ADB3 Driver Component 1.4.14 for Wind River VxWorks

Introduction

This release note accompanies the ADB3 Driver Component for Wind River VxWorks. The latest version of this driver can be found at:

ftp://ftp.alpha-data.com/pub/admxrcg3/vxworks

For support, send e-mail to support@alpha-data.com

The ADB3 Driver Component for Wind River VxWorks is a component that can be built into a VxWorks kernel image.

Operating systems supported

This release of the ADB3 Driver Component for VxWorks supports the following operating systems:

- Wind River VxWorks 6.8 and later

Hardware supported

This release of the ADB3 Driver Component for VxWorks supports the following Alpha Data hardware:

- ADM-XRC-6TL
- ADM-XRC-6T1
- ADM-XRC-6T-DA1
- ADM-XRC-6TGE and ADM-XRC-6TGEL
- ADM-XRC-6T-ADV8
- ADPE-XRC-6T and ADPE-XRC-6T-L
- ADPE-XRC-6T-ADV
- ADM-XRC-7K1
- ADM-XRC-7V1
- ADM-VPX3-7V2
- ADM-XRC-II (legacy hardware)

License agreement

Please refer to the files **license.rtf** or **license.txt** within this software package for the licensing terms that apply to this software. Please contact Alpha Data if alternative licensing terms are required.

Alpha Data reserves the right to use different licensing terms for future releases of this software.

Installation

Installation in a Windows VxWorks host

To install the ADB3 Driver Component, follow these steps:

- Unzip the **vxbAdb3-1.4.14.zip** into the **%WIND_BASE%** folder. If prompted about whether or not to allow merging of folders, respond with "allow".
- Next, the ADB3 Driver Component must be compiled for the CPU architecture of your kernel image. To do this, start a "VxWorks Development Shell" and enter the following commands:

  ```
  cd /d %WIND_BASE%\3rdparty\alphadata
  make CPU=<cpu> TOOL=<tool> VXBUILD="<options>"
  ```

  In the **make** command, **<cpu>**, **<tool>** and **<options>** must match the way your kernel image is built. For example, with most x86_64 SMP kernels, the **make** command is:

  ```
  make CPU=NEHALEM TOOL=gnu VXBUILD="LP64 SMP"
  ```

  For 32-bit, non-SMP kernel images, **VXBUILD** is generally omitted; for example:

  ```
  make CPU=PPC604 TOOL=diab
  ```

Note

The effect of building the ADB3 Driver Component for a particular CPU architecture is to create (or update) a vendor-specific archive with the object modules generated during compilation. For reference, the full pathname of this library is:

```
%WIND_BASE%\target\lib<_options>\<arch>\<cpu>\common\libalphadata.a
```

The individual object files that were stored in the archive are placed in:

```
%WIND_BASE%\target\lib<_options>\<arch>\<cpu>\common\alphadata
```

If the ADB3 Driver Component is successfully compiled for the CPU architecture used by your VxWorks kernel image, you will see it in the VxWorks kernel image configurator and will be able to include it. Otherwise, you will typically see that it is present in the VxWorks kernel image configurator but grayed out and cannot be included.

Installation in a Linux VxWorks host

To install the driver in Linux, follow these steps:

- Extract the **vxbAdb3-1.4.14.tar.gz** into the **$WIND_BASE** directory.
- Next, the ADB3 Driver Component must be compiled for the CPU architecture of your kernel image. To do this, start a "VxWorks Development Shell" and enter the following commands:

  ```
  cd $WIND_BASE/3rdparty/alphadata
  make CPU=<cpu> TOOL=<tool> VXBUILD="<options>"
  ```

  In the **make** command, **<cpu>**, **<tool>** and **<options>** must match the way your kernel image is built. For example, with most x86_64 SMP kernels, the **make** command is:

  ```
  make CPU=NEHALEM TOOL=gnu VXBUILD="LP64 SMP"
  ```

  For 32-bit, non-SMP kernel images, **VXBUILD** is generally omitted; for example:

```
make CPU=PPC604 TOOL=diab
```

Note

The effect of building the ADB3 Driver Component for a particular CPU architecture is to create (or update) a vendor-specific archive with the object modules generated during compilation. For reference, the full pathname of this library is:

```
$WIND_BASE/target/lib<_options>/<arch>/<cpu>/common/libalphadata.a
```

The individual object files that were stored in the archive are placed in:

```
$WIND_BASE/target/lib<_options>/<arch>/<cpu>/common/alphadata
```

If the ADB3 Driver Component is successfully compiled for the CPU architecture used by your VxWorks kernel image, you will see it in the VxWorks kernel image configurator and will be able to include it. Otherwise, you will typically see that it is present in the VxWorks kernel image configurator but grayed out and cannot be included.

Uninstallation

Uninstallation in a Windows VxWorks host

To uninstall the driver, first ensure that you have removed the component **DRV_ALPHADATA_VXBADB3** from all of your kernel image projects. Then:

- Manually delete the following folder and its contents:

```
%WIND_BASE%\target\3rdparty\alphadata\vxbAdb3
```

- Manually delete the following files:

```
%WIND_BASE%\target\config\comps\src\hwif\vxbAdb3.dc
%WIND_BASE%\target\config\comps\src\hwif\vxbAdb3.dr
%WIND_BASE%\target\config\comps\vxWorks\40vxbAdb3.cdf
```

You may also wish to remove any object modules and archives built from Alpha Data source code, as a result of following the instructions in Installation in a Windows VxWorks host. This can safely be done without damaging your VxWorks installation if only the following folder is deleted:

- `%WIND_BASE%\target\lib<_options>\<arch>\<cpu>\common\alphadata`

and only the following file is deleted:

- `%WIND_BASE%\target\lib<_options>\<arch>\<cpu>\common\libalphadata.a`

Note

In the VxWorks component build system, source code for all components from a particular vendor is built into the vendor-specific archive:

```
%WIND_BASE%\target\lib<_options>\<arch>\<cpu>\common\lib<vendor>.a
```

It is not possible to selectively delete compiled object modules from the vendor-specific archive. Thus, deleting the vendor-specific archive will delete all Alpha Data object code, not just that of the ADB3 Driver. If you have any other Alpha Data components installed, you must rebuild them afterwards.

Uninstallation in a Linux VxWorks host

To uninstall the driver, first ensure that you have removed the component **DRV_ALPHADATA_VXBADB3** from all of your kernel image projects. Then:

- Manually delete the following directory and its contents:

```
$WIND_BASE/target/3rdparty/alphadata/vxbAdb3
```

- Manually delete the following files:

```
$WIND_BASE/target/config/comps/src/hwif/vxbAdb3.dc
$WIND_BASE/target/config/comps/src/hwif/vxbAdb3.dr
$WIND_BASE/target/config/comps/vxWorks/40vxbAdb3.cdf
```

You may also wish to remove any object modules and archives built from Alpha Data source code, as a result of following the instructions in Installation in a Linux VxWorks host. This can safely be done without damaging your VxWorks installation if only the following folder is deleted:

- `$WIND_BASE/target/lib<_options>/<arch>/<cpu>/common/alphadata`

and only the following file is deleted:

- `$WIND_BASE/target/lib<_options>/<arch>/<cpu>/common/libalphadata.a`

Note

In the VxWorks component build system, source code for all components from a particular vendor is built into the vendor-specific archive:

`$WIND_BASE/target/lib<_options>/<arch>/<cpu>/common/lib<vendor>.a`

It is not possible to selectively delete compiled object modules from the vendor-specific archive. Thus, deleting the vendor-specific archive will delete all Alpha Data object code, not just that of the ADB3 Driver. If you have any other Alpha Data components installed, you must rebuild them afterwards.

VPD write-protection mechanism

The VPD write-protection mechanism described in the ADM-XRC Gen 3 SDK User Guide is exposed via the VxWorks kernel image configurator. If **VXBADB3_ENABLE_VPD_WRITE** is **TRUE**, the ADB3 Driver permits writes to VPD memory.

Note that for certain models, a board may additionally need to be operating in "Service Mode" in order to permit writes to VPD memory. Please refer to the User Manual for your reconfigurable computing board to determine whether or not "Service Mode" applies, and how to enter it.

VXBADB3_ENABLE_VPD_WRITE is the initial value of the global integer variable **adb3DrvEnableVpdWrite**. If necessary, this variable can be manipulated at runtime in order to enable or disable writes to VPD memory. This variable is checked every time an application attempts to write to the VPD memory, so changes to this variable take effect immediately rather than being sampled when the driver is started.

To set this value to 1 (thus enabling VPD writes at driver level) using the VxWorks shell, use:

```
-> adb3DrvEnableVpdWrite=(int)1
```

To set this value to 0 (thus disabling VPD writes at driver level) using the VxWorks shell, use:

```
-> adb3DrvEnableVpdWrite=(int)0
```

Common buffer support

The ADB3 Driver can allocate one or more "common buffers" at startup. The purpose of this feature is to support applications that use Direct Master data transfer, such as an ethernet-style I/O interface where the sizes and arrival times of packets of data are not known in advance by software running on the host. These buffers have the following characteristics:

- Persist until the driver is stopped.
- Guaranteed aligned to a specified power-of-2 address boundary.
- Allocated from the appropriate pool of memory in order to be contiguous and visible to bus-master devices.
- Can be mapped into the virtual address space of a user-mode process; see "ADMXRC3 API Specification 1.5.0" or later for details of the new ADMXRC3 API functions relating to common buffers.

This feature is exposed via the VxWorks kernel image configurator:

- **VXBADB3_NUM_COMMON_BUFFER** determines the number of common buffers allocated; the default is zero, meaning that no common buffers are allocated by default.
- **VXBADB3_COMMON_BUFFER_SIZE** determines the size in bytes of each common buffer; the default is 64 kiB (0x10000).
- **VXBADB3_COMMON_BUFFER_ALIGN** determines the power-of-2 address boundary to which each common buffer is guaranteed to be aligned; the default is 16 bytes.

Known issues

Modifications to certain BSPs needed for interrupt delivery

Certain BSPs for single-board computers require a modification to an interrupt vector table in order for interrupts to be delivered to the ADB3 Driver. If the driver behaves as if interrupts are not being delivered to it - for example, if DMA transfers hang, or the "ITest" example from the ADM-XRC Gen 3 SDK fails to work correctly - it may necessary to modify the file **hwConf.c** in your VxWorks kernel image project.

Although Alpha Data cannot in general provide precise instructions for doing this, for many BSPs the necessary steps are as follows:

1. Open **hwConf.c** in your VxWorks kernel image project in your favorite editor, and locate a table of this form:

   ```
   LOCAL const struct intrCtlrInputs loApicInputs[] = {
       { VXB_INTR_DYNAMIC, "yn", 0, 0 },
       { VXB_INTR_DYNAMIC, "gei", 0, 0 },
       ... other entries ...
   #if defined (INCLUDE_HPET_MSI)
       { INT_NUM_IA_HPET_TIMER0, "iaHpetTimerDev", 0, 0 },
       ... other entries ...
   #endif /* INCLUDE_HPET_MSI */
       ... other entries ...
   };
   ```

2. If you have a single Alpha Data Gen 3 Reconfigurable Computing Device in the target system, add the following entry to the end of the table:

   ```
   { VXB_INTR_DYNAMIC, "adb3", 0, 0 }
   ```

   If you have more than one device, add as many entries as you have devices, incrementing the number in the 3rd position of each entry for each entry. For example, if there are 4 devices, add the following entries to the end of the table:

   ```
   { VXB_INTR_DYNAMIC, "adb3", 0, 0 },
       { VXB_INTR_DYNAMIC, "adb3", 1, 0 },
       { VXB_INTR_DYNAMIC, "adb3", 2, 0 },
   ```

```
        { VXB_INTR_DYNAMIC, "adb3", 3, 0 }
```

3. Rebuild your VxWorks kernel image. It should now be capable of delivering interrupts to the ADB3 Driver.

Fixed-local addressing DMA transfers

The flag ADMXRC3_DMA_FIXEDLOCAL when used with the DMA functions in the ADMXRC3 currently has no effect for Gen 3 hardware.

Compiler warnings during builds

The following compiler warnings may be generated when building the driver for certain configurations:

- *../../../modules/admxrc3/admxrc3.c(858) (col. 45): warning #13365: XMM register was modified, but CG_allow_xmm was not specified*
  This warning arises because the 64-bit build specs for the Intel Compiler for VxWorks set the **-fno-implicit-fp-sse** option (no implicit floating point or SSE). This warning should appear only when building the ADMXRC3 API library, and can be safely ignored.

- *../../framework/vxworks/vxbus_pci.c: In function 'getRawBusResources':*
  ../../framework/vxworks/vxbus_pci.c:137: warning: unused variable 'f'
  This warning arises from variables that are required in some VxWorks configurations and not in others, and can safely be ignored.

- *icc: command line warning #10006: ignoring unknown option '-Wbad-function-cast'*
  icc: command line warning #10006: ignoring unknown option '-Wno-sign-conversion'
  This warning arises because the 64-bit build specs for the Intel Compiler for VxWorks set a couple of compiler options that are not supported for that particular version of the compiler. These warnings can safely be ignored.

- *"../../framework/vxworks/legacy_methods.c", line 154: warning (dcc:1500): function pciIntDisconnect2 has no prototype*
  This warning can be ignored, and occurs because the **pciIntDisconnect2** function appears to be missing from the VxWorks 5.5 header files, although it exists and can be used in some VxWorks kernel images.

Release history

Release 1.4.14

This release implements ADMXRC3 API Specification version 1.7.1.

This is the first release of the ADB3 Driver Component for VxWorks, and is broadly feature-equivalent to ADB3 Driver for VxWorks 1.4.14.