler for an x86 target.

```
typedef struct rt_cbuf
   {
     BT_U32BIT legal_wordcount __attribute__  ((aligned(2),packed));
     BT_U16BIT message_pointer __attribute__  ((aligned(2),packed));
   } RT_CBUF;
```

Above, packed and aligned are used in a single "__attribute__" declaration. Refer to the GNU CC reference manual for more information on the "__attribute__" keyword.

Endian-ness

The Intel x86 processor uses little Endian bit ordering, while PowerPC processors use big Endian bit ordering. The following lines are added to Busapi.h when using a PowerPC.

```
/* Big Endian Ordering */
#define NON_INTEL_BIT_FIELDS  /* Motorola Bit Ordering  */
#define NON_INTEL_WORD_ORDER  /* Motorola Word Ordering */
```

When converting between the two processors make sure you define these macros for PowerPC and "undef" them for Intel x86 Processors. Some PowerPC processors have PMC slots. If you are using a PowerPC with the PMC-1553, you need to define the WORD_SWAP symbol or use the Little Endian ordering available on some PowerPCs. The BusTools/1553-API automatically defines the correct symbols based on the processor type you specify when you build the API.