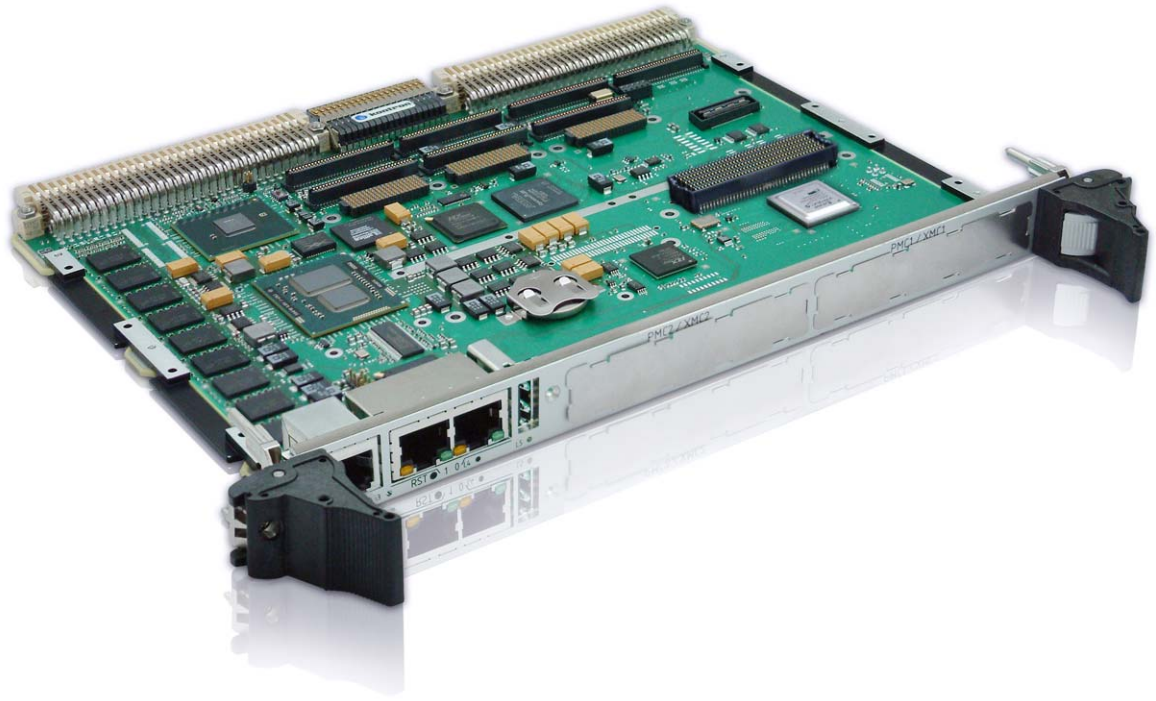
» VM6050 «



RHEL 6.4 BSP Release Notes on VM6050 Version 1.0 - ID 14276

SD.DT.G42-0e - November 2014

Revision History

Publication Title:	RHEL 6.4 BSP on VM6050	
Doc. ID:	SD.DT.G42-0e	
Rev.	Brief Description of Changes	Date of Issue
0e	Initial Version	11-2014

Proprietary Note

This document contains information proprietary to Kontron. It may not be copied or transmitted by any means, disclosed to others, or stored in any retrieval system or media without the prior written consent of Kontron or one of its authorized agents.

The information contained in this document is, to the best of our knowledge, entirely correct. However, Kontron cannot accept liability for any inaccuracies or the consequences thereof, or for any liability arising from the use or application of any circuit, product, or example shown in this document.

Kontron reserves the right to change, modify, or improve this document or the product described herein, as seen fit by Kontron without further notice.

Trademarks

This document may include names, company logos and trademarks, which are registered trademarks and, therefore, proprietary to their respective owners.

Environmental Protection Statement

This product has been manufactured to satisfy environmental protection requirements where possible. Many of the components used (structural parts, printed circuit boards, connectors, batteries, etc.) are capable of being recycled.

Final disposition of this product after its service life must be accomplished in accordance with applicable country, state, or local laws or regulations.

Environmental protection is a high priority with Kontron.

Kontron follows the DEEE/WEEE directive.

You are encouraged to return our products for proper disposal.

The Waste Electrical and Electronic Equipment (WEEE) Directive aims to:

- reduce waste arising from electrical and electronic equipment (EEE)
- make producers of EEE responsible for the environmental impact of their products, especially when they become waste
- encourage separate collection and subsequent treatment, reuse, recovery, recycling and sound environmental disposal of EEE
- improve the environmental performance of all those involved during the lifecycle of EEE

Conventions

This guide uses several types of notice: Note, Caution, ESD.

Note: this notice calls attention to important features or instructions.


Caution: this notice alert you to system damage, loss of data, or risk of personal injury.


ESD: This banner indicates an Electrostatic Sensitive Device.

All numbers are expressed in decimal, except addresses and memory or register data, which are expressed in hexadecimal. The prefix `0x' shows a hexadecimal number, following the `C' programming language convention.

The multipliers `k', `M' and `G' have their conventional scientific and engineering meanings of $*10^3$, $*10^6$ and $*10^9$ respectively. The only exception to this is in the description of the size of memory areas, when `K', `M' and `G' mean $*2^{10}$, $*2^{20}$ and $*2^{30}$ respectively.

When describing transfer rates, `k' `M' and `G' mean $*10^3$, $*10^6$ and $*10^9$ *not $*2^{10}$ $*2^{20}$ and $*2^{30}$.*

In PowerPC terminology, multiple bit fields are numbered from 0 to n, where 0 is the MSB and n is the LSB. PCI and CompactPCI terminology follows the more familiar convention that bit 0 is the LSB and n is the MSB.

Signal names ending with an asterisk (*) or a hash (#) denote active low signals; all other signals are active high.

Signal names follow the PICMG 2.0 R3.0 CompactPCI Specification and the PCI Local Bus 2.3 Specification.

For Your Safety

Your new Kontron product was developed and tested carefully to provide all features necessary to ensure its compliance with electrical safety requirements. It was also designed for a long fault-free life. However, the life expectancy of your product can be drastically reduced by improper treatment during unpacking and installation. Therefore, in the interest of your own safety and of the correct operation of your new Kontron product, you are requested to conform with the following guidelines.

High Voltage Safety Instructions


Warning!
All operations on this device must be carried out by sufficiently skilled personnel only.


Caution, Electric Shock!
Before installing a not hot-swappable Kontron product into a system always ensure that your mains power is switched off. This applies also to the installation of piggybacks. Serious electrical shock hazards can exist during all installation, repair and maintenance operations with this product. Therefore, always unplug the power cable and any other cables which provide external voltages before performing work.

Special Handling and Unpacking Instructions

ESD Sensitive Device!
Electronic boards and their components are sensitive to static electricity. Therefore, care must be taken during all handling operations and inspections of this product, in order to ensure product integrity at all times

Do not handle this product out of its protective enclosure while it is not used for operational purposes unless it is otherwise protected.

Whenever possible, unpack or pack this product only at EOS/ESD safe work stations. Where a safe work station is not guaranteed, it is important for the user to be electrically discharged before touching the product with his/her hands or tools. This is most easily done by touching a metal part of your system housing.

It is particularly important to observe standard anti-static precautions when changing piggybacks, ROM devices, jumper settings etc. If the product contains batteries for RTC or memory backup, ensure that the board is not placed on conductive surfaces, including anti-static plastics or sponges. They can cause short circuits and damage the batteries or conductive circuits on the board.

General Instructions on Usage

In order to maintain Kontron's product warranty, this product must not be altered or modified in any way. Changes or modifications to the device, which are not explicitly approved by Kontron and described in this manual or received from Kontron's Technical Support as a special handling instruction, will void your warranty.

This device should only be installed in or connected to systems that fulfill all necessary technical and specific environmental requirements. This applies also to the operational temperature range of the specific board version, which must not be exceeded. If batteries are present, their temperature restrictions must be taken into account.

In performing all necessary installation and application operations, please follow only the instructions supplied by the present manual.

Keep all the original packaging material for future storage or warranty shipments. If it is necessary to store or ship the board, please re-pack it as nearly as possible in the manner in which it was delivered.

Special care is necessary when handling or unpacking the product. Please consult the special handling and unpacking instruction.

Table Of Contents

Chapter 1 - Overview

Functional changes that differ from previous version of the document are identified by a vertical bar in the margin.

Linux, the Open Source Operating System is now taking a significant share of the OS market in Defense and Aerospace, after having taken ground initially in the enterprise server sector.

Red Hat Enterprise Linux is one of the leading platform for open source computing. It is sold by subscription, delivers continuous value and is certified by top enterprise hardware and software vendors.

The goal of this document is to help you through the installation process of the RHEL 6.4 BSP distribution on the Kontron VM6050 boards.

In this document, the term VM6050 is used for the VM6050 boards in standard or rugged conduction-cooled version:

VM6050 Single-slot 6U VME board

- VM6050-SA Standard Commercial version
- VM6050-RC Rugged Conduction-Cooled version

In this document, the term VM6050-RTM is associated to the VM6050 Rear Transition Module (RTM):

VM6050-RTM Rear Transition Module for the single-slot 6U VME board

- PBV36-P0-VM6-00

Chapter 2 - Release Content

The release is made of:

> **RHEL 6.4 BSP CD for VM6050 boards**

This distribution includes the BSP packages related to the VM6050. The RHEL 6.4 complete distribution should be order to RedHat.

You can choose to install this distribution in a graphical configuration or in a serial console configuration.

The Board Support Package (BSP) provides support for some specific features of the board:

> Sensors:	CPU Cores and Board temperatures and voltages.
> Vital Product Data (VPD) Tool:	Get board's serial number, order code, E.C. Level, ...
> LEDs:	Four Front Panel Tri-color LEDs
> GPIO:	Driver to support the GPIOs of the VM6050 boards
> Watchdog:	Drivers to setup the Watchdogs of the board.
> BIOS Update tool:	A command and script to update the BIOS of the board.
> CPLD register Tool (cpldtool):	Tool to deal with hardware registers of the onboard CPLD
> FRAM support:	Driver and special API file to read/write from/to the FRAM
> PBIT report:	sysvartool gives the report of the PBIT.
> I2C buses drivers:	i2c bus drivers for the local i2c bus and the two backplane i2c busses.

More information on VM6050 boards BSP in Chapter 7 “BSP Specific Features” page 14.

Chapter 3 - Associated Documentation

Kontron Documentation

- Hardware
  - VM6050 6U VME SBC User's Guide CA.DT.A93
  - VM6050 Hardware Release Notes CA.DT.A94

- Firmware
  - VM6050 BIOS User Manual SD.DT.F89

RHEL 6.4 Documentation

- Documentation available at: www.redhat.com

Chapter 4 - Required Configuration

4.1 Hardware Requirements

> A Kontron VM6050 board.
> The RHEL 6.4 release may be installed on one of the following bootable disks:
  - a SATA disk connected to the SATA connectors available on VM6050-RTM board.
  - Optional onboard USB Flash Disk.
> A USB DVD-ROM device (for installation from DVD-ROM).
> A console on serial line (text or VNC install).

For a graphic configuration on VM6050 a specific order code and a specific graphic module are required: VM6050-2SA34-12110 and MOD-GX-SA-00. The module provides two DP ports and a VGA connector.

4.2 Firmware Requirements

The version of the BIOS firmware must be at least:

> 14153

This version is displayed in the BIOS Setup.

4.3 Software Requirements

> The DVD-ROMs:
  - RHEL 6.4 x86_64 DVD images from RedHat
  - RHEL 6.4 BSP for VM6050 boards

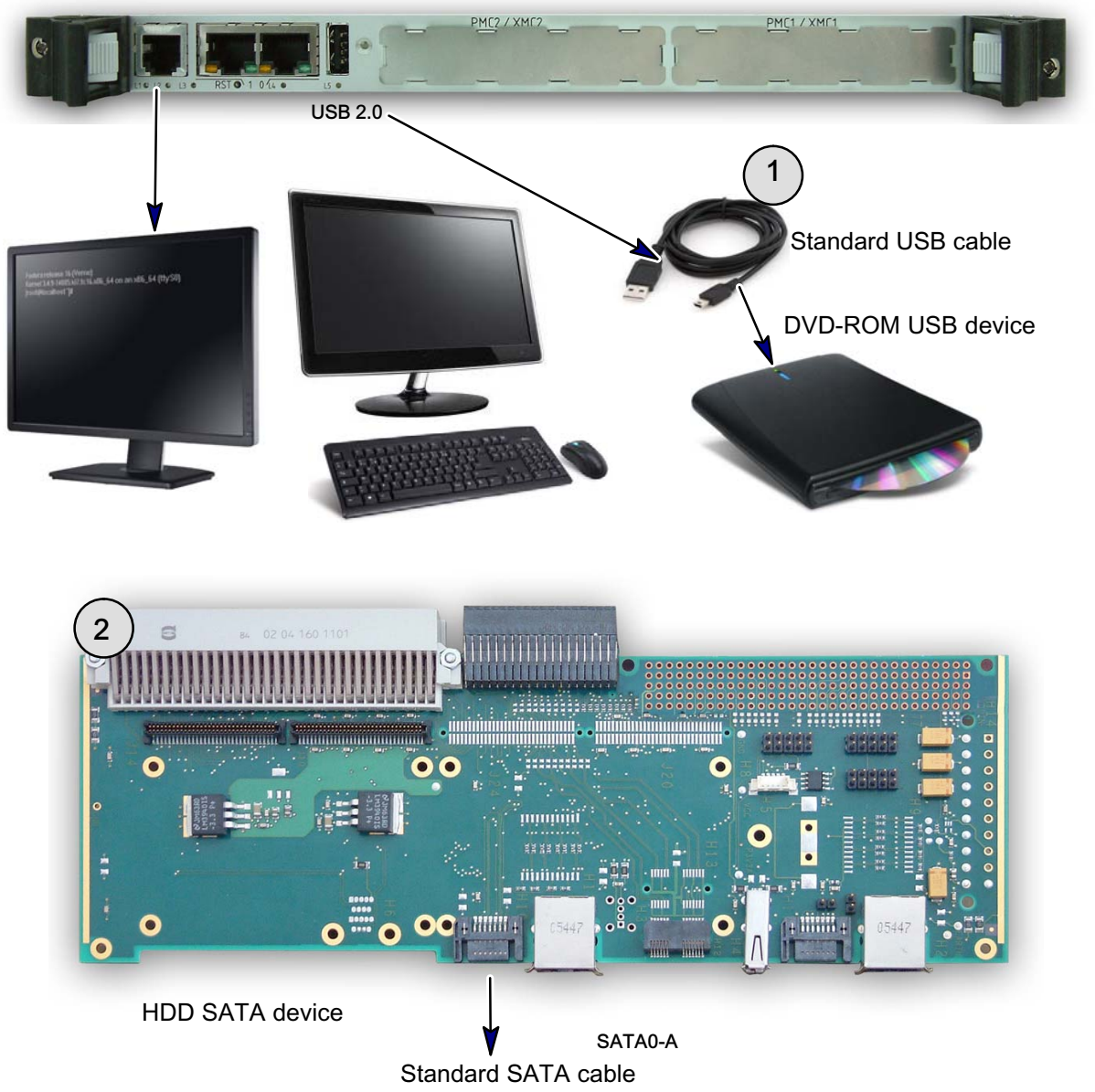
After the BSP is installed, the release version is saved in `/etc/ki7-release`:

```
# cat /etc/ki7-release
KI7 Board Support Package 1.0 [14276]
```

> ISO file image: the Kontron DVD can be delivered as ISO image format to customer.

4.4 DVD-ROM Installation Example

1. Plug a DVD-ROM USB device to the front panel USB connector using a standard USB cable.
2. Plug the HDD SATA device on the SATA0-A connector of the Rear Transition Module (VM6050-RTM).



Graphic configuration on VM6050 requires a specific order code and a specific graphic module: VM6050-2SA34-12110 and MOD-GX-SA-00. The module provides two DP ports and a VGA connector.

There are 2 USB ports available: one on the front panel and the other one on the RTM board if an RTM is available. So in order to use a USB keyboard and a USB mouse plus a USB DVD-ROM, a USB HUB is required.

Chapter 5 - Installation

This chapter describes the specific steps of the installation process of RHEL 6.4 on VM6050 boards.

5.1 RHEL 6.4 Installation on VM6050 Boards

This section describes the installation procedure from a USB DVD-ROM drive.

There is no major difference between the installation on the VM6050 boards and the standard RHEL 6.4 installation, so refer to the RedHat documentation to get more details on all the RedHat installation menus.

Nevertheless, the VM6050 boards may have graphic option, so the installation may be done in graphic mode or in text mode on the serial port 0.

1. Insert the DVD RHEL 6.4 in the DVD-ROM driver.
2. After a board reset, or a board power-on, type **<F7>** to get the the Boot Manager Menu or **<F2>** to get the Setup from the BIOS and to select the DVD-ROM device as the boot device.
3. Select the installation method using the menu, as explained above.
4. Then the standard RHEL 6.4 Installer menus from anaconda should be displayed.
5. Proceed as a standard RHEL 6.4 installation.
6. Note that the mezzanine USB Flash device, if present, is probed as another SATA drive.
7. At the end of the installation, reboot on the installed disk drive through the Boot Manager Menu **<F7>**. The BIOS Setup menu should be used to set the boot devices priorities. Refer to the BIOS Manual - SD.DT.F89.

It is strongly recommended to disable the swap partition if the installation is done on a USB or SATA flash device.

5.2 Installation of the BSP

To install the whole BSP RPMs package, mount the "RHEL 6.4 BSP for VM6050 Boards" CD-ROM, change to the mount point directory and use the `install.sh` script:

```
# mount /dev/cdrom /mnt
# cd /mnt
# ./install.sh -i
```

To check which BSP is currently installed:

```
# cat /etc/ki7-release
KI7 Board Support Package 1.0 [14276]
```

To check the log of the BSP installation:

```
# cat /usr/share/ki7_bsp/bsp_install.log
```

To list the Kontron rpms installed on the system:

```
# rpm -qa --queryformat "%{NAME}-%{VERSION} %{RELEASE}.%{PACKAGER}\n" | grep -i kontron | cut
-d' ' -f1
cpld-1.8.14276
cpld_i2c-1.8.2.6.32_358.el6.x86_64.14276
ki7_bsp-1.0.14276.el6
hwtools-1.3.7.14276
cpld_smi-1.3.14276
cpldtool-1.5.14276
cpld_gpio-1.3.2.6.32_358.el6.x86_64.14276
sensors_addons-1.8.14276
cpld_leds-1.3.14276
sysvartool-1.7.14276
fmram-1.1.14276
cpld_wdt-1.4.2.6.32_358.el6.x86_64.14276
vpdtool-1.11.14276
vmetoolkit-1.5.14276
```

Chapter 6 - RHEL 6.4 System Configuration

In this chapter, information related to some specific configuration items of the RHEL 6.4 system are detailed.

6.1 Network

6.1.1 Network Manager

With RHEL 6.4, the network interfaces are managed by the NetworkManager service by default.

For an embedded system, it is recommended to use the older network service instead which is easier to configure through configuration files.

For this:

> Disable the NetworkManager service:

```
[root@ki7]# chkconfig NetworkManager off
```

> Enable the network service:

```
[root@ki7]# chkconfig network on
```

> Stop the network manager: `service NetworkManager stop`

> Check the configuration files and modify them if needed :

- `/etc/sysconfig/network-scripts/ifcfg-ethx` files
- `/etc/resolv.conf`
- `/etc/sysconfig/network`

> Start the network service: `service network start`

> Reboot if `/etc/sysconfig/network` has been modified

6.1.2 MAC Address

By default, the MAC address is stored with the configuration parameters of each interface. If the MAC address of a device is found different from the one expected (board changed for example), the interface is not brought up. This is not suitable for an embedded system when boards must be changed for maintenance without requiring additional configuration.

To workaround this behavior, do not bind an Ethernet interface to a MAC address:

> Run `system-config-network`

> For each interface:

- click on Edit
- click on Hardware Device tab
- unselect Bind to MAC address

> Exit from `system-config-network` saving changes

This can be done also by editing the `/etc/sysconfig/network-scripts/ifcfg-eth*` files and removing the `HWADDR` lines.

Removing `HWADDR` only works if the service network (and not NetworkManager) is used.

6.1.3 Firewall

If the firewall must be disabled but has been enabled during the installation:

> Run `system-config-firewall`, click on disable and exit

OR run

```
[root@ki7]# service iptables stop
[root@ki7]# service ip6tables stop
```

> Make sure to disable the iptables service by running:

```
[root@ki7]# chkconfig iptables off
[root@ki7]# chkconfig ip6tables off
```

> Reboot

6.2 SELinux

SELinux stands for Security-Enhanced Linux. The Security-Enhanced Linux kernel enforces mandatory access control policies that confine user programs and system servers to the minimum amount of privilege they require to do their jobs.

If you experience some trouble running some services or have some permission issues, try to set the System Default Policy to Permissive instead of Enforcing by running the `system-config-selinux` tool, or from command line doing as follows:

- disable on boot by editing `/etc/selinux/config` to set `SELINUX=permissive` instead of `SELINUX=enforcing`
- disable now: `setenforce 0`

6.3 GRUB Boot Loader

If your console is on the serial line and the access to `grub` menu is required, open the `grub` configuration file `/boot/grub/grub.conf`, enter:

```
# vi /boot/grub/grub.conf
```

Append the following lines below "`hiddenmenu`" option:

```
serial --unit=1 --speed=115200
```

This line tells GRUB to use the first serial port at a baud rate of 115200.

The following line may be added in a dual head configuration to allow the selection of serial or graphic head as grub console.

```
terminal --timeout=8 console serial
```

The second line gives the user 9 seconds to decide where GRUB should output its information.

Please adjust port number and speed as per your setup.

Next make sure `splashimage` options is disabled as graphics can't be displayed across the serial port. Remove `splashimage` line or just comment it out by prefixing # symbol:

```
#splashimage=(hd0,0)/grub/splash.xpm.gz
```

6.4 SATA Speed

With some specific environmental constraints or with some SATA devices, it may be required to reduce the SATA speed of a specific SATA port.

The SATA speed may be 1.5 Gbps or 3 Gbps on VM6050, VX6060 or VX3030 and up to 6 Gbps for VX3035.

To properly manage the SATA speed, first of all check at the BIOS setup that AHCI mode is enabled.

Furthermore, AHCI mode should allow the access to Hotplug option (refer to section 6.5 page 12).

The current speed of the SATA ports may be checked at boot time by:

```
[root@ki7]# dmesg | egrep ata[1-9]:
[ 3.480624] ata1: SATA max UDMA/133 abar m2048@0xf1c02000 port 0xf1c02100 irq 45
[ 3.487984] ata2: SATA max UDMA/133 irq_stat 0x00400040, connection status changed irq 45
[ 3.496119] ata3: DUMMY
[ 3.498552] ata4: DUMMY
[ 3.500985] ata5: DUMMY
[ 3.503419] ata6: DUMMY
[ 3.809480] ata1: SATA link down (SStatus 0 SControl 300)
[ 4.226743] ata2: SATA link up 3.0 Gbps (SStatus 123 SControl 300)
```

Here the port 2 is set at 3.0 Gbps.

If the hard drive is finaly not supporting this rate, it is possible to decrease it by adding `libata.force=2:1.5` (for example) to the kernel command boot line to force at boot time the SATA speed limit of the port 2 at 1.5 Gbps.

```
[root@ki7]#    KERNEL=`grubby    --default-kernel`    ;    grubby    --update-kernel=$KERNEL
--args=libata.force=2:1.5
Then after a reboot:
[root@ki7]# dmesg | egrep ata[1-9]:
[ 3.480245] ata1: SATA max UDMA/133 abar m2048@0xf1c02000 port 0xf1c02100 irq 45
[ 3.487603] ata2: FORCE: PHY spd limit set to 1.5Gbps
[ 3.492627] ata2: SATA max UDMA/133 irq_stat 0x00400040, connection status changed irq 45
[ 3.500762] ata3: DUMMY
[ 3.503195] ata4: DUMMY
[ 3.505627] ata5: DUMMY
[ 3.508059] ata6: DUMMY
[ 3.814088] ata1: SATA link down (SStatus 0 SControl 300)
[ 4.231351] ata2: SATA link up 1.5 Gbps (SStatus 113 SControl 310)
```

It is mandatory for VM6050 to setup the SATA Speed of the onboard SSD Flash device to 1.5 Gbps.

In order to find the SATA bus number under Linux of the SSD Flash device type the following command:

```
[root@localhost ~]# dmesg | egrep GLS85
[    2.630945] ata5.00: ATA-8: GLS85LS1032A CS 32GBN A101D3, N A101D3, max UDMA/133
[    2.763694] scsi 4:0:0:0: Direct-Access     ATA      GLS85LS1032A CS  N A1 PQ: 0 ANSI: 5
```

In this case the `dmesg` indicates that the GLS85LS1032A with 32 GB is located to the `ata5` bus, means SATA number 5.

To keep a 1.5 Gbps speed for this interface `libata.force=5:1.5G` parameter must be added to the command line as described above.

6.5 SATA Hotplug

In AHCI mode, the SATA controller of the VM6050 boards provides a hotplug function. First of all this has to be setup at BIOS menus (Chipset->South Bridge Configuration->SATA Configuration). After the Hot Plug option is enabled for the SATA ports, boot the system.

» To remove a SATA device from the system:

Close all users of the device and backup device data as needed. Use `umount` to unmount any file systems that mounted the device.

Remove the device from any md and LVM volume using it. If the device is a member of an LVM Volume group, then it may be necessary to move data off the device using the `pvmove` command, then use the `vgreduce` command to remove the physical volume, and (optionally) `pvremove` to remove the LVM metadata from the disk.

If the device uses multipathing, run `multipath -l` and note all the paths to the device. Afterwards, remove the multipathed device using `multipath -f` device.

Run `blockdev -flushbufs` device to flush any outstanding I/O to all paths to the device. This is particularly important for raw devices, where there is no umount or vgreduce operation to cause an I/O flush.

Remove any reference to the device's path-based name, like `/dev/sd`, `/dev/disk/by-path` or the major:minor number, in applications, scripts, or utilities on the system. This is important in ensuring that different devices added in the future will not be mistaken for the current device.

Finally, remove each path to the device from the SCSI subsystem. To do so, use the command:

```
[root@ki7]# echo 1 > /sys/block/device-name/device/delete
```

where device-name may be `sde`, for example.

```
Nov 10 15:16:24 VM6050 kernel: [10018.256462] sd 1:0:0:0: [sdb] Synchronizing SCSI cache
Nov 10 15:16:24 VM6050 kernel: [10018.534156] sd 1:0:0:0: [sdb] Stopping disk
Nov 10 15:16:24 VM6050 kernel: [10018.934519] ata2.00: disabled
```

Then you can shut off the device.

» To add a SATA device:

When the system is up and running, power on the hotpluggable SATA device and the system should be warned that a new SATA device is available:

```
[root@ki7]# dmesg
...
Nov 10 15:22:40 Ki7 kernel: [10394.408164] ata2: irq_stat 0x00400040, connection status changed
Nov 10 15:22:40 Ki7 kernel: [10394.414149] ata2: SError: { RecovComm PHYRdyChg CommWake DevExch }
Nov 10 15:22:40 Ki7 kernel: [10394.420310] ata2: hard resetting link
Nov 10 15:22:43 Ki7 kernel: [10396.923493] ata2: SATA link up 3.0 Gbps (SStatus 123 SControl 300)
Nov 10 15:22:43 Ki7 kernel: [10396.925408] ata2.00: ATA-8: ST9160314AS, 0001SDM1, max UDMA/133
Nov 10 15:22:43 Ki7 kernel: [10396.925414] ata2.00: 312581808 sectors, multi 16: LBA48 NCQ (depth 31/32)
Nov 10 15:22:43 Ki7 kernel: [10396.927836] ata2.00: configured for UDMA/133
Nov 10 15:22:43 Ki7 kernel: [10396.927846] ata2: EH complete
Nov 10 15:22:43 Ki7 kernel: [10396.927991] scsi 1:0:0:0: Direct-Access ATA ST9160314AS\0001 PQ: 0 ANSI: 5
Nov 10 15:22:43 Ki7 kernel: [10396.928246] sd 1:0:0:0: Attached scsi generic sg1 type 0
Nov 10 15:22:43 Ki7 kernel: [10396.928299] sd 1:0:0:0: [sdb] 312581808 512-byte logical blocks: \
                                                                                (160 GB/149 GiB)
Nov 10 15:22:43 Ki7 kernel: [10396.928398] sd 1:0:0:0: [sdb] Write Protect is off
Nov 10 15:22:43 Ki7 kernel: [10396.928443] sd 1:0:0:0: [sdb] Write cache: enabled, \
                                                        read cache: enabled, doesn´t support DPO or FUA
Nov 10 15:22:43 Ki7 kernel: [10396.928772]  sdb: sdb1 sdb2
Nov 10 15:22:43 Ki7 kernel: [10396.955136] sd 1:0:0:0: [sdb] Attached SCSI disk
```

If automount services are enabled, the partitions should be mounted too.

6.6 VITA 57

Using VX3830 IO cards for VPX boards or VM6050, it is possible to use the VITA 57 option. Kontron is delivering a toolkit to help the setup and development of application using the onboard FPGA and the FMC modules defined though the VITA 57 standard. Refer to your representative to get more information about this option.

Chapter 7 - BSP Specific Features

7.1 Sensors

7.1.1 Sensors Overview

The BSP contains an RPM named `sensors_addons` that configures the standard `lm_sensor` software for the VM6050 boards.

To display sensors information:

```
[root@ki7]# sensors
```

» Example on VM6050

```
[root@vm6050-2 ~]# sensors
acpitz-virtual-0
Adapter: Virtual device
temp1:       +28.0°C  (crit = +119.0°C)
temp2:       +28.0°C  (crit = +119.0°C)

lm73-i2c-22-48
Adapter: I2C CPLD adapter
 LM73 sensor Temperature: +28.0°C  (low  = -30.0°C, high = +70.0°C)

lm73-i2c-22-49
Adapter: I2C CPLD adapter
 LM73 sensor Temperature: +29.0°C  (low  = -30.0°C, high = +70.0°C)

lm73-i2c-22-4a
Adapter: I2C CPLD adapter
 LM73 sensor Temperature: +24.0°C  (low  = -30.0°C, high = +70.0°C)

ads7830-i2c-8-4b
Adapter: SMBus I801 adapter at e000
12V VPX:     +11.89 V
5V VPX:       +4.92 V
3V3:          +3.34 V
2V5 6U:       +2.50 V
3V3 VPX SB:   +3.26 V
1V05S 3U:     +1.04 V
1V05 3U IBEX: +1.02 V
1V 3U:        +1.00 V

coretemp-isa-0000
Adapter: ISA adapter
Core 0:      +30.0°C  (high = +95.0°C, crit = +105.0°C)
Core 2:      +27.0°C  (high = +95.0°C, crit = +105.0°C)
```

The sensor command reveals the presence of low, high and critical thresholds. When the temperature temp1 goes beyond the critical threshold, an automatic reset of the board will occur.

When the temperature or the voltage goes beyond one of the limits low and high, an explicit alarm message will occur in the sensors command output. So, in order to track down this kind of event, run the following command:

```
[root@ki7]# sensors | grep ALARM
```

7.1.2 Sensors Values Limitations

The sensors named "`acpitz-virtual-0`" has some limitations.

For these sensors, which are internal to the CPU, Intel does not guarantee the validity of temperature value in high level range of temperature.

The Kontron BIOS, to inform the user that the temperature probe validity is not correct, return the value -56°C.

So if a probe of these sensors value returns -56°C, it does not mean that the board is currently running at -56°C, but it only means that the probe value can not be reliable.

7.2 CPLD-WDT

NAME

cpld-wdt – Kontron board cpld_wdt watchdog driver

DESCRIPTION

This man page describes how to use the watchdog implemented by the cpld on various Kontron boards including the VX304x, VX3035 and VM605x families.

The principle of a watchdog is to automatically provoke some action after a given time passes without the watchdog being prodded by some process. This would indicate that the process is no longer working correctly. The cpld_wdt watchdog actions are to do nothing, to reset the board, to generate an interrupt that can wake up some other process, or to reboot. The prodding is done by writing to the watchdog device, which restarts the timeout.

The cpld_wdt module implements the standard Linux watchdog API, detailed in file **Documentation**/**watchdog**/**watchdog – api.txt** in the kernel source rpm. The supported features are described below.

Loading Module
There can be many devices and modules providing the watchdog API. Ensure only the cpld_wdt module is being used. For example, the standard iTCO_wdt watchdog module is usually blacklisted by a file in /**etc**/**modprobe.d**/ to avoid it being loaded.

The cpld_wdt module has to be explicitly loaded, for example by creating a systemd file with a name ending in ".conf" in /**etc**/**modules – load.d**/ listing the modules to load, one per line, for example:

```
cpld_wdt
```

Module Options
To provide initial options to the module, create a file with a name ending in ".conf" in /**etc**/**modprobe.d**/ holding a line such as:

```
options cpld_wdt timeout=50 trigger_mode=1 nowayout=1
```

The module options, shown by the command "modinfo cpld_wdt", include
timeout

the integer timeout in seconds, from 0 to 510 with a resolution of 2 seconds. The default is 30.

trigger_mode

the action to do when the timeout expires. An integer value

- 0 to simply countdown with no action,
- 1, the default, to reset the board,
- 2 to generate an interrupt that can be used to wake up a read on the device, or to reboot if no read is pending.

nowayout

an integer 1 if there is no way to stop the watchdog. The default is 0, which stops the timer if the magic character "V" is written just before the device is closed.

Usage from Scripts
The watchdog is implemented with the standard device /**dev**/**watchdog**. This file can accept ioctls to

configure the watchdog, but can be used simply from a shell script as follows: Load the module with the required configuration, for example to interrupt, with a timeout of 10 seconds:

```
[root@ki7]# rmmod cpld_wdt
[root@ki7]# modprobe cpld_wdt timeout=10 trigger_mode=2 nowayout=0
```

Start the watchdog by writing to the device:

```
[root@ki7]# echo >/dev/watchdog
```

Probe the watchdog faster than every 10 seconds:

```
[root@ki7]# while sleep 5; do echo >/dev/watchdog; done
```

After a while stop the loop and wait for a timeout:

```
[root@ki7]# cat /dev/watchdog
```

This will hang for 10 seconds. If you wait another 10 seconds without issuing another read, the OS will reboot. To stop the watchdog instead:

```
[root@ki7]# echo V >/dev/watchdog
```

To avoid the need to be root, simply change the ownership or permissions of /dev/watchdog.

Watchdog Ioctl API
The following standard ioctls are supported.
WDIOC_GETSUPPORT

```
#include <linux/watchdog.h>
struct watchdog_info ident;
ioctl(fd, WDIOC_GETSUPPORT, &ident);
```

returns in the structure the fields
identity

the driver identification ”cPLD WDT”

firmware_version

is always 1

options

describes the supported features, namely **WDIOF_KEEPALIVEPING** and **WDIOF_SETTIMEOUT**.

WDIOC_SETTIMEOUT

```
int timeout = ...;
ioctl(fd, WDIOC_SETTIMEOUT, &timeout);
```

sets the timeout in seconds.

WDIOC_GETTIMEOUT

```
ioctl(fd, WDIOC_GETTIMEOUT, &timeout);
```

returns the current timeout setting in the argument (not the dynamically changing counter value).

WDIOC_SETOPTIONS

```
int options = ...;
ioctl(fd, WDIOC_SETOPTIONS, &options);
```

configures the given options. These are **WDIOS_DISABLECARD** to disable the watchdog, and **WDIOS_ENABLECARD** to enable the watchdog.

WDIOC_KEEPALIVE

```
ioctl(fd, WDIOC_KEEPALIVE, 0);
```

prods the watchdog, restarting the countdown timer.

SEE ALSO

cpld(4)
cpldtool(1)

FILES

/dev/watchdog
Documentation/watchdog/watchdog – api.txt

COPYRIGHT

Kontron

7.3 VPD Tool

NAME

vpdtool – display Kontron board VPDs (Vital Product Data)

SYNOPSIS

–a *boardtype*
– –boardtype
– –conffile or –f *file*
– –eclevel
– –help or –h
– –macaddr
– –serialnumber
– –variant

DESCRIPTION

vpdtool reads the VPDs (Vital Product Data) of many Kontron boards, including the following:

- ITC320/322 PENTXM2/4
- VM6050 VM6052 VM6054
- VM6250
- VX3020 VX3030 VX3035 VX3040
- VX3230 VX3240
- VX6060 VX6070 VX6080

As it reads hardware ports via /dev/mem you need to be root to run it.

OPTIONS

–**a** *boardtype*

forces the architecture of the board to that given, e.g. VX3020

– –**conffile or** –**f** *file*

uses the given features definition file. This file says how to convert the vpd binary encodings into text descriptions.

– –**help or** –**h**

prints an option summary.

– –**boardtype** – –**eclevel** – –**macaddr** – –**serialnumber** – –**variant**

displays only the requested information. The options can be combined.

EXAMPLE

```
[root@vm6050-2 sys]# vpdtool
VM6050 detected
 i2cbus_num = 22
Board type    : VM6050-2SA34-10110
EC Level      : 02005
Serial Number: 1813021380549
Variant       : 1000004180850010
```

FILES

/dev/mem
/dev/i2c/*

COPYRIGHT

Kontron

7.4 LEDs

The driver `leds_cpld` handles the front panel LEDs for user mode.

The driver `leds_cpld` creates a list of special files and classes in `/sys`:

```
[root@ki7]# ls /sys/class/leds/
led2:amber  led2:OFF  led3:amber  led3:OFF
led2:green  led2:red  led3:green  led3:red
```

For each LED, there are three different colors available: green, red, amber that can be set by addressing the related file.

For each color (which are exclusive), there are four different modes:

- ON (echo 0, see following example)
- slow blinking (echo 1, see following example)
- fast blinking (echo 2, see following example)
- OFF (echo 1, see following example)

Example, to set these different modes on the LED 2 in amber:

```
[root@ki7]# echo 0 > led2:amber/brightness The LED 2 is ON in AMBER
[root@ki7]# echo 1 > led2:amber/brightness The LED 2 blinks low in AMBER
[root@ki7]# echo 2 > led2:amber/brightness The LED 2 blinks fast in AMBER
[root@ki7]# echo 0 > led2:OFF/brightness The LED 2 is OFF
```

When setting led2 or led3, both led2 and led3 are set to user mode.

To exit from this mode, set led<n>:OFF/brightness to a non null value; for example

```
[root@ki7]# echo 1 > led2:OFF/brightness
```

> VM6050: On the front panel, the name of the led1 is L1, led2 is L2 and led3 is L3.

The led1 (L1) is not manageable at user level, so no special file for led1 is available.

> To read back the state of a led:

```
[root@vm6050-2 leds]# echo 2 > led2\:amber/brightness
[root@vm6050-2 leds]# cat led2\:amber/brightness
2
```

Note that the OFF entry may be cat to get the mode too:

```
[root@vm6050-2 leds]# cat led2\:OFF/brightness
2
[root@vm6050-2 leds]# echo 0 > led2\:OFF/brightness
[root@vm6050-2 leds]#
[root@vm6050-2 leds]# cat led2\:OFF/brightness
255
[root@vm6050-2 leds]# cat led2\:amber/brightness
255
```

7.5 Sysvartool and PBIT Report

To get the report of the PBIT (Power Built In Test), run the command:

```
[root@ki7]# sysvartool -A pbit -l
VX304x detected
 area = 2, arch = 2
POSTs configured to run from command line:
       mem_data: PASSED
       mem_addr: PASSED
   mem_pattern1: PASSED
   mem_pattern2: PASSED
   mem_pattern3: PASSED
   mem_pattern4: PASSED
    ether_loop0: PASSED (FAILED ONCE)
    ether_loop1: PASSED (FAILED ONCE)
    ether_loop2: PASSED
         system: PASSED

PASSED  : 10
FAILED  : 0
NOT RUN : 0
TOTAL   : 10

POSTs configured to run automatically from RAM:

PASSED  : 0
FAILED  : 0
NOT RUN : 0
TOTAL   : 0

POSTs configured to run automatically from ROM:

PASSED  : 0
FAILED  : 0
NOT RUN : 0
TOTAL   : 0
```

7.6 GPIOs

NAME

cpld-gpio - Kontron board gpio-cpld driver gpios

DESCRIPTION

This man page describes how to use the gpios implemented by the cpld on various Kontron boards including the VX304x, VX3035 and VM605x families.

The cpld gpio driver provides an API using files in the /sys filesystem. Reads and writes to the files are directly handled by the driver, provoking reads and writes of hardware registers. Reads normally return a short string terminated with a newline. Further reads need to seek back to the start of the file or re-open it. Writes should normally be of a single string, optionally terminated with a newline. Writing values other than those described below are undefined, but usually result in a write error with errno EINVAL Invalid argument.

Usage from Program
Operations can be easily done from a shell script. When used from a program, open the files for simultaneous read and write, except for ”intr_stat” (read only). Read in one go into a buffer big enough to accept the whole reply, including newline (i.e 11 characters). Use lseek(2) to rewind the file descriptor before each read. Write each value using a single system call. There is no need to lseek before writes.

Permissions
By default the files are owned by root and only writeable by the owner. If necessary, these ownerships and permissions can be changed (each time the driver is loaded), for example by a udev rule.

Sysfs Files
There is one directory per gpio, starting with gpio1, under the class directory:

/sys/class/gpio/

The number of gpios depends on the board. Each directory has the following files. The string values described are those that can be written, or will be read.

direction holds the value ”in” (incoming signal) or ”out” (outgoing). The default is ”in”. For example:

```
[root@ki7]# echo out >/sys/class/gpio/gpio4/direction
[root@ki7]# cat /sys/class/gpio/gpio4/direction
out
```

value holds the value ”0” (low signal) or ”1” (high). On read return the current value of the gpio. On write sets the value of an outgoing gpio. For example:

```
[root@ki7]# echo 0 > /sys/class/gpio/gpio4/value
[root@ki7]# cat /sys/class/gpio/gpio3/value
1
```

polarity holds the value ”hi” (interrupt when signal active on high level or rising edge), or ”lo” (active on low level or falling edge). The default is ”hi”.

mode holds the value ”edge” (interrupt on rising or falling edge) or ”level” (on high or low level). The default is ”edge”.

toggle holds the value ”on” (interrupt on any state change) or ”off” (interrupt only on the configured polarity and mode). When on it overrides the polarity and mode settings. The default is ”off”.

interrupt holds the value ”on” (enable interrupts) or ”off” (disable). When an interrupt occurs in level mode, this value is automatically reset to ”off” to avoid a permanent interrupt. The default is ”off”.

intr_stat is a read-only count of the number of interrupts received on the gpio. This is an unsigned 32 bit decimal value which can be up to 10 digits long, plus a newline. This file is read-only, and writing to it will result in a write error of EPERM Permission denied, or EIO Input/output error. For example:

```
$ cat /sys/class/gpio/gpio5/intr_stat
12
```

value_it on read waits for an interrupt on the gpio, and then returns a string of length zero. If interrupts have not been enabled, the file acts like the value file. The file acts like the value file for all writes.

To get an interrupt you must set the direction, polarity, mode, toggle and interrupt files appropriately, and then read value_it until it returns, or poll the intr_stat value until it changes.

Dual Purpose GPIO
On some boards, gpio2 is a dual purpose gpio which can also be used to generate a Maskable Reset. It is configured using cpld registers that can be set using cpldtool, or low level port accesses.

To check if the board provides this feature run the command:

```
[root@ki7]# cpldtool -a|grep MSKR2LOC
```

If MSKR2LOC is not found, gpio2 is not dual purpose. To configure gpio2 as a standard gpio which does not issue a local reset, on a VPX board:

```
[root@ki7]# cpldtool -f VPX_CONTROL MSKR2LOC 0
```

on a VME board:

```
[root@ki7]# cpldtool -f VME_CONTROL MSKR2LOC 0
```

or on either type of board specify the register by number instead:

```
[root@ki7]# cpldtool -f 0x70 MSKR2LOC 0
```

Use the same command with value 1 to restore the configuration. To do the same using low level port accesses, read the register and calculate the new value before writing it back. For example:

```
[root@ki7]# port 0x870
@0x870 = 0x1d
[root@ki7]# port 0x870 0xd
@0x870 <- 0x0d
[root@ki7]# port 0x870
@0x870 = 0x0d
```

SEE ALSO

cpld(4)
cpldtool(1)

FILES

/sys/devices/platform/cpld_gpios/gpio/gpio[1-8]

COPYRIGHT

Kontron

7.7 cpldtool

NAME

cpldtool – Kontron board cPLD register utility

SYNOPSIS

–a
–d *register*
–f *register fieldname hexvalue ...*
–i *register*
–s *register hexvalue*
–v

DESCRIPTION

This utility interprets, displays, and sets flags in registers of the cPLD on Kontron boards including the VX3230, VM6250, VX3030 and VX6060. The cPLD is a small device that controls some low–level aspects of the board. See cpld(4). You need to be root to access the device. **Do not change register values accept as advised by Kontron**.

OPTIONS

Registers should be given as a decimal number, or hexadecimal number prefixed by 0x, whereas register content hexvalues are always assumed to be hexadecimal, whether prefixed or not.

–a displays all registers and interprets the bitfield flags as appropriate.

–d *register* displays the current value of the given register.

–i *register* displays information on the given register, naming each bitfield and saying where it starts.

–s *register hexvalue* sets the given register to the given hexadecimal value.

–f *register fieldname hexvalue ...* sets the given register field by field. Each field is specified by field-name, as shown by option –i, and the hexadecimal value for that field.

–v more verbose output for the –a and –d options, detailing what some values mean.

EXAMPLES

Describe register 9:

```
[root@ki7]# cpldtool –i 9
Ref 0x9 – FLASH MEMORY PROTECT


Field name: Boot flash CS swap DIP (7)
Explanation:
Boot flash chip select configuration
 0 : Normal configuration
 1 : Rescue configuration
 ...
Field name: USER WP (3)
Explanation:
USER level WP hardware protection
 0 : No USER level WP correction
 1 : USER level WP active
```

Show the current value of fields in register 9:

```
[root@ki7]# cpldtool -d 9
Reg 0x9 - FLASH MEMORY PROTECT = 0x00
         Boot flash CS swap DIP=0x0
         Boot flash CS swap Valid#=0x0
         Boot both flash=0x0
         USER WP=0x0
         SYS WP=0x0
         VPD WP=0x0
         VPX NVMRO=0x0
```

Set the ”USER WP” bitfield in register 9:

```
[root@ki7]# cpldtool -f 9 'USER WP' 1
```

SEE ALSO

cpld(4)

COPYRIGHT

7.8 I2C Busses

The `cpld_i2c` driver is supporting the local i2c bus (I2C bus number 22) which is local to the board and used to address sensors and system eeproms.

There are also two other I2C busses routed to the backplane; the bus numbers are 23 and 24:

- `/dev/i2c-23` is the SMB bus connecting it to the I2C devices in the chassis (if any).
- `/dev/i2c-24` is the IPMB bus connecting all boards through the backplane

By default, those two busses are available only on the system controller.

It is possible to enable those busses on other slots by adding the option `force_i2c_extend=1` at the load of the module "`cpld`". Add this argument "`cpld.force_i2c_extend=1`" at the kernel command line to enable this option.

7.9 BIOS Update

The `flashrom` package provides a set of commands and scripts to update the BIOS of the boards.

To update the BIOS of the VM6050 boards, use the command `ki7updbios`:

Help command:

```
[root@vm6050-2 ~]# ki7updbios -r BIOS.bin
flashrom v0.9.9 on Linux 2.6.32-358.el6.x86_64 (x86_64)
flashrom is free software, get the source code at http://www.flashrom.org

Calibrating delay loop... OK.
Found chipset "Intel QM57". Enabling flash write... OK.
Found SST flash chip "SST25VF032B" (4096 kB, SPI) at physical address 0xffc00000.
Reading flash... done.
```

Read the current BIOS:

```
[root@ki7]# ki7updbios -r BIOS.bin
flashrom v0.9.4 on Linux 2.6.32.14-11035.vx304x.fc12.i686.PAE (i686), built with libpci 3.1.7,
GCC 4.4.2 20091027 (Red Hat 4.4.2-7), little endian
flashrom is free software, get the source code at http://www.flashrom.org

Calibrating delay loop... OK.
No coreboot table found.
Found chipset "Intel QM57", enabling flash write... OK.
This chipset supports the following protocols: FWH,SPI.
Found chip "SST SST25VF032B" (4096 KB, SPI) at physical address 0xffc00000.
Reading flash... done.
[root@ki7]# ls -ltr BIOS.bin
-rw-r--r-- 1 root root 4194304 2011-03-03 15:54 BIOS.bin
```

Write a new BIOS file:

```
-rw-r--r--  1 root root    3654 Oct  3  2014 mbm3k.cpusgl.2669.log
[root@vm6050-2 ~]# ki7updbios -w BIOS.bin
flashrom v0.9.9 on Linux 2.6.32-358.el6.x86_64 (x86_64)
flashrom is free software, get the source code at http://www.flashrom.org

Using region: "bios".
Calibrating delay loop... OK.
Found chipset "Intel QM57". Enabling flash write... OK.
Found SST flash chip "SST25VF032B" (4096 kB, SPI) at physical address 0xffc00000.
Flash image seems to be a legacy BIOS. Disabling coreboot-related checks.
Reading old flash chip contents... done.
Erasing and writing flash chip... Erase/write done.
Verifying flash... VERIFIED.
```

Take care that if the system is booted from the RESCUE BIOS flash (refer to the BIOS user manual), the update of the BIOS using the `ki7updbios` command will update the RESCUE BIOS Flash **which is not recommended**.

The update of the BIOS through the `ki7updbios` command does not preserve the setup parameters of the BIOS. For deployement of a BIOS version with its own setup parameters, preset one board with the desired parameters, backup this using `ki7updbios -r`, and use this version of BIOS+ setup to be deployed on other similar boards.

7.10 FMRAM Example

The `fmram` packages gives an example of how access to the FMRAM device which may be used to save some customer data which needs to be backuped.

```
[root@ki7]# fmram -h
```

Usage `fmram` [options]
a tool to read or write the FerroMagnetic RAM

Options are :

`-h`	: this help
`-r <value>`	: read at offset <value> (default 0)
`-w <value>`	: write at offset <value> (default 0)
`-s <value>`	: data size to read or write (default 4)
`-f <filename>`	: file name used to store (optional) or read (mandatory) data

Example:

`fmram -w 0x10 -s 0x11 -f data_file` :
store 17 bytes read from data_file to the ferromagnetic RAM at offset 16

With src package, the C code of this command is delivered as an example.

7.11 VME

7.11.1 ALMAVME

NAME

almavme – Alma VME Linux toolkit driver API

SYNOPSIS

Kernel API (from a kernel driver)
#include <almavme.h>

int alma_pci_channel_alloc(char *name, u32 vmeaddr, unsigned long pciaddr, u32 size, u32 flag);
int alma_pci_channel_free(int id);
unsigned long alma_vme_channel_alloc(char *name, u32 vmeaddr, u32 size, u32 flag);
int alma_vme_channel_free(unsigned long address);
u32 alma_cpuadr_to_vmeadr ”(unsigned long address);

int alma_user_dma(u32 vmeaddr,unsigned long pciaddr,u32 size,u32 flags);
int alma_user_dma_mcast(u32 vmeaddr, unsigned long pciaddr,u32 size,u32 flags,u32 select);

int alma_get_vme(void);
int alma_free_vme(void);
int alma_ctrl(int command, int config);

int request_vmeirq(unsigned int vector, void *handler);
int free_vmeirq(unsigned int vector);
void alma_vmeintcontrol(unsigned int mask, unsigned int ienable);
int disable_vmeirq(u32 level);
int enable_vmeirq(u32 level);
int alma_vmeinterrupt(unsigned char level, unsigned char vector);
int alma_phys_mem_alloc(struct alma_phys_mem_object *phys_mem);
int alma_phys_mem_free(struct alma_phys_mem_object *phys_mem);

User Space API (from an application)
#include <linux/almavme.h>
int fd = open(”/dev/almavme”, O_RDWR);
if(fd<0){
perror(”Error opening the almavme device”);
exit(1);
}
alma_ioctl_arg_t alma_ioctl;
alma_ioctl.intr.level = 1;
alma_ioctl.intr.vector = 2;
int error = ioctl(fd, VMEIOCTL_VMEINTR_GEN, &alma_ioctl);
close(fd);

See the User Space sections below.

DESCRIPTION

This describes the kernel service calls available for VME device drivers or application development on Kontron platforms using the almavme driver for all boards equipped with Alma2e and Alma2f.

Alma2f provides a highly integrated single chip solution to interface a VME64 bus with 2eSST protocol (two-edge Source-Synchroneous Transfer) and a 32-bit 66 MHz PCI Bus. All bridge features are programmable from the PCI bus or the VME bus. See the Alma2f User Manual CI.DT.A00.

In the following, functions are described from the kernel point of view, and examples show how to call them from a user application.

User Space Ioctls
All ioctls take a command argument, and an appropriate struct as third argument. As **alma_ioctl_arg_t** is a union of most of the possible struct types, it can be used as a suitable third argument for most of the ioctls. The descriptions below assume the following usage:

```
alma_ioctl_arg_t A;
A... = ...;
ioctl(fd,command,&A);
```

VME Access
All accesses to or from the VMEbus go through the PCI32 bus and the PCI-to-VME bridge hardware. No specific windows (slave or master) are opened at load time by the almavme device driver (except for the slave window to access Alma register space (A16) that is inherited from the firmware setup on VMPCx and PN3 boards). See the ”Mapping Service Calls” section below on how to open slave or master windows.

Mapping Service Calls
alma_pci_channel_alloc

int alma_pci_channel_alloc(char *name, u32 vmeaddr, unsigned long pciaddr,u32 size,u32 flag)

This allocates a slave channel from the VME to the PCI (i.e. incoming accesses to the board). Up to 15 channels can be opened, 8 of them support 2ESST (see the 2ESST section). The parameters of the routine describe the VME-PCI channel:

name is the name of this channel. It can be up to 12 bytes long, including the final null char. It is only used to label this channel in the list of channels printed by the almavmechan(1) utility.

vmeaddr is the VME physical address where the board will respond on the VME. It must be aligned to a 1MB boundary and also to the rounded-up channel size.

pciaddr is the physical address generated on the board when the PCI/VME bridge responds to the vmeaddr address on the VME bus. The address can be in PCI-IO space, PCI-MEM space or DRAM space (the flag field defines which one is used). It must be aligned to a 1 MB boundary and to the rounded-up channel size.

size is the size of the channel. It must be a multiple of 1 MB and it is rounded up to a power of 2.

flag contains the options for the channel. Options are ‘ored’ from the following:

MEFG_WRTPOST access is write posted (default NO)

VMEFG_READAHEAD access is read ahead (default NO)

VMEFG_LEBE_*xx* defines the little/big endian conversion mode (default is address coherency)

***** The pci space which is one of:

VMEFG_PCIIO access PCI-IO

VMEFG_PCIMEM access PCI-MEM

VMEFG_DRAM access DRAM (default).

***** The AM code, which is decoded on the VMEbus for this channel. The default is **VMEFG_AM_A32SDATA** (0x0D). An AM code is specified as either:

VMEFG_AM_*xxx* flags that predefine standard AM codes, or

VMEFG_AM_DIRECT ored with multiple AM selecting bitmasks. The bitmask 0x0FF00000 defines which AM[5..3] are valid and the bitmask 0x000FF000 defines which AM[2..0] are valid. Each acceptable value of AM[2..0], i.e. the eight values 0 to 7, is selected by setting a bit in the bitmask, (1<<12) for 0, (1<<13) for 1, etc. To simplify, the **VMEFG_AMSLV_***xxx* flags can be used when several AM codes must be decoded.

This routine returns an identifier from 0 to 15 on success and -1 on failure. It can be called from user space with the ioctl() **VMEIOCTL_PCI_CHAN_ALLOC.**

alma_pci_channel_free

int alma_pci_channel_free(int id)

This is used by the VME driver to free a VME-PCI channel allocated by alma_pci_channel_alloc(). The parameter id is the channel identifier returned by alma_pci_channel_alloc(). It returns 0 on success, -1 on failure. It can be called from user space with the ioctl **VMEIOCTL_PCI_CHAN_FREE.**

alma_vme_channel_alloc

unsigned long alma_vme_channel_alloc(char *name, u32 vmeaddr, u32 size, u32 flag)

This allocates an outgoing master channel from PCI to VME. The parameters of the routine describe the desired PCI-VME channel:

name is the name of this channel. It can be up to 12 bytes long, including the final null char. It is only used to label this channel in the list of channels printed by the alma_vmechan utility.

vmeaddr is the VME physical address generated by the board. It must be aligned to 8 MB.

size is the size of the channel. It must be a multiple of 8 MB, and less than 256 MB.

flag contains the options for the channel. Options are 'ored' from the following:

VMEFG_WRTPOST access is write posted (default NO). It must be used with a BLT or MBLT channel.

VMEFG_READAHEAD access is read ahead (default NO). It must be used with a BLT or MBLT channel.

VMEFG_LEBE_*xx* defines the little/big endian conversion mode (default address coherency).

VMEFG_PCIIO access is from PCI-IO to VME (default is access from PCI-MEM to VME). There is no difference between PCI-MEM and PCI-IO.

AM code. One of the predefined **VMEFG_AM_***xxx* flags. The default is **VMEFG_AM_A32SDATA** (0x0D AM code).

This routine returns the CPU physical address on success and -1 on failure. The CPU physical address to be used by the Operating System or a user program can be mapped using the routine mmap(). This routine can be called from user space with the ioctl **VMEIOCTL_VME_CHAN_ALLOC.**

alma_vme_channel_free

int alma_vme_channel_free(unsigned long address)

This is used by the VME driver to free a PCI-VME channel allocated by alma_vme_channel_alloc().

address is the pointer returned by alma_vme_channel_alloc().

This routine returns 0 on success and SYSERR on failure. It can be called from user space with the ioctl **VMEIOCTL_VME_CHAN_FREE.**

alma_cpuadr_to_vmeadr

u32 alma_cpuadr_to_vmeadr(unsigned long address)

This returns the VME address that can be used to get at the given cpu physical address, or -1 if not visible from the VME.

User Space Mapping Service

ioctl(fd, **VMEIOCTL_PCI_CHAN_ALLOC**, &A)

does alma_pci_channel_alloc(A.pcichan.name, A.pcichan.vmeaddr, A.pcichan.pciaddr, A.pcichan.-size, A.pcichan.flag)

ioctl(fd, **VMEIOCTL_PCI_CHAN_FREE**, &A)

calls alma_pci_channel_free(A.index)

ioctl(fd, **VMEIOCTL_VME_CHAN_ALLOC**, &A)

returns alma_vme_channel_alloc(A.vmechan.name, A.vmechan.vmeaddr, A.vmechan.size, A.vmechan.flag)

ioctl(fd, **VMEIOCTL_VME_CHAN_FREE**, &A)

calls alma_vme_channel_free(A.addr)

For example (from MasterSgl.c).

```
char *devName = "mychan";
alma_ioctl.vmechan.name = devName; // Name
alma_ioctl.vmechan.vmeaddr = aBaseAddr; // VME Physical Address
alma_ioctl.vmechan.size = PCIVME_BLKSIZE; // As small as possible (8 MB)
alma_ioctl.vmechan.flag = VMEFG_AM_A32UDATA;
physAddr = ioctl(fd, VMEIOCTL_VME_CHAN_ALLOC, &alma_ioctl);
if(physAddr==-1){
  printf("vmeOpen: VME channel open / VMEIOCTL_VME_CHAN_ALLOC fail:
%s\n",devName);
  return (ERROR);
}
virtualAddr = (unsigned long)mmap(0, PCIVME_BLKSIZE, PROT_READ |
PROT_WRITE,
                MAP_SHARED, fd, (off_t)(physAddr&0xFFFFFFFF));
if(virtualAddr==-1){
  printf("vmeOpen: VME channel open: mmap fail: %s\n",devName);
  ioctl(fd, VMEIOCTL_VME_CHAN_FREE, &physAddr);
}
```

DMA Operations

The VME/PCI bridge includes two physical DMA channels that can copy data between PCI and VME buses. DMA bounce buffers are allocated if necessary, i.e. if the flags say the source or destination area is a virtual address.

Note that by default each DMA transfer is 4 kB (a page). Contact Kontron for other possibilities.

alma_user_dma

int alma_user_dma(u32 vmeaddr,unsigned long pciaddr,u32 size,u32 flags)

This executes a dma between the given vmeaddr and a pci, cpu physical or user virtual address, depending on the flags. It returns 0 on success.

vmeaddr is the VME physical address.

pciaddr is by default the DRAM logical address and DMA buffers are allocated. If the flag VMEFG_DMA_PADDR is set, it is the cpu physical DRAM address and DMA buffers are not allocated, but the dart or iommu is programmed and the address converted to a pci bus physical address. If the flag VMEFG_DMA_PCIADDR is set, it is the pci bus physical DRAM address and DMA buffers are not allocated, and no change is made to the address, no dart or iommu entries are made.

size is the size of the transfer in bytes. It is divided by 4 kB to get the number of DMA transfers.

flags contains the options for the channel. Options are 'ored' from the following:

VMEFG_DMA_VMEPCI the DMA is VME read and PCI write (default is PCI read VME write)

VMEFG_LEBE_*xx* define the little/big endian conversion mode (default is address coherency)

AM code. One of the predefined **VMEFG_AM_*xxx*** flags can be used. The default is VMEFG_AM_A32SDATA (0x0D AM code).

VMEFG_DMA_CHBKSIZ changes the number of VME cycles in each block (default is 8). Note that the maximum number of cycles in a block is 256 and that the number of cycles are numbered starting from 0 (0 = 1 cycle, 1 = 2 cycles, etc.). The number of cycles uses the bitmask 0x0000FF00, so the blocksize has to be shifted left 8 and then 'ored' with a flag. For example, to set 256 VME cycles in each block, 'or' ((256-1)<<8)|VMEFG_DMA_CHBKSIZ with the other flags. Usually performance is improved by increasing the blocksize.

This does as alma_user_dma() but uses 2ESST broadcast cycles. Flags should include VMEFG_AM_2ESST and VMEFG_XAM_A64BDCST. The select mask has a bit set for the geographic id of each remote board that should accept the broadcast (little-endian bits 1..21). See the section on 2ESST below.

User Space DMA Operations

ioctl(fd, **VMEIOCTL_DMA_START**, &A)

calls alma_user_dma(A.dma.vmeaddr, A.dma.data, A.dma.size, A.dma.flag).

ioctl(fd, **VMEIOCTL_DMA_START_MCAST**, &A)

calls alma_user_dma_mcast(A.mcast.vmeaddr, A.mcast.data, A.mcast.size, A.mcast.flag, A.mcast.-select). This feature is not available on Alma2f.

For example:

```
dma_arg.vmeaddr = strtoul(argv[1], NULL, 0); // VME address
size = strtoul(argv[2], NULL, 0);
dma_arg.dmaaddr = malloc(size); // DMA buffer
if (dma_arg.dmaaddr == NULL) {
  fprintf(stderr," Error allocating DMA buffer\n");
  exit(1);
}
dma_arg.size = size; // size
if (argc <= 3) // Optional channel
  dma_arg.flag = VMEFG_AM_A32SDATA|VMEFG_DMA_VMEPCI;
else
  dma_arg.flag = strtoul(argv[3],NULL,0);
printf(" VME Addr: 0x%x, DMA buf: 0x%p, Size: 0x%x, Flags: 0x%x\n",
               dma_arg.vmeaddr,dma_arg.dmaaddr, dma_arg.size, dma_arg.-
flag);
err = ioctl(fd, VMEIOCTL_DMA_START, &dma_arg);
if (err != -1) {
  unsigned int *ptr = (unsigned int *) dma_arg.dmaaddr;
```

```
    for (i = 0; i < dma_arg.size/28; i++) {
      printf("[0x%p] %x %x %x %x %x %x %x\n",
      ptr, ptr[0], ptr[1], ptr[2],
      ptr[3], ptr[4], ptr[5], ptr[6]);
      ptr += 7;
    }
  }
```

Contiguous Memory Allocation

The following routines allocate and free the physically contiguous memory needed for dma.

alma_phys_mem_alloc

int alma_phys_mem_alloc(struct alma_phys_mem_object *phys_mem)

This allocates a contiguous physical memory area. In the phys_mem parameter, only phys_mem->size needs to be set to the requested size, which can be up to 64 Mbytes. It returns 0 on success, -1 on failure. It can be called from user space with the ioctl **VMEIOCTL_PHYSICAL_MEM_ALLOC.**

alma_phys_mem_free

int alma_phys_mem_free(struct alma_phys_mem_object *phys_mem)

This frees a previously allocated contiguous physical memory area. In the phys_mem parameter, only phys_mem->cpu_phys_addr needs to be set to the address of the physical memory area. It returns 0 on success, -1 on failure. It can be called from user space with the ioctl **VMEIOCTL_PHYSICAL_MEM_FREE.**

User Space Contiguous Memory Allocation

ioctl(fd, **VMEIOCTL_PHYSICAL_MEM_ALLOC**, struct alma_phys_mem_object *phys_mem)

calls alma_phys_mem_alloc(phys_mem).

ioctl(fd, **VMEIOCTL_PHYSICAL_MEM_FREE**, struct alma_phys_mem_object *phys_mem)

calls alma_phys_mem_free(phys_mem).

For example (taken from the almavmechan tool):

```
mem_obj.size = ...;
rc = ioctl(fd,VMEIOCTL_PHYSICAL_MEM_ALLOC,&mem_obj);
printf("0x%x\n",mem_obj.cpu_phys_addr);
mem_obj.cpu_phys_addr = ...;
rc = ioctl(fd,VMEIOCTL_PHYSICAL_MEM_FREE,&mem_obj);
```

Locking

The VME bus can be explicitly locked before one or more accesses are done. It must always be paired with a free. These routines return 0 on success. On the VM605x board, the VME bus should be locked before issuing a simgle VME access concurrently with VME DMA accesses.

alma_get_vme

int alma_get_vme(void)

alma_free_vme

int alma_free_vme(void)

User Space Locking

ioctl(fd, **VMEIOCTL_GET_VME**, 0)

calls alma_get_vme()

ioctl(fd, **VMEIOCTL_FREE_VME**, 0)

calls alma_free_vme().

For example:

```
void fcop_withcapturevmebus(u32 *dst, u32 *src, u32 size){
  int ret = ioctl(fd,VMEIOCTL_GET_VME,0); // Capture the VME Bus
  if (ret==0) {
    fcop(dst,src,size); // Do the job
    ret = ioctl(fd,VMEIOCTL_FREE_VME,0); // Release the VME Bus
  }
}
```

VME/PCI Bridge Configuration

Some parameters can be changed in the VME/PCI bridge, such as request level, release mode, and arbitration.

alma_ctrl

int alma_ctrl(int command, int config)

If command is **ACTL_GET_CONFIG** the config is ignored, and the current configuration is returned. If command is **ACTL_SET_CONFIG,** the given config becomes the new VME/PCI bridge configuration, and 0 is returned. The config format is the following (bit 0 is the LSB):

request level is coded in the 3 LSB: ALMACFG_REQLEVELx for level x (from 0 to 3 inclusive). The default value is level 3.

fair mode is coded in bit 3 (ALMACFG_FAIRMODE). If active, the VME/PCI bridge does not request the VME if another VME device requests it on the same level.

bus release mode is coded in bits 4..6 (ALMACFG_BUSREL_MSK). Four modes are supported: ALMACFG_BUSREL_ROR (release on request), ALMACFG_BUSREL_RWD (release when done), ALMACFG_BUSREL_ROC (release on clear) and ALMACFG_BUSREL_RNE (release never). The default mode is ALMACFG_BUSREL_ROR.

fair mode timeout is coded in bit 7 (ALMACFG_FAIRTIMEOUT).

system controller is coded in bit 17 (ALMACFG_SYS_CTRL). It is READ ONLY. If active, the VME/PCI bridge is the VME system controller.

arbitration type is coded in bit 16 (ALMACFG_ROT_PRIO). If the VME/PCI bridge is the system controller, this bit sets the arbitration type: 0 fixed priority, 1 rotating priority.

timeout is coded in bits 8..15 (ALMACFG_TIMEOUTMSK). If the VME/PCI bridge is the system controller, these bits are the timeout value in microseconds for data transfer on the VMEbus. For example, if the timeout is 120 and if there is no response to a VME request after 120 microseconds, the VME/PCI bridge will generate a bus error on the VMEbus.

VME speed is coded in bit 20 and 21 with **ALMACFG_SPEED_***XXXX.* There are three modes: slow, medium and fast. It programs the device to different speeds for VME 2ESST cycles.

This routine can be called from user space with the ioctls **VMEIOCTL_SET_ALMACFG**

and **VMEIOCTL_GET_ALMACFG.** It is possible to get the VMEbus and release the VMEbus by software; alma_get_vme() gets the VMEbus and alma_free_vme() releases the VME bus. These routines can be called from user space with the ioctls **VMEIOCTL_GET_VME** and **VMEIOCTL_FREE_VME.**

User Space VME/PCI Bridge Configuration

ioctl(fd, **VMEIOCTL_SET_ALMACFG**, &A)

calls alma_ctrl(ACTL_SET_CONFIG,A.almacfg)

ioctl(fd, **VMEIOCTL_GET_ALMACFG**, 0)

returns alma_ctrl(ACTL_GET_CONFIG,0)

Read() and Write() Primitives

The VME driver read() and write() primitives offer direct user access to VMEbus space, using the AM code specified by the VMEIOCTL_SET_RWAM ioctl command. The DMA is used to do reads and writes without remapping of VME buffers in kernel or user space. /**dev**/**almavme** and any /**dev**/**vme***X* may be used to read/write on the VME. The usual lseek(2) system call can be used to change the vmeaddr.

User Space Read() and Write() Configuration

ioctl(fd, **VMEIOCTL_SET_RWAM**, &A)

sets the AM code to be used by the driver's read() and write() entry points from A.defam.

ioctl(fd, **VMEIOCTL_GET_RWAM**, &A)

returns in A.defam the current AM code used by the driver's read() and write() entry points.

For example:

```
vmeaddr = strtoul(argv[1],NULL,0); // VME address
size = strtoul(argv[2], NULL, 0);
buffer = malloc(size); // DMA buffer
if (buffer == NULL) {
   fprintf(stderr," Error allocating DMA buffer\n");
   exit(1);
}
if (argc <= 3) // Optional channel
   flags = VMEFG_AM_A32SDATA;
else
   flags = strtoul(argv[3],NULL,0);
printf(" VME Addr: 0x%x, DMA buf: 0x%p, Size: 0x%x, Flags: 0x%x\n",
                 vmeaddr, buffer, size, flags);
err = ioctl(fd, VMEIOCTL_SET_RWAM, &flags); // Set the AM
err = lseek(fd, vmeaddr, SEEK_SET); // Set vmeaddr offset
err = read(fd, buffer, size); // Read data into buffer
if (err != -1) {
  unsigned int *ptr = (unsigned int *) buffer;
  for (i = 0; i < size/28; i++) {
    printf("[0x%p] %x %x %x %x %x %x %x\n",
      ptr, ptr[0], ptr[1], ptr[2],
      ptr[3], ptr[4], ptr[5], ptr[6]);
    ptr += 7;
  }
}
```

Interrupt Service Calls

The VME driver associates interrupt vectors with handlers by calling the routines request_vmeirq() and free_vmeirq(). The VME irq level may be masked/unmasked using alma_vmeintcontrol().

request_vmeirq

int request_vmeirq(unsigned int vector, void *handler)

Assign a handler to a specific VME vector. If the vector is between 8 and 15, it is for an addressed interrupt (see below). Otherwise, it is a VME irq vector from 0 to 255.

free_vmeirq

int free_vmeirq(unsigned int vector)

Release the specified vector.

alma_vmeintcontrol

void alma_vmeintcontrol(unsigned int mask, unsigned int ienable)

This masks or unmasks one or several interrupt levels.

mask is a 16 bit bitfield where each bit is associated with an interrupt level except bit 0. Bit n corresponds to the interrupt level n on the VMEbus.

ienable is 0 to mask and non-zero to unmask all the interrupt levels set in mask.

disable_vmeirq

int disable_vmeirq(u32 level)

This deprecated routine disables a single VME interrupt level, from 1 to 7.

enable_vmeirq

int enable_vmeirq(u32 level)

This deprecated routine enables a single VME interrupt level, from 1 to 7.

alma_vmeinterrupt

int alma_vmeinterrupt(unsigned char level, unsigned char vector)

This sends an interrupt on the VMEbus. Vector must be a multiple of 8. The interrupt vector generated is (vector+level).

User Space Interrupt Service

ioctl(fd, **VMEIOCTL_VMEINTR_GEN**, &A)

calls alma_vmeinterrupt(A.intr.level,A.intr.vector)

ioctl(fd, **VMEIOCTL_VMEINTR_CTRL**, &A)

calls alma_vmeintcontrol(A.intrctl.mask,A.intrctl.enable)

ioctl(fd, **VMEIOCTL_VMEIOINTSET**, &A)

connects interrupt A.vector to the VME driver, and returns an identifier that can be used with VMEIOCTL_INTRWAIT and VMEIOCTL_VMEIOINTCLR.

ioctl(fd, **VMEIOCTL_VMEIOINTCLR**, &A)

disconnects interrupt A.index from the VME driver.

ioctl(fd, **VMEIOCTL_INTRWAIT**, &A)

waits for an interrupt connected by VMEIOCTL_VMEIOINTSET. A.index is the identifier returned by VMEIOCTL_VMEIOINTSET.

For example, generating an interrupt:

```
lvl = strtol(argv[1], NULL, 0); // VME IRQ level
vect = strtol(argv[2], NULL, 0); // VME vector
printf("Level = 0x%lx, Vector = 0x%lx\n", lvl, vect);
alma_ioctl.intr.level = (u32) (lvl&0xFFFFFFFF);
alma_ioctl.intr.vector = (u32) (vect&0xFFFFFFFF);
err = ioctl(fd, VMEIOCTL_VMEINTR_GEN, &alma_ioctl);
```

For example, waiting for an interrupt:

```
lvl = strtol(argv[1], NULL, 0);
vect = strtol(argv[2], NULL, 0);
alma_arg.vector = (vect&0xFF); // Vector
printf("Init handling for interrupt level %d and vector 0x%x (0x%x)\n",
       lvl, vect, alma_arg.vector);
fflush(stdout);
Id = ioctl(fd, VMEIOCTL_VMEIOINTSET, &alma_arg);
// Enable the interrupt lvl
alma_arg.intrctl.mask = (1 << lvl); // VME IRQ level
alma_arg.intrctl.enable = 1; // Enable the IRQ
err = ioctl(fd, VMEIOCTL_VMEINTR_CTRL, &alma_arg);
printf("Wait for interrupt level %d (vector 0x%x) id=%d...\n", lvl, vect,
Id);
fflush(stdout);
alma_arg.index = Id; // Id in the user vector table
err = ioctl(fd, VMEIOCTL_INTRWAIT, &alma_arg);
printf("RECEIVED\n");
fflush(stdout);
// Disable the interrupt lvl
alma_arg.intrctl.mask = (1 << lvl); // VME IRQ level
alma_arg.intrctl.enable = 0; // Disable the IRQ
err = ioctl(fd, VMEIOCTL_VMEINTR_CTRL, &alma_arg);
printf("Disable VME IRQ level %d\n", lvl);
alma_arg.index = Id; // Id in the user vector table
err = ioctl(fd, VMEIOCTL_VMEIOINTCLR, &alma_arg);
```

Addressed Interrupts

There is another way to transmit an interrupt across the VMEbus using addressed interrupts. A byte write into a special register of the VME/PCI bridge will generate an interrupt on the board. A driver can choose to discriminate between eight different sources of addressed interrupts. The VME/PCI bridge registers can be accessed from the VMEbus in A16 mode. The A16 base address is setup by firmware (VMEID) on a 256 byte boundary, and extends for 256 bytes. To generate an addressed interrupt, write a byte at address (ALMAbase + 0xEE). The byte should have a value of (1<<x), where x is from 0 to 7 inclusive, chosen by the programmer. To receive addressed interrupts use request_vmeirq().

For example:

```
vmea16_addr = strtol(argv[3], NULL, 0); // VME A16 addr
printf("Adressed VME interrupt %ld remote A16:0x%lx\n", lvl-8,
vmea16_addr);
alma_ioctl.vmechan.name = "GENintr"; // Name
alma_ioctl.vmechan.vmeaddr =  0; // VME Physical Address
alma_ioctl.vmechan.size = PCIVME_BLKSIZE; // As small as possible (8 MB)
```

```
alma_ioctl.vmechan.flag = VMEFG_AM_A16S;
physaddr = ioctl(fd, VMEIOCTL_VME_CHAN_ALLOC, &alma_ioctl);
vaddr = (unsigned long) mmap(0, vmea16_addr + 0x100 /*size of A16 window*/,
  PROT_READ | PROT_WRITE, MAP_SHARED, fd, (off_t)(physaddr&0xFFFFFFFF));
if(vaddr==-1){
   printf("A16 VME channel open fail\n");
   ioctl(fd, VMEIOCTL_VME_CHAN_FREE, &physaddr);
   exit(1);
}
printf("VME_addr:0x%lx paddr:0x%lx vaddr:0x%lx\n", vmea16_addr, physaddr,
vaddr);
write_addr = (char*) (vaddr + vmea16_addr + 0xee);
write_val = (char)(1<<(lvl-8));
printf("WRITING 0x%x at 0x%p\n", write_val, write_addr);
*write_addr = write_val;
sync();
printf("READING 0x%x at 0x%p\n", *write_addr, write_addr);
sync();
munmap((void *)vaddr, 0x100); // size of A16 window
ioctl(fd, VMEIOCTL_VME_CHAN_FREE, &physaddr);
```

Further IOCTL() Commands
The following commands are also available:

ioctl(fd, **VMEIOCTL_VME_CHAN_GETIO**, struct pcivme_window *p)

copies the PCI-IO to VME channel descriptions into the third arg. It is N structures of type struct pcivme_window, where N is the number of PCI-IO to VME channels opened.

ioctl(fd, **VMEIOCTL_VME_CHAN_GETMEM**, struct pcivme_window *p)

copies the PCI-MEM to VME channel descriptions into the third arg. It is N structures of type struct pcivme_window, where N is the number of PCI-MEM to VME channels opened.

ioctl(fd, **VMEIOCTL_VME_CHAN_GETNB**, &A)

returns in A.vmechannb.mem the number of PCI-MEM to VME channels opened and in A.vmechannb.io the number of PCI-IO to VME channels opened.

ioctl(fd, **VMEIOCTL_PCI_CHAN_GET**, struct vmepci_window *p)

copies the VME to PCI channel descriptions into the third arg. It is 7 structures of type struct vmepci_window.

ioctl(fd, **VMEIOCTL_PCI_CHAN_GETEXT**, struct vmepci_window *p)

copies the VME to PCI channel descriptions into the third arg. It is MAXVMEPCIWIN structures of type struct vmepci_window.

Sysfs Files

The driver provides files under the sysfs "vme" class directory, `/sys/class/vme/almavme.` These can be read or written.

/sys/class/vme/almavme/vme_bus_error
A read of this file returns as a string the number of VME bus errors seen by the driver. Any write will clear the counter.

2ESST
The Alma PCI/VME bridge implements 2ESST (Two Edge Source Synchronous Vme Transfer). This is

an extension to the VME64 standard to allow the VME bus to operate at much greater speeds of nominally 160, 267 and 320 Mbytes/second, between boards supporting the extension.

2ESST transfers use a new address modifier (AM) code, and so are ignored by boards that do not support them. The supervisory/non-privileged and program/data sub-modes are replaced by a new extended address modifier (XAM). The data transmitter uses both edges of the clock to strobe data and does not wait for any acknowedgement (two edge source synchronous). 64 bits (8 bytes) of data are clocked on each edge. Transmissions must use an even number of edges, so the minumum unit of transfer is 16 bytes, though it is possible to flag the last cycle as ignored. It is not optimal to transfer only 16 bytes of data. A maximum of 128 beats of 2 edges of 8 bytes, ie 2048 bytes is allowed per transfer.

The initiator specifies in a three-part setup phase the maximum transfer rate. The slave can terminate the transfer with an error if it is unable to perform at this rate. It can also suspend the transfer if it does not currently have the resources, and the transfer will be retried later. The address transferred during setup phase does not specify the low 4 bits, so addresses are always aligned on 16 bytes.

There are physical limitations imposed by the distance between boards, the length of the backplane, and the effect of other boards in the rack, including a broadcast selecting many boards or not, so 2ESST transfers might work in one direction, and not in the reverse direction between the same two boards. For example, the maximum speed of 320 Mbytes/second might only be possible in a rack of 6 slots, with only 4 boards in place.

The Alma bridge provides 8 extra VME slave channels, for incoming VME accesses, allocated by the usual alma_pci_channel_alloc() routine when the VMEFG_AM_2ESST flag is used. These can be used for 2ESST accesses or ordinary accesses. (The first 8 VME channels cannot be used for 2ESST).

The maximum 2ESST rate that a channel will accept can be specified. Higher rates will cause the transfer to fail. The XAM must also be given and must match that used for the dma transfer.

2ESST transfers can only be generated by dma using the usual alma_queue_dmalist() routine, or alma_user_dma() its broadcast version alma_user_dma_mcast(). They must be targeted to a 2ESST window with the same AM, XAM, and the same or a higher rate.

The rate chosen for the dma can be throttled globally to a lower value by ioctl VMEIOCTL_SET_ALMACFG or kernel service alma_ctrl(), or the almavmechan(1) option -setalmastate.

Addresses for 2ESST dma must be aligned on 16 bytes.

The 2ESST protocol also provides for a broadcast transfer, where one board can do a single write that is received simultaneously by several boards. The boards to receive the broadcast are selected during the address phase by their geographic id. Broadcast AM therefore need to provide a bitmask selecting the remote boards that are to receive the write. Each board has a geographic id which is the slot number in which it is positioned. The bitmask is an ”or” of 1<<n where n is the geographic id of a targeted board.

Select mask decoding by the slave is not currently supported by the Alma device, and it will accept all broadcasts, even if it is not selected. One workaround is to setup several separate slave A64 windows targetting the same physical memory, and only enable the appropriate ones for a given select mask.

The almavmechan(1) command has an option -dmabdcst to generate 2ESST broadcasts.

EXAMPLES

Refer also to the example files Dma.c, Dma2.c, GEN_intr.c, RECV_intr.c, MasterSgl.c in /usr/share/vmetoolkit/examples.

RESTRICTIONS

Currently, the driver and ioctls do not provide meaningful errno values on error.

SEE ALSO

almavmechan(1), mbm3k(1)
Alma2f User Manual CI.DT.A00

FILES

/dev/almavme
/dev/vme*X*

7.11.2 almavmechan

NAME

almavmechan − VME bus utility for almavme driver

SYNOPSIS

−vmememalloc *name vmeaddr size flag*
−vmeioalloc *name vmeaddr size flags*
−vmefree *cpu_addr*
−pcialloc *name vmeadr cpuadr size fg*
−pcifree *id*
−printall
−vmeioprint
−vmememprint
−pciprint
−vmeintrgen *level vector*
−vmeintrctrl *mask on/off*
−vmeiointset *vector*
−vmeiointclr *vector*
−vmeintwait *id*
−setamrw *value*
−vmeedit
−getalmastate
−setalmastate *val*
−editalmastate
−dmastart *vmeadr dramadr size fg*
−dmabdcst *vmeadr pwbadr size fg sel*
−dmawait *id*
−physmemalloc *size*
−physmemfree *phys_addr*

DESCRIPTION

almavmechan can create and delete PCI to VME or VME to PCI channels using the ALMA PCI to VME bridge. It prints the current channels opened. It can also read and modify the bridge's state. This utility uses the ioctl's of the VME bus driver described in almavme(4).

OPTIONS

−vmememalloc *name vmeaddr size flag*

allocate a PCIMEM to VME channel

−vmeioalloc *name vmeaddr size flags*

allocate a PCIIO to VME channel

−vmefree *cpu_addr*

free a PCI to VME channel

−pcialloc *name vmeadr cpuadr size fg*

allocate a VME to PCI channel

−pcifree *id*

free a VME to PCI channel

–printall

print all allocated channels.

–vmeioprint

print all PCIIO to VME channels.

–vmememprint

print all PCIMEM to VME channels.

–pciprint

print all VME to PCI channels.

–vmeintrgen *level vector*

send a VME interrupt.

–vmeintrctrl *mask on/off*

mask or unmask VME interrupts.

–vmeiointset *vector*

connect a VME interrupt.

–vmeiointclr *vector*

disconnect a VME interrupt.

–vmeintwait *id*

wait for a VME interrupt.

–setamrw *value*

set AM (address modifier) to use for accesses through standard read() and write() routines.

–vmeedit

use the VME data editor to do direct VME bus read/writes (with VME addresses).

–getalmastate

print the current Alma state.

–setalmastate *val*

set the Alma state.

–editalmastate

change the Alma state interactively.

–dmastart *vmeadr dramadr size fg*

start a DMA between VME and PCI.

–dmabdcst *vmeadr pwbadr size fg sel*

start a broadcast DMA.

–dmawait *id*

wait for the end of a DMA.

–physmemalloc *size*

allocate a contiguous physical memory area.

–physmemfree *phys_addr*

free a contiguous physical memory area.

EXAMPLES

To create a PCI–MEM to VME channel at VME address 0x10000000, size of 8 MB with read ahead and write posting modes:

```
[root@ki7]# almavmechan –vmememalloc GREAT 0x10000000 0x8000000 0x3
  0xd2000000
```

The result 0xd2000000 is the CPU physical address needed to access VME address 0x10000000.

To print all opened channels and the state of the ALMA bridge:

```
[root@ki7]# almavmechan –printall


------------------------------------------------------------------------
                     No PCIIO to VME CHANNEL
------------------------------------------------------------------------
------------------------------------------------------------------------
                     PCIMEM to VME CHANNELS
------------------------------------------------------------------------
  Name      CPUaddr    PCIaddr    VMEaddr  size(Mb)    AM    Conv  WP  RH
------------------------------------------------------------------------
GREAT        0xd2000000 0x52000000 0x10000000  0128    A32SDATA ADDR  Yes
Yes
------------------------------------------------------------------------
------------------------------------------------------------------------
                     No VME to PCI CHANNEL
------------------------------------------------------------------------
Alma state:
        System controller: fixed priority.
                           timeout 100
        Fair mode timeout.
        Not Fair mode.
        VME bus release mode :  ROR
        Request level : BR3
        Dead lock logic ON.
        2ESST rate  FAST
```

SEE ALSO

almavme(4), mbm3k(1)

7.11.3 mbm3k

NAME

mbm3k − multiple benchmark manager

SYNOPSIS

mbm3k *[flags] command args*

Commands and their args:
cpu *period_10ms*
cpusgl r|w|v|v2|rv|vi d|n *vmeaddr size loops*
dmasgl r|w|v|v2|rv|vs|sv *vmeblksz vmeaddr size loops*
dmablt r|w|v|v2|rv|vs|sv *vmeblksz vmeaddr size loops*
dmamblt r|w|v|v2|rv|vs|sv *vmeblksz vmeaddr size loops*
dma2esst[320|267|160] [k]r|w|v|v2|rv|vs|sv *vmeblksz vmeaddr size loops [start][end]*
dma2ebdcst[320|267|160] [k]w *vmeblksz vmeaddr size loops [start][end]*
scsi r|w|v|v2 r|b *dir size loops*
net_client *port_number* **s|a** *preload i_addr size loops*
net_server *port_number*
help

Flags:
−f *logfile*
−p *physaddr*
−b

DESCRIPTION

mbm3k is a collection of benchmarking tools for the DRAM, VME, NETWORK, and SCSI busses.

Commands
cpu Evaluate the processor idleness and print the cpu usage every *period_10ms*/100 seconds.

net_server starts the network benchmark server. **net_client** should then be started on a different board to transfer data via a socket between the two processors. The same *port_number* must be specified for both commands. *i_addr* is the server's IP address, needed by the client.

scsi. The SCSI benchmark creates a file in directory *dir* (raw file or block file) and reads and writes this file.

The VME benchmarks **cpusgl ... dma2ebdcst** access the VME bus at the address *vmeaddr.* This can be done using the processor **cpu**xxx, or the ALMA DMA engine **dma**xxx. Several modes are available: xxx**sgl** for a single word transfer, xxx**blt** for block transfers, xxx**mblt** for 64bit block transfers, and VME 2−Edge Source Synchronous transfers at a given speed, **dma2esst**xxx and **dma2ebdcst**xxx.

OPTIONS

Flags
Flags must be given before the command.
−f *logfile* use given logfile (default mbm3k.log).
−p *physaddr* use a physical buffer as source of dma data.
−b capture VME bus before single VME access, for **cpusgl.**

Key Letters
The key letter args have the following meaning: **r|w|v|v2|rv** read|write|verify|verify twice|read verify
d|n Deadlock | No deadlock
s|n Snoop | No snoop
r|b Row | Block

s|a Sync | Async
k use KIOBBUF for io

Other Args
Other command args should be numbers in either decimal, or octal with a leading 0, or hexadecimal with a leading 0x. *loops* is the number of times to run the test. *start* and *end* say which test pattern should be used.

SEE ALSO

almavmechan(1), almavme(4)

7.12 CPLD

NAME

cpld – Kontron board facilities, cpld_i2c, cpld_smi, cpld_leds, cpld-gpio, cpld-wdt

DESCRIPTION

This man page describes some of the facilities made available by the cPLD (complex programmable logic device) on several Kontron boards. Not all boards provide all the facilities.

The cPLD is a small device that controls some low – level aspects of the board, including power up sequencing, reset, gpios, i2c buses including one for communication to the backplane, timer, watchdog, leds, and many configuration and control features.

The device is visible through 1 – byte registers in i/o port space from addresses 0x0800 to 0x08FF. The facilities are implemented in several drivers, each in its own rpm. The driver sources are provided.

cpld
The cpld driver is needed by the other drivers. It provides the geographical id (slot number) of the board in the file /**proc**/**geo_id**.

Some other basic features can be accessed with the cpldtool(1) utility, or by direct i/o on the appropriate port. For example, use dd on /**dev**/**port** or the **port** command (from rpm **hwtools**) as user root (beware: inappropriate i/o on ports may crash the system). As an example port 0x800 holds the version number of the cpld:

```
[root@ki7]# port 0x800
@0x800 = 0x06
```

Reset cpuB (only on VX6060)

On boards with two separate cpus, you can reset cpuB from cpuA or vice versa by setting lsb bit 0 to 0 in port 0x804. This is most easily done with the cpldtool utility:

```
[root@ki7]# cpldtool -f PWR_RST_CONFIG Software_Cross_Reset 0
```

The alternative is to read the port to find the current value, calculate the new value obtained by clearing bit 0 (which should always read as 1), and write the new value back. The bit will return to 1 on its own.

Access cpuB Serial Port (only on VX6060)

On boards with two separate cpus, you can enable access to cpuB's second console serial port from cpuA by dynamically setting the appropriate bit in cpld register 0xc, or permanently by configuring the bios. The two /dev/ttyS1 devices on the board are then linked together and can communicate via some utility like minicom or pyserial's miniterm.py:

```
[root@ki7]# cpldtool -f BOARD_CONFIGURATION SERIAL2_cfg 1
[root@ki7]# miniterm.py /dev/ttyS1 115200
```

To configure this permanently with the bios on cpuA follow the menus:

```
Kontron
Serial Configuration
COM1 Link Mode: [Enabled]
```

To see the cpuB bios on ttyS1, configure the cpuB bios with:

```
Advanced
Serial Port Console Redirection
```

```
COM1
Console Redirection [Enabled]
```

To have Linux use the ttyS1 console ensure it has the boot option:

```
console=ttyS1,115200
```

cpld_i2c
The cpld_i2c driver implements an i2c algorithm made available through the standard OS i2c interface.

cpld_smi (only on VX6060)
The cpld_smi driver provides a file and utility to program the onboard ethernet switch over the smi (Serial Management Interface) bus. The device /**dev**/**cpld_smi** has 2 ioctls to read and write a given register in the switch, snd the **swsmi** utility uses this device.

cpld_leds
The cpld_leds driver provides standard OS led class devices in /**sys**/**class**/**leds**/. They have filenames of the form *devicename:colour*.

Each cpu has 2 user settable leds, with devices named **led2** and **led3**. On the VM6050, each cpu also has 2 additional user settable leds, named **led4** and **led5**.

They implement 3 brightness levels by writing a value to *devicename:colour*/brightness

- 0 on
- 1 slow blink
- 2 fast blink

This switches each LED to user mode. Write 0 to *devicename*:OFF/brightness to switch an LED off, or any other value to exit from user mode. See the discussion in file **.../Documentation/leds – class.txt** (from rpm **kernel – doc**).

cpld – gpio
See the separate man page for the driver providing gpios implemented by the cpld.

cpld – wdt
See the separate man page for the driver providing a watchdog implemented by the cpld.

SEE ALSO
cpld – gpio(4)
cpld – wdt(4)
cpldtool(1)

FILES
/proc/geo_id
/dev/cpld_smi
/dev/port
/dev/i2c – 0 ...

COPYRIGHT

Kontron

Chapter 8 - RC Boards

8.1 How to Manage the Lack of RTC Battery

Some hardware constraints may prevent the use of the RTC's battery on the boards. This implies to workaround or disable some standard behaviour of Fedora services. The very first one is "`fsck`" which is run at boot time. `fsck` checks that the date of the last mount of the checked partition is prior to the current date. If not, the boot is stopped in a maintenance mode. This could happen after with the power off and without the RTC's battery. To workaround this, simply create a file named `/etc/e2fsck.conf` with the following content:

```
[option]
broken_system_clock=true
```

The same file has to be added to the `initramfs` too. To do this, simply run the command:

```
[root@ki7]# dracut --force
```

Without the right date for the system, some other services and functions may present problems. For example, the command "`make`" warns if there is some previous build detected in the future and this could lead to bad recompilation. If the network is available, it is recommended to setup the service `ntpd` to adjust the system date with a date server.

8.2 External Devices Connection

One impact of the RC board is the missing of front panel.

As a consequence, devices requiring access to the system such as:

- Serial line console.
- USB mouse
- USB keyboard
- Display monitor (through the mini DP or VGA)
- Ethernet

will be plugged in from the rear side through a Rear Transition Module (RTM) Paddle board.

The Kontron VM6050-RTM (Order Code: PBV36-P0-VM6-00) rear transition module is compliant to PMC I/O Module Standard VITA 36 - 199x Draft 0.1 July 19, 1999 (mechanical and PIM format) and is available for VM6050 boards.

For example, in order to perform a graphical linux installation stage on a VM6050-RC, plug in the required devices to the paddle board:

- USB mouse and USB keyboard through a USB HUB.
- The display monitor (through the mini DP).

After turning on the system, you should notice all devices have been recognized correctly allowing to take control of the graphical environment.

8.3 RC Specifications

The RC version of the boards is designed to work in different environmental constraints and for different levels of temperature and power dissipation.

This may imply specific setups (cpu frequencies, hardware parameters,…) to guarantee the specified behavior.

They are described in the “Hardware User’s Guide” of each board. Please check this document for your board and for the required environment.

Chapter 9 - Power Management

9.1 Introduction

At the core of power management is an understanding of how to effectively optimize energy consumption of each system component.

By studying the different tasks that your system performs, and configuring each component to ensure that its performance is just sufficient for the job, you can save energy, generate less heat.

Many of the principles for analysis and tuning of a system in regard to power consumption are similar to those for performance tuning.

To some degree, power management and performance tuning are opposite approaches to system configuration, because systems are usually optimized either towards performance or power.

Two types of tools are available:

- The ones to set-up the power management configuration:
  - BIOS menu and `cpufreq` (under linux).
- The ones to evaluate the impact of this setting on the system:
  - `powertop` under linux.

9.2 Power Management Setting

9.2.1 Under BIOS

A way to reduce the power consumption (the drawback being the decreasing of performance) is to disable the **Hyper-Threading** mode as well as the **Turbo-Mode**.

To disable the **Hyper-Threading**:

- Enter the bios and select the submenu **Advanced -> CPU Configuration**
- Then set the option **hyper threading** to **Disabled**.

To disable the Turbo Mode

- Enter bios and select the submenu **Advanced -> CPU PPM Configuration**
- Then set the option **Turbo mode** to **Disabled**.

An alternative is to use the BIOS TDP menu:

- This option is specific to the VX3044 board.

The thermal design power (TDP), sometimes called thermal design point, refers to the maximum amount of power the cooling system in a computer is required to dissipate.

The move to a 22nm process and Tri-Gate transistors alone should already account for some pretty significant power savings. But there are a few other changes in Ivy Bridge meant to optimize power consumption.

An important addition brought to mobile Ivy Bridge processors is the inclusion of a configurable TDP that allows them to switch between three different ratings:

nominal, a lower configurable TDP and an upper configurable TDP.

The lower configurable TDP implies the lowest power comsumption of course.The BIOS TDP menu is accessible under `enhanced/CPU Configuration/CPU PPM` Configuration.

9.2.2 Under Linux

The main power management tool available under linux is CPUFreq.

CPUfreq allows the clock speed of the processor to be adjusted on the fly.

This enables the system to run at a reduced clock speed to save power.

Different types of CPUfreq governors are available:

```
cpufreq_performance
```

The performance governor forces the CPU to use the highest possible clock frequency (no power saving benefit at all).

```
cpufreq_powersave
```

By contrast, the Powersave governor forces the CPU to use the lowest possible clock frequency.

```
cpufreq_ondemand
```

The ondemand governor is a dynamic governor that allows the CPU to achieve maximum clock frequency when system load is high and also minimum clock frequency when system is idle.

This is the default mode (best compromise between heat emission,power consumption,performance and manageability).

```
cpufreq_userspace
```

Allows userspace program to set the frequency. Used normally in conjunction with the cpuspeed daemon.

```
cpufreq_conservative
```

Similar to the cpufreq_ondemand mode but this mode switches between frequencies more gradually. Boot the target under linux and log as root.

At first, you can view which governor the system is currently using with the command:

```
[root@ki7]#cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

The result will be probably:

ondemand because this is the default value.

You can also view which governors are available for the CPUs:

```
[root@ki7]#cat
/sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
```

At this step, select the governor mode which fits the best your need with the command:

```
[root@ki7]#echo xxxx     > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

Control the result by:

```
[root@ki7]#cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

An alternative consists of selecting an explicit frequency by using a command such as:

```
[root@ki7]# cd /sys/devices/system/cpu
[root@ki7]# for i in 0 1 2 3
do ; cd cpu$i ; echo 1300000 > cpufreq/scaling_max_freq; cd ..; done
```

This sets frequency 1.3 GHZ for all the CPUs. If you want to know the set of available frequencies:

```
[root@ki7]# cd /sys/devices/system/cpu
[root@ki7]# for i in 0 1 2 3
do ; cd cpu$i ; cat cpufreq/scaling_available_frequencies; cd ..; done
```

It is also possible to edit the file `/etc/sysconfig/cpuspeed` to set governor, min and max frequencies as required.

Then restart `cpuspeed` service through:

```
service cpuspeed restart
```

Chapter 10 - Additional Information

10.1 Known Limitations

SUSPEND Mode not supported

The SUSPEND mode is not supported by the current graphic chipset hardware.

So, follow this procedure in order to disable it:

- Create a new configuration file under `/etc/polkit-1/localauthority`

```
# vi /etc/polkit-1/localauthority/50-local.d/50-disable-suspend.pkla
```

- Copy the following code into that file:

```
[Disable Suspend]
Identity=unix-user:*
Action=org.freedesktop.upower.hibernate;org.freedesktop.upower.suspend
ResultAny=no
ResultInactive=no
ResultActive=no
```

- Finally check that the permission was successfully revoked

```
$ pkcheck --action-id org.freedesktop.upower.suspend --process $$

Not authorized.
```

If you get no output from the `pkcheck` command (make sure to run it as a normal user, not root) the permission is still there.

That is it, after re-login the annoying Suspend will be replaced by a friendly Power Off.

MAILING ADDRESS
Kontron Modular Computers S.A.S.
150 rue Marcelin Berthelot - BP 244
ZI TOULON EST
83078 TOULON CEDEX - France

TELEPHONE AND E-MAIL
+33 (0) 4 98 16 34 00
Sales: Order-ATD-Toulon@Kontron.com
Support: GSS-ATD-Toulon@Kontron.com

For further information about other Kontron products, please visit our Internet web site: www.kontron.com.