acontis technologies GmbH

SOFTWARE

VxWin

VxWorks & MS Windows

Product Manual
Edition: 2018-12-04

Compact Table of Contents

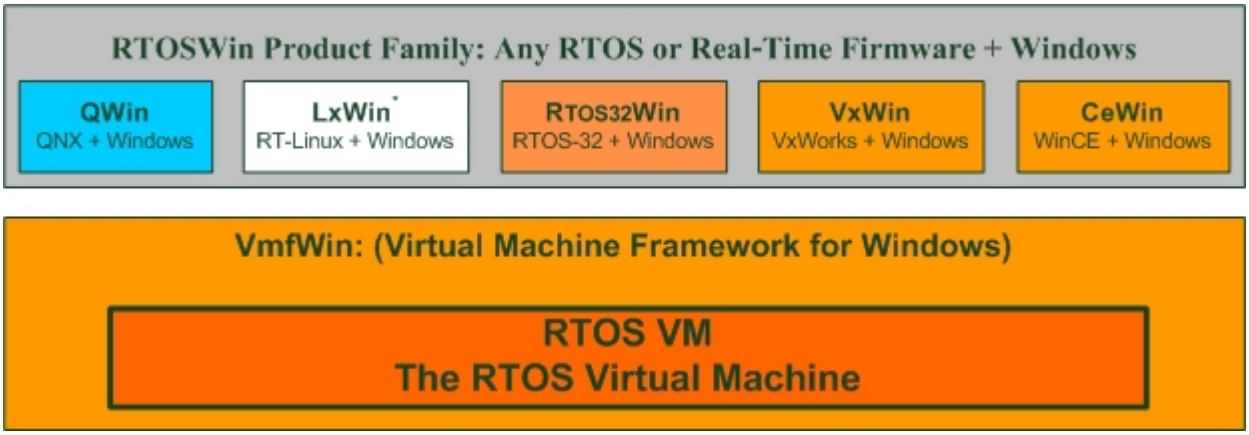
Table of Contents

1 Introduction

1.1 The ACONTIS RTOSWin Product Family

The ACONTIS RTOSWin family is a family of Windows virtualization solutions for multiple real-time operating systems and Microsoft Windows.

The key component of all these solutions is the RTOS Virtual Machine. The real-time operating systems are executed on top of the RTOS VM.



acontis technologies GmbH is providing a special VxWorks BSP. Using this BSP together with the RTOS VM runtime VxWorks can be executed together with Windows.

More details about the virtual machine can be found in the RTOS VM User Manual.

- Operating Modes
- Realtime Device Management (how to control hardware)
- RTOS VM configuration
- Booting the RTOS
- Communication Services: The RTOS Library

1.2 The ACONTIS RTOS Virtual Machine

The ACONTIS RTOS-VM provides a light-weight real-time virtualization platform for Windows.

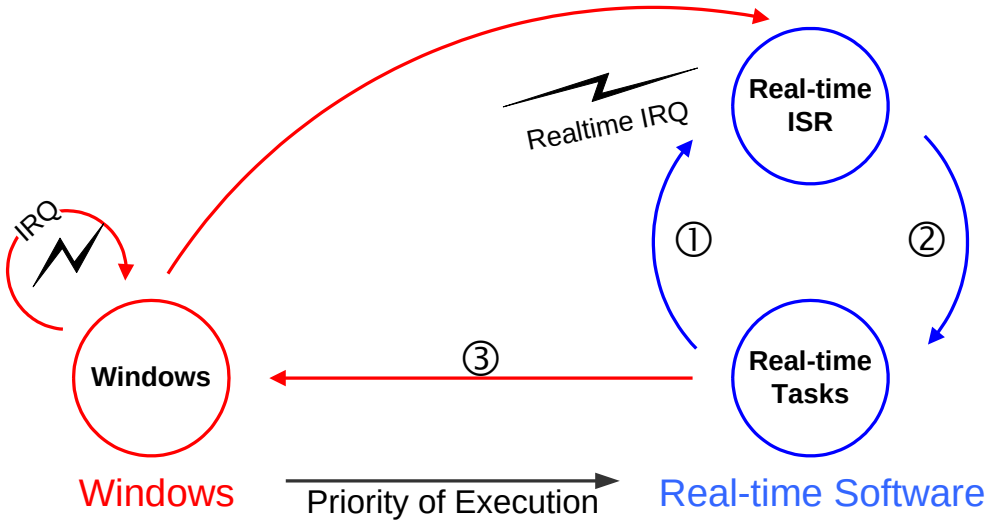
On top of this platform either real-time firmware, custom or off-the-shelf real-time operating systems can be executed.

When using multicore CPUs one can choose between two general operation modes. A more detailed description about possible operation modes can also be found in the RTOS VM User Manual.

1.2.1 Shared Mode Operation

Windows shall run on all CPU cores and only one CPU core shall additionally run the real-time software. If the Windows application needs a lot of CPU power (e.g. for image processing) this will be the appropriate operation mode even on multi-core CPUs. In shared mode operation Windows (on this core) will usually only get CPU time when the real-time software is idle.

The following diagram illustrates the flow of control:



Operating states of the RTOS-VM in shared mode

① Exception-handling or a higher priority interrupt becomes outstanding.

② Interrupt Service Routine optionally starts a new task and then finishes.

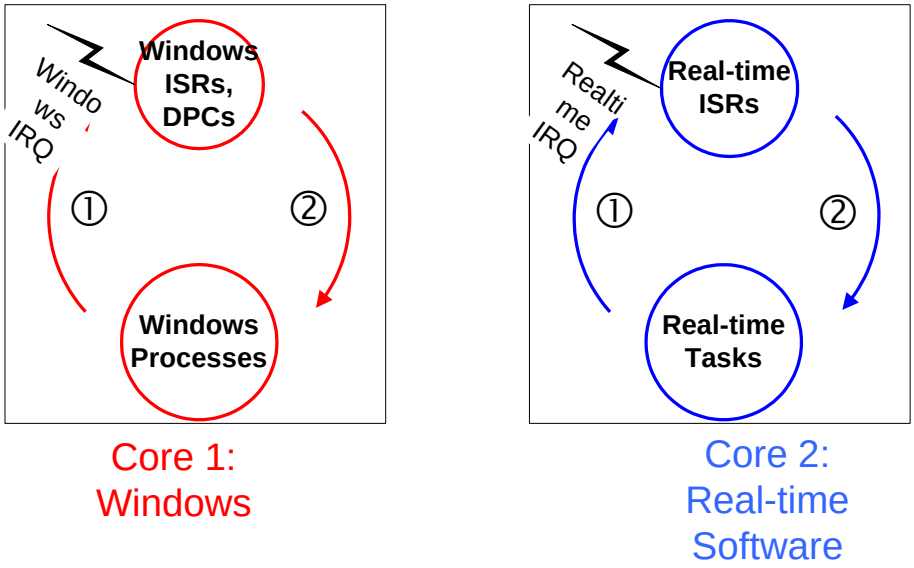
③ From the idle-state, VxWorks transfers control to Windows operating system.

Note: When running the RTOS-VM in shared mode on multiprocessor/multicore systems this state diagram is only applicable for one CPU core in the system (by default on the first core). All other CPU cores will run Windows only.

1.2.2 Exclusive Mode Operation

Windows and the real-time software shall run fully independently on different CPU cores. Using this mode will lead to much shorter interrupt and task latencies as there is no need to switch from Windows to the real-time software.

The following diagram, illustrates the flow of control on a dual core system:



Operating states of the RTOS-VM in exclusive mode

① Exception-handling or a higher priority interrupt becomes outstanding.

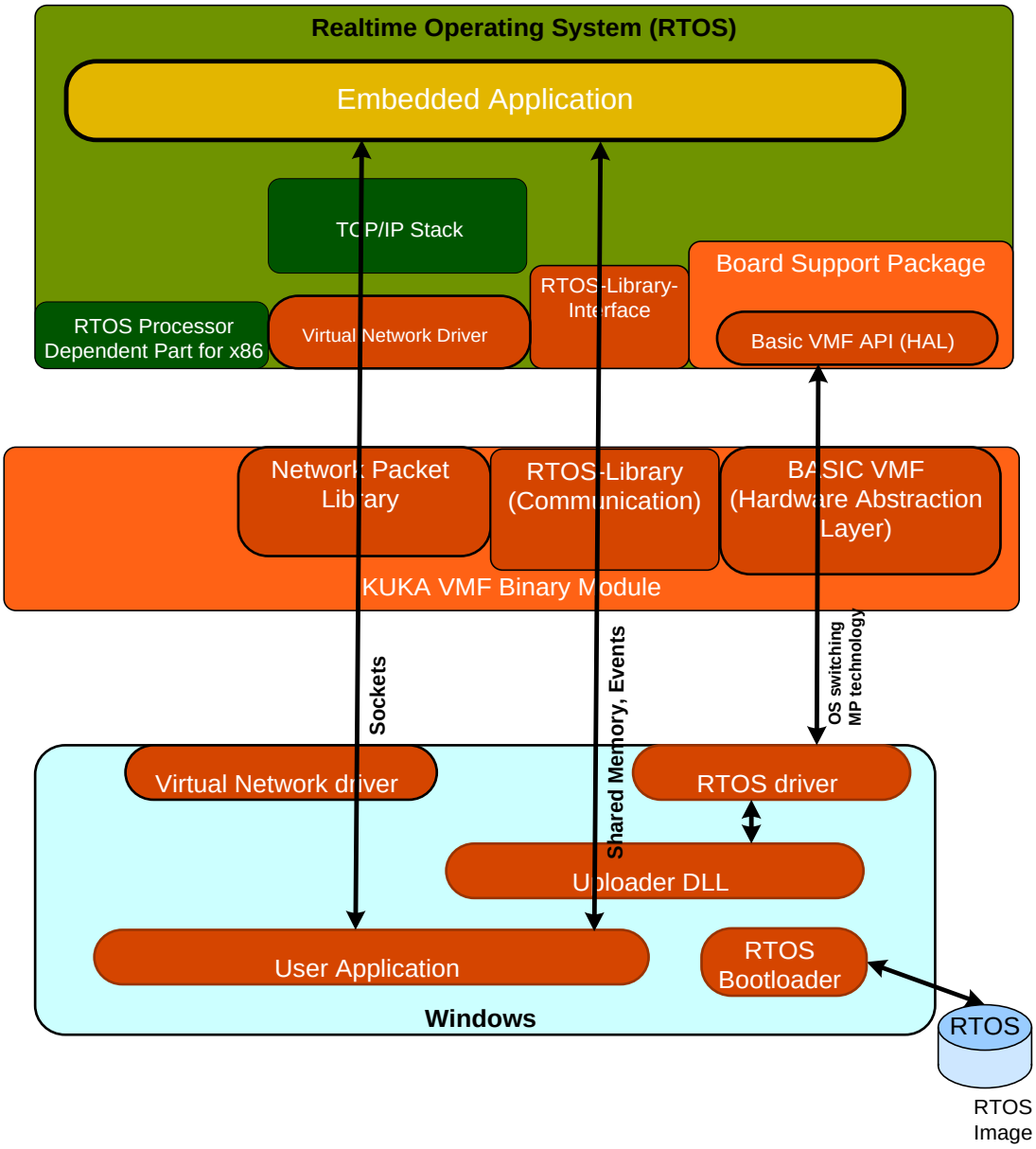
② Interrupt Service Routine optionally starts a new task and then finishes.

Note: When running the RTOS-VM in exclusive mode Windows will never be interrupted. Application and interrupt processing run concurrently and independently on both CPU cores. There is no need in the real-time software to enter the idle state.

1.3 VxWin Architecture Overview

VxWin is split into two main components:

a) The ACONTIS RTOS Virtual Machine runtime (VMF runtime)
b) The VxWorks BSP. This component is provided by acontis technologies.



1.4 Components

1.4.1 Development components

To develop software for VxWin the following components are provided.

a) VxWin BSP for VxWorks
   Important:
   The VxWin BSP is not part of the VxWin setup program. Please contact Customer Support for the latest VxWin BSP adequate to your desired WindRiver® VxWorks version.

b) VxWin development components (SDK + Documentation)
   The following files are shipped with VxWin for development support:
   - VxWin User Manual
   - RTOS VM User Manual (basic technology description)
   - RTOS Library for Windows/RTOS communication
   - SDK with Example applications

1.4.2 Runtime components

The runtime components are split into the following main parts:

a) VxWin runtime components (ACONTIS RTOS Virtual Machine)
b) Configuration files.
   For more information about the Configuration files see chapter 2.1.
c) VxWorks VxWin runtime images (this is the image built by VxWorks + VxWin BSP)

2 VxWin configuration settings

2.1 VxWin.config – the VxWin configuration file

All VxWin configuration settings are stored in the file VxWin.*config* (e.g. C:\Program Files\acontis_technologies\VxWin\Bin\Windows\Config\). If you wish to use a different configuration file, you may specify another filename and/or a different path by using the *–config* option of the Uploader command.

General configuration settings are described in the RTOS VM User manual (e.g. memory configuration, operation mode).

2.1.1 Syntax

The configuration file is an ASCII file that can be modified with a simple editor. Its syntax is similar to that of a Windows registry file.

2.1.1.1 General rules

- The first entry in a configuration file must be *RtosConfig*.
- Because comments are introduced by a semicolon (;), all characters following a semicolon will be ignored.
- No single line in the configuration file may exceed 256 characters.

2.1.1.2 Keys, Entries, and Values

The configuration settings are stored using keys. Specific settings are stored in entries. Every entry has both a name and a value. Every entry is subordinated to a specific key.

Example:

```
[Key]

      "EntryName"=EntryValue
```

There are different types of values:

- Single hexadecimal value (dword)

  "EntryName"=dword:1F
- Strings

  "EntryName"="This is a string"
- Multiple Strings

  "EntryName"=multi_sz:"First string","Second string","Last string"
- Multiple hexadecimal values

  "EntryName"=hex:XX,YY,…,ZZ

2.1.1.3 Include statements

Configuration commands and parameters can be split into multiple files. You may use an *include* statement to incorporate other configuration files into *VxWin.config*.

Examples:

```
#include "myOs.config"
#include "myApp.config"
```

Important notes:

If no additional path information is given, included files must be located in the same directory as *VxWin.config*.

Since nested includes are not supported, *include* statements may only be used in the main configuration file.

Common configuration files:

Filename	Meaning
VxWin.config	Main configuration file – contains includes to other configuration file(s) and maybe some common parameters
VxWinSmp.config	Configuration file used for SMP – contains additional IPI configuration.
General.config	General configuration file – contains information for the Uploader Utility

2.2 VxWin specific parameters

The user may define the following entries for VxWin under the [***rtos***] key:

Entry Name	Type	Description
Bootline	String	VxWorks boot line.
Console	string	VxWorks device name which shall be used for the console interface
VnetMACAddress	string	Virtual Network Adapter MAC address.
VnetPollPeriod	dword	Virtual Network Adapter polling period in timer ticks. 0 enabled interrupt mode
VnetNumCluster	dword	Number of network clusters for the Virtual Network Adapter.
AddNetworkX	string	Additional network interface where the IP stack shall be attached to. X==0 (AddNetwork0): first additional interface X==1 (AddNetwork1): second additional interface
LogNetworkInterfaceX	string	Network interface for which the network packet logger shall be enabled. X==0 (LogNetworkInterface0): first interface for packet capturing.
LogNetworkFileX	string	File where the network packet log shall be stored.
LogNetworkFileMaxSizeX	dword	Maximum capture file size in kByte

2.2.1 Bootline

As shown in the following example, the user may override the default VxWorks boot line by coding his/her own under the *[rtos]* key:

```
[rtos]
"Bootline" = "vnet(0,2)pc:vxWorks h=192.168.0.1 e=192.168.0.2 u=target pw=vxworks"
```

2.2.2 Console

By default the last serial channel is connected with the RTOS VM's virtual I/O channel. The VxWorks console will be redirected to the Target Console Window through the config file entry [Rtos] "Console"="/vio/0".
Alternatively the console can be redirectet to any serial port – for example first port – using "/tyCo/0".

2.2.3 VnetMACAddress

The Virtual Network adapter address [also known as the MAC (Media Control Address) address] is generally specified in the following format:
AA-BB-CC-DD-EE-02

2.2.4 VnetPollPeriod

The Virtual Network driver by default operates in polling mode. The polling period can be adjusted using this parameter. It is set in units of the system clock timer. A values of 0 enables the interrupt mode.

2.2.5 VnetNumCluster

Cluster size of the Virtual Network driver. In rare cases where extensive network traffic occurs this parameter has to be increased.

2.2.6 AddNetworkX

By default the IP stack will be attached to the virtual network. If additional network adapters shall be under control of VxWorks these adapters can be automatically attached to the IP stack using this parameter (the X has to be replaced by contiguous number beginning with 0). The following example will attach the fei0 device (first instance of the Intel PRO/100 network adapter) to the IP stack. The IP address and subnet mask will be set accordingly.

```
[rtos]
"AddNetwork0"="fei0:192.168.100.1:255.255.255.0"
```

2.2.7 LogNetworkXxx

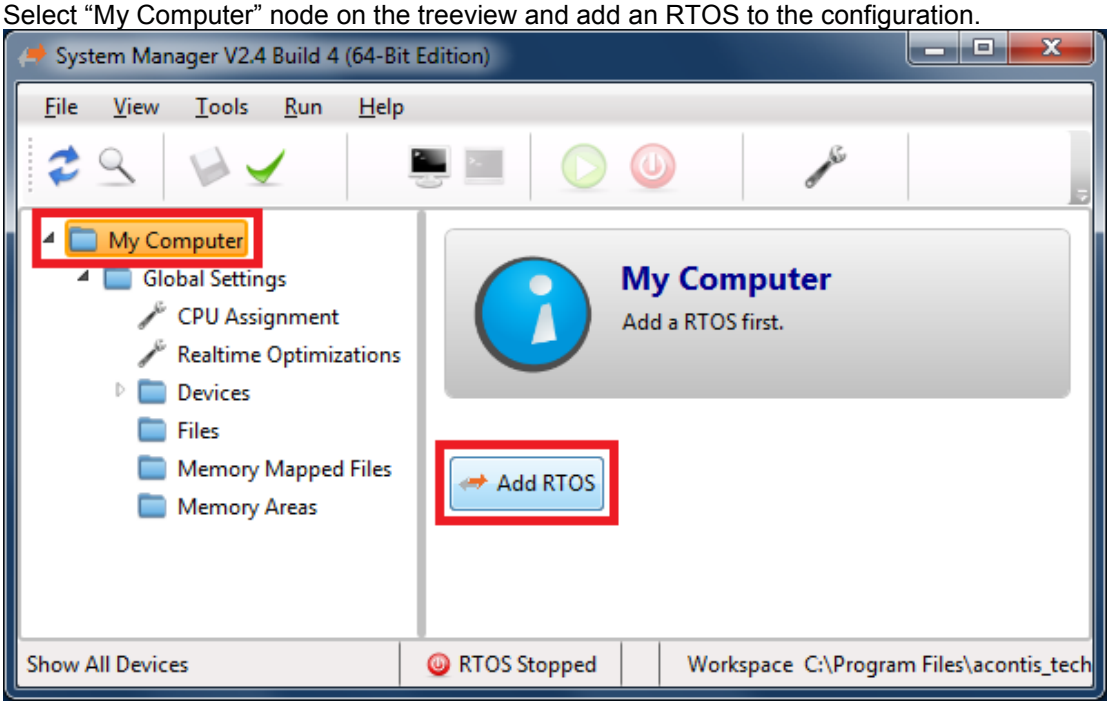
VxWin supports network packet logging which may help in solving network problems. Parameter *LogNetworkInterfaceX* selects the network interface where packets shall be captured. All captured packets will be stored in a file selected by parameter *LogNetworkFileX*. The maximum size of the file is limited by parameter LogNetworkFileMaxSizeX (kByte). The format of the file is compatible with PCAP.

3 Tutorials

3.1 Running a shipped VxWorks image and start the RealtimeDemo application

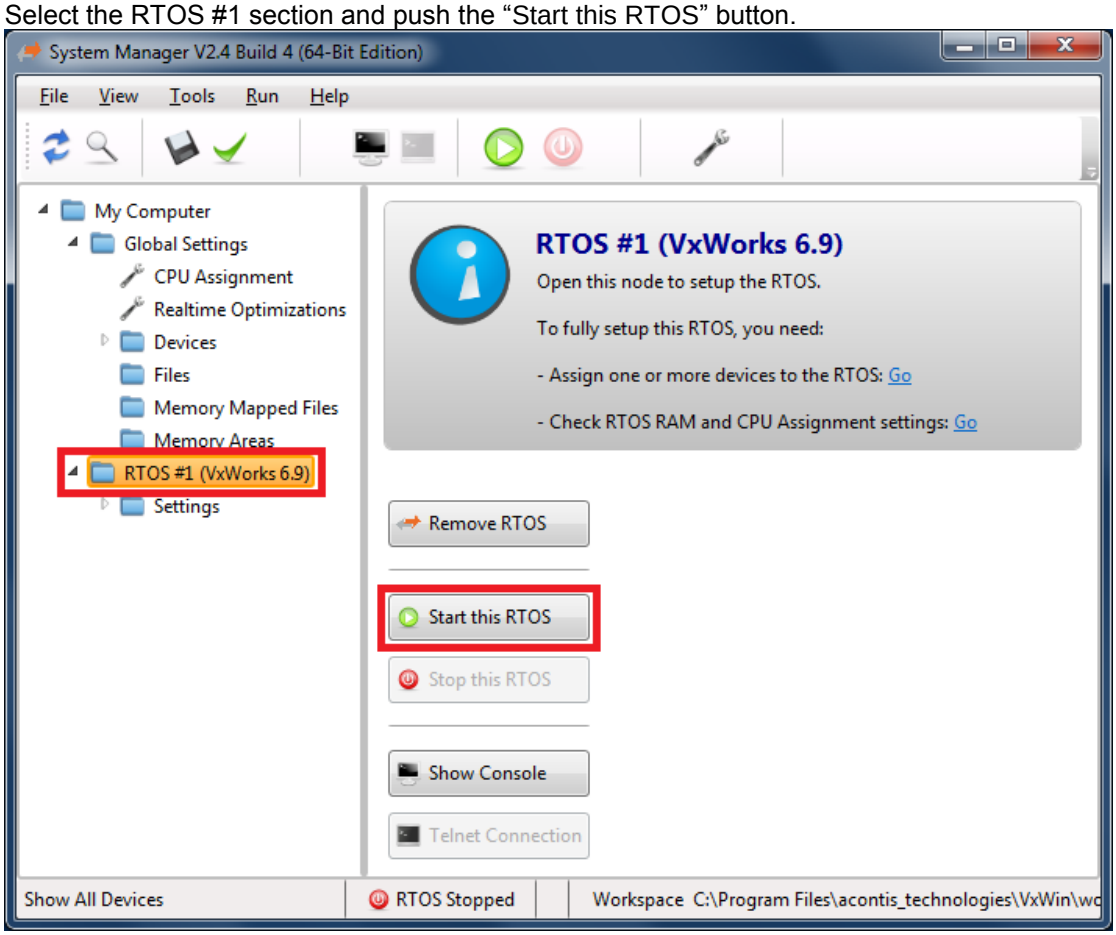
This tutorial shows how to run a shipped VxWorks image and start the RealtimeDemo example application.
It assumes that VxWin is installed on the PC.

- Start the System Manager
  → if it's the first launch of the System Manager opens a dialog to enter a workspace directory.
- Select "My Computer" node on the treeview and add an RTOS to the configuration.

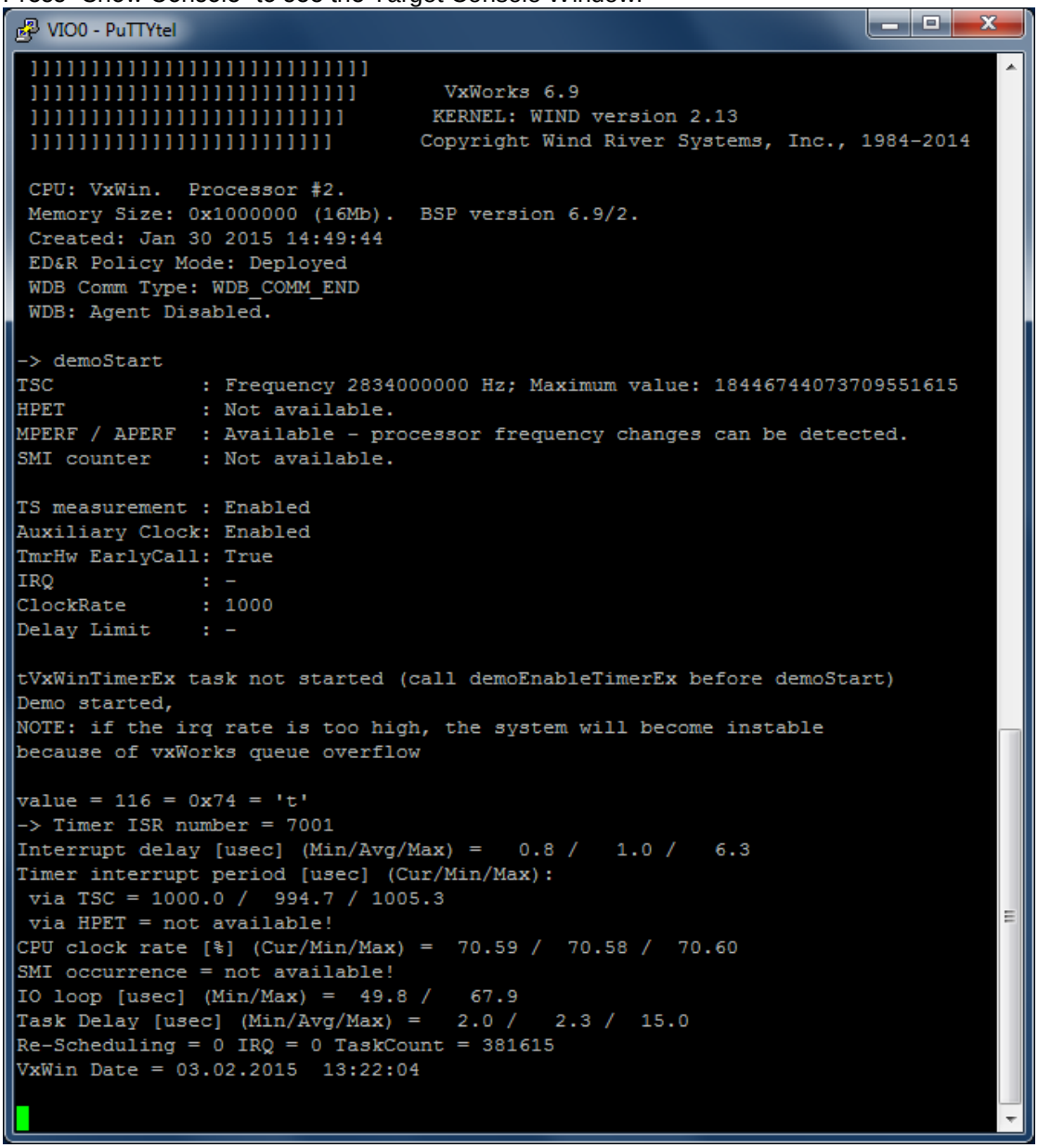


- Choose one of the shipped RTOS images to be used.

- Select the RTOS #1 section and push the “Start this RTOS” button.



- Press “Show Console” to see the Target Console Window:



Enter “demoStart” to start the Realtime Demo application.
The measure can be reset by entering “demoReset” and stopped by “demoStop”.

3.2 How to build the VxWorks example applications

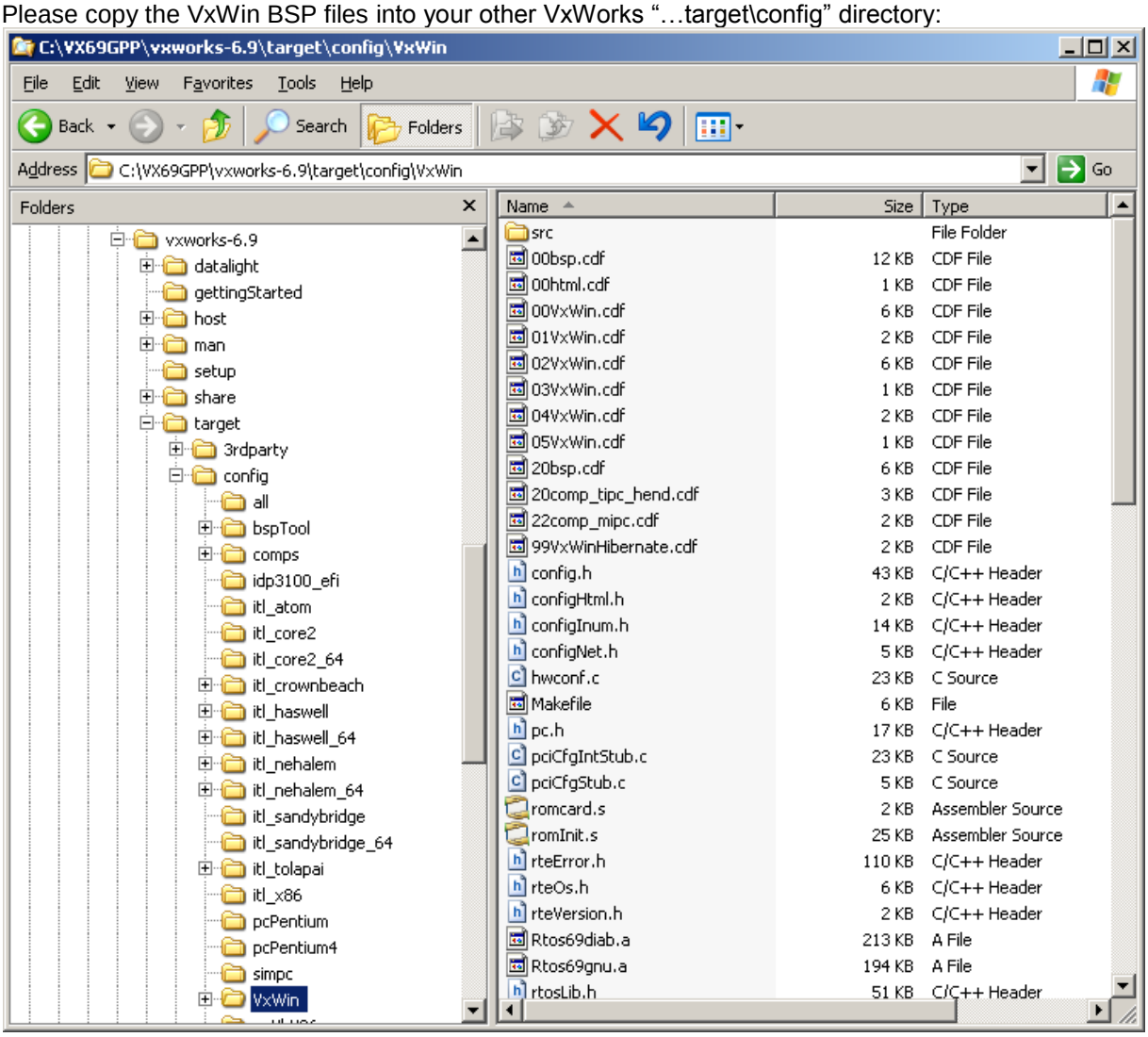
The different examples are described in detail in chapter 4 Example Applications. In the examples folder is additional a file HowToBuildTheExamples.txt. This file includes a short description how to build the examples for every example application.

3.3 How to make your own VxWin Image (BSP)

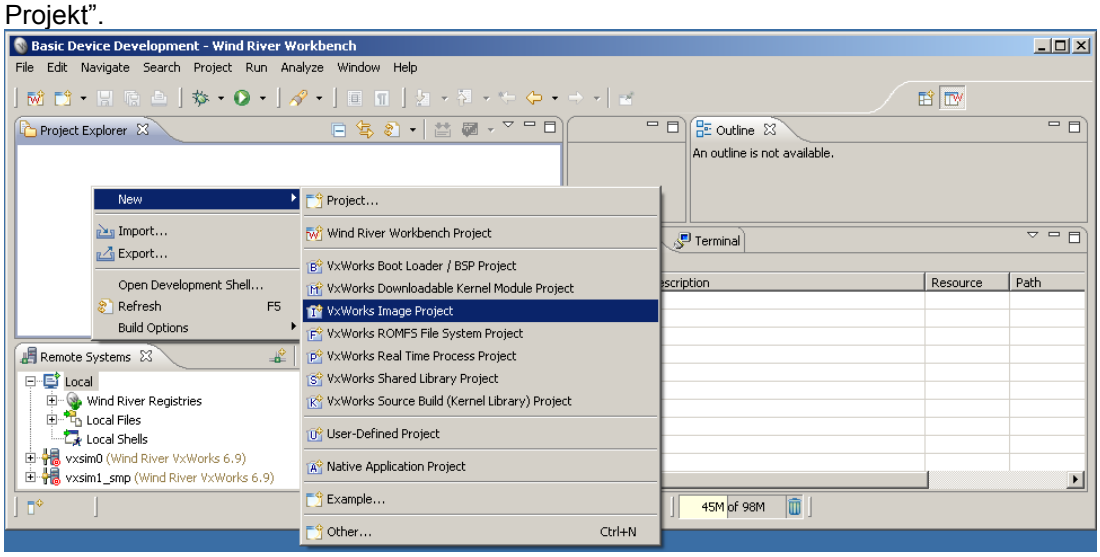
The VxWin BSP for VxWorks depends on your VxWorks version and is delivered separately and not part of the VxWin Setup.
See chapter “5.7

VxWin – Board Support Package" for additional information.

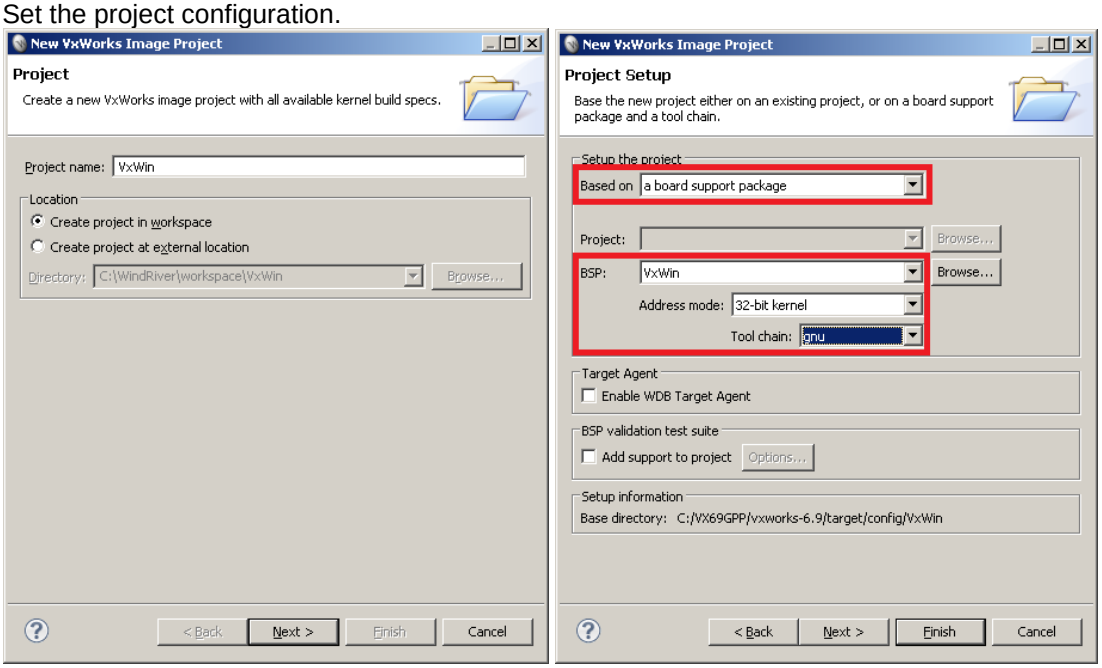
- Please copy the VxWin BSP files into your other VxWorks "…target\config" directory:



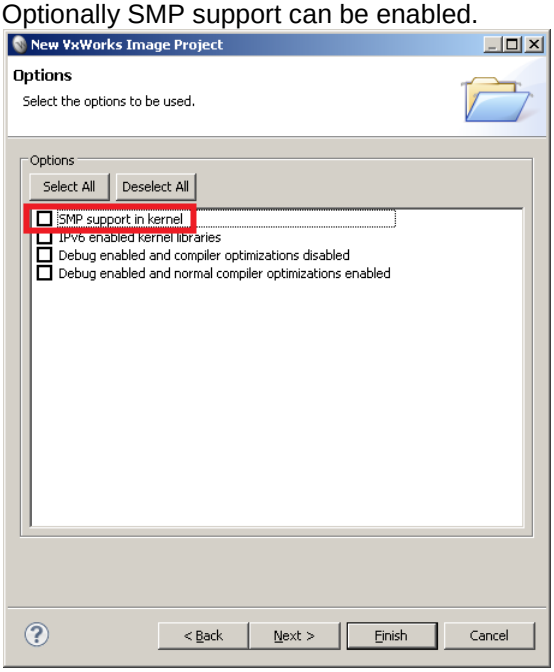
- Now Start your Workbench, right-click the “Project Explorer” and select “VxWorks Image Projekt”.



- Set the project configuration.

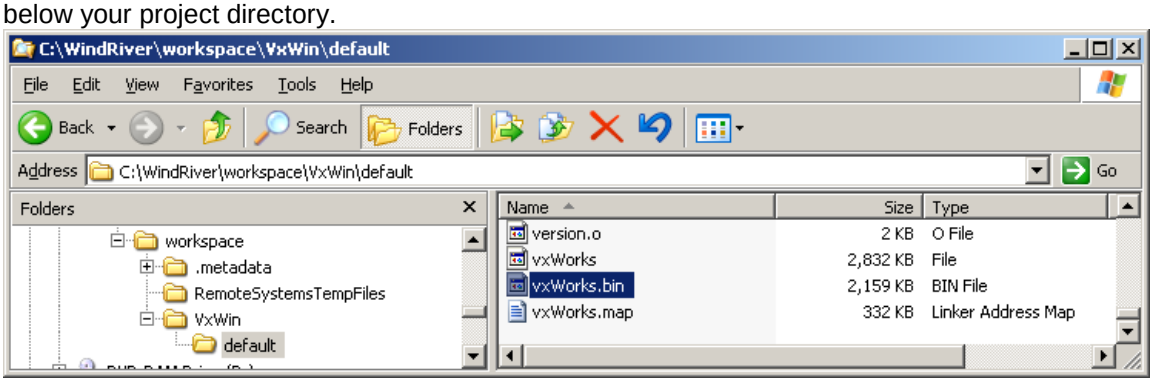


- Optionally SMP support can be enabled.

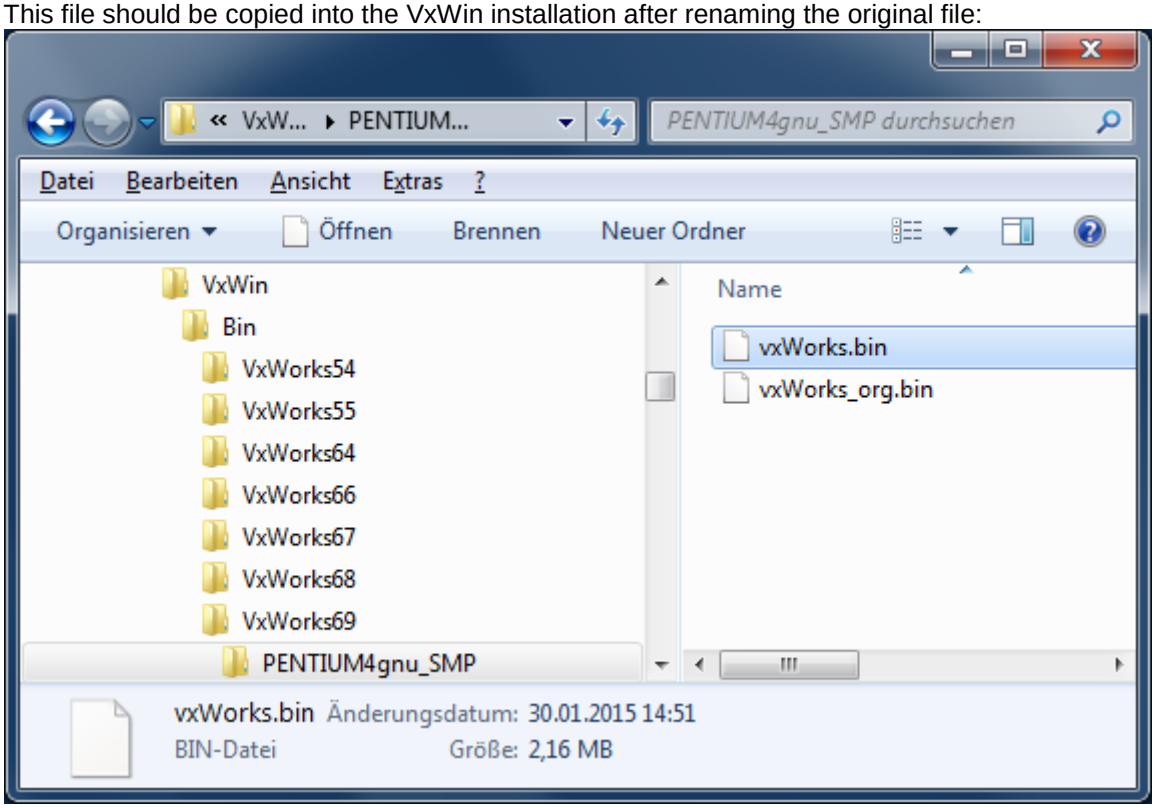


- Press “Finish” to create the project.

- After “Build” the project the file “vxWorks.bin” should be created in the “Default” directory below your project directory.



- This file should be copied into the VxWin installation after renaming the original file:



In this example the VxWorks 6.9 SMP file is replaced so in System Manager the RTOS "VxWorks 6.9 SMP" has to be selected to start the new image.

3.4 How to debug an Application

Please see chapter "5.1 Debugging - Using Tornado/Workbench with VxWin" for details.

3.5 How to debug the BSP

Please see chapter "5.1 Debugging - Using Tornado/Workbench with VxWin" for details.

4 Example Applications

4.1 General

To make your initial experiences in working with VxWin and VxWorks go smoothly, a number of example application programs have been provided with the product release. Some are intended as exercises help familiarize you with various system features; others are useful tools. Much of the code can serve as model software that can save you time in developing your own applications.

The example applications are located on the product release CD-ROM at *CD:\SDK\Examples*. Per default, setup copies the sample application programs to the directory: *C:\Program Files\acontis_technologies\VxWin\SDK\Examples*.

4.2 Building a Windows application

The Microsoft Windows example applications are located in “C:\Program Files\acontis_technologies\VxWin\SDK\Examples\Windows”.
You can create and debug these applications with Microsoft Visual Studio 2005 or newer.
A short description how to build the examples can be found in this directory (file HowToBuildTheExamples.txt).

4.3 Building a RTOS (VxWorks) application

The VxWorks example applications are located in “C:\Program Files\acontis_technologies\VxWin\SDK\Examples\VxWorks”.
A short description how to build the examples can be found in the directory of each example (file HowToBuildTheExamples.txt).

4.4 List of examples

These examples are based on the RTOS-Library. For more information about this library see the RTOS VM User Manual.

Windows	RTOS (VxWorks)	Description
CSharpDemo	-	The C# demo shows you how to use RtosLib functions from a C# application.
EventDemo	EventDemo	Use the Shared event demo to experiment with Shared Events between two OS.
-	FileDemo	The File demo shows you how a WinCE application can remotely access Windows file system.
InterlockDemo	InterlockDemo	The interlock demo shows you how to use the InterlockedCompareExchange function to synchronize shared memory access works between two OS.
MsgQueueDemo	MsgQueueDemo	Use the Message queue demo to see how a simple communication channel can be established between two OS using a shared memory based message queue to transfer data.
NotificationDemo	NotificationDemo	The Notification demo shows you, how to receive notifications when a blue screen error or other system events occur. The available events depend on the OS.
PipeDemo	PipeDemo	Use the Pipe demo to see how a simple communication channel can be established between two OS using a shared memory based pipe to transfer data.
ShmDemo	ShmDemo	Use the Shared memory demo to see how a simple communication channel can be established between two OS using Shared Memory to transfer data.
SocketDemo	SocketDemo	Use the Socket demo to see how a simple communication channel can be established between two OS using shared memory based socket to transfer data.
-	StdioFileDemo	The File demo shows you how a WinCE application can remotely access Windows file system using the C standard library functions.
UploadDemo	-	Use the Upload demo to see how a simple application can start and stop a RTOS.

5 User's Guide

5.1 Debugging - Using Tornado/Workbench with VxWin

The following debugging methods may be used:

Task-Mode Debugging

Customarily, Tornado/Workbench is used to debug one application task at a time. For this kind of debugging, a debugger must be initially attached to a task. Thereafter, the user can display and modify variables, perform stepped execution, or let the task run to a breakpoint.
In this debugging-mode, when a breakpoint is reached, only the task being tested is stopped, the rest of the system continues to run.
For detailed discussions of how to use task-mode debugging, refer to the Tornado User's Guide (especially Chapter 10), or the Workbench User's Guide (beginning at Chapter 18).
To debug Interrupt Service Routines (ISR's) or multiple threads, however, system-mode debugging must be used, as described in the next section.

System-Mode Debugging

System-mode debugging (external-mode debugging) is typically used to develop Interrupt Service Routines (ISR), because IRSs run outside of any real-time task context. It is also used whenever multiple threads are to be simultaneously debugged.
Under system-mode debugging, whenever a breakpoint is reached, the entire target system is stopped. For this reason, when using system-mode debugging, the Tornado/Workbench debugger must be hosted on an external system, not on the VxWin system.
For more detailed information about system-mode debugging, please refer to the Tornado User's Guide or the Workbench User's Guide.

—

5.1.1 Two development arrangements

To develop software for VxWin, the user may choose either of two basic host/target configurations:

- **The one-system method**: Develop a VxWorks application under Windows XP and then run it under VxWin on the same computer...

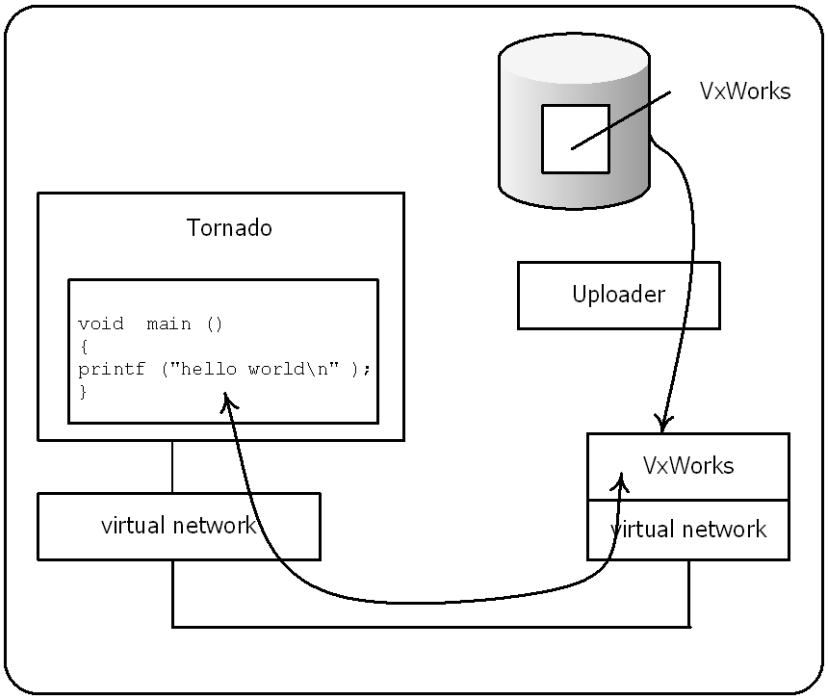
  -or-

- **The two-system method**: Develop a VxWorks application on one computer and run it on another computer, on which VxWin is installed. If you choose the two-system method, you can use a single- or a double-network technique.

In the following discussion, reference is made to a "target server" and a "run-time" or "target" agent. These debugging elements are part of Tornado/Workbench. For details on how to configure and use them, please refer to Wind River's *Tornado User's Guide* or *Workbench User's Guide*.

5.1.2 The one-system method

In a combined host/target system the connection between Windows and the target is made using the VxWin Virtual Network.



In this configuration, the development software runs under Windows XP and the run-time target's components run under VxWorks, which was installed on the same PC. The development process is as follows:

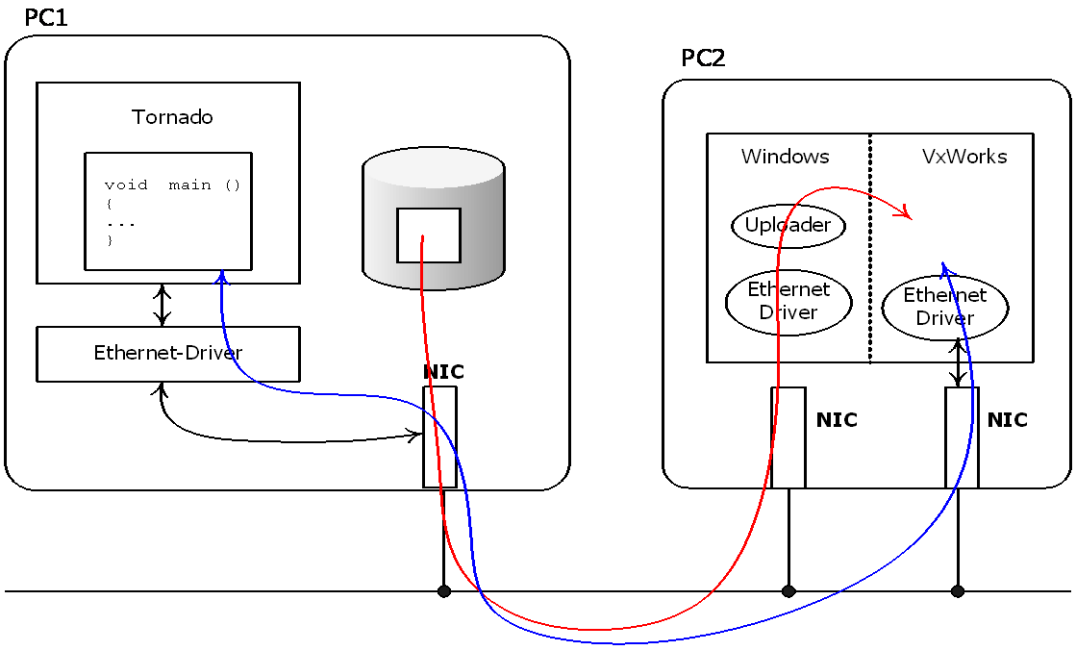
- Step 1: Build a VxWorks image.
- Step 2: Using the Uploader Utility, transfer the VxWorks image into the program memory.
- Step 3: Under Windows XP, create a target-server connection to a VxWorks target agent.
- Step 4: Download the application and start debugging it.

5.1.3 The two-system method

When host and target systems are physically separated, the connection between them can be made using either of two techniques.

Two systems – three network adapters

Using this technique (as illustrated below), the target system has two network adapters, one for Windows alone and one for VxWin:

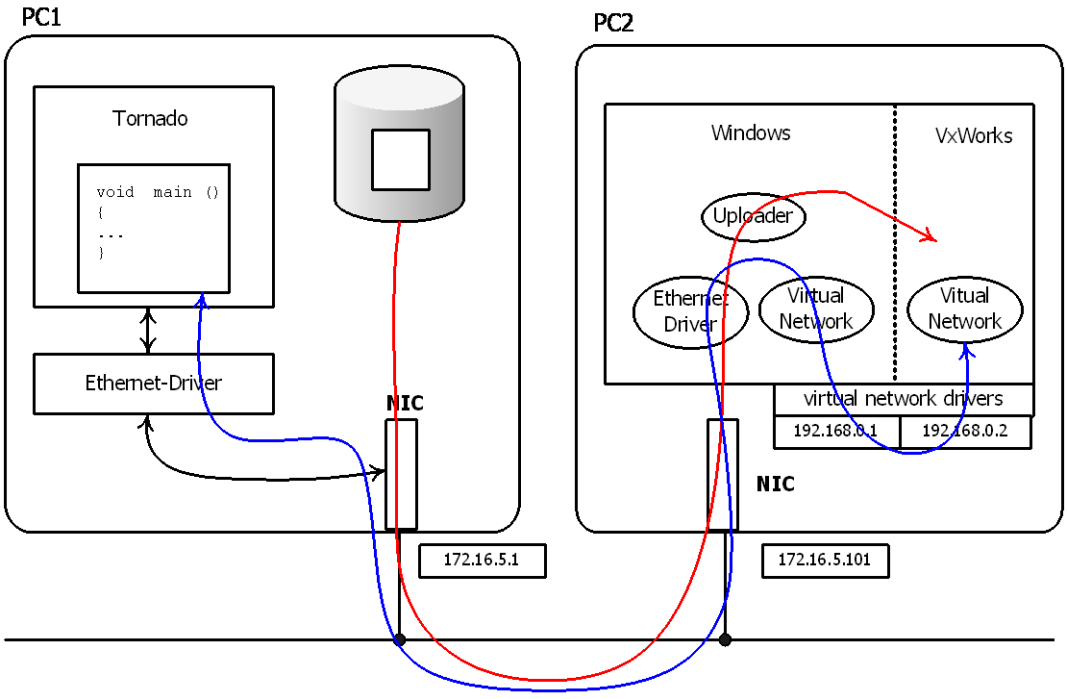


Development components are installed on the host computer – PC1, while run-time components are installed on the target computer – PC2. The development process is as follows:

- Step1: Build a new VxWorks image on PC1.
  Note: As shipped, VxWorks images may not contain support for a network adapter
- Step 2: To make the image on PC1 accessible from PC2, you must manually create a network share on PC1. Then, using the Uploader Utility on PC2, load the VxWorks image.
- Step 3: Create a target-server connection on PC1 to a VxWorks target agent on PC2.
- Step 4: Download the application and start debugging it.

Two systems – two network adapters

Using this technique, the target system has only one network adapter which is used by both Windows and VxWorks. The dual use of one adapter is accomplished by using "packet routing," described elsewhere in this manual. See Packet routing: VxWorks Uses the Windows network adapter. Packet routing ensures that all data packets sent to or originating from VxWorks on PC2 will be sent through PC2's Ethernet adapter.



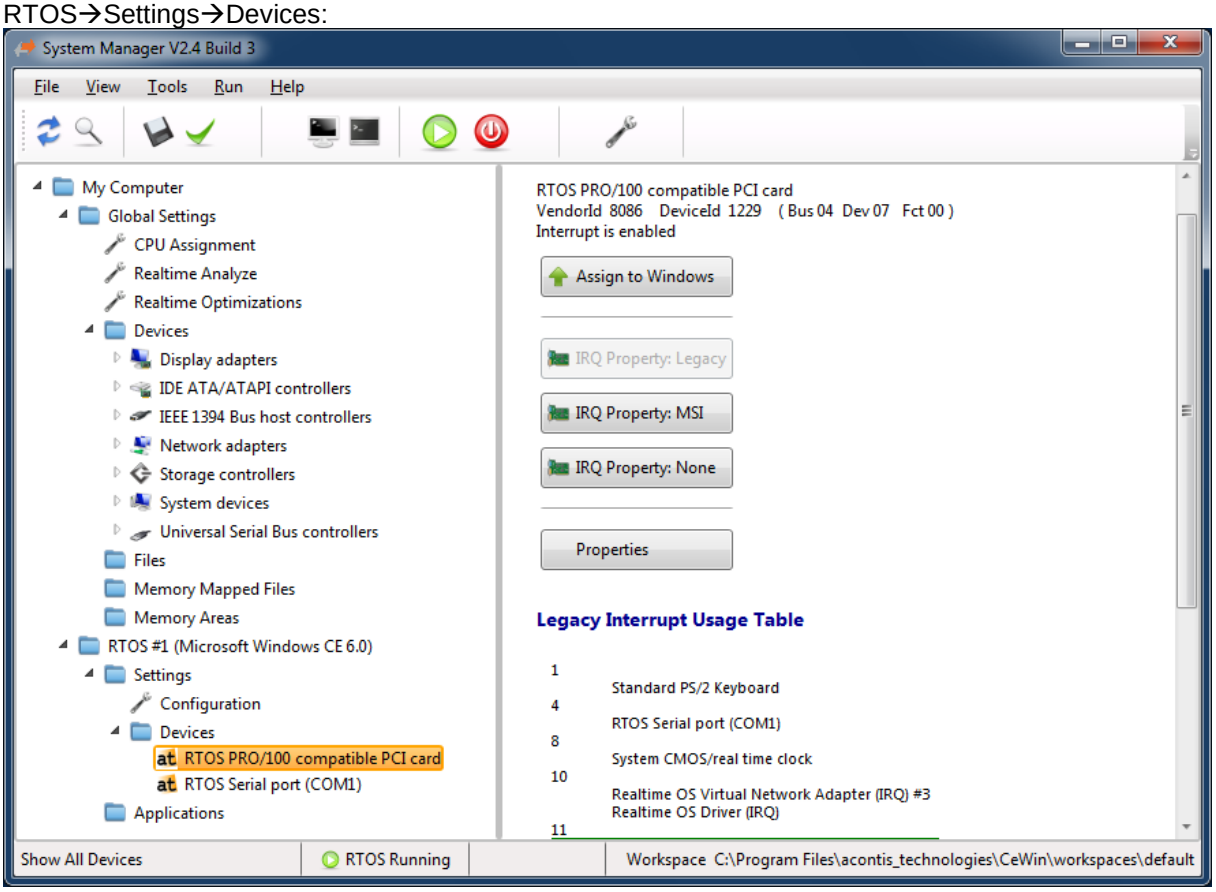
Here too, development components must be installed on PC1 while run-time components must be installed on PC2.

5.2 Accessing PCI cards

When PCI cards shall be controlled by VxWorks the PC will have to be partitioned first.
By default, all hardware belongs to Windows.
To being able to access hardware from within VxWorks it will have to be separated from Windows.
This is described in detail in the RTOS VM User Manual.

After separation is done, the usual methods within VxWorks can be used to access PCI cards. See the VxWorks manuals for more information.

Within the System Manager all devices assigned to a RTOS will appear below RTOS→Settings→Devices:



5.3 RTOS Library

The RTOS library provides higher level communication services for synchronizing Windows with VxWorks or to exchange data between the operating systems. The RTOS library is based on VMF-functions which provide the basic communication functionality.
A description of the RTOS Library can be found in the document RtosVM User Manual chapter 8.1.

5.4 Virtual Network

Using a virtual network connection Windows and VxWorks can set up TCP or UDP communication without adding additional hardware.

5.4.1 Windows XP / 7 / 8 configuration

A detailed description how to set up the Windows side of the network can be found at the RTOS-VM user manual.

5.4.2 VxWorks configuration

5.4.2.1 Bootline

To configure VxWorks for networking, the user should enter his own prototype boot line in the VxWin configuration file (***general.config***).

When VxWin starts up, the Uploader Utility copies the prototype boot line from the configuration file into memory at a predefined address. Then, when VxWorks boots, it reads and interprets the network parameters in that boot line. Among the various parameters that must be specified are: network addresses, network devices, host-computer designation, script filename, and so on.

The following describes the general VxWorks boot line syntax:

```
bootdev(0,procnum)hostname:filename h=# e=# b=# g=# u=userid pw=passwd
tn=targetname s=startupscript o=other
```

Parameter		**Description**
bootdev	=	“shm” for the Virtual Network, “ultra” for the SMC Elite Ultra Ethernet adapter card, and so on. (Refer to Wind River documentation.)
procnum	=	The processor number of the Virtual Network. (Must always be 1.)
hostname	=	The name of the host computer from which the boot will be performed.
filename	=	The filename of the VxWorks image to be executed.
e	=	The IP address of the Ethernet port. This field can have an additional partial net-mask. <inet_adrs>:<subnet_mask>
b	=	The IP address of the VxWorks Virtual Network interface. This field can also have an additional partial net-mask.
h	=	The host's Internet-Address.
g	=	The Internet address of the host system's gateway. Note: When the host is on the same network, this field should be left blank.
u	=	A valid user-name on the host.

Parameter		Description
pw	=	The user's password on the host system. When specified, FTP will be used for data transfers.
tn	=	The name of the VxWorks target system.
s	=	The name of a script file containing commands in text form to be interpreted by VxWorks Target Shell Interpreter during startup. This script can access the following "o" parameters.
o	=	"other" – a character string that may be accessed either by commands in the –s script file or a task under VxWorks. This string may be used, e.g., to parameterize the boot process.

Note: Internet addresses are specified in "dot" notation (e.g., 90.0.0.2)
Note: The order of assigned parameters is arbitrary.
Example: The following boot line could be used to configure the Virtual Network:

```
vnet(0,1)pc:VxWorks h=192.168.0.1 e=192.168.0.2 u=target pw=vxworks
```

Note: IP addresses must, of course, match those used on the Windows side of the Virtual Network.

5.4.2.2 Virtual Network adapter address

The network adapter address [also known as the MAC (Media Access Control) address] is generally set in accord with the following syntax:

AA-BB-CC-DD-EE-00

You can use a parameter in the VxWin configuration (***general.config***) file to specify a network address for your Virtual Network card. Refer to NetworkAddress in Global parameters in the VxWin configuration file. For information on setting MAC addresses on both VxWorks and Windows, also refer to Setting MAC Addresses for the Virtual Network Driver.

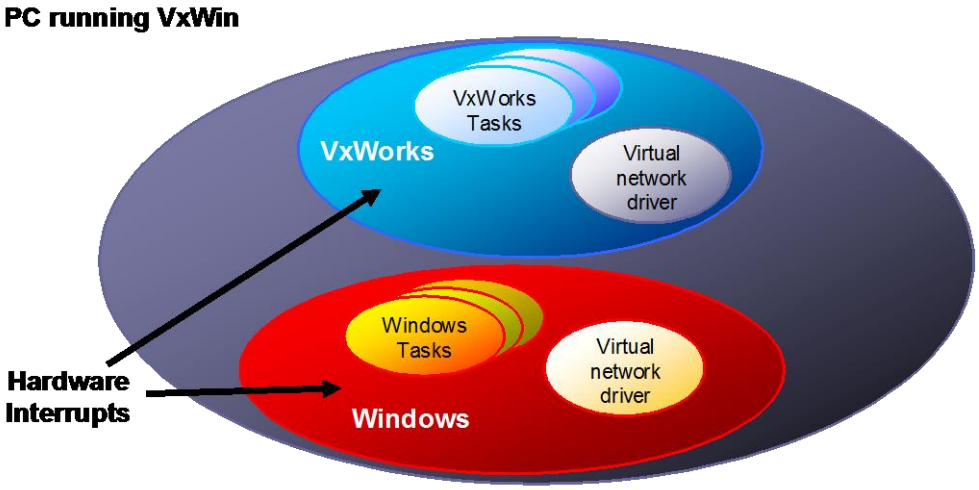
5.4.2.3 Virtual Network adapter polling or interrupt mode

The virtual network adapter can be configured either for polling or interrupt mode (***general.config***).

[Rtos]
"VnetPollPeriod"=dword:0 ; polling period in ticks (0 = interrupt mode)

5.5 Interrupts and Exceptions

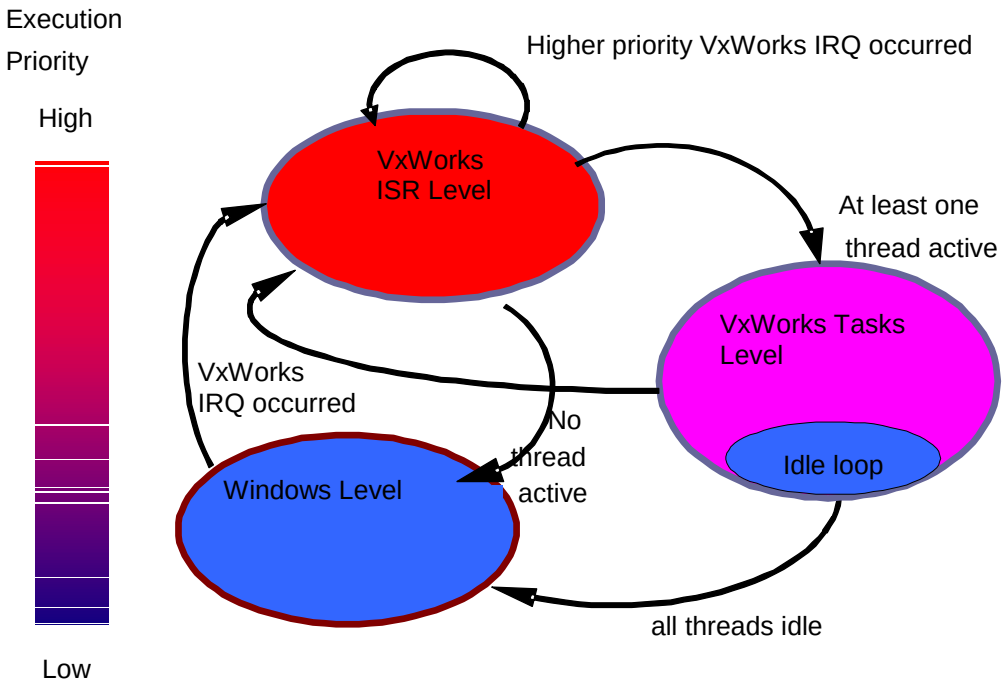
Hardware devices, with the help of their associated controller/adapters, generate interrupts to signal when they require attention from the CPU.
Upon receiving an interrupt signal, the CPU will promptly set aside its current processing to attend instead to the needs of the interrupting device. This rapid response assures that real-time operating systems respond to external events as quickly as possible.
Note: While peripheral devices may share an interrupt under *either* of the two operating systems, any attempt to share an interrupt *between* the two systems under VxWin will result in unpredictable and certainly incorrect system behavior.



All interrupts for VxWorks as well as those for Windows are connected to the PC's Interrupt Controller which, upon the occurrence of a connected event, will interrupt the CPU.
VxWin guarantees that interrupts controlled by VxWorks always have a higher priority than any program or any interrupt under Windows. If VxWorks is running on the same CPU core as Windows (shared mode) then within microseconds after an interrupt occurs, whichever Windows program or lower priority VxWorks task was running is stopped and control is transferred to the VxWorks Interrupt Service Routine (ISR) that the user assigned to that interrupt during initialization.
After the ISR finishes executing, control is returned to the VxWorks kernel, which then checks to see if any other real-time threads are ready to run. If that is the case, the kernel will activate the now-ready-to-run task. VxWorks tasks will continue to run until all are suspended or blocked. After which, the system will enter the VxWorks idle loop, which in turn returns control to Windows (when running in shared mode).

Since Windows is re-activated only when all VxWorks tasks are idle, one could say that Windows runs as if it were part of VxWorks idle loop (when running in shared mode).

State diagram of VxWin (shared mode)



5.5.1 Exception-Handling (shared mode)

The processor hardware is able to detect and respond to certain kinds of serious errors such as faulty addressing (page/segment errors) or floating-point processor errors (such as divide by zero). This class of hardware errors are generically called exceptions. When an exception occurs, it triggers a high-priority interrupt. The correct reaction to the occurrence to such an exception depends on which operating system, Windows or VxWorks, was running at the time. To enable them to respond, each in its own way, VxWin provides two different exception tables (IDT), one for each system context, which it swaps at appropriate times. Whenever a real-time interrupt activates VxWorks, VxWin replaces the Windows exception table with one defined for VxWorks. Correspondingly, whenever control is returned to Windows, the Windows exception table is likewise restored.

Exception-table-swapping, is only one of the mechanisms VxWin implements to enable the operating systems to share a single execution platform.

5.5.2 usrRoot Task Exceptions (shared mode)

If the usrRoot task (VxWorks) generates an exception, i.e., a page error, the entire system will crash. While experience has shown that this occurs only

infrequently, when it does occur, the most common reason for it, it has been determined, is that VxWorks had been incorrectly configured.

5.5.3 Connecting an ISR to an interrupt

After VxWorks has started, you can use the VxWorks ***intConnect( )*** function to connect an Interrupt Service Routine (ISR) to a specific interrupt level.
In the following example, the Interrupt Service Routine *proc* is connected to interrupt level 7. Thereafter, interrupt level 7 in the interrupt controller is enabled.

```
intLvl = 7;
intNum = INT_NUM_GET(intLvl);
intVec = INUM_TO_IVEC(intNum);
intConnect(intVec, proc, param);
sysIntEnablePIC(intLvl);
```

Note: *intLvl* is the interrupt level, *intNum* is the interrupt number, *intVec* is the interrupt vector. The macro *INT_NUM_GET* supplies the required interrupt vector from either of two arrays in accord with the interrupt controller type. One can also use predefined macros, such as *INT_NUM_COM1* or *INT_VEC_COM1*, which are defined in the header file *configInum.h* . While *proc* is the C routine to be called in response to the interrupt (the ISR), *param* is a parameter to be passed to the routine each time it is called.
For more information about this topic, see Wind River's documentation, *VxWorks Programmer's Guide*.

5.5.4 Additional interrupt functions – VxWorks

When your VxWorks application must use other functions that pertain to interrupts, you should use the standard functions as defined by Wind River for VxWorks. Please refer first to the *intLib* and *intArchLib* libraries in the Wind River documentation (*VxWorks OS Libraries API Reference* and *VxWorks Programmer's Guide*).

5.6 VxWorks System overload

It is possible to overwhelm a real-time system with so many interrupts that it can not dispose of the resulting workload. This causes work-queues to become full so that they cannot accept additional entries. For most real-time applications this would lead to an overall system malfunction.

Note: This is not a limitation of VxWin. System overload is a condition that can arise in any real-time system.

Therefore, it is important that real-time system designers plan their systems carefully, taking into account the worst-case demands that will be placed on it, so that this kind of error does not occur.

When a native VxWorks detects an internal work-queue overrun, it typically re-boots itself. After which, the BSP routine ***sysToMonitor*** is called. If we were dealing with a VME system, for example, under these circumstances, the ROM-monitor would be called.

But a VxWin system has no ROM-monitor. In this case, a call to ***sysToMonitor*** causes all interrupts to be deactivated and a message box will be shown by Windows.

5.7 VxWin – Board Support Package

5.7.1 Introduction

The VxWin board support package (BSP) provides special adaptations for Wind River's VxWorks to the run-time environment of VxWin.
Several files and functions are designated by Wind River Systems as platform or run-time specific such as the ***sysLib.c*** file.
This manual describes such components and additional extensions to support the operation of VxWorks concurrently with Windows.

Important*:*
The BSP is not part of the VxWin setup program. Please contact Customer Support for the latest BSP adequate to your desired VxWorks Version.

5.7.2 Important files in the VxWin BSP

In the following sections, some important files included in VxWin Board Support Package have been listed along with brief explanations of their purpose.
The files are grouped primarily to reflect their general role in the system and secondarily to reflect their file-type. Only files that were developed or modified for VxWin are listed here.

VxWin Source Files

File	Description
RtosLib.h	All definitions pertinent to VxWin Interface, including those for shared events and shared-memory API.
vxwinVioSio.c	The SIO driver for the VMF VIO interface
vxwinVnetEnd.c	The END driver for the VMF VNET interface
vxwinConsoleLoader.c	Determine the console device using the following entries of the VxWin config file: [Rtos] "Console"="/vio/0" ; use "/tyCo/0" for first COM port.
vxwinNetworkLoader.c	Attach END interface to the network stack using the following entries of the VxWin config file: [Rtos] "AddNetwork0"="fei0:192.168.100.1:255.255.255.0" "AddNetwork1"="gei0:192.168.200.1:255.255.255.0"
vxwinNetworkLogger.c	Handle the network logging functionality using the “LogNetworkFilter” entries in the [Rtos] section of the VxWin config file.
vxwinRamDiskMounter.c	Automatically mount a RAM disk device to an eventually VMF general purpose shared memory area named “RamDisk”. Please refer to RTOS VM User Manual. [SharedMemory\RamDisk] "Name"="RamDisk" "Description"="RAM disk example configuration" "Base"=dword:2000000 "Alignment"=dword:0 "Size"=dword:300000 "File"="C:\\RamDisk.img" "Initialize"=dword:2 "Save"=dword:1 "AccessDefault"=dword:0 [SharedMemory\RamDisk\AccessModes] "0"=dword:1

Additional VxWorks Test Tools

File	Description
vmfInterface.h	The VMF interface API header file
vxwinDemo.c	A demo-program (e.g. latency measurement) Start the demo: function demoStart() Stop the demo: function demoStop()
vxwinCpuMon.c	CPU Monitoring. Displays performance data for each CPU. Start monitoring: function vxwinCpuMonStart() Reset monitoring: function vxwinCpuMonReset() Stop monitoring: function vxwinCpuMonStop() Show data: function vxwinCpuMonShow() To get performance data on exclusive cores, enable 'IdleOs' by setting an appropriate value for "ProcessorMask" in the section [Idle] in the config-file.

Additional VxWin Libraries

File	Description
RtosXxYy.a	VxWin kernel library for vxworks version Xx and tool Yy (diag or gnu)
RtosXxYy.so	VxWin shared object library for vxworks version Xx and tool Yy (diag or gnu)

5.8 Board Support Package for VxWorks 7 native

5.8.1 Development Host: Copy VxWin platform and BSP

Copy the VxWin platform directory vxwin_common-1.0.6.0 into %WIND_BASE%\vxworks-7\pkgs\os\platform\itl_core_psl

Copy the VxWin BSP directory vxwin_generic-1.0.6.0 into %WIND_BASE%\vxworks-7\pkgs\os\board\ia

5.8.2 Target System Configuration

The following VxWin configuration settings are mandatory:

- Disable wait to RTOS communication subsystem (it is not yet supported).
- Assure the VMF is located below physical address 0x20000000
- Minimum memory size is 48 Mbyte

See settings located in folder VxWin-TargetConfig, file vxworks7native.config:

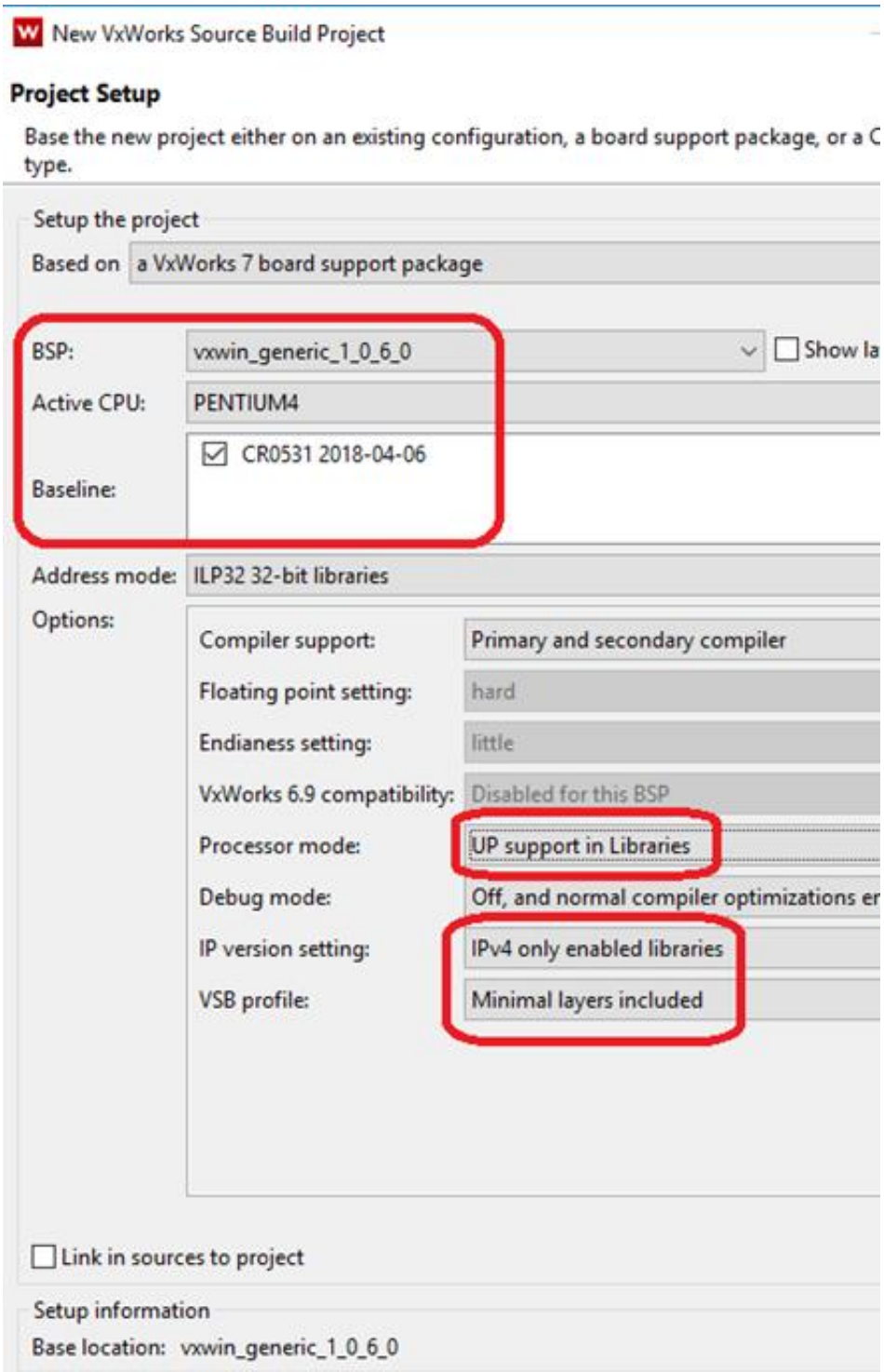
```
[Upload]
"WaitForRtosCommSubsystems"=dword:0
[Vmf]
"AddressMax"=hex:FF,FF,FF,1F,00,00,00,00
[Rtos]
"MemorySize"=dword:3000000
```

5.8.3 Shipped Binary image

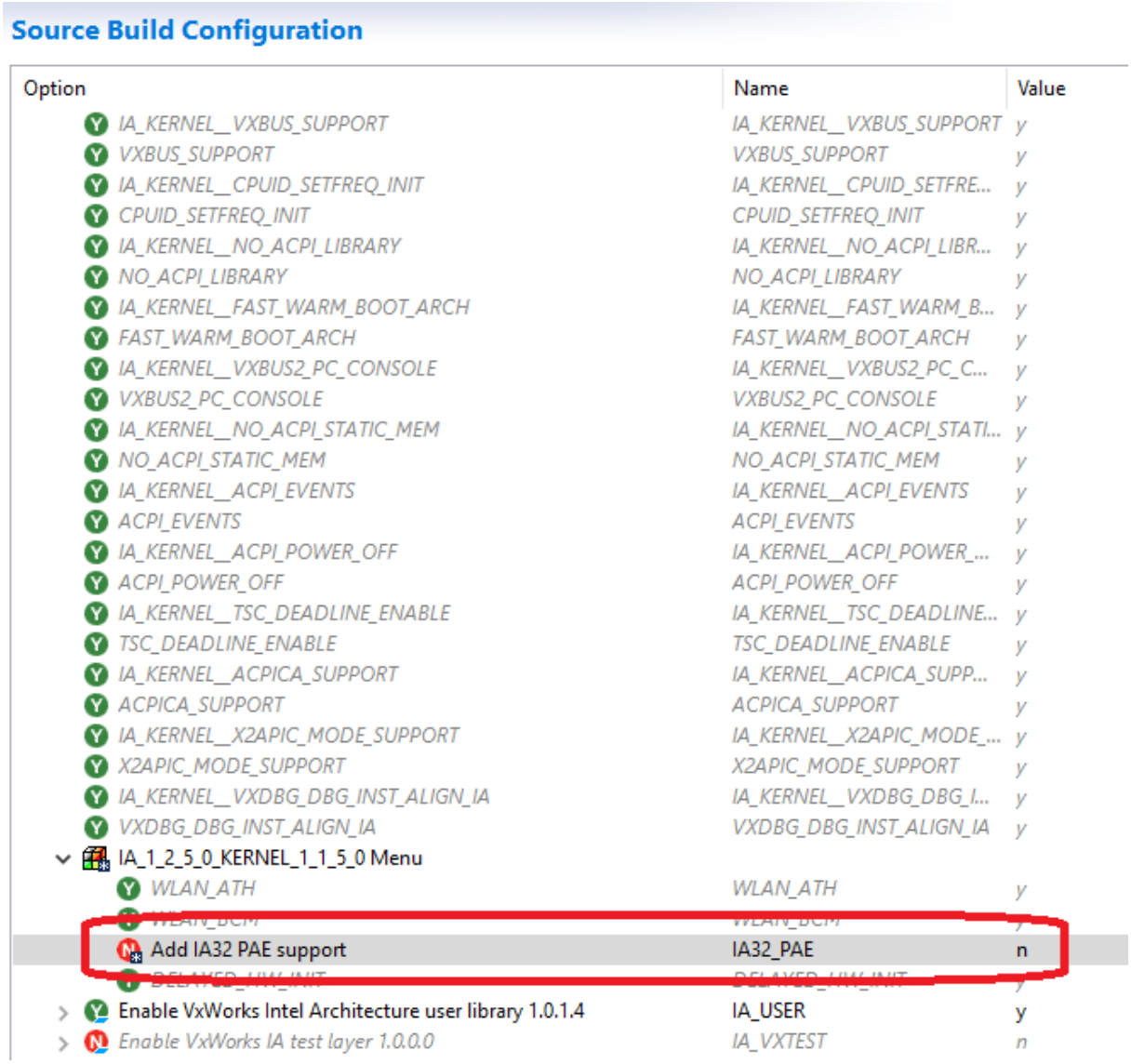
You can find a bootable VxWorks 7 native binary image (vxWorks.bin) in folder VxWin-Target

5.8.4 VxWorks Source Build VSB

Create a new VxWorks Source Build Project. It has to be based on the vxwin_generic_1_0_6_0 BSP, the PENTIUM4 CPU and Baseline CR0531 2018-04-06 has to be selected. SMP and IPv6 is not supported and must be unselected. Also, Minimal layers have to be included:

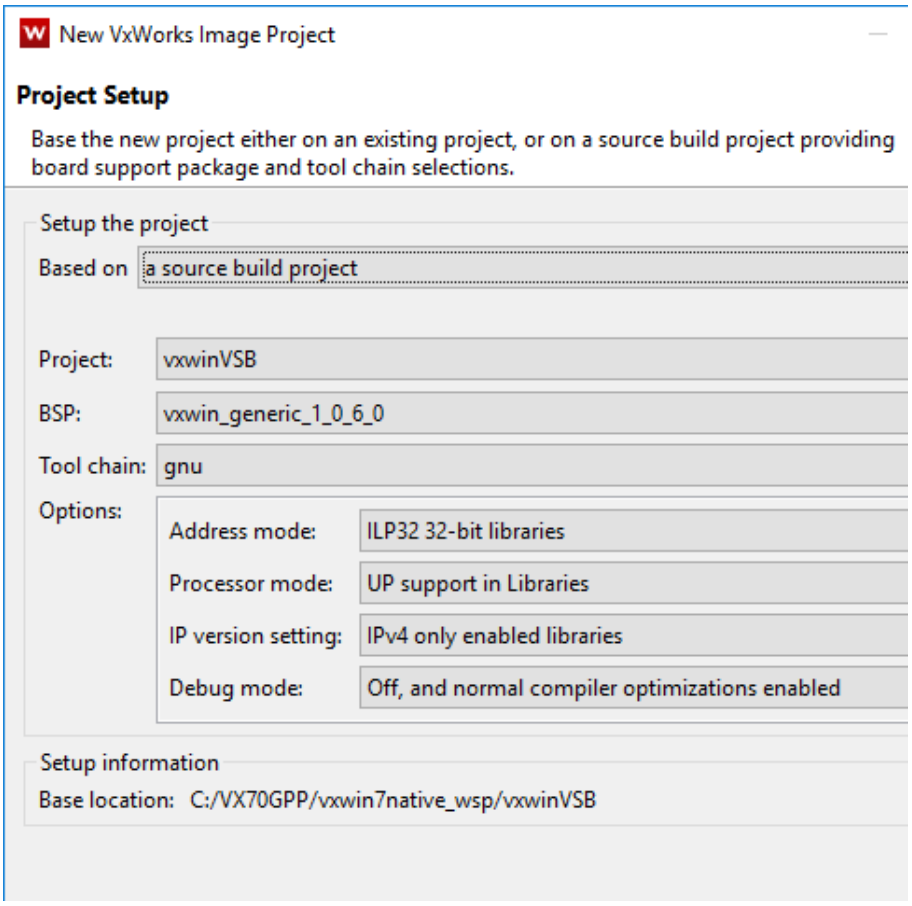


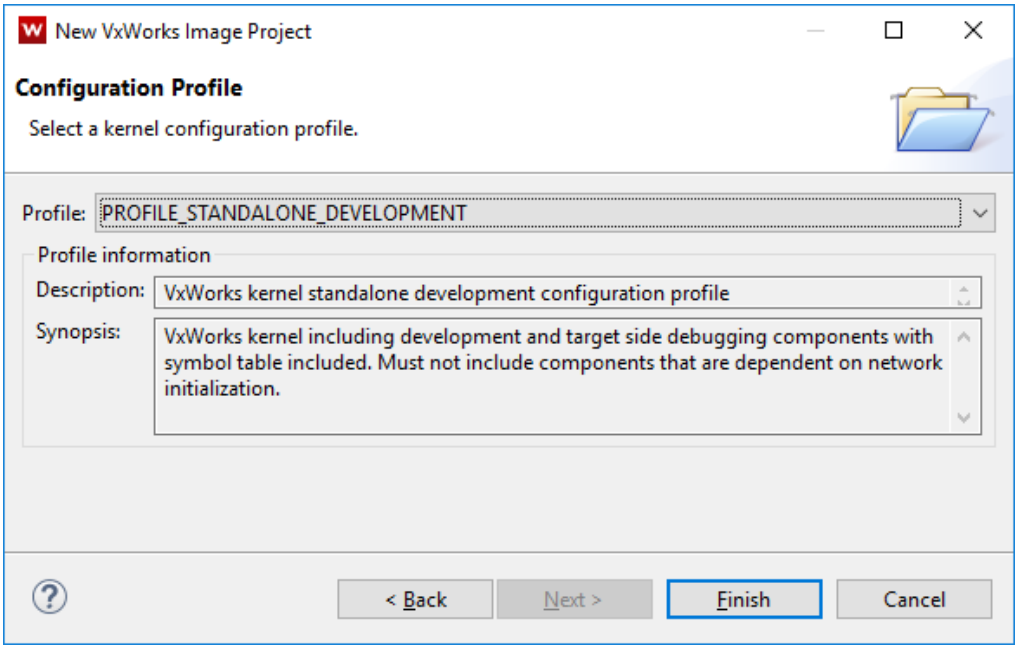
Important Note: Turn off IA32_PAE before building the VSB!



5.8.5 VxWorks Kernel Image Project VIP

Create a new VxWorks Kernel Image Project. It has to be based on the VSB which has been created before. Supported profiles: PROFILE_STANDALONE_MINIMAL and PROFILE_STANDALONE_DEVELOPMENT

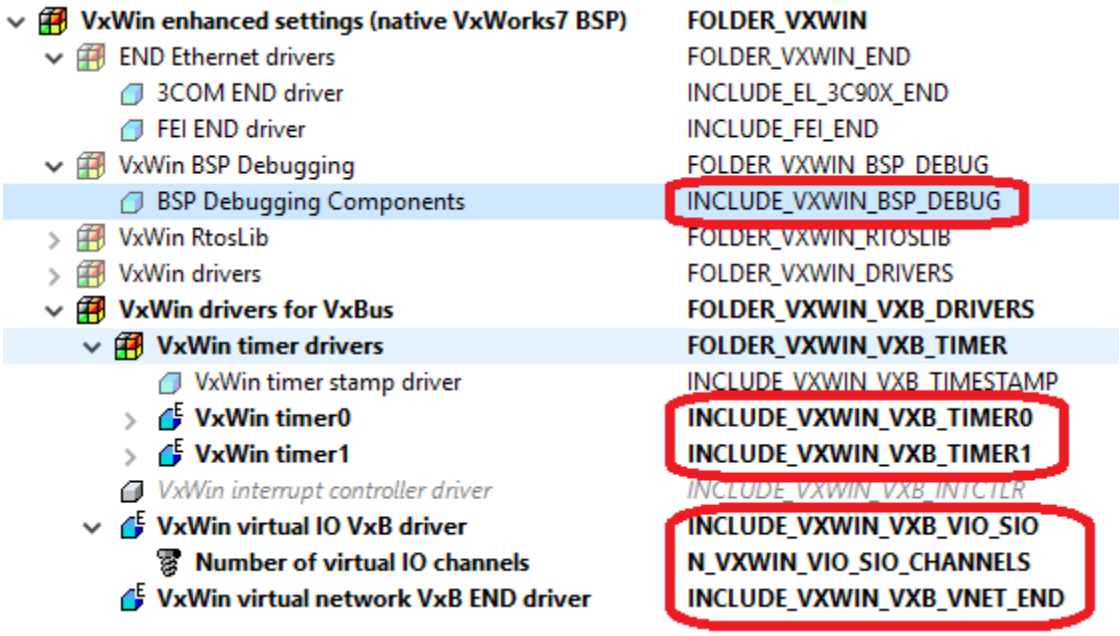




Supported Components:

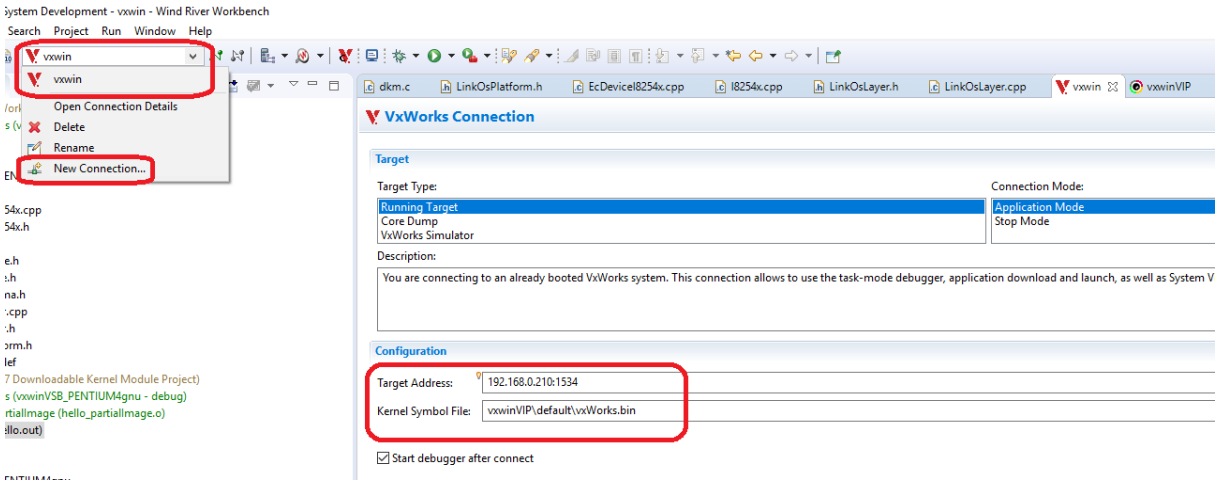
- Two virtual timers (for sysClk and auxClk):
  INCLUDE_VXWIN_VXB_TIMER0/1
- One Virtual serial channel for the Kernel Shell:
  INCLUDE_VXWIN_VXB_VIO_SIO
- Virtual network driver: INCLUDE_VXWIN_VXB_VNET_END
- BSP Debugging (for extended BSP debugging):
  INCLUDE_VXWIN_BSP_DEBUG

All other VxWin components MUST NOT be included/enabled.
The resulting VxWorks image will be located in %VIPDIR%\default\vxWorks.bin



5.8.6 Target connection to debug a DKM

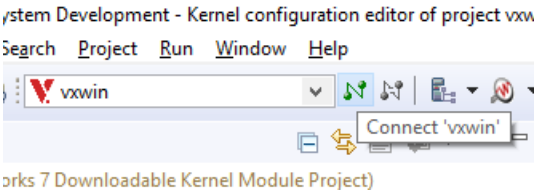
At the very first time a “New Connection” has to be created. Properties, see below screenshot.



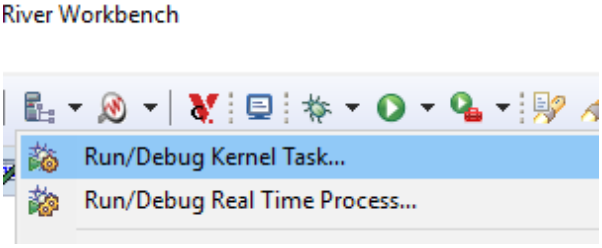
5.8.7 Debugging a DKM

The VIP has to built with the appropriate Debug Agents included:

After booting the VIP, select the previously created connection (in the example below its name is “vxwin”) and press the Connect button.



Select the DKM project. Press the “Target Actions” button and select “Run/Debug Kernel Task…”.



5.9 VxWorks 7 Kernel Image with 512 Mb memory

5.9.1 VxWorks 7 native

To build a VxWorks 7 native image with 512 Mb, following settings have to be adjusted:

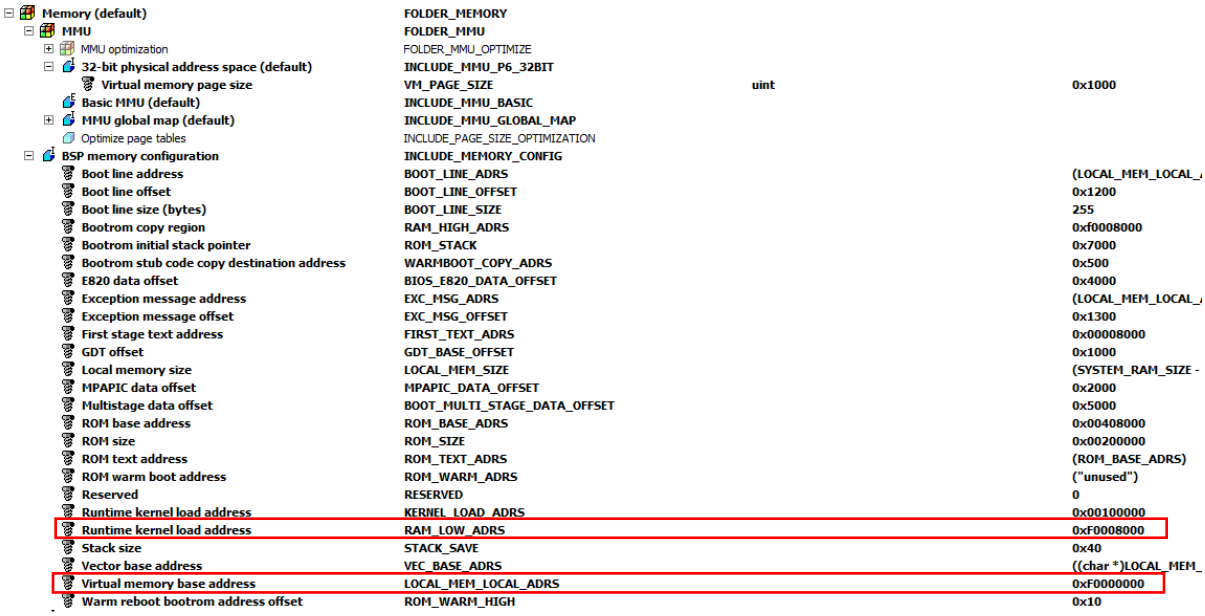
- File systemmgr_user.config, located in folder VxWin-TargetConfig:

  [Rtos]

  "MemorySize"=dword:20000000 ; 512Mb

  "MemoryStartAddress"=dword:10000000 ; 256Mb

- Components in the VIP project:



  Set LOCAL_MEM_LOCAL_ADRS to 0xF0000000

  Set RAM_LOW_ADRS to 0xF0008000

5.9.2 VxWorks 7 legacy

To build a VxWorks 7 legacy image with 512 Mb, following settings have to be adjusted:

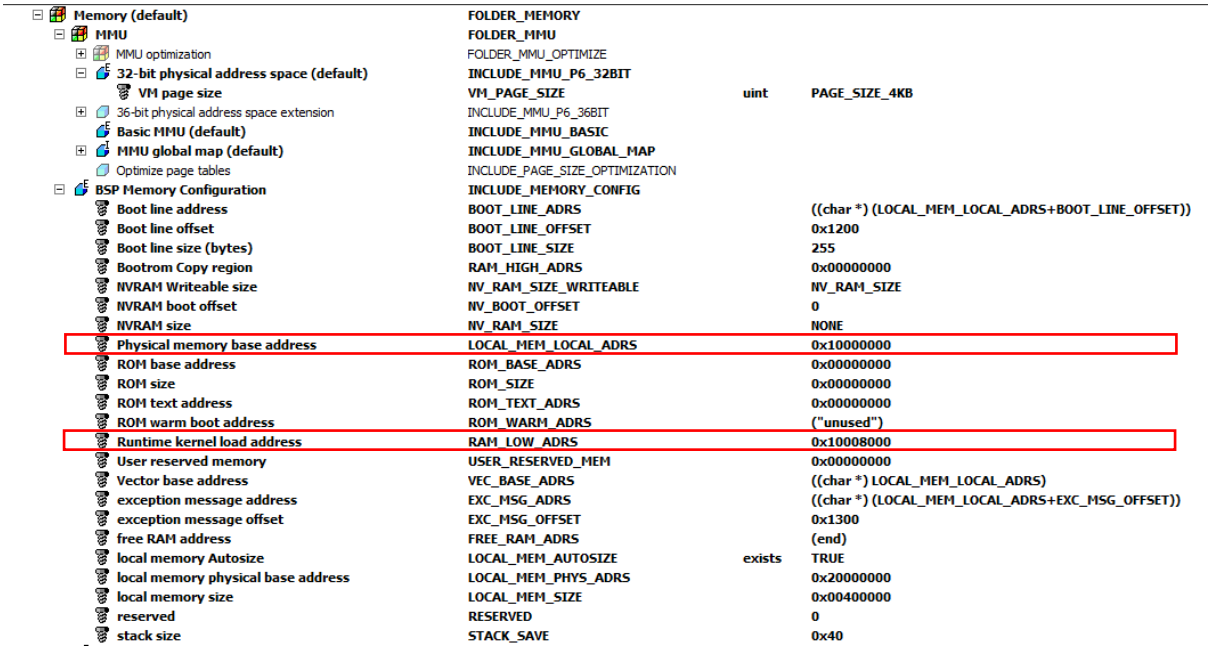
- File systemmgr_user.config, located in folder VxWin-TargetConfig:

  [Rtos]

  "MemorySize"=dword:20000000 ; 512Mb

  "MemoryStartAddress"=dword:10000000 ; 256Mb

- Components in the VIP project:



Memory (default)	FOLDER_MEMORY		
MMU	FOLDER_MMU		
MMU optimization	FOLDER_MMU_OPTIMIZE		
32-bit physical address space (default)	INCLUDE_MMU_P6_32BIT		
VM page size	VM_PAGE_SIZE	uint	PAGE_SIZE_4KB
36-bit physical address space extension	INCLUDE_MMU_P6_36BIT		
Basic MMU (default)	INCLUDE_MMU_BASIC		
MMU global map (default)	INCLUDE_MMU_GLOBAL_MAP		
Optimize page tables	INCLUDE_PAGE_SIZE_OPTIMIZATION		
BSP Memory Configuration	INCLUDE_MEMORY_CONFIG		
Boot line address	BOOT_LINE_ADRS		((char *) (LOCAL_MEM_LOCAL_ADRS+BOOT_LINE_OFFSET))
Boot line offset	BOOT_LINE_OFFSET		0x1200
Boot line size (bytes)	BOOT_LINE_SIZE		255
Bootrom Copy region	RAM_HIGH_ADRS		0x00000000
NVRAM Writeable size	NV_RAM_SIZE_WRITEABLE		NV_RAM_SIZE
NVRAM boot offset	NV_BOOT_OFFSET		0
NVRAM size	NV_RAM_SIZE		NONE
Physical memory base address	LOCAL_MEM_LOCAL_ADRS		0x10000000
ROM base address	ROM_BASE_ADRS		0x00000000
ROM size	ROM_SIZE		0x00000000
ROM text address	ROM_TEXT_ADRS		0x00000000
ROM warm boot address	ROM_WARM_ADRS		("unused")
Runtime kernel load address	RAM_LOW_ADRS		0x10008000
User reserved memory	USER_RESERVED_MEM		0x00000000
Vector base address	VEC_BASE_ADRS		((char *) LOCAL_MEM_LOCAL_ADRS)
exception message address	EXC_MSG_ADRS		((char *) (LOCAL_MEM_LOCAL_ADRS+EXC_MSG_OFFSET))
exception message offset	EXC_MSG_OFFSET		0x1300
free RAM address	FREE_RAM_ADRS		(end)
local memory Autosize	LOCAL_MEM_AUTOSIZE	exists	TRUE
local memory physical base address	LOCAL_MEM_PHYS_ADRS		0x20000000
local memory size	LOCAL_MEM_SIZE		0x00400000
reserved	RESERVED		0
stack size	STACK_SAVE		0x40

  Set LOCAL_MEM_LOCAL_ADRS to 0x10000000

  Set RAM_LOW_ADRS to 0x10008000

5.10 Multiple Instances

For systems with multi OS configurations each instance of a Vxworks image must be created separately with individual memory settings. The memory settings herein must correspond to the memory settings in the config file. The maximum needed memory size will be calculated by following formula: size(Image1) + size(Image2) + size(Image3) + ...

Each image could have individual sizes, but the minimum start addresses, etc depends on the previous image start address and size. It's possible to leave a gap in memory between each image.

Without gap:

```
|<- [.config:Upload] RteMemorySize ->|
:                                    :
+---------+---------------+----------+
| image 1 |    image 2    | image 3  |
+---------+---------------+----------+
:         :               :          :
|<- s0  ->|<- s1 (size1)->|<- s2   ->|
Off0      Off1            Off2


Off0: [.config:Upload] RteMemoryStartAddress
Off1: Off0 + s0;
Off2: Off1 + s1;
Off3: ...

Min. RteMemorySize: s0 + s1 + s2 + ...
```

With gap (between each image):

```
|<-   [.config:Upload] RteMemorySize   ->|
:                                        :
+---------+-+---------------+-+----------+
| image 1 |g|    image 2    |g| image 3  |
+---------+-+---------------+-+----------+
:         : :               : :          :
|<- s0  ->| |<- s1 (size1)->| |<- s2   ->|
Off0        Off1              Off2


Off0: [.config:Upload] RteMemoryStartAddress
Off1: Off0 + s0 + g;
Off2: Off1 + s1 + g;
Off3: ...

g: gap of variable size

Min. RteMemorySize: s0 + g + s1 + g + s2 + ...
```

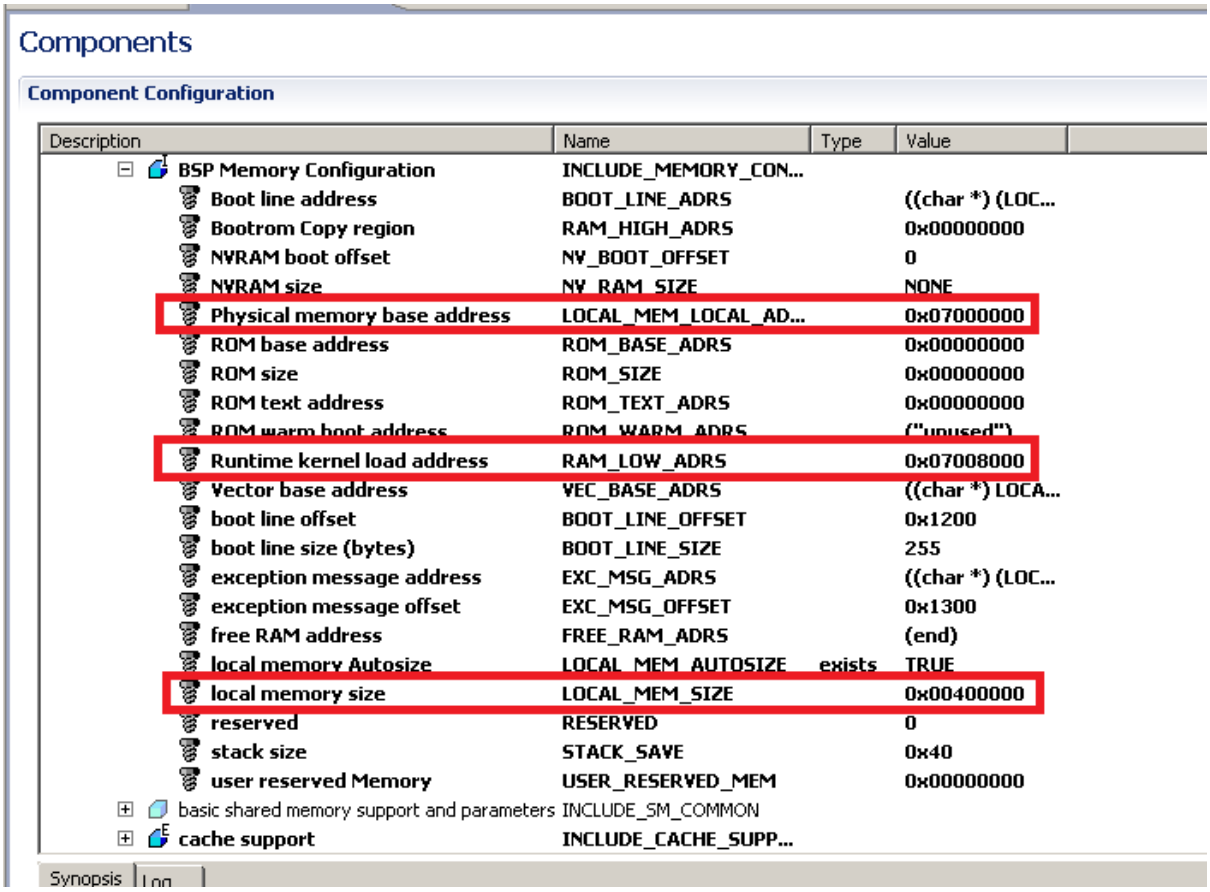
Remark A: The gap between each image can be of different size.
Remark B: There could be an initial gap before first image and a gap after last image. In those cases the RteMemorySize calculation must be adjusted accordingly.

5.10.1 Individual Image Creation

The followings steps are required to get the desired images:

1. Create a Wind River VxWorks 6.x Image project based on VxWin BSP
2. Go to Kernel Konfiguration
3. Search node *memory (default) \ BSP Memory Configuration*
4. Adjust value: *Physical memory base address* with **Off0** for 1st image, **Off1** for 2nd image, and so on.
5. Adjust value: *Runtime kernel load address* with **Off0 + 0x8000** for 1st image, **Off1 + 0x8000** for 2nd image, and so on.
6. Adjust value: *local memory size* with **s0** for 1st image, **s1** for 2nd image, and so on.
7. Compile image

Hint: The steps 4 & 5 must be adjusted for **each** desired image. Step 6 only, if the size of the images vary.



5.10.2 Further Config File Dependencies

Each configured rtos have a separate entry in the .config file. The offset values (OffX) and the size values (sX) correspond directly to the rtos specific entries in the .config file.
Hint: Using of the System Manager the values will be automatically adjusted without manual interference!

1st rtos:

```
[Rtos]
    "MemoryStartAddress"=dword:XXXXXXXX  : Value of Off0!
    "MemorySize"=dword:XXXXXXXX  : Value of s0!
```

2nd rtos:

```
[Rtos1]
    "MemoryStartAddress"=dword:XXXXXXXX  : Value of Off1!
    "MemorySize"=dword:XXXXXXXX  : Value of s1!
```

3rd rtos:

```
[Rtos2]
    "MemoryStartAddress"=dword:XXXXXXXX  : Value of Off2!
    "MemorySize"=dword:XXXXXXXX  : Value of s2!
```

6 Built-in tools and utilities

6.1 RtosService

RtosService.exe is needed when using the RtosLib.lib.
If you look at the default General.config file, you will see that RtosService.exe is automatically launched.
RtosService.exe starts the communication between Windows and WinCE. Also it starts the time synchronisation between Windows and WinCE.

6.2 Boot Line for VxWorks

When VxWorks is booted up in an embedded system, among the first things it does is read a line of text from a pre-defined location in RAM memory[1]. This so-called *boot line*[2] contains parameters that VxWorks uses to dynamically characterize itself during its boot-up phase. Boot-line parameters convey information such as: network addresses, boot device designation, password, and start-up script identifier.
When VxWin is started, it retrieves boot-line parameters from the boot line in the VxWin configuration file. By this means, VxWin users can influence, upon each startup, how VxWorks boots, without having to generate a new image.

[1] The boot line is stored in the variable ***sysBootLine*** as defined in the VxWorks ***sysLib.c*** library. The boot line is a simple ASCII text-string.

[2] "Boot line," a VxWorks term, is described in greater detail in Wind River Systems documentation.

7 Version History

A general version history containing information about new features, migration hints and improvements can be found in the release notes file “ReleaseHistory.txt”.