



ALPHA DATA

Common Host Utilities for Windows & Linux Release: 1.13.0

**Document Revision: 1.4
12 Jun 2017**

© 2017 Copyright Alpha Data Parallel Systems Ltd.

All rights reserved.

This publication is protected by Copyright Law, with all rights reserved. No part of this publication may be reproduced, in any shape or form, without prior written consent from Alpha Data Parallel Systems Ltd.

Head Office

Address: Suite L4A, 160 Dundee Street,
Edinburgh, EH11 1DQ, UK
Telephone: +44 131 558 2600
Fax: +44 131 558 2700
email: sales@alpha-data.com
website: <http://www.alpha-data.com>

US Office

10822 West Toller Drive, Suite 250
Littleton, CO 80127
(303) 954 8768
(866) 820 9956 - toll free
sales@alpha-data.com
<http://www.alpha-data.com>

All trademarks are the property of their respective owners.

Table Of Contents

1	Introduction	1
1.1	Directory structure	1
2	Building the Common Host Utilities	3
2.1	Building with Visual Studio 2012	3
2.2	Building with Visual Studio 2013	3
2.3	Building in Linux	3
2.3.1	Building SYSMON in Linux	4
3	Common Host Utilities	5
3.1	AVR2UTIL / AVR2UTIL-S utility	5
3.1.1	Available commands	10
3.1.1.1	build-info command	10
3.1.1.2	version command	10
3.1.1.3	product-id command	10
3.1.1.4	enter-service-mode command	11
3.1.1.5	exit-service-mode command	11
3.1.1.6	getclk command	11
3.1.1.7	setclk command	12
3.1.1.8	getclknv command	12
3.1.1.9	setclknv command	12
3.1.1.10	i2c-read-to-file command	13
3.1.1.11	i2c-verify-with-file command	13
3.1.1.12	i2c-write-from-file command	14
3.1.1.13	i2c-read command	14
3.1.1.14	i2c-write command	14
3.1.1.15	update-brdcfg command	15
3.1.1.16	verify-brdcfg command	15
3.1.1.17	save-brdcfg command	15
3.1.1.18	update-firmware command	15
3.1.1.19	verify-firmware command	16
3.1.1.20	save-firmware command	16
3.1.1.21	update-vpd command	16
3.1.1.22	verify-vpd command	16
3.1.1.23	save-vpd command	17
3.1.1.24	display-vpd command	17
3.1.1.25	display-vpd-raw command	17
3.1.1.26	display-sensors command	17
3.1.1.27	display-sensors-raw command	18
3.1.1.28	override-sensor command	18
3.1.1.29	release-sensor command	18
3.1.1.30	spi-info command	19
3.1.1.31	spi-raw command	19
3.1.2	Commands requiring non-Service Mode	19
3.1.3	Commands requiring Service Mode	19
3.2	BITSTRIP utility	21
3.3	DMADUMP utility	22
3.4	DUMP utility	26
3.5	FLASH utility	30
3.5.1	Region to address range mapping	33
3.6	INFO utility	35
3.7	IPROG utility	38
3.8	LOADER utility	41
3.9	MONITOR utility	43
3.10	SYSMON utility	45

3.10.1	<i>SYSMON device information tab</i>	46
3.10.2	<i>SYSMON sensor information tab</i>	46
3.10.3	<i>SYSMON sensor readout tab</i>	47
3.10.4	<i>SYSMON device status tab</i>	47
3.10.5	<i>SYSMON clock generator tab</i>	48
3.10.6	<i>SYSMON sensor data logging</i>	49
3.11	<i>VPD utility</i>	52
3.11.1	<i>VPD write-protection mechanisms</i>	55
Appendix A AVR2UTIL clock generator indices		58
A.1	<i>ADM-XRC-KU1</i>	58
A.2	<i>ADM-PCIE-8V3</i>	58
A.3	<i>ADM-PCIE-8K5</i>	59

List of Tables

Table 1	Utilities for Windows and Linux	1
Table 2	Exit codes for AVR2UTIL utility	6
Table 3	Exit codes for BITSTRIP utility	21
Table 4	Exit codes for DMADUMP utility	25
Table 5	Exit codes for DUMP utility	29
Table 6	Exit codes for FLASH utility	32
Table 7	Exit codes for INFO utility	37
Table 8	Exit codes for IPROG utility	40
Table 9	Exit codes for LOADER utility	42
Table 10	Exit codes for MONITOR utility	44
Table 11	Exit codes for VPD utility	55
Table 12	AVR2UTIL clock generator indices (ADM-XRC-KU1)	58
Table 13	AVR2UTIL clock generator indices (ADM-PCIE-8V3)	58
Table 14	AVR2UTIL clock generator indices (ADM-PCIE-8K5)	59

List of Figures

Figure 1	Directory structure	1
Figure 2	SYSMON user interface	45
Figure 3	SYSMON notification area icon	46
Figure 4	SYSMON sensor information tab	46
Figure 5	SYSMON sensor readout tab	47
Figure 6	SYSMON device status tab	48
Figure 7	SYSMON clock generator tab	49
Figure 8	SYSMON Action menu in Linux	49
Figure 9	SYSMON Action menu in Windows	50

1 Introduction

This document describes the Common Host Utilities for Windows & Linux, for Alpha Data Gen 3 Reconfigurable Computing Hardware. In this context, "common" refers to the fact that these utilities, with some exceptions, can be used with all models in Alpha Data's range of Gen 3 Reconfigurable Computing Hardware:

- Embedded system products:
 - 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
 - ADM-XRC-7K1
 - ADM-XRC-7V1
 - ADM-VPX3-7V2
 - ADM-XRC-KU1
- Datacenter products:
 - ADM-PCIE-7V3
 - ADM-PCIE-KU3
 - ADM-PCIE-8V3
 - ADM-PCIE-8K5

Table 1 lists the available utilities for Windows and Linux.

AVR2UTIL	Utility for manipulating microcontroller firmware and related data (for certain models only).
BITSTRIP	Utility for removing the header from a .bit file, leaving only the SelectMap data
DMADUMP	Utility for reading and writing using DMA engines
DUMP	Utility for reading and writing memory windows
FLASH	Utility for programming FPGA bitstream (.BIT) files in user-programmable Flash memory
INFO	Utility for displaying information about a reconfigurable computing device
IPROG	Utility for software-initiated reconfiguration of a target FPGA from Flash memory
LOADER	Utility for configuring a target FPGA with a bitstream file
MONITOR	Utility that displays sensor readings
SYSMON	Utility that combines the functionality of the INFO and MONITOR utilities in a graphical user interface
VPD	Utility that allows the Vital Product Data of a reconfigurable computing device to be read or written

Table 1 : Utilities for Windows and Linux

1.1 Directory structure

The files and folders making up the Common Host Utilities are organized as in Figure 1 below:

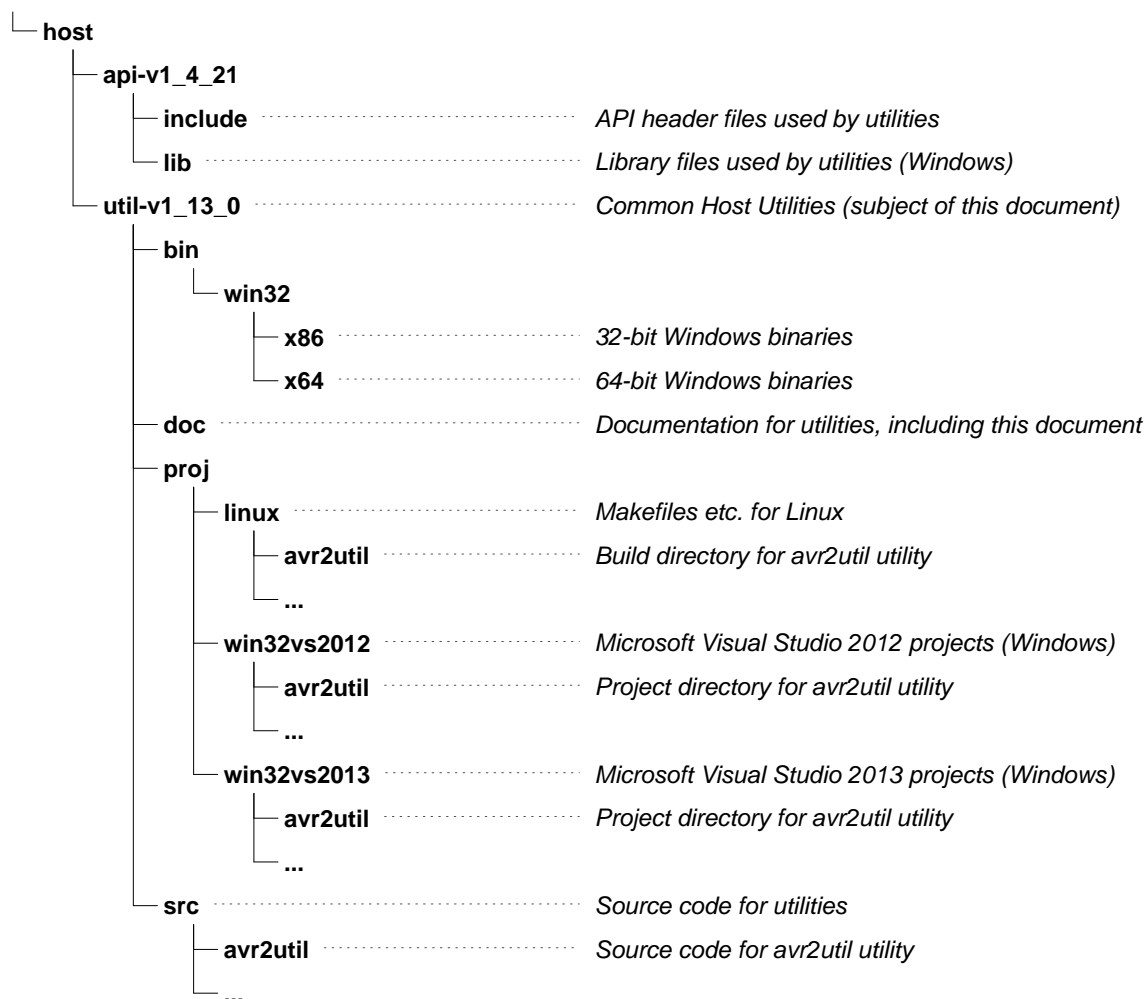


Figure 1 : Directory structure

The root of this package, i.e. the directory which forms the root of tree of directories and files making up this package, is referred to in the remainder of this document as **(root)**.

The base directory of the Common Host Utilities, i.e. **(root)/host/util-v1_13_0/** is referred to in the remainder of this document as **(util)**, or as **\$(util)** in example shell commands.

2 Building the Common Host Utilities

2.1 Building with Visual Studio 2012

A Microsoft Visual Studio 2012 solution **(util)/proj/win32vs2012/util.sln** is provided, containing projects for all of the utilities. To build all of the utilities, use the "Batch Build" command in the Microsoft Visual Studio 2012 IDE.

Note

Building the utilities does not update the **(util)/bin/x86/** or **(util)/bin/x64/** folders.

2.2 Building with Visual Studio 2013

A Microsoft Visual Studio 2013 solution **(util)/proj/win32vs2013/util.sln** is provided, containing projects for all of the utilities. To build all of the utilities, use the "Batch Build" command in the Microsoft Visual Studio 2013 IDE.

Note

Building the utilities does not update the **(util)/bin/x86/** or **(util)/bin/x64/** folders.

2.3 Building in Linux

To build all of the host utilities at once, excluding the **SYSMON** utility, enter the following shell commands in a BASH shell:

```
$ cd $(util)/proj/linux
$ make clean all
```

Note

The **SYSMON** utility must be built separately, because it depends upon certain packages being installed in the system. For further details, refer to [Section 2.3.1](#).

A number of variables, passed on the **make** command line, and certain environmental variables can modify the way the utilities are built:

- **BIARCH**

For most x86_64 Linux distributions, it is possible to build both a native (64-bit) executable and a 32-bit executable. To do this, set **BIARCH** variable to **yes** on the **make** command-line. For example:

```
make BIARCH=yes clean all
```

Assuming that building is successful, the 32-bit executable has the suffix "32" whereas the native executable has no suffix. For example, in the case of the **INFO**, the executables are named **info** (native 64-bit) and **info32** (32-bit).

- **CROSS_COMPILE**

To build using a cross-compiler, set the **CROSS_COMPILE** environment variable to the prefix of the toolchain binaries, ensuring that the toolchain is in the **PATH**. For example

```
export PATH=/path/to/toolchain:$PATH
export CROSS_COMPILE=arm-none-linux-gnueabi-
make clean all
```

- **SYSROOT**

Generally used only when cross-compiling, the value of **SYSROOT** points to the target system's root filesystem. This may be required if the toolchain used for cross-compiling does not have the required defaults for paths to system header files and libraries directories. For example:

```
export PATH=/path/to/toolchain:$PATH
export CROSS_COMPILE=arm-none-linux-gnueabi-
```

```
make SYSROOT=/path/to/arm-rootfs clean all
```

2.3.1 Building SYSMON in Linux

The Linux version of the **SYSMON** utility uses **GTKMM-2.4** or **GTKMM-3.0** for building its graphical user interface. This package is present in recent Linux distributions, but may not be present in older Linux distributions. For this reason, **SYSMON** is built separately from the other example applications. A non-exhaustive list of the packages that are required to build **SYSMON** is as follows:

gtkmm24-devel or gtkmm30-devel	cairomm-devel
libsiggc++20-devel	glibmm24-devel
pangomm-devel	pkgconfig

To run **SYSMON**, the corresponding runtime packages are required:

gtkmm24 or gtkmm30	cairomm
libsiggc++20	glibmm24
pangomm	

To build the "Release" configuration of **SYSMON**, enter the following commands in a BASH shell:

```
$ cd $(util)/proj/linux/sysmon  
$ make CONFIG=Release clean all
```

The executable's path is then **(util)/proj/linux/sysmon/bin/Release/sysmon**.

3 Common Host Utilities

3.1 AVR2UTIL / AVR2UTIL-S utility

AVR2UTIL-S

This section also covers the **AVR2UTIL-S** utility. It is functionally identical to **AVRUTIL** except that the AVR2 API library, which provides USB communication functionality, is statically built into the program for simpler deployment on Linux-based platforms.

Any example command line shown below that utilizes "avr2util" can be converted into a command line for AVR2UTIL-S by replacing "avr2util" with "avr2util-s".

Command line

```
avr2util [option ...] build-info
avr2util [option ...] version
avr2util [option ...] product-id
avr2util [option ...] enter-service-mode
avr2util [option ...] exit-service-mode
avr2util [option ...] getclk                clockgen-index
avr2util [option ...] setclk                clockgen-index frequency
avr2util [option ...] getclknv             clockgen-index
avr2util [option ...] setclknv             clockgen-index frequency
avr2util [option ...] i2c-read-to-file     bus device address count output-filename
avr2util [option ...] i2c-verify-from-file bus device address input-filename
avr2util [option ...] i2c-write-from-file  bus device address input-filename
avr2util [option ...] i2c-read            bus device address count
avr2util [option ...] i2c-write            bus device address data-byte ...
avr2util [option ...] update-brdcfg        config-filename
avr2util [option ...] verify-brdcfg        config-filename
avr2util [option ...] save-brdcfg          config-save-filename
avr2util [option ...] update-firmware      firmware-filename
avr2util [option ...] verify-firmware      firmware-filename
avr2util [option ...] save-firmware        firmware-save-filename
avr2util [option ...] update-vpd           vpd-filename
avr2util [option ...] verify-vpd           vpd-filename
avr2util [option ...] save-vpd             vpd-save-filename
avr2util [option ...] display-vpd
avr2util [option ...] display-vpd-raw
avr2util [option ...] display-sensors
avr2util [option ...] display-sensors-raw
avr2util [option ...] override-sensor      sensor-index unscaled-value
avr2util [option ...] release-sensor       sensor-index
avr2util [option ...] spi-info             chip-index
avr2util [option ...] spi-raw              chip-index read-count write-byte ...
```

The following options are accepted:

-index <index>	Specifies the index of the card to open (default 0).
-sn <#>	Specifies the serial number of the card to open.
-usbcom <port>	Specifies the USB serial port to open.
+verbose	Displays commands sent to the microcontroller and its responses, for debug purposes.

Exit codes

When **AVR2UTIL** successfully executes the requested command, the exit code is 0. When an error occurs, one of the following exit codes is returned:

Symbolic name	Value	Meaning
EXIT_OK	0	Success.
EXIT_UNRECOGNIZED_COMMAND	1	Invalid command passed as 1st positional argument.
EXIT_INSUFFICIENT_ARGS	2	Not enough positional arguments following command.
EXIT_INVALID_INDEX	3	Index value out of range or not a valid number.
EXIT_INVALID_FREQUENCY	4	Frequency value out of range or not a valid number.
EXIT_READ_FIRMWARE_FAILED	5	Failed to read firmware file.
EXIT_FIRMWARE_TOO_LARGE	6	Firmware file is too large for uC.
EXIT_ALLOCATION_FAILED	7	Failed to allocate buffer for firmware or board config. data.
EXIT_VERIFY_FAILED	8	Errors found when verifying updated firmware or board config. data.
EXIT_UNSUPPORTED_MODEL	9	Attempting to use this utility on an unsupported model.
EXIT_BAD_COMMAND_LINE	10	Illegal basic command-line syntax.
EXIT_USB_NOT_SUPPORTED	11	Access to AVR2 uC via USB is currently not supported for this OS.
EXIT_MODE_CHANGE_FAILED	12	AVR2 uC did not enter or exit Service Mode as requested.
EXIT_WRONG_MODE	13	Device is in the wrong mode (Service Mode vs. non-Service Mode) for the requested command.
EXIT_UNRECOGNIZED_PRODUCTID	14	Product ID not recognized; aborting as a precaution against firmware corruption.
EXIT_INVALID_I2C_BUS	15	I2C bus number is not valid.
EXIT_INVALID_I2C_DEVICE	16	I2C device number is not valid.
EXIT_INVALID_I2C_ADDRESS	17	I2C address is not valid.
EXIT_I2C_READ_FILE_FAILED	18	Failed to read data for I2C write from file.
EXIT_I2C_WRITE_FILE_FAILED	19	Failed to write data from I2C read to file.
EXIT_INVALID_I2C_COUNT	20	I2C data byte count is not valid.

Table 2 : Exit codes for AVR2UTIL utility (continued on next page)

Symbolic name	Value	Meaning
EXIT_INVALID_I2C_BYTEVAL	21	I2C data byte value is not valid.
EXIT_WRITE_FIRMWARE_FAILED	22	Failed to write firmware/VPD/board config to a file.
EXIT_INVALID_OVERRIDE	23	Sensor override value out of range or not a valid number.
EXIT_INVALID_SPI_INDEX	24	SPI chip index is not valid.
EXIT_INVALID_SPI_READ_COUNT	25	SPI read byte count is not valid.
EXIT_INVALID_SPI_WRITE_COUNT	26	SPI write byte count is not valid.
EXIT_INVALID_SPI_BYTEVAL	27	SPI data byte value is not valid.
EXIT_DEVICE_OPEN_ERROR	100	Failed to open device.
EXIT_AVR2_STATUS_ERROR	101	Failed to get AVR2 uC status.
EXIT_AVR2_COMMAND_ERROR	102	Failed to send command to AVR2 uC.
EXIT_AVR2_BAD_STATUS	103	AVR2 uC returned nonzero status for operation.
EXIT_AVR2_SHORT_RESPONSE	104	AVR2 uC's response was too short (< 2 bytes) to be valid.
EXIT_LIBRARY_NOT_FOUND	105	Could not find AVR2 or ADB3 shared library/DLL.

Table 2 : Exit codes for AVR2UTIL utility

Summary

This utility performs maintenance functions on the firmware of the microcontroller and associated data on the following models in Alpha Data's range of reconfigurable computing hardware:

- ADM-XRC-KU1
- ADM-PCIE-8V3
- ADM-PCIE-8K5

AVR2UTIL supports the following use-cases:

- Checking the version of the microcontroller firmware.
- Programming the clock generator on a board in a nonvolatile manner, so that it powers up providing a user-specified frequency at a given clock input on the FPGA.
- Upgrading the microcontroller firmware and its associated data.

Description

AVR2UTIL can communicate with the microcontroller in one of two ways:

- (1) Via PCI Express, using the ADB3 Driver. This requires the ADB3 Driver to be installed and running. PCIe communication mode is the default, and will be used unless the **-usbcom** option is passed on the command line.

Examples:

```
avr2util version
```

```
avr2util -index 1 version
```

When communicating via PCI Express, **AVR2UTIL** must be run from an elevated command prompt (i.e. one that has been launched as Administrator, in Windows) or root (in Linux).

- (2) Via a USB serial connection. This requires the microcontroller to be connected to the host system via a USB cable. USB communication mode is selected by passing the **-usbcom** option on the command line, followed by the device name.

- In Windows 10, the built-in USB serial communication class driver should automatically be used. For Windows Vista, 7 and 8/8.1, a **.inf** file is available from Alpha Data in the event that Windows fails to locate a suitable driver; this merely instructs Windows to use its built-in USB serial communication class driver.

The device name that follows the **-usbcom** option is of the form **COM n** where n is a number such as 1, 2, 3, ... etc.

Example:

```
avr2util -usbcom COM3 version
```

- In Linux, the built-in USB serial communication class driver is used, and the device name that follows the **-usbcom** option is of the form **/dev/ttyACM n** where n is a number such as 0, 1, 2, ... etc.

Example:

```
avr2util -usbcom /dev/ttyACM0 version
```

When communicating with the microcontroller via USB, **AVR2UTIL** need not be run from an elevated command prompt (Windows) or as root (Linux).

NOTE: Currently, the BootMan2 firmware, which executes when the microcontroller is in Service Mode, does not support USB communication. This means that if the microcontroller is in Service Mode, no USB serial communication device will be visible to the operating system on the host machine. In order to permit USB communication with the microcontroller, please ensure that it is not in Service Mode when using **AVR2UTIL**.

The **AVR2UTIL** utility currently has the following commands:

- **build-info**
Displays the version of **AVR2UTIL** itself and information about how it was built.
- **version**
Displays the version of the microcontroller firmware.
- **product-id**
Displays the Product ID of the firmware.
- **enter-service-mode**
Commands the microcontroller to enter service mode.
- **exit-service-mode**
Commands the microcontroller to exit service mode.
- **getclk <clockgen-index>**
Gets the (volatile) current output frequency for the particular clock generator selected by *<clockgen-index>*.
- **setclk <clockgen-index> <frequency>**
Sets the (volatile) current output frequency for the particular clock generator selected by *<clockgen-index>* to *<frequency>* Hz.
- **getclknv <clockgen-index>**
Gets the nonvolatile override frequency for the particular clock generator selected by *<clockgen-index>*.
- **setclknv <clockgen-index> <frequency>**
Sets the nonvolatile override frequency for the particular clock generator selected by *<clockgen-index>* to *<frequency>* Hz.
- **i2c-read-to-file <bus> <device> <address> <count> <output filename>**
Reads data from a nonvolatile memory device on a particular I2C bus and saves it into a file.
- **i2c-verify-with-file <bus> <device> <address> <input filename>**
Verifies data in a nonvolatile memory device on a particular I2C bus against the contents of a file.
- **i2c-write-to-file <bus> <device> <address> <input filename>**

Writes the contents of a file to a nonvolatile memory device on a particular I2C bus.

NOTE: This command should be used only under guidance from Alpha Data, because incorrect usage can corrupt board control devices.

- **i2c-read** *<bus> <device> <address> <count>*

Performs an individual I2C read of a device on a particular I2C bus, displaying the byte(s) read.

- **i2c-write** *<bus> <device> <address> <data byte> ...*

Performs an individual I2C write of a one or more bytes to a device on a particular I2C bus.

NOTE: This command should be used only under guidance from Alpha Data, because incorrect usage can corrupt board control devices.

- **update-brdcfg** *<brdcfg-filename>*

Writes the board-specific configuration area used by the microcontroller firmware with the contents of the file *<brdcfg-filename>*.

NOTE: This command should be used only under guidance from Alpha Data, because incorrect usage can corrupt data required by the microcontroller's firmware.

- **verify-brdcfg** *<brdcfg-filename>*

Verifies the board-specific configuration area used by the microcontroller firmware against the contents of the file *<brdcfg-filename>*.

- **save-brdcfg** *<brdcfg-save-filename>*

Reads the board-specific configuration area used by the microcontroller firmware from the board and saves it into the file *<brdcfg-save-filename>*.

- **update-firmware** *<firmware-filename>*

Writes the firmware of the microcontroller with the contents of the file *<firmware-filename>*.

NOTE: This command should be used only under guidance from Alpha Data, because incorrect usage can corrupt the microcontroller's firmware.

- **verify-firmware** *<firmware-filename>*

Verifies the firmware of the microcontroller against the contents of the file *<firmware-filename>*.

- **save-firmware** *<firmware-filename>*

Reads the firmware of the microcontroller from the board and saves it into the file *<firmware-save-filename>*.

- **update-vpd** *<vpd-filename>*

Writes the Vital Product Data (VPD) for the board with the contents of the file *<vpd-filename>*.

NOTE: This command should be used only under guidance from Alpha Data, because incorrect usage can corrupt the board's VPD.

- **verify-vpd** *<vpd-filename>*

Verifies the Vital Product Data (VPD) for the board against the contents of the file *<vpd-filename>*.

- **save-vpd** *<vpd-filename>*

Reads the Vital Product Data (VPD) from the board and saves it into the file *<vpd-save-filename>*.

- **display-vpd**

Reads the Vital Product Data (VPD) from the board and displays it in human-readable form.

- **display-vpd-raw**

Reads the Vital Product Data (VPD) from the board and displays it as raw bytes.

- **display-sensors**

Reads the Sensor Page from the board and displays it in human-readable form.

- **display-sensors-raw**

Reads the Sensor Page from the board and displays it as raw bytes.

- **override-sensor** <sensor-index> <unscaled-value>
Facilitates firmware testing by Alpha Data; permits a particular value to be injected into a sensor, overriding its natural value.
- **release-sensor** <sensor-index>
Facilitates firmware testing by Alpha Data; undoes the override-sensor command, returning a sensor to normal operation.
- **spi-info** <chip-index>
Displays the Serial Flash Discoverable Parameter (SFDP) information for a given SPI Flash chip attached to the microcontroller.
- **spi-raw** <chip-index> <read-count> <write-byte> ...
Performs a sequence of writes to (using the specified bytes of write data) and/or reads from a given SPI Flash chip attached to the microcontroller, and displays the bytes read.
NOTE: This command is intended for use by Alpha Data for test purposes.

3.1.1 Available commands

3.1.1.1 build-info command

The **build-info** command returns the version number of **AVR2UTIL** itself along with some information about how it was built:

- Whether dynamically or statically linked to the AVR2 API library (used when communicating with the uC via a USB serial connection), and the version of the AVR2 API header files.
- Whether dynamically or statically linked to the ADB3 API library (used when communicating with the uC via the PCIe host interface of a reconfigurable computing card), and the version of the ADB3 API header files.

No communication with the microcontroller on any board is performed, so none of the options do anything when used in conjunction with this command. The only useful form of this command is therefore:

```
avr2util build-info
```

3.1.1.2 version command

The **version** command displays the version of the microcontroller firmware in the form *a.b.c.d*. This command can be used as a further check in order to verify that a previous **update-firmware** command was successful.

If the microcontroller is in Service Mode, this gives the version of the Boot Manager II (BootMan2) firmware; otherwise, it gives the version of the Board Manager II (BoardMan2) firmware.

The general form of this command is:

```
avr2util [option ...] version
```

Usage example 1 - The following two commands are equivalent and display the microcontroller firmware version of the first device in the system (according to system-defined PCIe enumeration order):

```
avr2util version  
avr2util -index 0 version
```

Usage example 2 - Display the microcontroller firmware version of the device connected by a USB cable and appearing as COM3:

```
avr2util -usbcom COM3 version
```

3.1.1.3 product-id command

Displays the Product ID of the firmware. The following known Product IDs exist:

- 1320 (0x528), 720 (0x2D0) or 184320 (0x2D000) => Boot Manager II (BootMan2)

- 1321 (0x529), 721 (0x2D1) or 184321 (0x2D001) => Board Manager II (BoardMan2)

This provides a way to determine whether or not the microcontroller is in Service Mode; if the Product ID corresponds to BootMan2, the microcontroller is in Service Mode.

The general form of this command is:

```
avr2util [option ...] product-id
```

Usage example 1 - Display the microcontroller's Product ID for the second device in the system (according to system-defined PCIe enumeration order):

```
avr2util -index 1 version
```

Usage example 2 - Display the microcontroller's Product ID for the device connected by a USB cable and appearing as COM3:

```
avr2util -usbcom COM3 version
```

3.1.1.4 enter-service-mode command

Commands the microcontroller to enter service mode; if already in Service Mode, this command does nothing. The microcontroller remains in Service Mode until commanded to exit Service Mode or until a power cycle occurs.

NOTE: As of writing, the BootMan2 firmware does not support USB communication. If a command to enter Service Mode is sent via a USB serial connection and BootMan2 does not support USB communication, it will immediately exit Service Mode in order to avoid becoming "stuck" in Service Mode.

The general form of this command is:

```
avr2util [option ...] enter-service-mode
```

Usage example 1 - Command the microcontroller of the second device in the system (according to system-defined PCIe enumeration order) to enter service mode:

```
avr2util -index 1 enter-service-mode
```

3.1.1.5 exit-service-mode command

Commands the microcontroller to exit service mode; if not in Service Mode, this command does nothing.

The general form of this command is:

```
avr2util [option ...] exit-service-mode
```

Usage example 1 - The following two commands are equivalent and command the microcontroller of the first device in the system (according to system-defined PCIe enumeration order) to exit service mode:

```
avr2util exit-service-mode  
avr2util -index 0 exit-service-mode
```

3.1.1.6 getclk command

The **getclk** command returns the current frequency, in Hz, for a particular clock generator.

This command requires the microcontroller not to be in Service Mode in order to succeed. See [3.1.2](#) below.

The general form of this command is:

```
avr2util [option ...] getclk <clockgen-index>
```

For the correspondence of *<clockgen-index>* to physical clock nets, refer to [Appendix A](#).

Usage example 1 - For the second device in the system (according to system-defined PCIe enumeration order), display the current frequency of clock output 1:


```
avr2util -index 1 getclk 1
```

Usage example 2 - For the device connected by a USB cable and appearing as COM3, display the current frequency of clock output 3:

```
avr2util -usbcom COM3 getclk 3
```

3.1.1.7 setclk command

The **setclk** command sets the current frequency, in Hz, for a particular clock generator, effective immediately. The operation performed is volatile, and a power cycle returns all clock generators to their default frequencies. This command requires the microcontroller not to be in Service Mode in order to succeed. See 3.1.2 below.

The general form of this command is:

```
avr2util [option ...] setclk <clockgen-index> <frequency>
```

For the correspondence of <clockgen-index> to physical clock nets, refer to [Appendix A](#). Unlike the **setclknv** command, the **setclk** command can reprogram all clock outputs (albeit in a volatile way), even those whose nonvolatile frequency cannot be overridden.

Usage example 1 - For the first device in the system (according to system-defined PCIe enumeration order), set the current frequency of clock output 1 to 250 MHz:

```
avr2util setclk 1 250000000
```

3.1.1.8 getclknv command

The **getclknv** command returns the current nonvolatile override frequency, in Hz, for a particular clock generator. At power-on, the microcontroller inspects each clock generator's nonvolatile override frequency in turn. If set to a value other than 0 or 4294967295, it programs the clock generator to output a clock of that frequency. Otherwise, the clock generator remains at its factory default frequency.

This command requires the microcontroller not to be in Service Mode in order to succeed. See 3.1.2 below.

The general form of this command is:

```
avr2util [option ...] getclknv <clockgen-index>
```

For the correspondence of <clockgen-index> to physical clock nets, refer to [Appendix A](#).

Usage example 1 - For the second device in the system (according to system-defined PCIe enumeration order), display the override frequency for clock output 1:

```
avr2util -index 1 getclknv 1
```

Usage example 2 - For the device connected by a USB cable and appearing as COM3, display the override frequency for clock output 3:

```
avr2util -usbcom COM3 getclknv 3
```

3.1.1.9 setclknv command

The **setclknv** command sets the nonvolatile override frequency, in Hz, for a particular clock generator. This command does **not** cause the specified clock generator's actual output frequency to change immediately.

At power-on, the microcontroller inspects each clock generator's nonvolatile override frequency in turn. If set to a value other than 0 or 4294967295, it programs the clock generator to output a clock of that frequency. Otherwise, the clock generator remains at its factory default frequency.

To unset the nonvolatile override frequency for a particular clock generator, set it to 0 or 0xFFFFFFFF.

This command requires the microcontroller not to be in Service Mode in order to succeed. See 3.1.2 below.

The general form of this command is:

```
avr2util [option ...] setclknv <clockgen-index> <frequency>
```

For the correspondence of *<clockgen-index>* to physical clock nets, refer to [Appendix A](#).

Usage example 1 - For the first device in the system (according to system-defined PCIe enumeration order), set the override frequency for clock output 1 to 250 MHz:

```
avr2util setclknv 1 250000000
```

Usage example 2 - For the device connected by a USB cable and appearing as COM3, unset (remove) the override frequency for clock output 3:

```
avr2util -usbcom COM3 setclknv 3 0xFFFFFFFF
```

3.1.1.10 i2c-read-to-file command

The **i2c-read-to-file** command is primarily for in-house use by Alpha Data, but may also be used by end users under guidance from Alpha Data support personnel. This command performs multiple single-byte reads of an I2C device (usually a PROM), saving the data read into a file (usually with a **.bin** extension).

The first two arguments after the command identify the I2C bus number and I2C device (on that bus), respectively. The third argument is the address within the I2C device at which to begin reading. The fourth argument is the number of consecutively-addressed bytes to read, whilst the fifth argument is the filename into which to save the data.

This command requires the microcontroller not to be in Service Mode in order to succeed. See [3.1.2](#) below.

The general form of this command is:

```
avr2util [option ...] i2c-read-to-file <bus> <device> <address> <count> <save-filename>
```

Usage example 1 - For the first device in the system (according to system-defined PCIe enumeration order), read 256 bytes from I2C bus 0, device 0x57 starting at address 0 within the device and save the data into the file **save_file.bin**:

```
avr2util i2c-read-to-file 0 0x57 0 256 /path/to/save_file.bin
```

3.1.1.11 i2c-verify-with-file command

The **i2c-verify-with-file** command is primarily for in-house use by Alpha Data, but may also be used by end users under guidance from Alpha Data support personnel. This command performs multiple single-byte reads of an I2C device (usually a PROM), and verifies that the data read matches the contents of a file (usually with a **.bin** extension).

The first two arguments after the command identify the I2C bus number and I2C device (on that bus), respectively. The third argument is the address within the I2C device at which to begin verifying. The fourth argument is the name of the file against which the data read is compared.

This command requires the microcontroller not to be in Service Mode in order to succeed. See [3.1.2](#) below.

The general form of this command is:

```
avr2util [option ...] i2c-verify-with-file <bus> <device> <address> <verify-filename>
```

Usage example 1 - For the first device in the system (according to system-defined PCIe enumeration order), verify that the data in I2C bus 0, device 0x57 starting at address 0 within the device matches the contents of the file **verify_file.bin**:

```
avr2util i2c-verify-with-file 0 0x57 0 /path/to/verify_file.bin
```

3.1.1.12 i2c-write-from-file command

The **i2c-write-from-file** command is primarily for in-house use by Alpha Data, but may also be used by end users under guidance from Alpha Data support personnel. This command performs multiple single-byte writes to an I2C device (usually a PROM), obtaining the data to be written from a file (usually with a **.bin** extension).

The first two arguments after the command identify the I2C bus number and I2C device (on that bus), respectively. The third argument is the address within the I2C device at which to begin writing. The fourth argument is the name of the file containing the data to be written.

This command requires the microcontroller not to be in Service Mode in order to succeed. See [3.1.2](#) below.

The general form of this command is:

```
avr2util [option ...] i2c-write-from-file <bus> <device> <address> <write-filename>
```

Usage example 1 - For the first device in the system (according to system-defined PCIe enumeration order), write I2C device 0x57 on bus 0, starting at address 0 within the device, with the contents of the file **data_file.bin**:

```
avr2util i2c-write-from-file 0 0x57 0 /path/to/data_file.bin
```

3.1.1.13 i2c-read command

The **i2c-read** command is primarily for in-house use by Alpha Data, but may also be used by end users under guidance from Alpha Data support personnel. This command performs an individual read of one or more bytes from an I2C device, displaying the bytes read.

The first two arguments after the command identify the I2C bus number and I2C device (on that bus), respectively. The third argument is the address within the I2C device at which to perform a read. The fourth argument is the length of the I2C read, in bytes.

This command requires the microcontroller not to be in Service Mode in order to succeed. See [3.1.2](#) below.

The general form of this command is:

```
avr2util [option ...] i2c-read <bus> <device> <address> <count>
```

Usage example 1 - For the first device in the system (according to system-defined PCIe enumeration order), perform a 4-byte read from I2C bus 1, device 0x30 at address 0x10 within the device:

```
avr2util i2c-read 1 0x30 0x10 4
```

3.1.1.14 i2c-write command

The **i2c-write** command is primarily for in-house use by Alpha Data, but may also be used by end users under guidance from Alpha Data support personnel. This command performs an individual write of one or more bytes to an I2C device, with the bytes to be written obtained from the command-line.

The first two arguments after the command identify the I2C bus number and I2C device (on that bus), respectively. The third argument is the address within the I2C device at which to perform a write. The fourth and later arguments are the bytes written to the I2C device.

This command requires the microcontroller not to be in Service Mode in order to succeed. See [3.1.2](#) below.

The general form of this command is:

```
avr2util [option ...] i2c-write <bus> <device> <address> <data byte> ...
```

Usage example 1 - For the first device in the system (according to system-defined PCIe enumeration order), perform a 4-byte write to I2C bus 1, device 0x30 at address 0x10 within the device, where the individual bytes written are 0x12, 0x34, 0x56 and 0x78:

```
avr2util i2c-write 1 0x30 0x10 0x12 0x34 0x56 0x78
```

3.1.1.15 update-brdcfg command

NOTE: This command should be used only under guidance from Alpha Data, because incorrect usage can corrupt data required by the microcontroller's firmware.

The **update-brdcfg** command writes a block of data required by the microcontroller firmware with the contents of the specified file (usually with a **.bin** extension).

This command requires the microcontroller to be in Service Mode, at least temporarily. See [3.1.3](#) below.

The general form of this command is:

```
avr2util [option ...] update-brdcfg <brdcfg-filename>
```

Usage example 1 - For the first device in the system (according to system-defined PCIe enumeration order), write the board configuration area with the contents of a file:

```
avr2util update-brdcfg /path/to/brdcfg_file.bin
```

3.1.1.16 verify-brdcfg command

The **verify-brdcfg** command verifies that the block of data required by the microcontroller firmware matches the contents of the specified file (usually with a **.bin** extension).

This command requires the microcontroller to be in Service Mode, at least temporarily. See [3.1.3](#) below.

The general form of this command is:

```
avr2util [option ...] verify-brdcfg <brdcfg-filename>
```

Usage example 1 - For the first device in the system (according to system-defined PCIe enumeration order), verify that the board configuration area matches the contents of a file:

```
avr2util verify-brdcfg /path/to/brdcfg_file.bin
```

3.1.1.17 save-brdcfg command

The **save-brdcfg** command reads the block of data required by the microcontroller, from the nonvolatile memory in a board in which it resides, and saves it into the specified file (usually with a **.bin** extension). Before updating the board configuration of a board using the **update-brdcfg** command, **save-brdcfg in a board** can be used to take a copy of the existing board configuration data, should it be necessary to revert the board configuration data.

This command requires the microcontroller to be in Service Mode, at least temporarily. See [3.1.3](#) below.

The general form of this command is:

```
avr2util [option ...] save-brdcfg <brdcfg-save-filename>
```

Usage example 1 - For the first device in the system (according to system-defined PCIe enumeration order), save its board configuration data into a file:

```
avr2util save-brdcfg /path/to/brdcfg_save_file.bin
```

3.1.1.18 update-firmware command

NOTE: This command should be used only under guidance from Alpha Data, because incorrect usage can corrupt the microcontroller's firmware.

The **update-brdcfg** command writes the firmware of the microcontroller with the contents of the specified file (usually with a **.bin** extension).

This command requires the microcontroller to be in Service Mode, at least temporarily. See [3.1.3](#) below.

The general form of this command is:

```
avr2util [option ...] update-firmware <firmware-filename>
```

Usage example 1 - For the first device in the system (according to system-defined PCIe enumeration order),

write the BoardMan2 firmware with the contents of a file:

```
avr2util update-firmware /path/to/firmware_file.bin
```

3.1.1.19 verify-firmware command

The **verify-firmware** command verifies the firmware of the microcontroller against the contents of the specified file (usually with a **.bin** extension).

This command requires the microcontroller to be in Service Mode, at least temporarily. See [3.1.3](#) below.

The general form of this command is:

```
avr2util [option ...] verify-firmware <firmware-filename>
```

Usage example 1 - For the first device in the system (according to system-defined PCIe enumeration order), verify that the BoardMan2 firmware matches the contents of a file:

```
avr2util verify-firmware /path/to/firmware_file.bin
```

3.1.1.20 save-firmware command

The **save-firmware** command reads the firmware of the microcontroller, from the nonvolatile memory in a board in which it resides, and saves it into the specified file (usually with a **.bin** extension). Before updating the firmware of a board using the **update-firmware** command, **save-firmware** can be used to take a copy of the existing firmware, should it be necessary to revert the firmware.

This command requires the microcontroller to be in Service Mode, at least temporarily. See [3.1.3](#) below.

The general form of this command is:

```
avr2util [option ...] save-firmware <firmware-save-filename>
```

Usage example 1 - For the first device in the system (according to system-defined PCIe enumeration order), save the BoardMan2 firmware into a file:

```
avr2util save-firmware /path/to/firmware_save_file.bin
```

3.1.1.21 update-vpd command

NOTE: This command should be used only under guidance from Alpha Data, because incorrect usage can corrupt the board's VPD.

Writes the Vital Product Data (VPD) with the contents of the specified file (usually with a **.bin** extension).

This command requires the microcontroller to be in Service Mode, at least temporarily. See [3.1.3](#) below.

The general form of this command is:

```
avr2util [option ...] update-vpd <vpd-filename>
```

Usage example 1 - For the first device in the system (according to system-defined PCIe enumeration order), write the VPD area with the contents of a file:

```
avr2util update-vpd /path/to/vpd_file.bin
```

3.1.1.22 verify-vpd command

Verifies the Vital Product Data (VPD) against the contents of the specified file (usually with a **.bin** extension).

This command requires the microcontroller to be in Service Mode, at least temporarily. See [3.1.3](#) below.

The general form of this command is:

```
avr2util [option ...] verify-vpd <vpd-filename>
```

Usage example 1 - For the first device in the system (according to system-defined PCIe enumeration order),

verify that the VPD area matches the contents of a file:

```
avr2util verify-vpd /path/to/vpd_file.bin
```

3.1.1.23 save-vpd command

The **save-vpd** command reads the Vital Product Data (VPD), from the nonvolatile memory in a board in which it resides, and saves it into the specified file (usually with a **.bin** extension).

This command requires the microcontroller to be in Service Mode, at least temporarily. See 3.1.3 below. Before updating the VPD of a board using the **update-vpd** command, **save-vpd** can be used to take a copy of the existing VPD, should it be necessary to revert the VPD.

The general form of this command is:

```
avr2util [option ...] save-vpd <vpd-save-filename>
```

Usage example 1 - For the first device in the system (according to system-defined PCIe enumeration order), save the VPD area into a file:

```
avr2util save-vpd /path/to/vpd_save_file.bin
```

3.1.1.24 display-vpd command

The **display-vpd** command reads the Vital Product Data (VPD) area, from the nonvolatile memory in a board in which it resides, and displays it in human-readable form.

This command works whether or not the microcontroller is in Service Mode.

The general form of this command is:

```
avr2util [option ...] display-vpd
```

Usage example 1 - For the first device in the system (according to system-defined PCIe enumeration order), display the VPD in human-readable form:

```
avr2util display-vpd
```

3.1.1.25 display-vpd-raw command

The **display-vpd** command reads the Vital Product Data (VPD) area, from the nonvolatile memory in a board in which it resides, and displays it as raw bytes.

This command works whether or not the microcontroller is in Service Mode.

The general form of this command is:

```
avr2util [option ...] display-vpd-raw
```

Usage example 1 - For the second device in the system (according to system-defined PCIe enumeration order), display the VPD as raw bytes:

```
avr2util -index 1 display-vpd-raw
```

3.1.1.26 display-sensors command

The **display-sensors** command requests the Sensor Page from the microcontroller and displays it in human-readable form.

This command requires the microcontroller not to be in Service Mode, at least temporarily, in order to succeed. See 3.1.2 below.

The general form of this command is:

```
avr2util [option ...] display-sensors
```

Usage example 1 - For board accessible via USB serial device /dev/ttyACM1 (a Linux device name), display the Sensor Page in human-readable form:

```
avr2util -usbcom /dev/ttyACM1 display-sensors
```

3.1.1.27 display-sensors-raw command

The **display-sensors-raw** command requests the Sensor Page from the microcontroller and displays it as raw bytes.

This command requires the microcontroller not to be in Service Mode, at least temporarily, in order to succeed. See 3.1.2 below.

The general form of this command is:

```
avr2util [option ...] display-sensors-raw
```

Usage example 1 - For board accessible via USB serial device /dev/ttyACM1 (a Linux device name), display the Sensor Page as raw bytes:

```
avr2util -usbcom /dev/ttyACM1 display-sensors-raw
```

3.1.1.28 override-sensor command

The **override-sensor** command facilitates firmware testing by Alpha Data. It injects a value into a particular sensor, overriding its natural value. The sensor remains overridden until at least one of the following occurs:

- A power cycle, including removal and reapplication of standby power.
- The microcontroller enters and exits Service Mode, either using the **enter-service-mode** and **exit-service-mode** commands or by toggling the physical Service Mode switch.
- The **release-sensor** command is used to explicitly release the sensor.

This command requires the microcontroller not to be in Service Mode, at least temporarily, in order to succeed. See 3.1.2 below.

The first argument after the command identifies the sensor to be overridden, and the second argument is the sensor-specific value to be injected.

The general form of this command is:

```
avr2util [option ...] override-sensor <sensor index> <unscaled value>
```

Usage example 1 - For the board accessible via USB serial device COM5 (a Windows device name), inject a value of 0 into sensor 11:

```
avr2util -usbcom COM5 override-sensor 11 0
```

3.1.1.29 release-sensor command

The **release-sensor** command facilitates firmware testing by Alpha Data. It undoes a previous **override-sensor** command, returning a particular sensor to normal operation.

This command requires the microcontroller not to be in Service Mode, at least temporarily, in order to succeed. See 3.1.2 below.

The single argument after the command identifies the sensor which is to return to normal operation (i.e. no longer overridden).

The general form of this command is:

```
avr2util [option ...] release-sensor <sensor index>
```

Usage example 1 - For the board accessible via USB serial device COM5 (a Windows device name), return sensor 11 to normal operation.

```
avr2util -usbcom COM5 release-sensor 11
```

3.1.1.30 spi-info command

The **spi-info** reads the Serial Flash Discoverable Parameters (SFDP) information from a SPI Flash chip that is accessible from the microcontroller, and displays it in human-readable form.

This command requires the microcontroller not to be in Service Mode, at least temporarily, in order to succeed. See 3.1.2 below.

The single argument after the command is the zero-based index of the SPI Flash chip whose SFDP information is to be displayed.

The general form of this command is:

```
avr2util [option ...] spi-info <chip index>
```

Usage example 1 - For the board accessible via USB serial device COM5 (a Windows device name), display the SFDP information for the 2nd SPI Flash chip (index 1).

```
avr2util -usbcom COM5 spi-info 1
```

3.1.1.31 spi-raw command

The **spi-raw** command is primarily for in-house use by Alpha Data for test purposes. This command performs a transaction consisting of zero or more bytes written to an SPI Flash chip followed by zero or more bytes read from the same SPI Flash chip, with the bytes to be written obtained from the command-line. The bytes read are displayed in raw form, i.e. as individual byte values.

The first argument after the command is the zero-based index of the SPI Flash chip. The second argument is the number of bytes read, and the third and later arguments are the bytes written to the SPI Flash chip.

This command requires the microcontroller not to be in Service Mode in order to succeed. See 3.1.2 below.

The general form of this command is:

```
avr2util [option ...] spi-raw <chip index> <read count> <write byte> ...
```

Usage example 1 - For the first device in the system (according to system-defined PCIe enumeration order), read and display (in the form of raw bytes) the 8-byte SFDP header from the first SPI Flash chip (index 0). JEDEC document **JESD216B** describes the SFDP mechanism in SPI Flash chips.

```
avr2util spi-raw 0 8 0x5A 0 0 0 0
```

3.1.2 Commands requiring non-Service Mode

The following commands can only perform their main operation while **not** in Service Mode:

- **getclk, getclknv, setclk, setclknv**
- **i2c-read-to-file, i2c-verify-with-file, i2c-write-from-file**
- **i2c-read, i2c-write**
- **display-sensors, display-sensors-raw**
- **override-sensor, release-sensor**

If communicating with the microcontroller via PCIe and it is in Service Mode, these commands temporarily switch the microcontroller out of Service Mode, perform the necessary operations and then switch the microcontroller back into Service Mode.

3.1.3 Commands requiring Service Mode

The following commands can only perform their main operation while in Service Mode:

- **update-brdcfg, save-brdcfg, verify-brdcfg**

- **update-firmware, save-firmware, verify-firmware**
- **update-vpd, save-vpd, verify-vpd**

If communicating with the microcontroller in PCIe mode and it is not in Service Mode, these commands temporarily switch the microcontroller into Service Mode, perform the necessary operations, and then switch the microcontroller out of Service Mode.

If communicating with the microcontroller in USB mode, these commands will not attempt to switch the microcontroller into Service Mode. Instead, they abort with an error message, returning exit code EXIT_WRONG_MODE (13). This is because a switch to Service Mode reboots the microcontroller and causes the USB serial communication device to disappear (from the point of view of the host). As of writing, the BootMan2 firmware, which executes when the microcontroller is in Service Mode, does not support USB communication.

3.2 BITSTRIP utility

Command line

```
bitstrip [option ...] <input .bit filename>
```

where the following options are accepted:

`-o <output filename>` Specifies the name of the file into which the SelectMap data is written.

Summary

Reads an FPGA bitstream (**.bit**) file, displays certain information from the header, and optionally writes the SelectMap data (without the header) to another file.


Description

To simply display information from a **.bit** file's header, use

```
bitstrip <input .bit filename>
```

To display information from a **.bit** file's header and write the SelectMap data to another file, use

```
bitstrip -o <output filename> <input .bit filename>
```

The data written to *<output filename>* is suitable for sending to a target FPGA using [ADMXRC3_ConfigureFromBuffer](#) .

Exit codes

When **BITSTRIP** runs successfully, the exit code is 0. When an error occurs, one of the following exit codes is returned:

Symbolic name	Value	Meaning
EXIT_OK	0	Success.
EXIT_BAD_COMMAND_LINE	1	Illegal basic command-line syntax.
EXIT_INSUFFICIENT_ARGS	2	Not enough positional arguments.
EXIT_LOAD_BIT_ERROR	3	Failed to read the input .bit file.
EXIT_SAVE_BIN_ERROR	4	Failed to write the output file.

Table 3 : Exit codes for BITSTRIP utility

3.3 DMADUMP utility

Command line

```
dmadump [option ...] fb channel address [n] [fill value]
dmadump [option ...] fw channel address [n] [fill value]
dmadump [option ...] fd channel address [n] [fill value]
dmadump [option ...] fq channel address [n] [fill value]
dmadump [option ...] rb channel address [n]
dmadump [option ...] rw channel address [n]
dmadump [option ...] rd channel address [n]
dmadump [option ...] rq channel address [n]
dmadump [option ...] wb channel address [n] [data ...]
dmadump [option ...] ww channel address [n] [data ...]
dmadump [option ...] wd channel address [n] [data ...]
dmadump [option ...] wq channel address [n] [data ...]
```

where

<i>channel</i>	is the index of the DMA engine / DMA channel to use.
<i>address</i>	is the local address (generally an OCP address) at which to begin reading or writing.
<i>n</i>	is the number of bytes to read or write (optional).
<i>fill value</i>	is an optional fill value, valid for fill commands.
<i>data</i>	is an optional data item, valid for write commands.

and the following options are accepted:

-index <index>	Specifies the index of the card to open (default 0).
-sn <#>	Specifies the serial number of the card to open.
-be	Causes the data to be read or written to be treated as little-endian (default).
+be	Causes the data to be read or written to be treated as big-endian.
-hex	Causes write values to be interpreted as decimal unless prefixed by '0x' (default).
+hex	Causes write values to be interpreted as hexadecimal always.

DMADUMP opens a reconfigurable computing device with intent to modify its state, and for this reason it must be run in an elevated command prompt (i.e. as Administrator in Windows) or as root (in Linux).

Summary

Displays data read from a target FPGA using a DMA engine, or writes data to a target FPGA using a DMA engine.

Description

The **DMADUMP** utility operates in one of three modes:

- Reading data from a target FPGA using a DMA engine and displaying it; for this mode, use the **rb**, **rw**, **rd** or **rq** commands.
- Writing data to a target FPGA using a DMA engine; for this mode, use the **wb**, **ww**, **wd** or **wq** commands.
- Filling an address region in a target FPGA using a DMA engine with a particular value; for this mode, use the **fb**, **fw**, **fd** or **fq** commands.

The option **+be** may be specified, before the command. This causes the **DMADUMP** utility to use big-endian byte ordering convention as opposed to little-endian (the default).

Read commands

The read command implies the word width for displaying data:

- **rb**
Byte (8-bit) reads; data is displayed as bytes.
- **rw**
Word (16-bit) reads; data is displayed as words.
- **rd**
Doubleword (32-bit) reads; data is displayed as doublewords.
- **rq**
Quadword (64-bit) reads; data is displayed as quadwords.

After the read command, a DMA channel/engine index and an address must be supplied, in that order. This specifies the DMA engine to use, and where in that DMA engine's address space to begin reading data. An optional length parameter, in bytes, can also be supplied. If omitted, the length is equal to the word width implied by the read command. If present, the length parameter specifies how many bytes to read and display. The length should be an integer multiple of the width; if not, the length is rounded down.

Write commands

The write command implies the word width to be used when performing writes:

- **wb**
Data is written as bytes (8-bit).
- **ww**
Data is written as words (16-bit).
- **wd**
Data is written as doublewords (32-bit).
- **wq**
Data is written as quadwords (64-bit).

After the write command, a DMA channel/engine index and an address must be supplied, in that order. This specifies which DMA engine to use for writing the data, and where in that DMA engine's address space to begin writing the data. An optional length parameter, in bytes, can also be supplied. If omitted, the length is equal to the word width implied by the write command. If present, the length parameter specifies how many bytes to write. The length should be an integer multiple of the width; if not, the length is rounded down.

The program obtains the values to be written in two ways: from any additional parameters on the command line after the length parameter, and then from the standard input stream (stdin). This works as follows:

- (1) Any remaining command line arguments, if present after the length parameter, are interpreted as data values to be written. These values are assumed to be of the word width implied by the command, and are written using the specified DMA engine, incrementing the address with each word written. If there are enough values passed on the command line to satisfy the byte count, the program terminates.
- (2) If there are insufficient data values passed on the command line, the program waits for values to be entered on the standard input stream. Values entered this way are also assumed to be of the word width implied by the command, and are written using the specified DMA engine, incrementing the address with each word written. When the entire byte count that was specified in the length parameter has been satisfied or end-of-file is encountered, the program terminates.

Fill commands

The fill command implies the word width to be used when performing a fill:

- **fb**
The fill value is a byte value (8-bit).
- **fw**
The fill value is a word value (16-bit).
- **fd**
The fill value is a double word value (32-bit).
- **fq**
The fill value is a quadword value (64-bit).

After the fill command, a DMA channel/engine index and an address must be supplied, in that order. This specifies the DMA engine to use, and where in that DMA engine's address space to begin writing data. An optional length parameter, in bytes, can also be supplied. If omitted, the length is equal to the word width implied by the write command. If present, the length parameter specifies how many bytes to write. The length should be an integer multiple of the width; if not, the length is rounded down.

The fill value is obtained in one of two ways: from an additional parameter on the command line after the length parameter, or from the standard input stream (stdin). There must be exactly one fill value.

Example session

Assuming that the target FPGA is currently configured with an FPGA bitstream which has a RAM-like region in the OCP/AXI address space of DMA channel 0 at address 0x80000, an example session looks like this:

```
# ./dmadump fb 0 0x80000 0x20 0xee
# ./dmadump rd 0 0x80000 0x40
Dump of memory at 0x00000000_00080000 + 64(0x40) bytes:
          00          04          08          0C
0x00000000_00080000: EEEEEEEE EEEEEEEE EEEEEEEE EEEEEEEE .....
0x00000000_00080010: EEEEEEEE EEEEEEEE EEEEEEEE EEEEEEEE .....
0x00000000_00080020: 00000000 00000000 00000000 00000000 .....
0x00000000_00080030: 00000000 00000000 00000000 00000000 .....
# ./dmadump wd 0 0x80004 0x4 0x12345678
0x00000000_00080004: 0x12345678
# ./dmadump rd 0 0x80000 0x40
Dump of memory at 0x00000000_00080000 + 64(0x40) bytes:
          00          04          08          0C
0x00000000_00080000: EEEEEEEE 12345678 EEEEEEEE EEEEEEEE .....
0x00000000_00080010: EEEEEEEE EEEEEEEE EEEEEEEE EEEEEEEE .....
0x00000000_00080020: 00000000 00000000 00000000 00000000 .....
0x00000000_00080030: 00000000 00000000 00000000 00000000 .....
```

Remarks

When entering data for write commands, values are expressed in decimal by default. To express data as hexadecimal, prefix it with '0x' or use the **+hex** option.

Each DMA engine has its own address space. This means that in general, unless an FPGA design explicitly makes a shared resource available to multiple DMA engines, writing data using one DMA engine and then attempting to read it back using a different DMA engine will not return the data just written.

The Direct Slave address space is separate from the address space of each DMA engine. This means that in general, writing data using the **DMADUMP** utility and attempting to read it back via the **DUMP** utility will not return the same data. However, it is possible to create an FPGA design which explicitly makes a shared resource available in both the Direct Slave address space and the address space of a DMA engine. In that case, data written by one channel can be read back via another channel.

Exit codes

When **DMADUMP** successfully executes the requested command, the exit code is 0. When an error occurs, one of the following exit codes is returned:

Symbolic name	Value	Meaning
EXIT_OK	0	Success.
EXIT_BAD_COMMAND_LINE	1	Illegal basic command-line syntax.
EXIT_UNRECOGNIZED_COMMAND	2	Invalid command passed as 1st positional argument.
EXIT_ALLOCATION_FAILURE	3	A memory allocation failed.
EXIT_DEVICE_OPEN_ERROR	4	Failed to open ADMXRC3 device.
EXIT_WRITEDMA_ERROR	5	A DMA write call failed.
EXIT_READDMA_ERROR	6	A DMA read call failed.
ERROR_COUNT_TOO_LARGE	7	Requested byte count is too large.

Table 4 : Exit codes for DMADUMP utility

3.4 DUMP utility

Command line

```
dump [option ...] fb window offset [n] [fill value]
dump [option ...] fw window offset [n] [fill value]
dump [option ...] fd window offset [n] [fill value]
dump [option ...] fq window offset [n] [fill value]
dump [option ...] rb window offset [n]
dump [option ...] rw window offset [n]
dump [option ...] rd window offset [n]
dump [option ...] rq window offset [n]
dump [option ...] wb window offset [n] [data ...]
dump [option ...] ww window offset [n] [data ...]
dump [option ...] wd window offset [n] [data ...]
dump [option ...] wq window offset [n] [data ...]
```

where

<i>window</i>	is the memory window to read or write.
<i>offset</i>	is the offset into the window at which to begin reading or writing.
<i>n</i>	is the number of bytes to read or write (optional).
<i>fill value</i>	is an optional fill value, valid for fill commands.
<i>data</i>	is an optional data item, valid for write commands.

and the following options are accepted:

-index <index>	Specifies the index of the card to open (default 0).
-sn <#>	Specifies the serial number of the card to open.
-be	Causes the data to be read or written to be treated as little-endian (default).
+be	Causes the data to be read or written to be treated as big-endian.
-hex	Causes write values to be interpreted as decimal unless prefixed by '0x' (default).
+hex	Causes write values to be interpreted as hexadecimal always.

DUMP opens a reconfigurable computing device with intent to modify its state, and for this reason it must be run in an elevated command prompt (i.e. as Administrator in Windows) or as root (in Linux).

Summary

Displays data read from a memory access window, or writes data to a memory access window.

Description

The **DUMP** utility operates in one of three modes:

- Reading data from a memory access window and displaying it; for this mode, use the **rb**, **rw**, **rd** or **rq** commands.
- Writing data to a memory access window; for this mode, use the **wb**, **ww**, **wd** or **wq** commands.
- Filling a region of a memory access window with a particular value; for this mode, use the **fb**, **fw**, **fd** or **fq** commands.

The option **+be** may be specified, before the command. This causes the **DUMP** utility to use big-endian byte

ordering convention as opposed to little-endian (the default).

Read commands

The read command implies the word width for displaying data:

- **rb**
Byte (8-bit) reads; data is displayed as bytes.
- **rw**
Word (16-bit) reads; data is displayed as words.
- **rd**
Doubleword (32-bit) reads; data is displayed as doublewords.
- **rq**
Quadword (64-bit) reads; data is displayed as quadwords.

After the read command, a window index and an offset must be supplied, in that order. This specifies the memory access window to be read, and where in that window to begin reading data. An optional length parameter, in bytes, can also be supplied. If omitted, the length is equal to the word width implied by the read command. If present, the length parameter specifies how many bytes to read and display. The length should be an integer multiple of the width; if not, the length is rounded down.

Write commands

The write command implies the word width to be used when performing writes:

- **wb**
Data is written as bytes (8-bit).
- **ww**
Data is written as words (16-bit).
- **wd**
Data is written as doublewords (32-bit).
- **wq**
Data is written as quadwords (64-bit).

After the write command, a window index and an offset must be supplied, in that order. This specifies the memory access window to be written, and where in that window to begin writing data. An optional length parameter, in bytes, can also be supplied. If omitted, the length is equal to the word width implied by the write command. If present, the length parameter specifies how many bytes to write. The length should be an integer multiple of the width; if not, the length is rounded down.

The program obtains the values to be written in two ways: from any additional parameters on the command line after the length parameter, and then from the standard input stream (stdin). This works as follows:

- (1) Any remaining command line arguments, if present after the length parameter, are interpreted as data values to be written. These values are assumed to be of the word width implied by the command, and are written to the memory window, incrementing the offset with each word written. If there are enough values passed on the command line to satisfy the byte count, the program terminates.
- (2) If there are insufficient data values passed on the command line, the program waits for values to be entered on the standard input stream. Values entered this way are also assumed to be of the word width implied by the command, and are written to the memory window, incrementing the offset with each word written. When the entire byte count that was specified in the length parameter has been satisfied or end-of-file is encountered, the program terminates.

Fill commands

The fill command implies the word width to be used when performing a fill:

- **fb**
The fill value is a byte value (8-bit).
- **fw**
The fill value is a word value (16-bit).
- **fd**
The fill value is a double word value (32-bit).
- **fq**
The fill value is a quadword value (64-bit).

After the fill command, a window index and an offset must be supplied, in that order. This specifies the memory access window to be written, and where in that window to begin writing data. An optional length parameter, in bytes, can also be supplied. If omitted, the length is equal to the word width implied by the write command. If present, the length parameter specifies how many bytes to write. The length should be an integer multiple of the width; if not, the length is rounded down.

The fill value is obtained in one of two ways: from an additional parameter on the command line after the length parameter, or from the standard input stream (stdin). There can be only one fill value.

Example session

Assuming that the target FPGA is currently configured with an FPGA bitstream which has a RAM-like region in the OCP/AXI address space of the Direct Slave channel at address 0x80000, an example session looks like this:

```
# ./dump rd 0 0x80000 0x40
Dump of memory at 0x00000000_00080000 + 64(0x40) bytes:
00 04 08 0C
0x00000000_00080000: 00000000 00000000 00000000 00000000 .....
0x00000000_00080010: 00000000 00000000 00000000 00000000 .....
0x00000000_00080020: 00000000 00000000 00000000 00000000 .....
0x00000000_00080030: 00000000 00000000 00000000 00000000 .....
# ./dump fw 0 0x80000 0x20 0x1234
# ./dump rd 0 0x80000 0x40
Dump of memory at 0x00000000_00080000 + 64(0x40) bytes:
00 04 08 0C
0x00000000_00080000: 12341234 12341234 12341234 12341234 4.4.4.4.4.4.4.4.
0x00000000_00080010: 12341234 12341234 12341234 12341234 4.4.4.4.4.4.4.4.
0x00000000_00080020: 00000000 00000000 00000000 00000000 .....
0x00000000_00080030: 00000000 00000000 00000000 00000000 .....
# ./dump wd 0 0x80004 0x8 0xdeadbeef 0xcafeface
0x00000000_00080004: 0xDEADBEEF
0x00000000_00080008: 0xCAFEFACE
# ./dump rd 0 0x80000 0x40
Dump of memory at 0x00000000_00080000 + 64(0x40) bytes:
00 04 08 0C
0x00000000_00080000: 12341234 DEADBEEF CAFEFACE 12341234 4.4.....4.4.
0x00000000_00080010: 12341234 12341234 12341234 12341234 4.4.4.4.4.4.4.4.
0x00000000_00080020: 00000000 00000000 00000000 00000000 .....
0x00000000_00080030: 00000000 00000000 00000000 00000000 .....
```

Remarks

When entering data for write commands, values are expressed in decimal by default. To express data as hexadecimal, prefix it with '0x' or use the **+hex** option.

The **DUMP** utility uses store instructions for writes whose widths correspond to the word width specified on the command line, if possible. This is not possible if the CPU architecture in use does not have store instructions of the required width or if the offset specified on the command line would result in unaligned stores. In the case of an unaligned offset, writes are performed as a sequence of byte stores, because unaligned stores are illegal on some CPU architectures. FPGA designs that use byte enables to mask writes to byte lanes will work correctly

regardless of the size of a store instruction generated by the compiler.

The Direct Slave address space is separate from the address space of each DMA engine. This means that in general, writing data using the **DUMP** utility and attempting to read it back via the **DMADUMP** utility will not return the same data. However, it is possible to create an FPGA design which explicitly makes a shared resource available in both the Direct Slave address space and the address space of a DMA engine. In that case, data written by one channel can be read back via another channel.

Exit codes

When **DUMP** successfully executes the requested command, the exit code is 0. When an error occurs, one of the following exit codes is returned:

Symbolic name	Value	Meaning
EXIT_OK	0	Success.
EXIT_BAD_COMMAND_LINE	1	Illegal basic command-line syntax.
EXIT_UNRECOGNIZED_COMMAND	2	Invalid command passed as 1st positional argument.
EXIT_ALLOCATION_FAILURE	3	A memory allocation failed.
EXIT_DEVICE_OPEN_ERROR	4	Failed to open ADMXRC3 device.
EXIT_MAPWINDOW_ERROR	5	Failed to map window into process' virtual address space.

Table 5 : Exit codes for DUMP utility

3.5 FLASH utility

Command line

```
flash [option ...] info      target-index
flash [option ...] chkblank target-index
flash [option ...] erase     target-index
flash [option ...] program   target-index filename
flash [option ...] verify    target-index filename
```

where

target-index is the index of a target FPGA.
filename is the name of a .BIT file (program or verify commands only).

and the following options are accepted:

-index <index>	Specifies the index of the card to open (default 0).
-sn <#>	Specifies the serial number of the card to open.
-failsafe	Causes the command to target the default region of the Flash.
+failsafe	Causes the command to target the failsafe of the Flash; see Section 3.5.1 below.
+force	Causes a mismatch between the target FPGA device and the .BIT file device to be ignored.
-force	Causes a mismatch between the target FPGA device and the .BIT file device to result in an error (default).
-range <address>,<length>	Overrides the program logic that determines how to map a region index to a range of Flash addresses. See Section 3.5.1 below.
-region <n>	Causes the command to target region <i>n</i> of the Flash; see Section 3.5.1 below.

The **-failsafe**, **-range** and **-region** options are all mutually exclusive to one another; at most, one of these options can be passed on the command line.

The **FLASH** utility opens a reconfigurable computing device with intent to modify its state, and for this reason it must be run in an elevated command prompt (i.e. as Administrator in Windows) or as root (in Linux).

Summary

Blank-checks, erases, programs or verifies a target FPGA bitstream region in the user-programmable Flash memory of a device.

Description

The **FLASH** utility has five commands:

- **chkblank** <target-index>
Verifies that a region is blank, i.e. all bytes are 0xFF.
- **erase** <target-index>
Erases a region so that it becomes blank, i.e. all bytes are 0xFF.
- **info** <target-index>
Displays information about the Flash memory that holds a region.

- **program** <target-index> <filename>
Programs the specified bitstream (.BIT) file into a region so that the target FPGA is configured from a particular region at power-on or reset.
- **verify** <target-index> <filename>
Verifies that a region contains the specified bitstream (.BIT) file.

chkblank command

The **chkblank** command verifies that a target FPGA region is blank, i.e. all bytes are 0xFF, but does not modify the Flash memory bank. Following the command, an index of a target FPGA in the device must be specified. The index of the target FPGA is normally zero but may be nonzero in models with multiple target FPGAs.

For example, to blank-check the default region for target FPGA 0:

```
flash chkblank 0
```

erase command

The **erase** command erases a target FPGA region so that it becomes blank, i.e. all bytes are 0xFF. It automatically performs a blank-check after erasing. Following the command, an index of a target FPGA in the device must be specified. The index of the target FPGA is normally zero but may be nonzero in models with multiple target FPGAs.

For example, to erase the default region for target FPGA 0:

```
flash erase 0
```

info command

The **info** command displays information about the Flash memory and then exits, without doing anything else. Following the command, an index of a target FPGA in the device must be specified. The index of the target FPGA is normally zero but may be nonzero in models with multiple target FPGAs. For example:

```
flash info 0
```

program command

The **program** command programs a target FPGA region with the data in the specified bitstream (.BIT) file. Following the command, an index of a target FPGA in the device and the name of a bitstream (.BIT) filename must be specified. The index of the target FPGA is normally zero but may be nonzero in models with multiple target FPGAs.

If the device in the .BIT file does not match the target FPGA, this command fails with an error and does not program the target FPGA region, unless the **+force** option is passed. Verification is automatically performed after programming.

For example, to program the default region for target FPGA 0 with a bitstream file called **my_design.bit**:

```
flash program 0 /path/to/my_design.bit
```

verify command

The **verify** command verifies that a target FPGA region contains the data in the specified bitstream (.BIT) file, but does not modify the Flash memory bank. Following the command, an index of a target FPGA in the device and the name of a bitstream (.BIT) filename must be specified. The index of the target FPGA is normally zero but may be nonzero in models with multiple target FPGAs.

If the device in the .BIT file does not match the target FPGA, this command fails with an error unless the **+force**

option is passed. If discrepancies between the target FPGA region and the data in the .BIT file are found, they are displayed (up to a certain number of erroneous bytes), followed by a failure message.

For example, to verify that the default region for target FPGA 0 contains the data in a bitstream file called **my_design.bit**:

```
flash verify 0 /path/to/my_design.bit
```

Exit codes

When **FLASH** successfully executes the requested command, the exit code is 0. When an error occurs, one of the following exit codes is returned:

Symbolic name	Value	Meaning
EXIT_OK	0	Success.
EXIT_BAD_COMMAND_LINE	1	Illegal basic command-line syntax.
EXIT_BAD_RANGE	2	Value for -range option is not valid.
EXIT_UNRECOGNIZED_COMMAND	3	Command not recognized.
EXIT_INSUFFICIENT_ARGS	4	Too few positional arguments following command.
EXIT_ILLEGAL_FLAGS	5	An illegal combination of flags was passed.
EXIT_INVALID_RANGE	6	A Flash address range was invalid.
EXIT_ALLOCATION_FAILURE	7	A memory allocation failed.
EXIT_BLOCK_QUERY_FAILURE	8	Failed to query a Flash block (ADMXRC3_GetFlashBlockInfo failed).
EXIT_CREATE_FILE_ERROR	9	Failed to open file for writing readback data.
EXIT_WRITE_FILE_ERROR	10	Failed to write to readback data file.
EXIT_READ_FLASH_ERROR	11	Failed to read from Flash (ADMXRC3_ReadFlash failed).
EXIT_WRITE_FLASH_ERROR	12	Failed to write to Flash (ADMXRC3_WriteFlash failed).
EXIT_ERASE_FLASH_ERROR	13	Failed to erase Flash (ADMXRC3_EraseFlash failed).
EXIT_READ_BIT_ERROR	14	Failed to read .bit file.
EXIT_SANITY_ERROR	15	A sanity check failed.
EXIT_FPGA_MISMATCH	16	The FPGA identifier in the .bit file does not match the FPGA in the device.
EXIT_BOOT_FLAG_VERIFY_ERROR	17	Boot flag read back has incorrect value.
EXIT_DEVICE_OPEN_ERROR	18	Failed to open ADMXRC3 device.
EXIT_CARDINFO_ERROR	19	Failed to get information about ADMXRC3 device.
EXIT_FAILSAFE_NOT_SUPPORTED	20	Writing to failsafe region not supported for this ADMXRC3 device.
EXIT_ILLEGAL_TARGET_INDEX	21	Target FPGA index illegal for this ADMXRC3 device.
EXIT_UNSUPPORTED_MODEL	22	The ADMXRC3 device is an unsupported model.
EXIT_FPGAINFO_ERROR	23	Failed to get information about target FPGA.

Table 6 : Exit codes for FLASH utility (continued on next page)

Symbolic name	Value	Meaning
EXIT_FLASHINFO_ERROR	24	Failed to get information about Flash memory.
EXIT_ILLEGAL_REGION	25	Region index passed is illegal for this ADMXRC3 device.
EXIT_SET_CLOCK_FAILED	26	Failed to set LCLK clock generator to nominal frequency.
EXIT_NOT_BLANK	27	Blank check failed; region is not blank.
EXIT_VERIFY_FAILED	28	Verification failed; data read back did not match data written.
EXIT_NULL_FILENAME	29	Illegal NULL string passed for a filename.

Table 6 : Exit codes for FLASH utility


3.5.1 Region to address range mapping

WARNING

The **+failsafe**, **-region** and **-range** options must be used with care on models that feature a Virtex-6 target FPGA, because they can be used to overwrite the **failsafe region** of the Flash memory. The failsafe region is factory-programmed with a bitstream that protects against sub-micron effects that might otherwise degrade the performance of the target FPGA over time.

Xilinx answer record 35055* elaborates on protecting Virtex-6 GTX transceivers from performance degradation over time.

Alpha Data recommends that the failsafe region should not be erased or overwritten. If overwritten on a model that features a Virtex-6 target FPGA, the customer must ensure that it is written with a valid, known-good bitstream that satisfies the requirements for protecting the target FPGA from sub-micron effects.

Most of Alpha Data's reconfigurable computing cards have Flash memory capable of storing multiple **.bit** files, and are divided into two or more regions. The address map for each Flash memory bank, including information about regions, is presented in the [ADMXRC3 API Hardware Addendum](#) .


The default behaviour of the **chkblank**, **erase**, **program** and **verify** commands is to use the **default region** of the Flash memory bank, which varies between models. A region is identified by its zero-based index. For example, for the ADM-XRC-6T1, the default region is 0.

The **+failsafe** option modifies the behaviour of the **flash** utility so that it targets the **failsafe region**. This is a region of Flash, with a different index to that of the default region. For example, for the ADM-XRC-6T1, the failsafe region is 1. The failsafe region normally contains a factory-programmed bitstream that serves one of two purposes:

- In models which feature a dedicated, non-user-programmable PCIe interface chip, such as the ADM-XRC-6T1, the failsafe region is used to configure the target FPGA at power-on if a valid bitstream is not found in the default region. As noted in the warning above, this protects the target FPGA from sub-micron effects that might otherwise degrade the performance of the target FPGA over time.
- In models which feature a single FPGA that serves as both PCIe interface and target FPGA, such as the ADM-PCIE-7V3, the failsafe region can be selected by switches so that the FPGA is configured with the failsafe bitstream at power-on. This enables recovery from programming a "bad" **.bit** file into the default region. Here, a "bad" **.bit** file is defined to be one that does not include a working PCIe interface, thus preventing further Flash programming via PCIe.

The **-region** option supports models that have more than two regions, such as the ADM-PCIE-7V3. It must be followed by an argument that is the region index, e.g. **-region 0**. The **-region** option can be used to target the failsafe region or the default region (or any other region in models with more than two regions) by passing the

appropriate region index as the argument.

The final option that modifies the behaviour of **flash** utility is the **-range** option. This option overrides the program logic that maps a region index to a range of Flash memory addresses, and must be followed by an argument of the form **<address>,<length>**. This option can be used to achieve the same effect as **+failsafe** and **-region** options, provided that the user knows the correct address range to use. Refer to the [ADMXRC3 API Hardware Addendum](#)  for Flash memory bank address maps, by model.

Some examples of using the above three options follow:

- The following commands are all equivalent on the ADM-XRC-6T1, and perform a blank-check on the failsafe region (1), which should fail assuming that the factory-programmed bitstream is still present:

```
flash +failsafe chkblank 0
flash -region 1 chkblank 0
flash -range 2900000,1700000 chkblank 0
```
- The following commands are all equivalent on the ADM-PCIE-7V3, and write a bitstream into the default region (1):

```
flash program 0 /path/to/my_design.bit
flash -failsafe program 0 /path/to/my_design.bit
flash -region 1 program 0 /path/to/my_design.bit
flash -range 2000000,2000000 program 0 /path/to/my_design.bit
```
- On the ADM-PCIE-7V3, the following commands write and then verify a small compressed bitstream (must be less than or equal to 10 MiB) in the uppermost 10 MiB of region 3:

```
flash -range 7600000,A00000 program 0 /path/to/my_design.bit
flash -range 7600000,A00000 verify 0 /path/to/my_design.bit
```

3.6 INFO utility

Command line

```
info [option ...]
```

where the following options are accepted:



-flash	Causes Flash bank information not to be shown (default).
+flash	Causes Flash bank information to be shown.
-index <index>	Specifies the index of the card to open (default 0).
-io	Causes I/O module information not to be shown (default).
+io	Causes I/O module information to be shown.
-sensor	Causes sensor information not to be shown (default).
+sensor	Causes sensor information to be shown.
-sn <#>	Specifies the serial number of the card to open.

Because this utility does not attempt to change any state in the selected device, it need not be run with elevated privileges (i.e. as Administrator in Windows) or as root (in Linux).

Summary

Displays information about a reconfigurable computing device.



Description

The **INFO** utility demonstrates the use of most of the informational functions in the ADMXRC3 API. It uses [ADMXRC3_OpenEx](#)  to open a device in passive mode, meaning that an unprivileged user can successfully run it. The output consists of several sections, the first of which is obtained using [ADMXRC3_GetVersionInfo](#) .



```
API information
API library version      1.4.17
Driver version           1.4.17
```

The second section shows information obtained using [ADMXRC3_GetCardInfoEx](#) , and shows the information in the [ADMXRC3_CARD_INFOEX](#)  structure:

```
Card information
Model                ADM-VPX3-7V2
Serial number        200 (0xC8)
Number of programmable clocks 2
Number of DMA channels 4
Number of target FPGAs 1
Number of local bus windows 4
Number of sensors     23
Number of I/O module sites 1
Number of memory banks 4
Bank presence bitmap  0xF
```

The third section uses the **NumTargetFpga** member of the [ADMXRC3_CARD_INFOEX](#)  structure and [ADMXRC3_GetFpgaInfo](#)  to enumerate the target FPGAs in the device:

```
Target FPGA information
FPGA 0                7VX690TFFG1761-2I
```

The fourth section uses the **NumMemoryBank** member of the [ADMXRC3_CARD_INFOEX](#)  structure and [ADMXRC3_GetBankInfo](#)  to enumerate the memory banks (non-Flash) in the device:

```
Memory bank information
Bank 0                      SDRAM, DDR3, 262144(0x40000) kiW x 32 + 0 bits
                             303.0 MHz - 800.0 MHz
                             Connectivity mask 0x1
Bank 1                      SDRAM, DDR3, 262144(0x40000) kiW x 32 + 0 bits
                             303.0 MHz - 800.0 MHz
                             Connectivity mask 0x1
... (other memory banks) ...
```

The fourth section uses the **NumWindow** member of the [ADMXRC3_CARD_INFOEX](#) structure and [ADMXRC3_GetWindowInfo](#) to enumerate the memory access windows in the device:

```
Local bus window information
Window 0 (Target FPGA 0 pre Bus base      0xC0400000 size 0x400000
                             Local base    0x0 size 0x400000
                             Virtual size  0x400000
Window 1 (Target FPGA 0 non Bus base      0x64EC0000 size 0x400000
                             Local base    0x0 size 0x400000
                             Virtual size  0x400000
... (other windows) ...
```

The next section appears if the **+flash** option is passed on the command line. It uses the **NumFlashBank** member of the [ADMXRC3_CARD_INFOEX](#) structure and [ADMXRC3_GetFlashInfo](#) to enumerate the Flash memory banks in the device:

```
Flash bank information
Bank 0                      Numonyx Axcell P30 (Symm bl), 65536(0x10000) kiB
                             Useable area 0x1200000-0x3FFFFFFF
```

The next section appears if the **+io** option is passed on the command line. It uses the **NumModuleSite** member of the [ADMXRC3_CARD_INFOEX](#) structure and [ADMXRC3_GetModuleInfo](#) to enumerate the I/O module sites in the device and show what is fitted, if anything:

```
I/O module information
Module 0                      Product FMC-CLINK-MINI
                             Part number FMC-CLINK-MINI
                             Manufacturer Alpha Data
                             Serial number 112
                             Manufacture time 8995680 minutes since 00:00 1/1/1996
                             I/O voltage 1.80
                             Flags 0x0
```

The final section appears if the **+sensor** option is passed on the command line. It uses the **NumSensor** member of the [ADMXRC3_CARD_INFOEX](#) structure and [ADMXRC3_GetSensorInfo](#) to enumerate the sensors in the device:

```
Sensor information
Sensor 0                      12V VPX power rail
                             V, double, exponent 0, error 0.000
Sensor 1                      5V VPX power rail
                             V, double, exponent 0, error 0.000
... (other sensors) ...
```

Exit codes

When **INFO** runs successfully, the exit code is 0. When an error occurs, one of the following exit codes is returned:

Symbolic name	Value	Meaning
EXIT_OK	0	Success.
EXIT_BAD_COMMAND_LINE	1	Illegal basic command-line syntax.
EXIT_DEVICE_OPEN_ERROR	2	Failed to open ADMXRC3 device.

Table 7 : Exit codes for INFO utility

3.7 IPROG utility

Command line

```
iprog [option ...] abort      target-index
iprog [option ...] from-now  target-index address [delay-ms]
iprog [option ...] on-stop   target-index address [delay-ms]
iprog [option ...] status    target-index
```

where

<i>target-index</i>	is the index of a target FPGA.
<i>address</i>	specifies where in Flash memory the SelectMap data begins, and can be a region index, a Flash address (hex), or a WBSTAR register value. See below for details.
<i>delay-ms</i>	is the reconfiguration countdown time, in milliseconds (optional, default 30000).

and the following options are accepted:

-index <index>	Specifies the index of the card to open (default 0).
-sn <#>	Specifies the serial number of the card to open.
-verbose	Suppresses most informational messages (default).
+verbose	Displays informational messages.

The **IPROG** utility opens a reconfigurable computing device with intent to modify its state, and for this reason it must be run in an elevated command prompt (i.e. as Administrator in Windows) or as root (in Linux).

Summary

Initiates or aborts software-initiated reconfiguration of the target FPGA on models that support it.

Description

The **IPROG** utility has four commands:

- **abort** <target-index>
Aborts software-initiated reconfiguration of the target FPGA.
- **from-now** <target-index> <address> [delay-ms]
Schedules software-initiated reconfiguration to occur *delay-ms* milliseconds from when the command is issued.
- **on-stop** <target-index> <address> [delay-ms]
Schedules software-initiated reconfiguration to occur *delay-ms* milliseconds from when the device is stopped (i.e. by unloading ADB3 Driver).
- **status** <target-index>
Displays whether or not software-initiated reconfiguration is currently scheduled, and the remaining countdown time if scheduled.

abort command

The **abort** command aborts software-initiated reconfiguration of the target FPGA. The first argument is the index of the target FPGA for which reconfiguration is to be aborted.



For example, to abort software-initiated reconfiguration of target FPGA 0:

```
iprogram abort 0
```

from-now & on-stop commands

The **from-now** command schedules software-initiated reconfiguration to occur at a particular time from when the command is issued, whereas the **on-stop** command schedules software-initiated reconfiguration to occur at a particular time from when the device is stopped (i.e. the ADB3 Driver is unloaded). The **on-stop** command is generally preferable to **from-now** because the former more or less eliminates the risk that the target FPGA is reconfigured while the ADB3 Driver is communicating with it.

For both commands, the first argument is the index of the target FPGA to be reconfigured. This must be followed by a value that indicates from where in Flash memory the SelectMap data is obtained, which must be in one of the following forms:

- **address:<a>**
where <a> is a hexadecimal number representing the byte address in Flash memory at which the SelectMap data begins. Details of Flash memory maps, by model, are found in [ADMXRC3 API Hardware Addendum](#) .
- **region:<i>**
where <i> is the decimal zero-based index of a region of Flash memory, as described in [ADMXRC3 API Hardware Addendum](#) .
- **wbstar:<value>**
where <value> is a hexadecimal number which is the value for the WBSTAR register in the ICAP interface of a 7 Series FPGA. Refer to Xilinx UG470, "IPROG Reconfiguration" for further details.

The last argument, which is optional, is the countdown delay in milliseconds (default 30000). For the **from-now** command, it is the delay from issuing the command to when the FPGA is reconfigured. For the **on-stop** command, it is the delay from stopping the device (i.e. unloading the ADB3 Driver) to when the FPGA is reconfigured. The default of 30 seconds is intended to provide sufficient time to stop the device even in a system which is heavily loaded. For the **on-stop** command, a delay of 2000 milliseconds is expected to be safe, and a value of less than 500 milliseconds is not recommended except in a lightly-loaded system.

For example, to schedule software-initiated reconfiguration after 15 seconds, from the beginning of region 1 of the Flash memory on an ADM-PCIE-7V3, the following commands are all equivalent:

```
iprogram from-now 0 address:2000000 15000
iprogram from-now 0 region:1 15000
iprogram from-now 0 wbstar:60000000 15000
```

To schedule software-initiated reconfiguration after 2 seconds from device stop, from the middle of region 1 of the Flash memory on an ADM-PCIE-7V3, the following two commands are equivalent:

```
iprogram on-stop 0 address:3000000 2000
iprogram on-stop 0 wbstar:60800000 2000
```

status command

The **status** command displays whether or not software-initiated reconfiguration is currently scheduled and how much time remains for the countdown. The first argument is the index of the target FPGA for which status is to be requested.

For example, to request the status of software-initiated reconfiguration of target FPGA 0:

```
iprogram status 0
```

Exit codes

When **Iprog** successfully executes the requested command, the exit code is 0. When an error occurs, one of the following exit codes is returned:

Symbolic name	Value	Meaning
EXIT_OK	0	Success.
EXIT_BAD_COMMAND_LINE	1	Illegal basic command-line syntax.
EXIT_UNRECOGNIZED_COMMAND	2	Invalid command passed as 1st positional argument.
EXIT_INSUFFICIENT_ARGS	3	Not enough positional arguments after command.
EXIT_INVALID_WBSTAR	4	Invalid WBSTAR address.
EXIT_DEVICE_OPEN_ERROR	5	Failed to open ADMXRC3 device.
EXIT_ABORTIPROG_ERROR	6	Call to ADB3_AbortIprog failed.
EXIT_SCHEDULEIPROG_ERROR	7	Call to ADB3_ScheduleIprog failed.
EXIT_STATUSIPROG_ERROR	8	Call to ADB3_StatusIprog failed.
EXIT_UNSUPPORTED_MODEL	9	Iprog operations not supported for this model.
EXIT_ILLEGAL_TARGET_INDEX	10	Iprog operations not supported for the specified target FPGA index.
EXIT_ILLEGAL_REGION	11	Flash region index is not valid for this model & target FPGA index.
EXIT_INVALID_FLASH_ADDRESS	12	Flash address is not valid for this model & target FPGA index.

Table 8 : Exit codes for Iprog utility

3.8 LOADER utility

Command line

```
loader [option ...] <target FPGA index> <bitstream filename>
```

where the following options are accepted:

-index <index>	Specifies the index of the card to open (default 0).
-sn <#>	Specifies the serial number of the card to open.
-binary	Treat <bitstream filename> as a .bit file, and attempt to parse it accordingly.
+binary	Treat <bitstream filename> as a binary file containing only SelectMap data.
-checklink	Do not verify that communications with the target FPGA have been established before exiting.
+checklink	Verify that communications with the target FPGA have been established before exiting (default).

The **LOADER** utility opens a reconfigurable computing device with intent to modify its state, and for this reason it must be run in an elevated command prompt (i.e. as Administrator in Windows) or as root (in Linux).



Summary

Configures a target FPGA with a .bit file, and then exits.

Description

The **LOADER** utility configures the target FPGA, identified by <target FPGA index>, within a reconfigurable computing device with the bitstream file identified by <bitstream filename>, and then exits.

By default, **LOADER** expects <bitstream filename> to name a .bit file, i.e. a file generated by the Xilinx ISE or Vivado design tools, and attempts to parse the file accordingly. However, the **+binary** option causes **LOADER** to treat <bitstream filename> as a file containing raw SelectMap data. Such a file can be obtained in a number of ways, including a user-created program or by using the [BITSTRIP utility](#).

Because the **LOADER** utility uses [ADMXRC3_ConfigureFromFile](#)  or [ADMXRC3_ConfigureFromBuffer](#) , it normally checks that communications with the target FPGA have been established before exiting. Passing the **-checklink** option causes this check to be omitted, and is needed for an FPGA design that has no MPTL host interface, such as a stand-alone Ethernet design.

Exit codes

When **LOADER** runs successfully, the exit code is 0. When an error occurs, one of the following exit codes is returned:

Symbolic name	Value	Meaning
EXIT_OK	0	Success.
EXIT_BAD_COMMAND_LINE	1	Illegal basic command-line syntax.
EXIT_DEVICE_OPEN_ERROR	2	Failed to open ADMXRC3 device.
EXIT_STAT_FILE_ERROR	3	Failed to stat() .bit file.
EXIT_FILE_TOO_LARGE	4	.bit file too large to load into memory.
EXIT_ALLOCATION_FAILED	5	Failed to allocate buffer for firmware or board config data.
EXIT_FILE_OPEN_ERROR	6	Failed to open .bit file for reading.
EXIT_FILE_READ_ERROR	7	Failed to read .bit file.
EXIT_CONFIGURATION_FAILED	8	ADMXRC3_Configure{FromBuffer,FromFile} failed.

Table 9 : Exit codes for LOADER utility

3.9 MONITOR utility

Command line

```
monitor [option ...]
```

where the following options are accepted:




-index <index>	Specifies the index of the card to open (default 0).
-period <delay>	Specifies the update period, in seconds.
-repeat <n>	Specifies the number of updates to perform (default 0); a value of zero means "repeat for ever".
-sn <#>	Specifies the serial number of the card to open.

Summary

Displays readings from all sensors.

Description

The **MONITOR** utility repeatedly displays sensor readings in the command shell at the interval specified by the **-period** option. The number of updates to perform before terminating can be specified on the command line using the **-repeat** option, but by default, the program runs until interrupted with CTRL-C.

It makes use of the [ADMXRC3_GetSensorInfo](#)  and [ADMXRC3_ReadSensor](#)  functions from the ADMXRC3 API, and because it opens a device in passive mode using [ADMXRC3_OpenEx](#) , it can run alongside other reconfigurable computing applications without disturbing them.

The output looks like this:

```
Model:                257 (0x101) => ADM-XRC-6TL
Serial number:        101 (0x65)
Number of sensors:    10
  Sensor 0            1V supply rail: 0.987000 V
  Sensor 1            1.5V supply rail: 1.509186 V
  Sensor 2            1.8V supply rail: 1.803192 V
  Sensor 3            2.5V supply rail: 2.508896 V
  Sensor 4            3.3V supply rail: 3.268082 V
  Sensor 5            5V supply rail: 5.017990 V
  Sensor 6            XMC variable power rail: 12.000000 V
  Sensor 7            XRM I/O voltage: 2.495712 V
  Sensor 8            LM87 internal temperature: 49.000000 deg C
  Sensor 9            Target FPGA temperature: 57.000000 deg C
```

Exit codes

When **MONITOR** runs successfully, the exit code is 0. When an error occurs, one of the following exit codes is returned:

Symbolic name	Value	Meaning
EXIT_OK	0	Success.
EXIT_BAD_COMMAND_LINE	1	Illegal basic command-line syntax.
EXIT_DEVICE_OPEN_ERROR	2	Failed to open ADMXRC3 device.
EXIT_CARDINFO_ERROR	3	Failed to get information about ADMXRC3 device.
EXIT_NO_SENSORS_FOUND	4	No sensors in the selected device.

Table 10 : Exit codes for MONITOR utility

3.10 SYSMON utility

Command line




`sysmon`

Summary

Utility presenting device information and hardware sensors in a graphical user interface.

Description

The **SYSMON** utility combines the information shown by the INFO and MONITOR utilities with a graphical user interface. Its main function is graphical display of hardware sensor data, and it can be minimized to the notification area of the desktop (the "System Tray" in Windows) in order to run unobtrusively.

It makes use of the [ADMXRC3_GetSensorInfo](#)  and [ADMXRC3_ReadSensor](#)  functions from the ADMXRC3 API, and because it initially opens a device in passive mode using [ADMXRC3_OpenEx](#) , it can run alongside other reconfigurable computing applications without disturbing them.

The user interface of the Linux version of SYSMON is as follows, upon starting the utility:

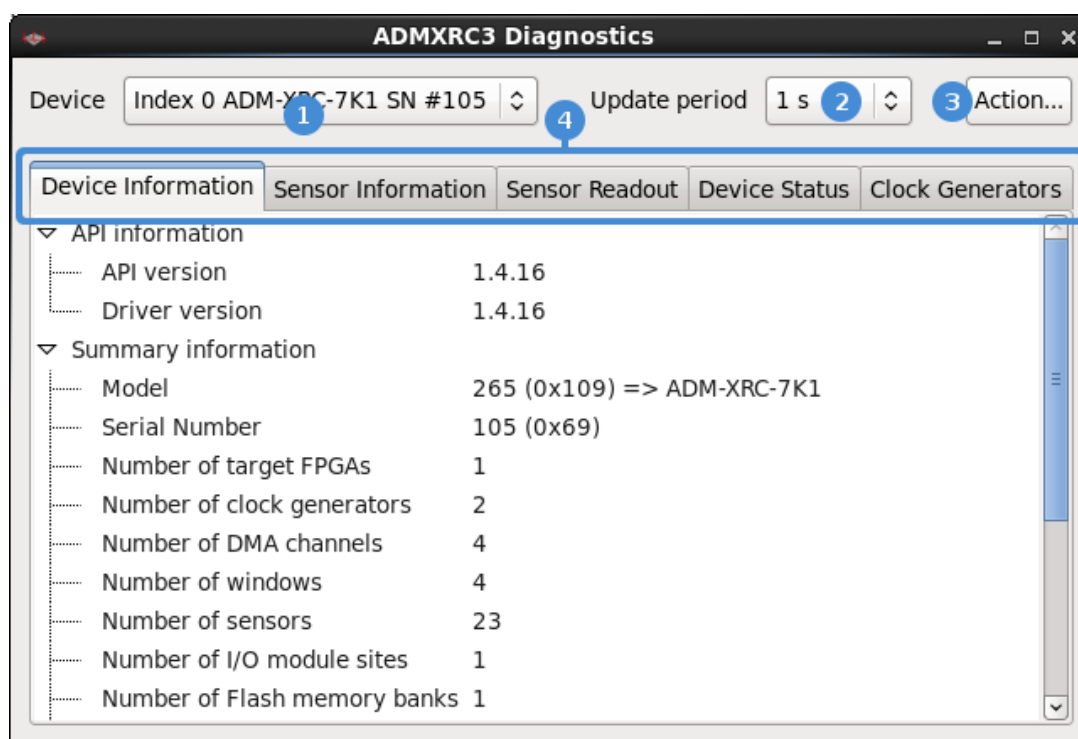


Figure 2 : SYSMON user interface

Referring to [Figure 2](#), the user interface elements are as follows:

- 1 A combo box that specifies which reconfigurable computing device to use.
- 2 A combo box that selects the time interval between sensor readings.
- 3 A button that reveals the **Action** menu when clicked. The **Action** menu allows sensor data logging as described in [Section 3.10.6](#) below. The Windows version of SYSMON does not have this button, but instead hosts equivalent functionality via the system menu.
- 4 A tab control whose tabs are as follows:
 - The [device information tab](#) shows information about the currently selected device.

- The [sensor information tab](#) shows information about the available sensors in the currently selected device.
- The [sensor readout tab](#) displays sensor data graphically in up to four 'scopes'.
- The [device status tab](#) displays any error conditions in the currently selected device, and permits them to be cleared if the user that launched SYSMON has the necessary privileges.
- The [clock generator tab](#) displays clock generator frequencies for the currently selected device, and permits them to be changed if the user that launched SYSMON has the necessary privileges.

When minimized (item 5), **sysmon** appears in the notification area of the desktop:

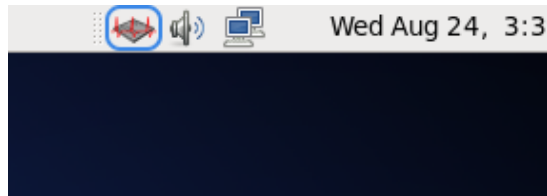


Figure 3 : SYSMON notification area icon

The icon shown in the notification area has a context menu activated by a right-click, and this can be used to restore the application to the desktop, as well as offering the same logging functions as the **Action** menu. Refer to [Section 3.10.6](#) for a description of data logging.

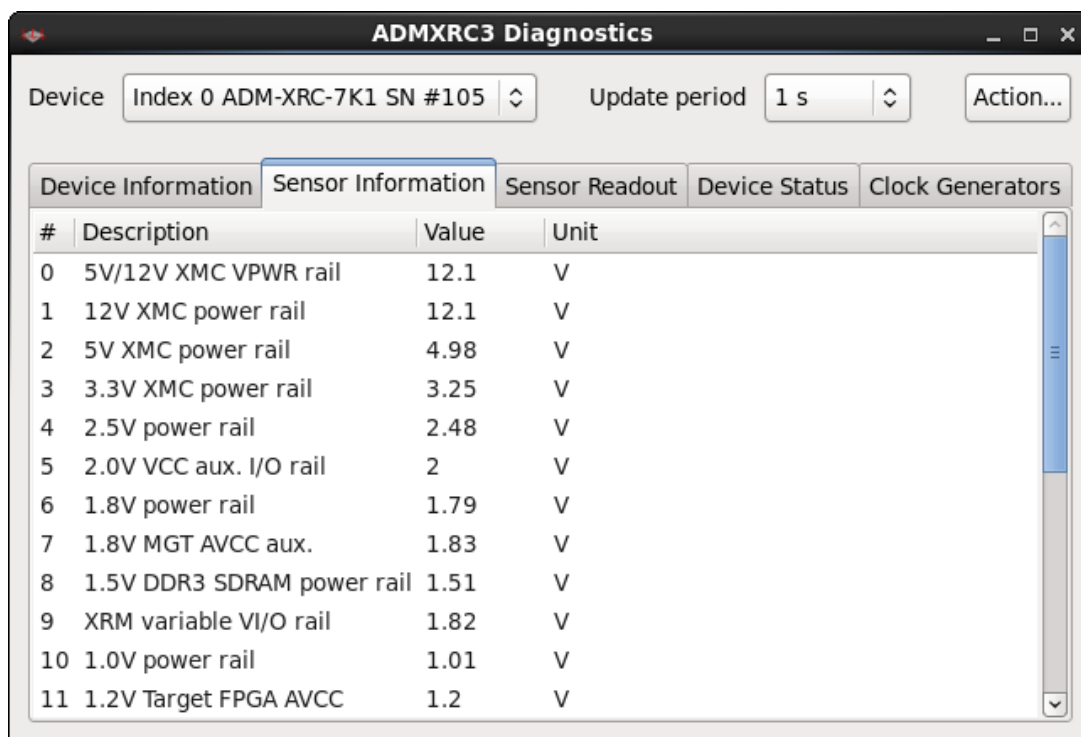
To actually close the application as opposed to minimizing it, click the close button of the window.

3.10.1 SYSMON device information tab

The set of information shown in the device information tab is approximately the same as that shown by the command-line [INFO](#) utility, but with a collapsible tree structure.

3.10.2 SYSMON sensor information tab

The sensor information tab is a tabular view of the available sensors, including the current reading for each sensor:



#	Description	Value	Unit
0	5V/12V XMC VPWR rail	12.1	V
1	12V XMC power rail	12.1	V
2	5V XMC power rail	4.98	V
3	3.3V XMC power rail	3.25	V
4	2.5V power rail	2.48	V
5	2.0V VCC aux. I/O rail	2	V
6	1.8V power rail	1.79	V
7	1.8V MGT AVCC aux.	1.83	V
8	1.5V DDR3 SDRAM power rail	1.51	V
9	XRM variable VI/O rail	1.82	V
10	1.0V power rail	1.01	V
11	1.2V Target FPGA AVCC	1.2	V

Figure 4 : SYSMON sensor information tab

3.10.3 SYSMON sensor readout tab

The sensor readout tab displays sensor readings in graphical form:

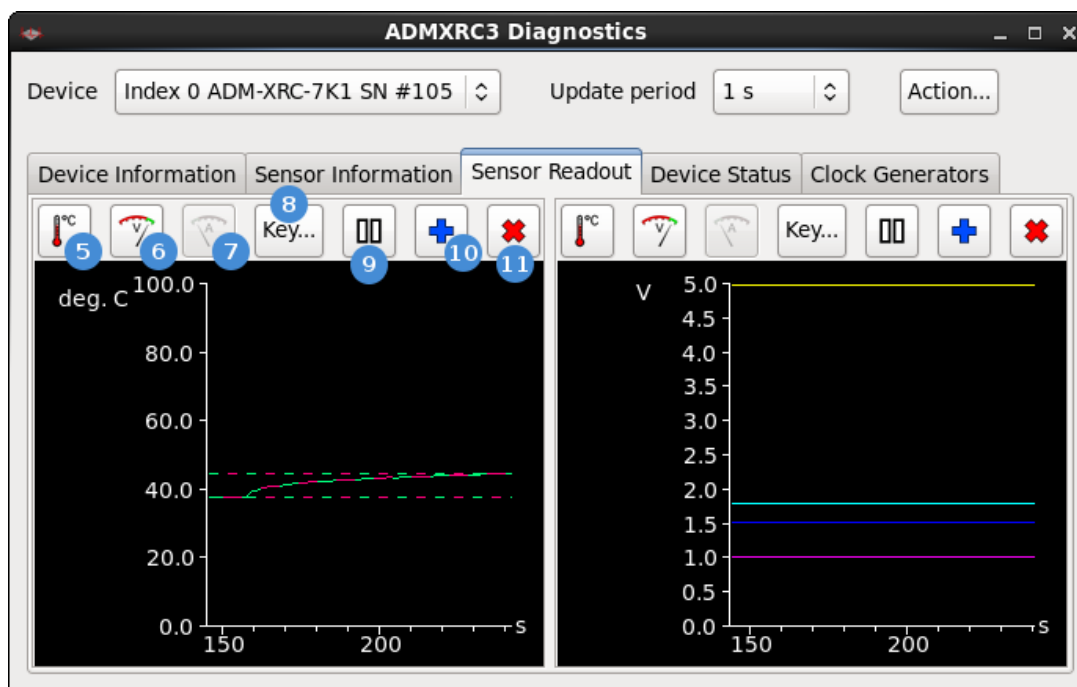


Figure 5 : SYSMON sensor readout tab

Initially, the 'scope is empty and displays no sensors. The above figure shows two scopes, one showing temperatures and the other showing voltages. The user interface elements of the 'scope toolbar are as follows:

- 5 The temperature button sets the 'scope to display all temperature sensors in the device, and starts updates.
- 6 The voltage button sets the 'scope to display all voltage sensors in the device, and starts updates.
- 7 The current button sets the 'scope to display all current sensors in the device, and starts updates.
- 8 Mouse over the key to see which sensor corresponds to which coloured trace.
- 9 The pause / resume button can be used to pause and resume update of the 'scope.
- 10 A button that adds another 'scope when clicked, to a maximum of 4, so that various types of sensor can be viewed at the same time.
- 11 A button that destroys a 'scope when clicked. If there is only one 'scope, the button is disabled.

3.10.4 SYSMON device status tab

The device status tab shows any error conditions that are present for a device, and allows the user to clear error conditions, provided that he or she has sufficient privileges:

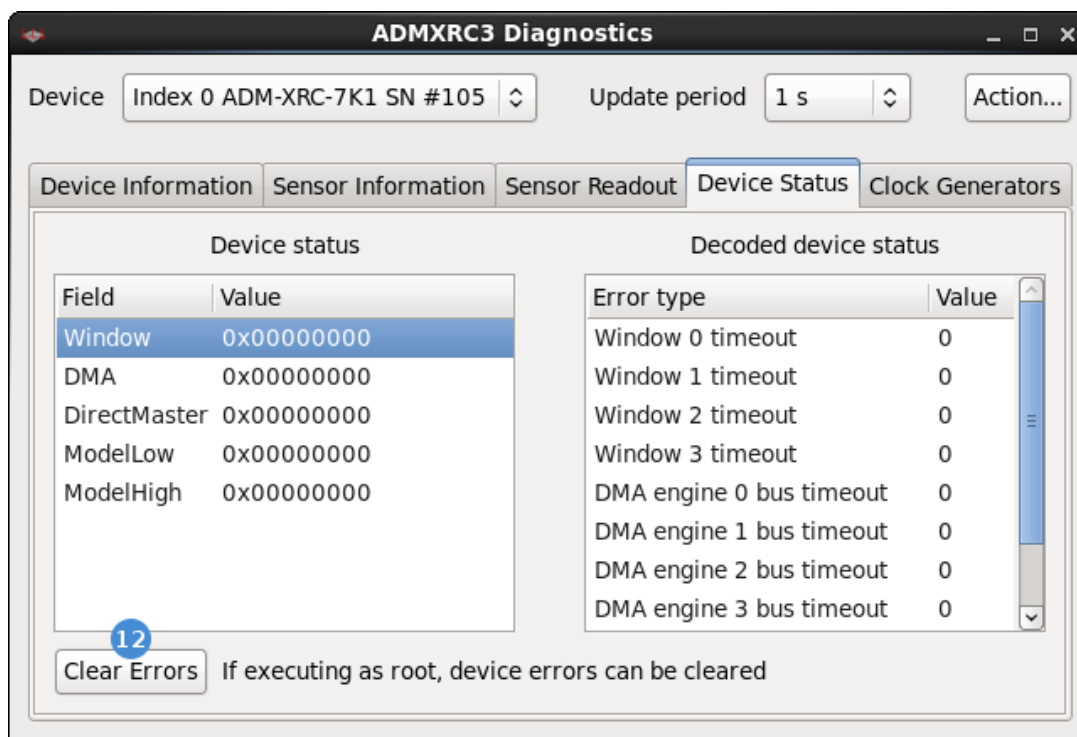


Figure 6 : SYSMON device status tab

The **Clear Errors** button, labeled 12 in Figure 6, allows device errors to be cleared, but in order for this button to be functional, **SYSMON** must be launched by a user with sufficient privileges to reopen the device in active mode. This works follows:

- In Linux, **SYSMON** must be launched by a user that has privileges to reopen the device in active mode. The **Clear Errors** button will then be functional.
In most cases, this means that the user must be **root**, but if the system administrator has customized the **udev** rules file for the **ADB3 Driver**, it is possible for non-root users to have privileges to reopen the device in active mode.
- In versions of Windows without User Account Control (UAC), i.e. Windows XP and earlier, **SYSMON** must be launched by a user in the Administrators group. The **Clear Errors** button will then be functional.
- In versions of Windows with User Account Control (UAC), i.e. Windows Vista and earlier, **SYSMON** must be launched by a user in the Administrators group. If the **Clear Errors** button is clicked and **SYSMON** is not running elevated, **SYSMON** re-launches itself elevated (this will incur a UAC prompt). The **Clear Errors** button will then be functional.

Alternatively, launch **SYSMON** elevated by right-clicking on its shortcut or executable file and selecting "Run as administrator". The **Clear Errors** button will then be functional.

3.10.5 SYSMON clock generator tab

The clock generator tab shows current frequencies of clock generators, and allows the user to change their frequencies, provided that he or she has sufficient privileges:

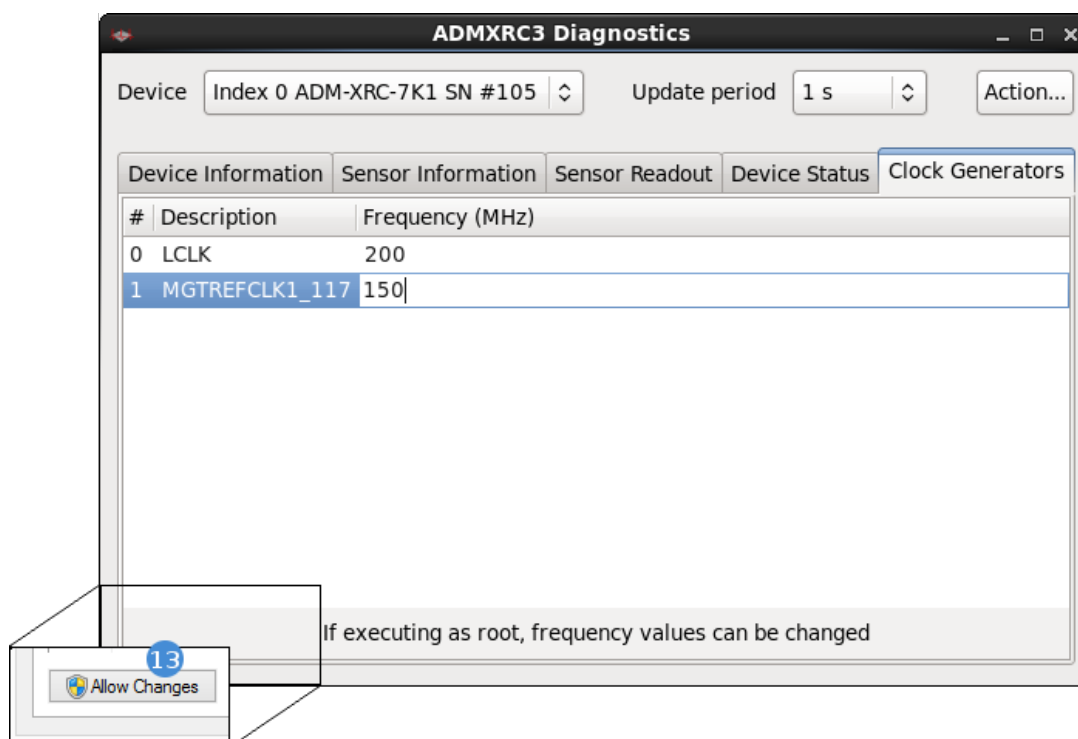


Figure 7 : SYSMON clock generator tab

In Linux, provided that **SYSMON** has been launched by a user with privileges to reopen the device in active mode, frequencies can be changed by clicking on them and entering a new frequency. In most cases, this means that the user must be **root**, but if the system administrator has customized the **udev** rules file for the **ADB3 Driver**, it is possible for non-root users to have privileges to reopen the device in active mode.

In Windows, provided that **SYSMON** is executing elevated so that it can reopen the device in active mode, frequencies can be double-clicked and a new frequency entered in order to change them. This works as follows:

- In versions of Windows without User Account Control (UAC), i.e. Windows XP and earlier, **SYSMON** must be launched by a user in the Administrators group, which means that it runs elevated. This is sufficient to allow frequencies to be changed.
- In versions of Windows with User Account Control (UAC), i.e. Windows Vista and earlier, **SYSMON** must be launched by a user in the Administrators group, which enables the **Allow Changes** button (labeled 13 in Figure 7). Clicking **Allow Changes** causes **SYSMON** to re-launch itself elevated. Once **SYSMON** is running elevated, frequencies can be changed.

Alternatively, launch **SYSMON** elevated by right-clicking on its shortcut or executable file and selecting "Run as administrator". This will permit frequencies to be changed.

3.10.6 SYSMON sensor data logging

In Linux, SYSMON can log sensor data over an arbitrary time period via the **Action** menu:

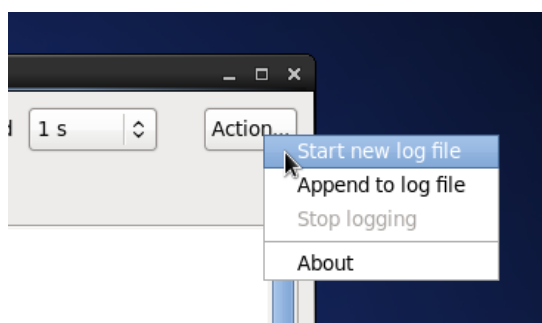


Figure 8 : SYSMON Action menu in Linux

In Windows, the **Action** button is not present, and the **Action** menu items are located in the system menu:

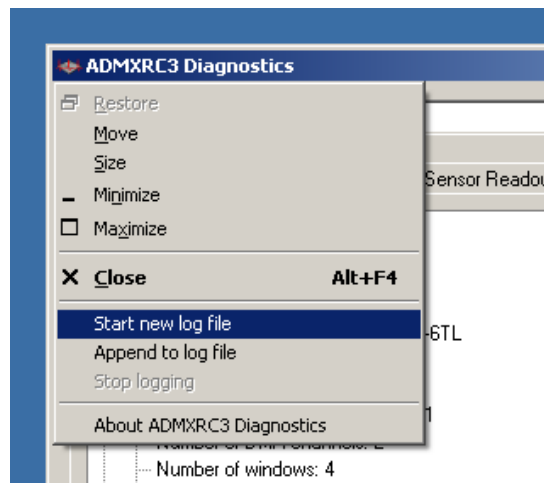


Figure 9 : SYSMON Action menu in Windows

Data logging works as follows:

- The **Start new log file** option prompts for a filename into which sensor data is to be logged. If a file of that name already exists, it will be overwritten.
- The **Append to log file** option prompts for a filename into which sensor data is to be logged, but unlike **Start new log file**, if a file of that name already exists, new data will be appended to it.
- The **Stop logging** option is only enabled after logging has successfully been started using **Start new log file** or **Append to log file**, and causes SYSMON to cease logging data.

The files created are in comma-separated value (CSV) format (some rows and columns deleted for brevity):

```
START,11:59:07 23 Aug 2011
COMMENT,MODEL,SERIAL#
DEVICE,ADM-XRC-6TL,102
COMMENT,SENSOR#,Description,Unit
SENSOR,0,1V supply rail,V
SENSOR,1,1.5V supply rail,V
SENSOR,2,1.8V supply rail,V
SENSOR,3,2.5V supply rail,V
...
SENSOR,12,Bridge VCCAUX,V
COMMENT,TIMESTAMP,1V supply rail,1.5V supply rail,1.8V supply rail,2.5V supply rail,...
COMMENT,ms,V,V,V,V,...
DATA,583,0.987000,1.509186,1.812988,2.508896,...
DATA,1584,0.987000,1.509186,1.812988,2.508896,...
DATA,2645,0.987000,1.509186,1.812988,2.508896,...
DATA,3646,0.987000,1.509186,1.812988,2.508896,...
...
DATA,13661,0.987000,1.509186,1.812988,2.508896,...
DATA,14663,0.987000,1.509186,1.812988,2.508896,...
STOP,11:59:22 23 Aug 2011
```

The string in column 1 of each row indicates what information a row contains:

- **START** signifies the start of a logging session, in case the file contains multiple sessions that were obtained using the **Append to log file** option.
- **STOP** signifies the end of a logging session, in case the file contains multiple sessions that were obtained using the **Append to log file** option.
- **COMMENT** signifies a comment, for the benefit of human readers, and can be filtered out by a program that reads the file.
- **DEVICE** identifies the model and serial number, in the second and third cells respectively, of the physical card from which the data was collected.

- **SENSOR** signifies information about a sensor. The second cell is the sensor index, the third cell is the sensor's description and the fourth cell is the unit for that sensor.
- **DATA** signifies a set of sensor readings at a given instant. The second cell is a timestamp, in milliseconds, relative to the time and date in the **START** row. The third and subsequent cells are individual sensor values, where the third cell corresponds to the **SENSOR** row whose sensor index is 0, the fourth cell corresponds to the **SENSOR** row whose sensor index is 1 etc.

3.11 VPD utility

Command line

```
vpd [option ...] fb address n [data]
vpd [option ...] fw address n [data]
vpd [option ...] fd address n [data]
vpd [option ...] fq address n [data]
vpd [option ...] fs address n [string]
vpd [option ...] rb address [n]
vpd [option ...] rw address [n]
vpd [option ...] rd address [n]
vpd [option ...] rq address [n]
vpd [option ...] wb address [n] [data ...]
vpd [option ...] ww address [n] [data ...]
vpd [option ...] wd address [n] [data ...]
vpd [option ...] wq address [n] [data ...]
vpd [option ...] ws address [n] [string ...]
```

where

<i>address</i>	is the address in VPD memory at which to begin reading or writing.
<i>n</i>	is the number of bytes to read or write.
<i>data</i>	is a numeric data item, valid for fill and write commands.
<i>string</i>	is a string data item, valid for fill and write commands.

and the following options are accepted:

-index <index>	Specifies the index of the card to open (default 0).
-sn <#>	Specifies the serial number of the card to open.
-be	Causes the data to be read or written to be treated as little-endian (default).
+be	Causes the data to be read or written to be treated as big-endian.
-hex	Causes numeric data values to be interpreted as decimal unless prefixed by '0x' (default).
+hex	Causes numeric data values to be interpreted as hexadecimal always.

The **VPD** utility opens a reconfigurable computing device with intent to modify its state, and for this reason it must be run in an elevated command prompt (i.e. as Administrator in Windows) or as root (in Linux).


Summary

Displays data read from Vital Product Data (VPD) memory, or writes data to VPD memory. VPD memory contains information about a reconfigurable computing card, such as the type of FPGA fitted, memory bank sizes etc.

Description

VPD write-protection mechanisms


The **VPD** utility must be used with care, particularly when using its capability to write or fill VPD memory. Corrupting the VPD of a reconfigurable computing card can impair its functionality until the VPD is restored to its correct values.

To avoid corrupting VPD, please ensure that you are aware of the address map of VPD Space for the particular reconfigurable computing card in use. This information is provided by [ADMXRC3 API Hardware Addendum](#) .

Writing to VPD writes requires a software enable to be activated in the ADB3 Driver. Additionally, on certain models in Alpha Data's range of reconfigurable computing hardware, a card must be put into **Service Mode** before VPD memory can be accessed. For further details, refer to [Section 3.11.1](#) below.

The **VPD** utility operates in one of three modes:

- Filling a region of VPD memory with a value or string; for this mode, use the **fb**, **fw**, **fd**, **fq** or **fs** commands.
- Reading data from VPD memory and displaying it; for this mode, use the **rb**, **rw**, **rd** or **rq** commands.
- Writing numeric or string data to a region of VPD memory; for this mode, use the **wb**, **ww**, **wd**, **wq** or **ws** commands.

The address space accessed by the VPD utility is called **VPD Space**, and the address map of VPD Space is model-specific. Details of address maps of VPD Space for supported models are given in [ADMXRC3 API Hardware Addendum](#) .

The option **+be** may be specified, before the command. This causes the **VPD** utility to use big-endian byte ordering convention as opposed to little-endian (the default).

Fill commands

When filling a region of VPD memory with data, the fill command specifies whether the data is numeric or string data. In the case of numeric data, the command also implies the word width of the data. The available fill commands are:

- **fb**
Fill value is a byte (8-bit).
- **fw**
Fill value is a word (16-bit).
- **fd**
Fill value is a doubleword (32-bit).
- **fq**
Fill value is a quadword (64-bit).
- **fs**
Fill value is an ASCII string (8-bit characters).

The next 3 arguments after the fill command must be:

- (a) *address* - the byte address within VPD memory at which to begin filling.
- (b) *n* - byte count; the number of bytes of VPD memory to fill.
- (c) *data* or *string* - the numeric or string value to place in the specified region of VPD memory.

If the command is **fs** and the string value is shorter than the byte count *n*, the string is repeated until the byte count is satisfied. If the string is longer than the byte count *n*, only the first *n* characters are used. If a string contains spaces, it must be quoted on the command line so that it is not interpreted by the shell as two or more separate arguments.

For the numeric fill commands **fb**, **fw**, **fd** and **fq**, the numeric value is repeated until the byte count is satisfied.

Read commands

The read command implies the word width used for displaying the data:

- **rb**
Byte (8-bit) reads; data is displayed as bytes.
- **rw**
Word (16-bit) reads; data is displayed as words.
- **rd**
Doubleword (32-bit) reads; data is displayed as doublewords.
- **rq**
Quadword (64-bit) reads; data is displayed as quadwords.

After the read command, an address must be supplied, which specifies where in VPD memory to begin reading. An optional length parameter, in bytes, can also be supplied. If omitted, the length is equal to the word width implied by the read command. If present, the length parameter specifies how many bytes to read and display. The length should be an integer multiple of the width; if not, the length is rounded down.

Write commands

The write command specifies whether the data is numeric or string data. In the case of numeric data, the command also implies the word width of the data. The available write commands are:

- **wb**
Data is written as bytes (8-bit).
- **ww**
Data is written as words (16-bit).
- **wd**
Data is written as doublewords (32-bit).
- **wq**
Data is written as quadwords (64-bit).
- **ws**
Data is supplied as one or more ASCII strings (8-bit characters).

After the write command, an address must be supplied, which specifies where in VPD memory to begin writing data. An optional length parameter, in bytes, can also be supplied. If omitted, the length is equal to the word width implied by the write command. If present, the length parameter specifies how many bytes to write. The length should be an integer multiple of the width; if not, the length is rounded down.

The program obtains the values to be written in two ways: from any additional parameters on the command line after the length parameter, and then from the standard input stream (stdin). This works as follows:

- (1) Any remaining command line arguments, if present after the length parameter, are interpreted as data values to be written. Numeric values are assumed to be of the word width implied by the command parameter. As each value is written to VPD memory, the address is incremented. If there are enough values passed on the command line to satisfy the byte count, the program terminates.
- (2) If there are insufficient data values passed on the command line, the program waits for values to be entered on the standard input stream. Numeric values entered this way are also assumed to be of the word width implied by the command. As each value is written to VPD memory, the address is incremented. When the entire byte count that was specified in the length parameter has been satisfied or end-of-file is encountered, the program terminates.

Example session

The following session was captured under Linux using an ADM-XRC-6T1. The base address 0x100000 is used because that is the VPD-space address of the user-definable area of VPD memory in the ADM-XRC-6T1.

```
# ./vpd rb 0x100000 0x60
Dump of VPD at 0x00100000 + 96(0x60) bytes:
0x00100000: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
0x00100010: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
0x00100020: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
0x00100030: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
0x00100040: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
0x00100050: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
# ./vpd fs 0x100008 20 "hello world!"
# ./vpd wd 0x100020 12
0x00100020: 0xdeadbeef
0x00100024: 0xcafeface
0x00100028: 0x12345678
# ./vpd fw 0x100031 10 0xa55a
# ./vpd rb 0x100000 0x60
Dump of VPD at 0x00100000 + 96(0x60) bytes:
0x00100000: FF FF FF FF FF FF FF 68 65 6C 6C 6F 20 77 6F .....hello wo
0x00100010: 72 6C 64 21 00 68 65 6C 6C 6F 20 77 FF FF FF FF rld!.hello w....
0x00100020: EF BE AD DE CE FA FE CA 78 56 34 12 FF FF FF FF .???.?..?xV4.....
0x00100030: FF 5A A5 5A A5 5A A5 5A A5 5A A5 5A FF FF FF FF .Z?Z?Z?Z?Z?.....
0x00100040: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
0x00100050: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
```

Remarks

When entering data for fill or write commands, values are expressed in decimal by default. To express data as hexadecimal, prefix it with '0x' or use the **+hex** option.

Exit codes

When **VPD** successfully executes the requested command, the exit code is 0. When an error occurs, one of the following exit codes is returned:

Symbolic name	Value	Meaning
EXIT_OK	0	Success.
EXIT_BAD_COMMAND_LINE	1	Illegal basic command-line syntax.
EXIT_UNRECOGNIZED_COMMAND	2	Invalid command passed as 1st positional argument.
EXIT_ALLOCATION_FAILURE	3	A memory allocation failed.
EXIT_DEVICE_OPEN_ERROR	4	Failed to open ADMXRC3 device.
EXIT_READVPD_ERROR	5	A call to ADMXRC3_ReadVPD failed.
EXIT_WRITEVPD_ERROR	6	A call to ADMXRC3_WriteVPD failed.

Table 11 : Exit codes for VPD utility

3.11.1 VPD write-protection mechanisms

In order to be able to write to VPD memory, a software protection mechanism must be disabled:

- In Windows, the registry value **EnableVpdWrite** must be set to 1. The full path of the registry value is:
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\adb3\Parameters\EnableVpdWrite
A change to this value takes immediate effect, and it is **not** necessary to reboot the machine or to disable and re-enable devices using Windows Device Manager.
NOTE: Each time the ADB3 Driver is installed or reinstalled, this value is set to 0.
- In Linux, the ADB3 Driver must be started with the **EnableVpdWrite** parameter set to 1. For example:
modprobe adb3 EnableVpdWrite=1

When omitted from the **modprobe** command line, **EnableVpdWrite** defaults to 0.

As well as the software protection mechanism, some models in Alpha Data's range of reconfigurable computing hardware must be in **Service Mode** in order for any access to VPD memory (whether reading or writing) to succeed. This applies to **all** models that have a switch setting for **Service Mode**. Please contact the User Manual for your reconfigurable computing hardware in order to determine whether or not this applies.

Appendix A: AVR2UTIL clock generator indices

A.1 ADM-XRC-KU1

In the ADM-XRC-KU1, the frequencies of clock generators with indices 1 and 3 may be overridden using the **setclknv** command, whereas the clock generators with indices 0 and 2 may not (because their frequencies must be fixed in order for the board to function correctly).

clockgen-index	Net(s) [1]	Purpose	Factory default (MHz)	ADMXRC3 API index [2]	Note
0	REFCLK250M_N0 REFCLK250M_N1	MPTL reference clock	250	N/A	[3]
1	PROGCLK_N0 PROGCLK_N1 PROGCLK_N2	Reference clock for user-definable MGTs	156.25	1	
2	REFCLK300M_N0 REFCLK300M_N1 REFCLK300M_N2 REFCLK300M_N3	Reference clock for DDR4 SDRAM and other logic	300	N/A	[3]
3	FABRIC_CLK_N	General purpose clock	300	2	

Table 12 : AVR2UTIL clock generator indices (ADM-XRC-KU1)

Note:

- [1] For differential clocks, only the negative side of a differential pair is listed.
- [2] This is the clock generator index used in calls such as **ADMXRC3_SetClockFrequency**.
- [3] Not user-programmable. Attempting to set an override frequency using **AVR2UTIL** will fail with exit code 103. Not exposed by ADMXRC3 API.

A.2 ADM-PCIE-8V3

In the ADM-PCIE-8V3, the frequencies of clock generators with indices 0, 1 and 2 may be overridden using the **setclknv** command, whereas the clock generator with index 3 may not (because its frequency must be fixed in order for the board to function correctly).

clockgen-index	Net(s) [1]	Purpose	Factory default (MHz)	ADMXRC3 API index [2]	Note
0	GTY_CLK_0B_N GTY_CLK_0C_N	QSFP+ 0 reference clock QSFP+ 1 reference clock	161.1328125	0	
1	GTY_CLK_1B_N GTY_CLK_1C_N	FireFly 0 reference clock FireFly 1 reference clock	161.1328125	1	
2	MEM_CLK_0_N MEM_CLK_1_N	Reference clock for DDR4 SDRAM	300	2	
3	FABRIC_CLK_N	General purpose clock	300	N/A	[3]

Table 13 : AVR2UTIL clock generator indices (ADM-PCIE-8V3)

Note:

- [1] For differential clocks, only the negative side of a differential pair is listed.
- [2] This is the clock generator index used in calls such as **ADMXRC3_SetClockFrequency**.
- [3] Not user-programmable. Attempting to set an override frequency using **AVR2UTIL** will fail with exit code 103. Not exposed by ADMXRC3 API.

A.3 ADM-PCIE-8K5

In the ADM-PCIE-8K5, the frequencies of all four clock generators, with indices 0 to 3, may be overridden using the **setclknv** command.

clockgen-index	Net(s) [1]	Purpose	Factory default (MHz)	ADMXRC3 API index [2]	Note
0	GTY_CLK_0_N	SFP+ 0 reference clock	156.25	0	
1	GTY_CLK_1_N	SFP+ 1 reference clock	156.25	1	
2	MEM_CLK_0_N MEM_CLK_1_N	Reference clock for DDR4 SDRAM	300	2	
3	GTH_CLK_2_N	FireFly 1 reference clock	156.25	3	

Table 14 : AVR2UTIL clock generator indices (ADM-PCIE-8K5)

Note:

- [1] For differential clocks, only the negative side of a differential pair is listed.
- [2] This is the clock generator index used in calls such as **ADMXRC3_SetClockFrequency**.

Revision History

Date	Revision	Nature of change
12 Aug 2015	1.0	Initial version.
16 May 2016	1.1	Documented new utility: avr2util. Documented support for new models: ADM-XRC-KU1, ADM-PCIE-8V3, ADM-PCIE-8K5.
30 Aug 2016	1.2	General overhaul to correct typographical errors Documented USB communication support in AVR2UTIL.
3 Mar 2017	1.3	Documented new commands in AVR2UTIL 2.5.0: i2c-read-to-file, i2c-verify-from-file, i2c-write-from-file, i2c-read, i2c-write, save-brdcfg, save-firmware, save-vpd, display-vpd, display-vpd-raw, display-sensors, display-sensors-raw, override-sensor, release-sensor
12 Jun 2017	1.4	Documented new commands in AVR2UTIL 2.7.1: getclk, setclk, spi-info, spi-raw