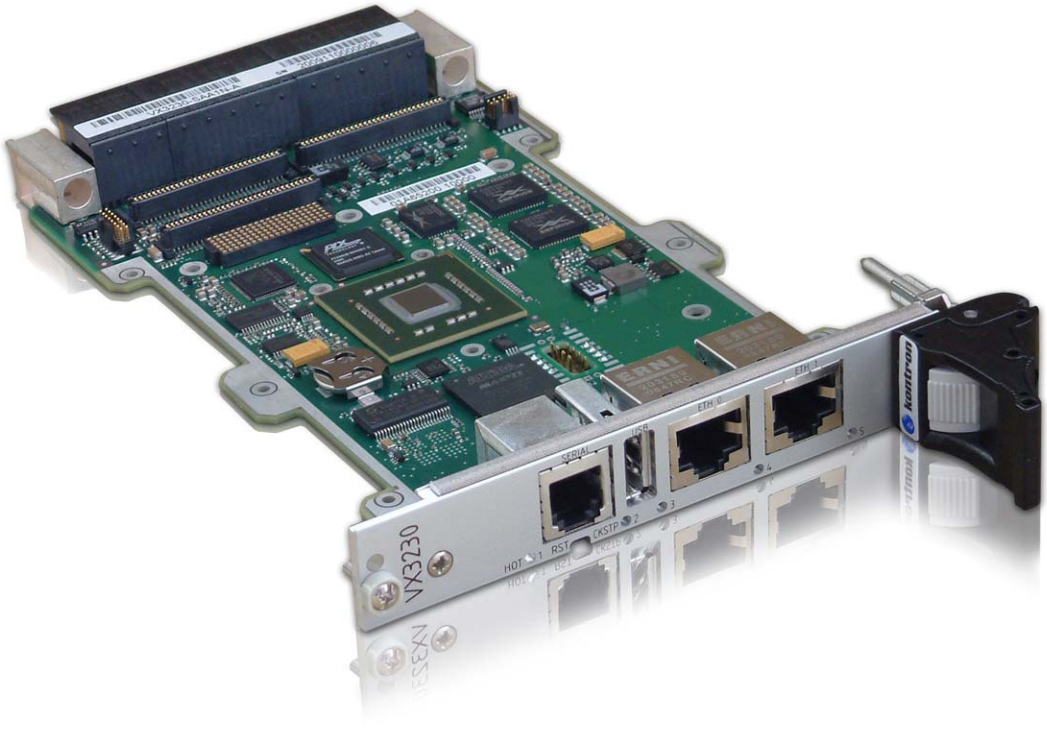
» VX3230 «



U-Boot User Manual

SD.DT.F46-0e - October 2009

If it's embedded, it's Kontron.

Revision History

Publication Title:	VX3230 U-Boot User Manual	
Doc. ID:	SD.DT.F64-0e	
Rev.	**Brief Description of Changes**	**Date of Issue**
0e	Initial Version	10-2009

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 on the previous page of this manual.

Table Of Contents

Chapter 1 - U-Boot Overview

1.1 General Purpose

This document describes the firmware available on the Kontron VX3230 board.

The firmware used on the VX3230 board is based on U-Boot 1.3.3 from U-Boot Project and the first Kontron release is U-Boot 1.3.3 ID 09268.

The VX3230 U-Boot firmware includes commands to:

- Display and modify the memories and registers (Flash, SRAM, DDR SDRAM, Registers, ...)
- Access the devices
- Boot Operating Systems (Linux, VxWorks, ...)
- Run the Power-on Self-Test (POST)
- Configure the boot of the board

1.2 U-Boot Project

The U-Boot project is fully described on the web site: http://www.denx.de/wiki/U-Boot.

1.3 Associated Documentation

Kontron Documentation

VX3230 3U VPX Board User's Guide	CA.DT.A63
VX3230 Hardware Release Notes	CA.DT.A64
VX3230 U-Boot User Manual (this manual)	SD.DT.F46
VX3230 PBIT User's Guide	SD.DT.F48
VX3230 Release Notes Fedora 9	SD.DT.F47

DENX Software Engineering Documentation

Documentation available at	http://www.denx.de/en/Documents/WebHome

1.4 U-Boot for VX3230

1.4.1 Overview

The U-Boot Mapping on System Flash is fully described in section 1.4.2 "U-Boot Mapping on System Flash" page 3.

The U-Boot firmware may be used in Rescue Mode or User Mode, depending on the VX3230 configuration.

Refer to Chapter 2 “U-Boot in User Mode" page 5, or Chapter 3 “U-Boot in Rescue Mode" page 7 for detailed information.

Power-On Built-In Test (PBIT) are available on the VX3230 if customer has selected this option. Refer to document SD.DT.F48 for detailed information on PBIT.

For a VX3230 board @1GHz with 1 GB of memory, the boot time:

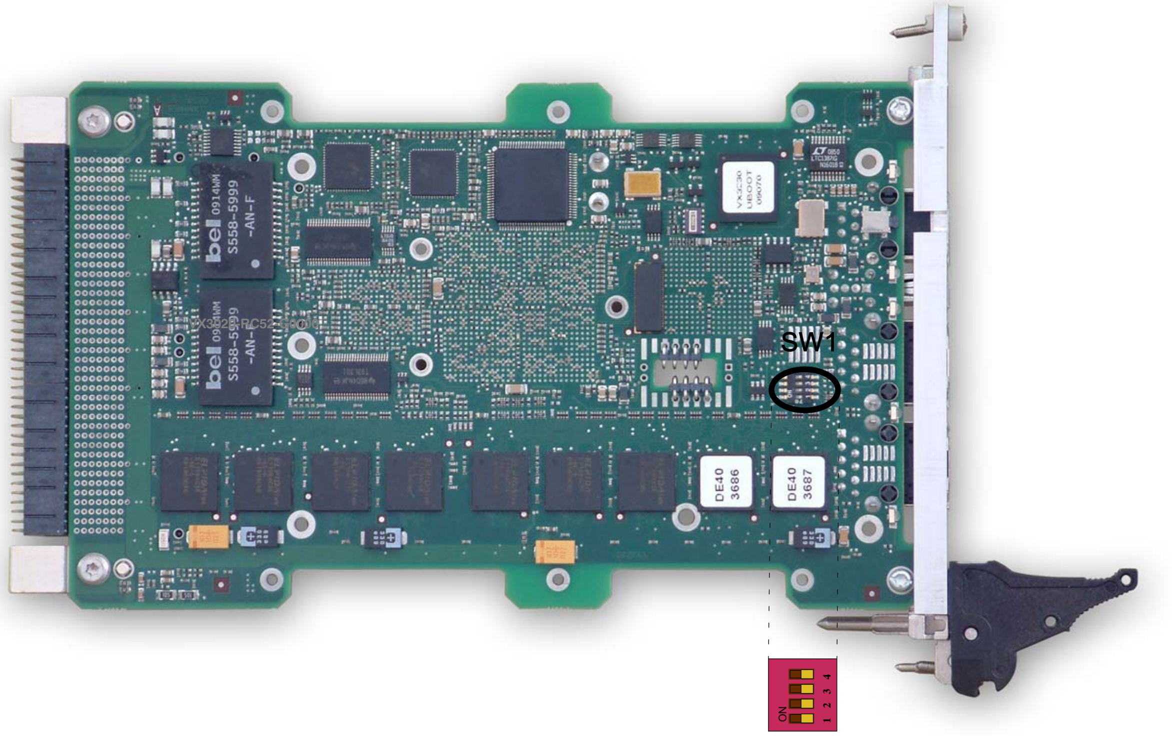
- without running any power-on self-test is ~ 11s
- Refer to SD.DT.F48 documentation for boot time including PBIT execution.

1.4.2 U-Boot Mapping on System Flash

The VX3230 has 4 MB system flash dedicated to U-Boot.

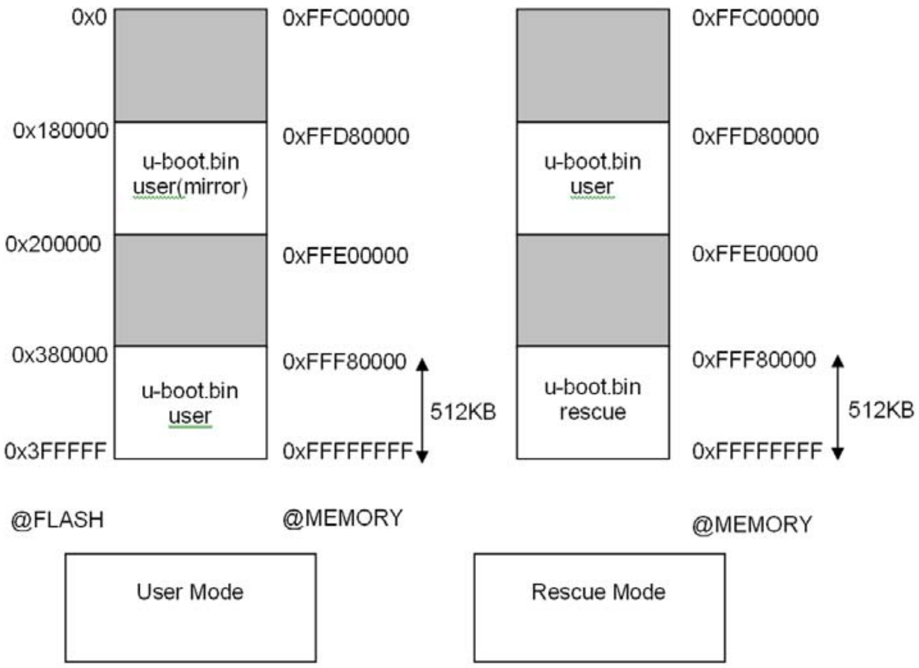
Depending of the DIP Switch SW1 setting, either the full content can be seen, or the last 2 MB are mirrored to the first 2 MB :

- When DIP Switch SW1, location 2 (SW1_2: Flash Boot Mode) is set to OFF, the FLASH #0 is the boot system flash (MAIN Flash) and U-Boot prompt is **VX3230>**
- When DIP Switch SW1, location 2 (SW1_2: Flash Boot Mode) is set to ON, the FLASH #1 is the boot system flash (RESCUE Flash) and U-Boot prompt is **VX3230-RESCUE>**



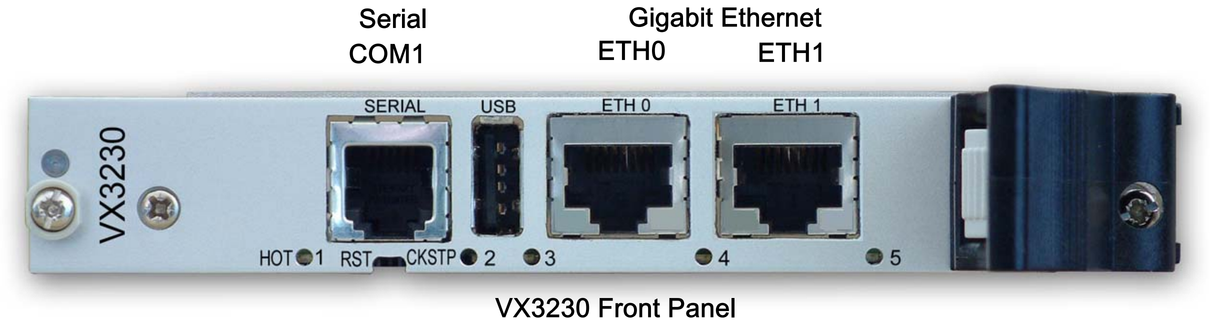
DIP Switch SW1	Function	Description
1	PCI-E EE CS#	ON (0) PCI-E EEPROOM R/W Enabled OFF (1) PCI-E EEPROM R/W Disabled
2	Flash Boot Mode#	ON (0) Boot in Rescue Mode OFF (1) Boot in Standard Mode
3	Boot Flash WP#	ON (0) Boot Flash Write Protected OFF (1) Boot Flash Write Enabled
4	Factory Mode#	ON (0) Factory Mode OFF (1) Normal Mode

System Flash Mapping

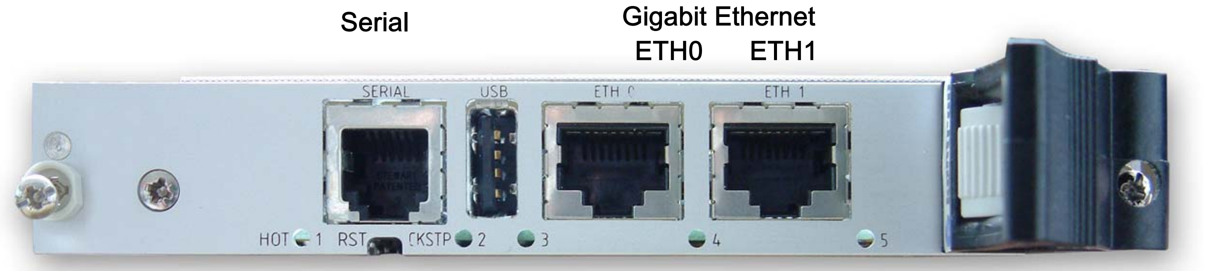


Chapter 2 - U-Boot in User Mode

Connect a serial console to the VX3230 board with serial speed set to 115200 bauds and an Ethernet cable either on the front panel or via Rear Transition Module (RTM) depending on the VX3230 board class (SA/RC).



VX3230 Front Panel



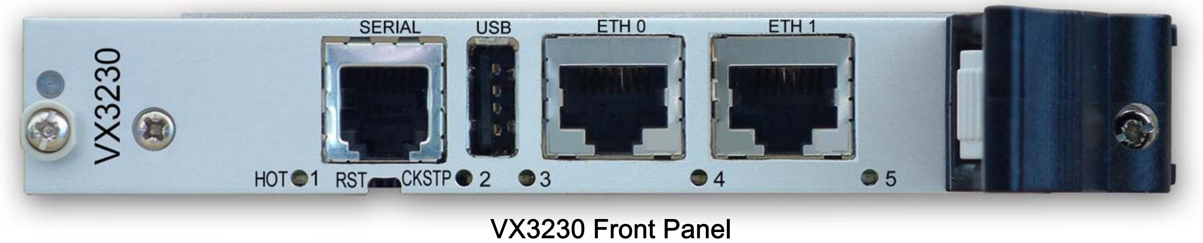
VX3230-RTM Front Panel

Traces displayed when the VX3230 is booted up through the serial console in User mode with SW1_2 set to OFF:

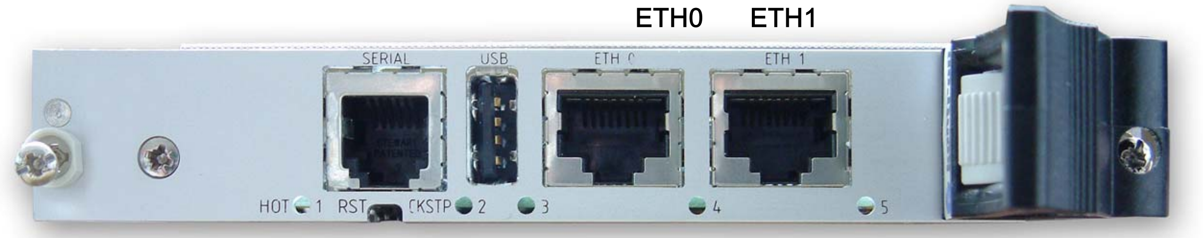
```
U-Boot 1.3.3 ID09268 for KONTRON VX3230 board

CPU:   8533_E, Version: 1.1, (0x803c0011)
Core:  E500, Version: 2.2, (0x80210022)
Clock Configuration:
       CPU: 800 MHz, CCB: 400 MHz,
       DDR: 200 MHz (400 MT/s data rate), LBC:  25 MHz
L1:    D-cache 32 kB enabled
       I-cache 32 kB enabled
Board: CANNES
I2C:   ready
DRAM:  Initializing
        -> DDR: Init mem CTL 1 done
    DDR: 1024 MB
L2 cache 256KB: enabled
VPD EEPROM:   ready
FLASH:  4 MB
    eTSEC1 is in sgmii mode.
    eTSEC3 is in sgmii mode.

    PCIE3 connected to SATA as Root Complex (base address e000b000)
               Scanning PCI bus 01
        Bus: 01  Dev: 00  Ven_ID: 1095  Dev_ID: 3132  Class: 0180  Int: 0x08 (8)
    PCIE3 on bus 00 - 01
    PCIE1 connected to Slot2 as Root Complex (base address e000a000)
    PCIE1 on bus 02 - 02
    VPX System Controller (GA = 30)

    PCIE2 connected to Slot 1 as Root Complex (base address e0009000)
               Scanning PCI bus 04
        Bus: 05  Dev: 01  Ven_ID: 10b5  Dev_ID: 8608  Class: 0604  Int: 0x06 (6)
        Bus: 05  Dev: 05  Ven_ID: 10b5  Dev_ID: 8608  Class: 0604  Int: 0x06 (6)
        Bus: 05  Dev: 07  Ven_ID: 10b5  Dev_ID: 8608  Class: 0604  Int: 0x06 (6)
        Bus: 05  Dev: 09  Ven_ID: 10b5  Dev_ID: 8608  Class: 0680  Int: 0x04 (4)
        Bus: 04  Dev: 00  Ven_ID: 10b5  Dev_ID: 8608  Class: 0604  Int: 0x07 (7)
    PCIE2 on bus 03 - 08

    PCI: 32 bit, 33 MHz, async, host, arbiter (base address e0008000)
               Scanning PCI bus 09
        Bus: 09  Dev: 11  Ven_ID: 1131  Dev_ID: 1561  Class: 0c03  Int: 0x02 (2)
        Bus: 09  Dev: 11  Ven_ID: 1131  Dev_ID: 1561  Class: 0c03  Int: 0x02 (2)
        Bus: 09  Dev: 11  Ven_ID: 1131  Dev_ID: 1562  Class: 0c03  Int: 0x02 (2)
PCI on bus 09 - 09
In:    serial
Out:   serial
Err:   serial
Net:   eTSEC1, eTSEC3
SATA:
        SATA0 : No SATA Device Found
        SATA1 : No SATA Device Found
VX3230 =>
```

Chapter 3 - U-Boot in Rescue Mode

Connect a serial console to the VX3230 board with serial speed set to 115200 bauds and an Ethernet cable either on the front panel or via Rear Transition Module (RTM) depending on the VX3230 board class (SA/RC).



VX3230 Front Panel



VX3230-RTM Front Panel

Traces displayed when the VX3230 is booted up through the serial console in Rescue mode with SW1_2 set to ON:

```
U-Boot 1.3.3 ID09268 for KONTRON VX3230 board

CPU:   8533_E, Version: 1.1, (0x803c0011)
Core:  E500, Version: 2.2, (0x80210022)
Clock Configuration:
       CPU: 800 MHz, CCB: 400 MHz,
       DDR: 200 MHz (400 MT/s data rate), LBC:  25 MHz
L1:    D-cache 32 kB enabled
       I-cache 32 kB enabled
Board: CANNES
I2C:   ready
DRAM:  Initializing
        -> DDR: Init mem CTL 1 done
    DDR: 1024 MB
L2 cache 256KB: enabled
VPD EEPROM:   ready
FLASH:  4 MB
    eTSEC1 is in sgmii mode.
    eTSEC3 is in sgmii mode.

    PCIE3 connected to SATA as Root Complex (base address e000b000)
               Scanning PCI bus 01
        Bus: 01  Dev: 00  Ven_ID: 1095  Dev_ID: 3132  Class: 0180  Int: 0x08 (8)
    PCIE3 on bus 00 - 01
    PCIE1 connected to Slot2 as Root Complex (base address e000a000)
    PCIE1 on bus 02 - 02
    VPX System Controller (GA = 30)

    PCIE2 connected to Slot 1 as Root Complex (base address e0009000)
               Scanning PCI bus 04
        Bus: 05  Dev: 01  Ven_ID: 10b5  Dev_ID: 8608  Class: 0604  Int: 0x06 (6)
        Bus: 05  Dev: 05  Ven_ID: 10b5  Dev_ID: 8608  Class: 0604  Int: 0x06 (6)
        Bus: 05  Dev: 07  Ven_ID: 10b5  Dev_ID: 8608  Class: 0604  Int: 0x06 (6)
        Bus: 05  Dev: 09  Ven_ID: 10b5  Dev_ID: 8608  Class: 0680  Int: 0x04 (4)
        Bus: 04  Dev: 00  Ven_ID: 10b5  Dev_ID: 8608  Class: 0604  Int: 0x07 (7)
    PCIE2 on bus 03 - 08

    PCI: 32 bit, 33 MHz, async, host, arbiter (base address e0008000)
               Scanning PCI bus 09
        Bus: 09  Dev: 11  Ven_ID: 1131  Dev_ID: 1561  Class: 0c03  Int: 0x02 (2)
        Bus: 09  Dev: 11  Ven_ID: 1131  Dev_ID: 1561  Class: 0c03  Int: 0x02 (2)
        Bus: 09  Dev: 11  Ven_ID: 1131  Dev_ID: 1562  Class: 0c03  Int: 0x02 (2)
PCI on bus 09 - 09
In:    serial
Out:   serial
Err:   serial
Net:   eTSEC1, eTSEC3
SATA:
        SATA0 : No SATA Device Found
        SATA1 : No SATA Device Found
VX3230 -RESCUE=>
```

Chapter 4 - Environment Parameters

To print and set the environment variables, we recommend to use following commands under the U-Boot User Mode. These commands are also described in Chapter 7 “U-Boot Command Line Interface" page 48.

4.1 Print the Current Environment Parameters

The U-Boot is delivered on the VX3230 with default parameters. These parameters are displayed by entering the following command:

```
VX3230 => printenv
```

Below an example of display on VX3230 board:

```
 bootcmd=setenv bootargs root=/dev/$bdev rw console=$consoledev,$baudrate
$othbootargs;tftp $loadaddr $bootfile;tftp $fdtaddr $fdtfile;bootm $loadaddr - $fdtaddr
ramboot=setenv bootargs root=/dev/ram rw console=$consoledev,$baudrate $othbootargs;tftp
$ramdiskaddr $ramdiskfile;tftp $loadaddr $bootfile;tftp $fdtaddr $fdtfile;bootm $loadaddr
$ramdiskaddr $fdtaddr
nfsboot=setenv bootargs root=/dev/nfs rw nfsroot=$serverip:$rootpath
ip=$ipaddr:$serverip:$gatewayip:$netmask:$hostname:$netdev:off
console=$consoledev,$baudrate $othbootargs;tftp $loadaddr $bootfile;tftp $fdtaddr
$fdtfile;bootm $loadaddr - $fdtaddr
baudrate=115200
loads_echo=1
ethaddr=00:00:DE:40:36:4A
eth1addr=00:00:DE:40:36:4B
ipaddr=192.93.159.142
serverip=192.93.167.102
rootpath=/nfs/mpc85xx
gatewayip=192.93.159.46
netmask=255.255.0.0
hostname=8544ds_unknown
bootfile=8544ds/uImage.uboot
loadaddr=1000000
netdev=eth0
uboot=cannes.bin
tftpflash=tftpboot $loadaddr $uboot; update user $loadaddr
consoledev=ttyS0
ramdiskaddr=2000000
ramdiskfile=8544ds/ramdisk.uboot
fdtaddr=c00000
fdtfile=8544ds/mpc8544ds.dtb
bdev=sda3
ethact=eTSEC1
bootdelay=-1
stdin=serial
stdout=serial
stderr=serial

Environment size: 1156/2044 bytes
VX3230 =>
```

4.2 Set Ethernet Parameters

The VX3230 board has two Ethernet interfaces that the user needs to initialize before updating the firmware using network interface. Depending on the VX3230 board class (SA or RC), the Ethernet ports are accessible either on front panel or on rear P0 backplane via Rear Transition Module (RTM).

The two front panel Ethernet ports are named eTSEC1 and eTSEC3 and the rear Ethernet ports are named eTSEC2 and eTSEC4.

In order to update firmware from network interface, set the following Ethernet parameters:

ipaddr: IP address of the VX3230

gatewayip: IP address of the gateway between the server and the VX3230

serverip: IP address of the server

> Example:

VX3230 IP address:	192.93.167.109
Gateway IP address:	192.193.167.46
Server IP address:	192.93.167.102

```
VX3230 => setenv ipaddr 192.93.167.109
VX3230 => setenv gatewayip 192.93.167.46
VX3230 => setenv serverip 192.93.167.102
```

At this point, all parameters are valid and saved in the SDRAM U-Boot area. But, these parameters will be lost if the board is switched off.

So, to definitely store those parameters in a non-volatile memory, the user needs to type the following command:

```
VX3230 => saveenv
Saving Environment to EEPROM...
VX3230 =>
```

All saved parameters in the non-volatile memory will be restored at the next startup of the board.

To check that the parameters are properly set, try a basic ping command under U-Boot by typing:

```
VX3230 => ping 192.93.167.102
Enet starting in 1000BT/FD
Speed: 1000, full duplex
Using eTSEC2 device
host 192.93.167.102 is alive
VX3230 =>
```

4.3 Create and Run User Parameters

U-Boot firmware on the VX3230 allows the user to create his own parameters using the U-Boot **setenv** command.

For example:

- To download an executable file “**example.bin**" from the network using the tftp protocol, load to address 0x20000000 (which is a safe SDRAM address enought to store a 128-MB file) and to execute the associated code,
- and to create a user parameter named “**flash_file**", use the procedure described below:

The network parameters are supposed to be set as described in the sections 4.2 “Set Ethernet Parameters".

```
VX3230 =>  setenv flash_file ’tftp 0x2000000 example.bin;go 0x2000000’
VX3230 =>  saveenv
VX3230 =>
```

The character "**;**" separates a command to another on the same line.

At this point the user parameter named "**flash_file**" is stored in the VX3230 non-volatile memory. To execute this parameter, type the following command:

```
VX3230 =>  run flash_file
VX3230 =>
```

Another example:

- To set Ethernet parameters and donwload an image file "**u-boot.bin**", create a user parameter named “**dluboot**", using the procedure described below:

```
VX3230 =>  setenv dluboot  ’setenv ipaddr 192.93.167.109; setenv gatewayip 192.93.167.46;
setenv serverip 192.93.167.102; tftp 0x2000000 u-boot.bin’
VX3230 =>  saveenv
VX3230 =>
```

To execute this parameter, type the following command:

```
VX3230 =>  run dluboot
Speed: 1000, full duplex
Using eTSEC1 device
TFTP from server 192.93.167.102; our IP address is 192.93.167.109
Filename ’uboot.bin’.
Load address: 0x100000
Loading: ####################################
done
Bytes transferred = 524288 (80000 hex)
VX3230 =>
```

Emptying a parameter is the only way to delete a parameter:

```
VX3230 =>  setenv dluboot
VX3230 =>  saveenv
VX3230 =>
```

4.4 Specific User Parameters

» VPX_Disable

By default, U-Boot firmware on the VX3230 probe its PCIExpress 2 where there is the VPX; to disable the probe, the variable VPX_Disable has to be defined :

```
VX3230 => setenv VPX_Disable 1
VX3230 => saveenv
VX3230 => reset

U-Boot 1.3.3 ID09268 for KONTRON VX3230 board

CPU:   8533_E, Version: 1.1, (0x803c0011)
Core:  E500, Version: 2.2, (0x80210022)
Clock Configuration:
       CPU: 800 MHz, CCB: 400 MHz,
       DDR: 200 MHz (400 MT/s data rate), LBC:  25 MHz
L1:    D-cache 32 kB enabled
       I-cache 32 kB enabled
Board: CANNES
I2C:   ready
DRAM:  Initializing
        -> DDR: Init mem CTL 1 done
    DDR: 1024 MB
L2 cache 256KB: enabled
VPD EEPROM:   ready
FLASH:  4 MB
    eTSEC1 is in sgmii mode.
    eTSEC3 is in sgmii mode.

    PCIE3 connected to SATA as Root Complex (base address e000b000)
               Scanning PCI bus 01
        Bus: 01  Dev: 00  Ven_ID: 1095  Dev_ID: 3132  Class: 0180  Int: 0x08 (8)
    PCIE3 on bus 00 - 01

    PCIE1 connected to Slot2 as Root Complex (base address e000a000)
    PCIE1 on bus 02 - 02
    VPX System Controller (GA = 30)
    VPX_Disable defined
    VPX keept closed ==> PCIE2: disabled

    PCI: 32 bit, 33 MHz, async, host, arbiter (base address e0008000)
               Scanning PCI bus 03
        Bus: 03  Dev: 11  Ven_ID: 1131  Dev_ID: 1561  Class: 0c03  Int: 0x02 (2)
        Bus: 03  Dev: 11  Ven_ID: 1131  Dev_ID: 1561  Class: 0c03  Int: 0x02 (2)
        Bus: 03  Dev: 11  Ven_ID: 1131  Dev_ID: 1562  Class: 0c03  Int: 0x02 (2)
PCI on bus 03 - 03
In:    serial
Out:   serial
Err:   serial
Net:   eTSEC1, eTSEC3
SATA:
        SATA0 : No SATA Device Found
        SATA1 : No SATA Device Found
VX3230 =>
```

serialconf

By default, U-Boot firmware on the VX3230 uses serial line 0 for the console. It can be changed to serial line 1 using the variable **serialconf**.

In RESCUE mode, the console remains on serial line 0.

To change to serial line 1:

```
VX3230 => setenv serialconf 1
VX3230 => saveenv
VX3230 =>
```

To change to back to serial line 0, just remove the variable:

```
VX3230 => setenv serialconf
VX3230 => saveenv
VX3230 =>
```

usb1panel

Two USB ports are available on the VX3230. Usually, usb1 drives the USB flash soldered on the VX3230. Switching to the second USB port on the rear transition module is done when setting **usb1panel** to rear:

```
VX3230 => setenv usb1panel rear
VX3230 => saveenv
VX3230 =>
```

usb0panel, ethpanel, eth1panel

These variables are only relevant on a VX3230/SA card without a front panel for a PMC (no XMC/PMC slot manufacturing option).

On a VX3230/RC card or on a VX3230/SA card with a front panel for a PMC (XMC/PMC slot manufacturing option), usb0 and both ethernet interfaces are automatically directed to the rear transition module.

On a basic VX3230/SA card (no XMC/PMC slot manufacturing option), user can redirect usb0 and/or ethernet interfaces using the following commands:

```
VX3230 => setenv usb0panel rear
VX3230 => setenv ethpanel rear
VX3230 => setenv eth1panel rear
VX3230 => saveenv
VX3230 =>
```

Chapter 5 - Update U-Boot Firmware

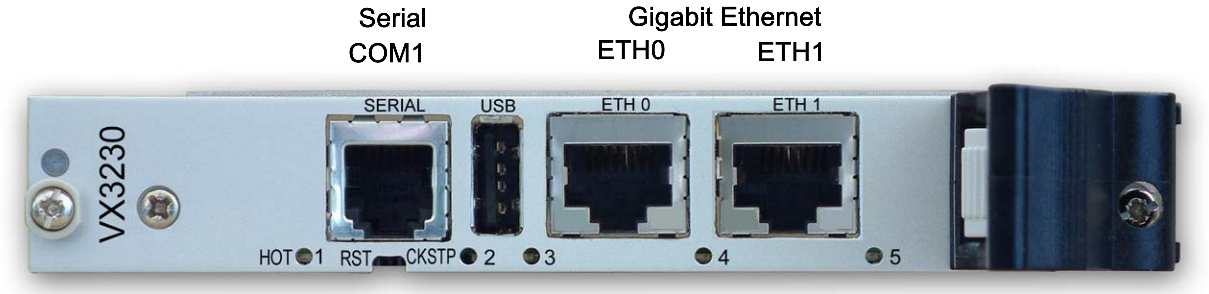
The VX3230 U-Boot user firmware on Main System Flash can be either updated (User Mode) or restored from RESCUE Flash (Rescue mode).

THE U-Boot ON THE RESCUE FLASH IS NEVER UPDATED !!!

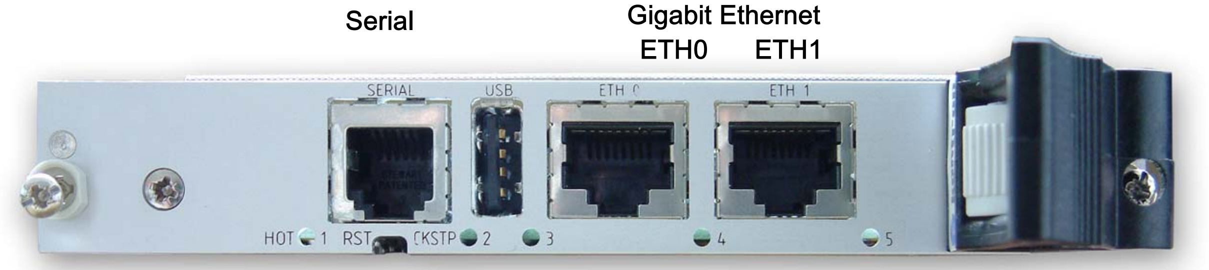
Before updating the firmware file, the type of the file to be downloaded is checked, and a checksum is done after the download.

The network parameters need to be set by user, as described in previous chapter.

5.1 Update MAIN Flash from User Mode



VX3230 Front Panel



VX3230-RTM Front Panel

The procedure to update U-Boot via Ethernet on MAIN Flash is:

1. Check first if the DIP Switch SW1_2 on VX3230 board is set to OFF (Boot on MAIN FLASH)
2. Connect a serial console via a RJ-12 cable to the front serial port of the VX3230 board
3. Connect an Ethernet cable either on Ethernet Port 0 (ETH0) or Port 1 (ETH1) on front panel or on Ethernet Port 2 (ETH0) or Port 3 (ETH1) on the RTM depending on the VX3230 board class (SA/RC)
4. Set the speed serial console to **115200 bauds**
5. Power-On the VX3230 board and wait for the **VX3230=>** prompt

6. Set the correct Ethernet parameters:

```
VX3230 => setenv ipaddr 192.93.167.109
VX3230 => setenv gatewayip 192.93.167.46
VX3230 => setenv sererip 192.93.167.102
VX3230 => setenv loadaddr 0x1000000
VX3230 => setenv uboot cannes.bin
VX3230 => saveenv
VX3230 =>
```

7. Type run tftpflash:

> Example:

```
VX3230 => run tftpflash
Speed: 1000, full duplex
Using eTSEC1 device
TFTP from server 192.93.167.102; our IP address is 192.93.167.109
Filename ’cannes.bin’.
Load address: 0x1000000
Loading: ####################################
done
Bytes transferred = 524288 (80000 hex)
........ done
Un-Protected 8 sectors

........ done
Erased 8 sectors
Copy to Flash... done
........ done
Protected 8 sectors
Total of 524288 bytes were the same
VX3230  =>
```

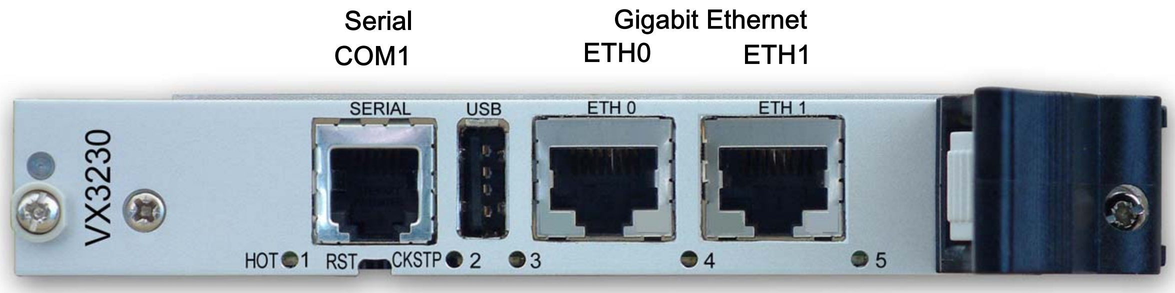
8. Type reset on VX3230 prompt or Power Off/On the rack or push Reset Button of the board and wait for the prompt VX3230=>.

   The new version is displayed by running ver command.

> Example :

```
VX3230 => ver

U-Boot 1.3.3 ID09268 for KONTRON VX3230 board
VX3230 =>
```

5.2 Update MAIN Flash from Rescue Mode



VX3230 Front Panel

Sometimes, the MAIN Flash might be corrupted and it is not possible to have the U-Boot VX3230 prompt.

In order to repair the corrupted Flash, the user can switch on the RESCUE Flash by setting the DIP Switch SW1_2 to ON.

The procedure to restore a MAIN corrupted Flash is shown below:

1. Check first, if DIP Switch SW1_2 on VX3230 board is set to ON (Boot on RESCUE FLASH).
2. Connect a serial console via a RJ-12 cable to the front serial port of the VX3230 board.
3. Connect an Ethernet cable either on Ethernet Port 0 (ETH0) or Port 1 (ETH1) on front panel of the board.
4. Set the speed serial console to 115200 bauds.
5. Power-on the VX3230 board and wait for **VX3230-RESCUE=>** prompt.
6. Set the correct Ethernet parameters:

```
VX3230-RESCUE => setenv ipaddr 192.93.167.109
VX3230-RESCUE => setenv gatewayip 192.93.167.46
VX3230-RESCUE => setenv sererip 192.93.167.102
VX3230-RESCUE => setenv loadaddr 0x1000000
VX3230-RESCUE => setenv uboot cannes.bin
VX3230-RESCUE => saveenv
VX3230-RESCUE =>
```

7. Type **run tftpflash**:

   > Example:

```
VX3230-RESCUE => run tftpflash
Speed: 1000, full duplex
Using eTSEC1 device
TFTP from server 192.93.167.102; our IP address is 192.93.167.109
Filename ’cannes.bin’.
Load address: 0x1000000
Loading: ####################################
done
Bytes transferred = 524288 (80000 hex)
........ done
Un-Protected 8 sectors

........ done
Erased 8 sectors
Copy to Flash... done
........ done
Protected 8 sectors
Total of 524288 bytes were the same
VX3230-RESCUE  =>
```

8. Power-off the VX3230.
9. Set DIP Switch SW1_2 to **OFF.**
10. Power-on the VX3230 and wait for **VX3230=>** prompt.

Chapter 6 - U-Boot Command Line Interface

The U-Boot implementation delivered with VX3230 contains the complete boot code infrastructure. Our Quality Insurance process has thoroughly qualified proper operation of the U-Boot commands required to operate VX3230. Such commands are indicated as **Qualified** in the following sections.

Conversely, U-Boot other commands are **Delivered** as part of the U-Boot core. They should not depend on the VX3230 hardware and have not formally be qualified by Kontron.

In case of trouble, please report issues first to the U-Boot community at url http://bugs.denx.de/databases/u-boot/new and then to support-kom-sa@kontron.com.

There is no formal commitment from Kontron to issue formal fixes for **Delivered** features.

6.1 Commands Summary

6.1.1 Proprietary Qualified Commands

List of U-Boot proprietary qualified commands available only for U-Boot on VX3230 boards.

Command Name	Description	Command Type	See section
diag	perform board diagnostics	Proprietary	6.2.1 page 21
update	Updates U-Boot Firmware	Proprietary	6.2.2 page 21
vpdutil show	Display Vital Product Data (VPD) information of the board	Proprietary	6.2.3 page 22
vpdutil pld	Display current version of PLD on the board	Proprietary	6.2.3 page 22

6.1.2 Standard Qualified Commands

List of U-Boot standard qualified commands available on VX3230 boards.

Command Name	Description	Command Type	See section
?	alias for 'help'	Miscellaneous	6.11.11 page 46
ads	Voltage sensors	Miscellaneous	6.11.1 page 44
bdinfo	print Board Info structure	Information	6.3.1 page 23
bootd	boot default, i.e., run 'bootcmd'	Environment Var.	6.8.1 page 34
bootelf	boot from an ELF image in memory	Execution Control	6.6.2 page 30
bootline	boot parameters	Execution Control	6.9.1 page 36
bootm	boot application image from memory	Execution Control	6.6.3 page 30
bootp	boot image via network using bootp/TFTP protocol	Network	6.7.1 page 32
bootvx	boot VxWorks from an ELF image	Execution Control	6.9.2 page 36
cmp	memory compare	Memory	6.4.3 page 25
coninfo	print console devices and information	Information	6.3.2 page 23
cp	memory copy	Memory Flash Memory	6.4.4 page 25
crc32	checksum calculation	Memory	6.4.2 page 25
date	get/set/reset date & time	Miscellaneous	6.11.2 page 44
dtt	Digital Thermometer and Thermostat	Miscellaneous	6.11.3 page 44

Command Name	Description	Command Type	See section
echo	echo args to console	Miscellaneous	6.11.4 page 44
erase	erase FLASH memory	Flash Memory	6.5.2 page 28
ext2load	load binary file from a filesystem image	Filesystem Sup.	6.9.4 page 37
ext2ls	list files in a directory (default/)	Filesystem Sup.	6.9.3 page 36
flinfo	print FLASH memory information	Information Flash Memory	6.3.3 page 23 6.5.3 page 28
go	start application at addr 'addr'	Execution Control	6.6.4 page 31
help	print online help	Information	6.3.6 page 24
i2c	I2C Sub-system	Special	6.10.1 page 39
icrc32	checksum calculation	Special	6.10.2 page 39
iloop	infinite loop on address range	Special	6.10.3 page 39
imd	I2C memory modify (auto-incrementing)	Special	6.10.4 page 39
iminfo	print header information for application image	Information	6.3.4 page 24
imm	I2C memory modify (auto-incrementing)	Special	6.10.5 page 39
imw	I2C memory write (fill)	Special	6.10.6 page 40
inm	I2C memory modify (constant address)	Special	6.10.7 page 40
iprobe	probe to discover valid I2C chip addresses	Special	6.10.8 page 40
loop	infinite loop on address range	Memory	6.4.5 page 26
md	memory display	Memory	6.4.6 page 26
mii	MII utility command	Network	6.7.5 page 33
mm	memory modify (auto-incrementing)	Memory	6.4.7 page 26
mtest	Simple RAM Test	Memory	6.4.8 page 26
mw	memory write (fill)	Memory	6.4.9 page 27
nfs	boot image via network using NFS protocol	Network	6.7.6 page 33
nm	memory modify (constant address)	Memory	6.4.10 page 27
pci	list and access PCI configuration space	Special	6.10.9 page 40
ping	send ICMP ECHO_REQUEST to network host	Network	6.7.7 page 33
printenv	print environment variables	Environment Var.	6.8.2 page 34
protect	enable or disable FLASH write protection	Flash Memory	6.5.4 page 29
rarpboot	boot image via network using RARP/TFTP protocol	Network	6.7.8 page 33
reset	perform reset of the CPU	Miscellaneaous	6.11.7 page 45
run	run commands in an environment variable	Environment Var.	6.8.3 page 34
sata	SATA sub system	Special	6.10.10 page 41
saveenv	save environment variables to persistent storeage	Environment Var.	6.8.4 page 34
setenv	set environment variables	Environment Var.	6.8.5 page 35
sleep	delay execution for some time	Miscellaneous	6.11.8 page 45
tftpboot	boot image via network using TFTP protocol	Network	6.7.9 page 33
usb	USB sub system	Special	6.10.11 page 42
usbboot	Boot from USB device	Special	6.10.12 page 43
version	print monitor version	Miscellaneous	6.11.10 page 45

6.1.3 Standard Delivered Commands

List of U-Boot standard delivered commands available on VX3230 boards.

Command Name	Description	Command Type	See section
autoscr	run script from memory	Execution Control	6.6.1 page 30
base	print or set address offset	Memory	6.4.1 page 25
exit	exit script	Execution Control	6.11.5 page 45
fatinfo	print information about filesystem	Filesystem Sup.	6.9.5 page 37
fatload	load binary file from a dos filesystem	Filesystem Sup.	6.9.6 page 37
fatls	list files in a directory (default/)	Filesystem Sup.	6.9.7 page 37
fdt	flattened device tree utility commands	Filesystem Sup.	6.9.8 page 38
imls	list all images found in flash	Information	6.3.5 page 24
imxtract	extract a part of a multi-image	Filesystem Sup.	6.9.10 page 38
itest	return true/false on integer compare	Miscellaneous	6.11.6 page 45
loadb	load binary file over serial line (kermit mode)	Network	6.7.2 page 32
loads	load s-record file over serial line	Network	6.7.3 page 33
loady	load binary file over serial line (ymodem mode)	Network	6.7.4 page 33
test	minimal test like /bin/sh	Miscellaneous.	6.11.9 page 45

6.2 Proprietary Commands

6.2.1 diag - perform board diagnostics

This command is fully described in document SD.DT.F48 - ”VX3230 PBIT User's Guide”.

6.2.2 update - Update U-Boot firmware

The command **update** can be used to update the U-boot firmware.

It first checks the U-Boot image, then it updates the user part of the System Flash.

```
VX3230 => help update
update user addr
 update the U-boot firmware from address in memory, in user mode

VX3230 => update user 1000

Image found @0x00001000 has bad magic
VX3230 => update user 1000000
........ done
Un-Protected 8 sectors

........ done
Erased 8 sectors
........ done
Protected 8 sectors
VX3230 =>
```

6.2.3 vpdutil - manage VPD (Vital Product Data)

```
VX3230 => help vpdutil
vpdutil read offset length - read from VPD EEPROM
vpdutil write offset value - write to VPD EEPROM
vpdutil update - update vpd data of the board
vpdutil show - show vpd data of the board
vpdutil pld - show revision of PLD
vpdutil default - fill VPD EEPROM with default values
VX3230 =>
```

Customer does not need to use those commands except for debug information purpose, for example to provide at Kontron support the revision of the PLD or any useful information of the board.

Example:

```
VX3230 => vpdutil pld
-> PLD revision: 01.00
VX3230 => vpdutil show

************* VPD *************

-> Board Data 0

Board Id              : 00470000
Hardware Index        : 0013
Software Index        : 0003
Main Memory Size      : 40000000
Flash Memory Size     : 00400000
Nvsram Memory Size    : 00000000

-> Board Data 1

Serial Number         : 1109110000002
CPU Id                : 0047
Frequency 0           : 1f78a400
Frequency 1           : 4ead9aa8
Num of MAC            : 02
MAC 0                 : 0000de40364a
MAC 1                 : 0000de40364b
CRC Board Data 1      : 0082ce

-> Production Data

Order Code            : VX3230-PROTO-SA81N-A
EC Level              : 1000
Variant               : 01C61200
VX3230 =>
```

6.3 Information Commands

6.3.1 bdinfo - print Board Info structure

The **bdinfo** command (short: **bdi**) prints the information that U-Boot passes about the board such as memory addresses and sizes, clock frequencies, MAC address, etc. This information is mainly needed to be passed to the O.S. kernel.

```
VX3230 => help bdinfo
memstart      = 0x00000000
memsize       = 0x40000000
flashstart     = 0xFFC00000
flashsize     = 0x00400000
flashoffset   = 0x00000000
sramstart     = 0x00000000
sramsize      = 0x00000000
immr_base     = 0xE0000000
bootflags     = 0xF8013F80
intfreq       = 799.999 MHz
busfreq       = 399.999 MHz
ethaddr       = 00:00:DE:40:36:4A
eth1addr      = 00:00:DE:40:36:4B
IP addr       = 192.93.167.109
baudrate      = 115200 bps
VX3230 =>
```

6.3.2 coninfo - print console devices and information

The **coninfo** command (short: **conin**) displays information about the available console I/O devices.

```
VX3230 => coninfo
List of available devices:
serial   80000003 SIO stdin stdout stderrhelp coninfo
VX3230 =>
```

6.3.3 flinfo - print FLASH memory information

The command **flinfo** (short: **fli**) can be used to get information about the available flash memory (see Flash Memory Commands below).

```
VX3230 => help flinfo
flinfo
- print information for all FLASH memory banks
flinfo N
- print information for FLASH memory bank # N
VX3230 =>
```

6.3.4 iminfo - print header information for application image

iminfo (short: **imi**) is used to print the header information for images like Linux kernels or ramdisks. It prints (among other information) the image name, type and size and verifies that the CRC32 checksums stored within the image are OK.

```
VX3230 => help iminfo
iminfo addr [addr ...]
- print header information for application image starting at
address 'addr' in memory; this includes verification of the
image contents (magic number, header and payload checksums)
VX3230 =>
```

6.3.5 imls - list all images found in flash

```
VX3230 => help imls
imls
- Prints information about all images found at sector
boundaries in flash.
VX3230 =>
```

6.3.6 help - print online help

The **help** command (short: **h or ?**) prints online help. Without any arguments, it prints a list of all U-Boot commands that are available in your configuration of U-Boot. You can get detailed information for a specific command by typing its name as argument to the **help** command:

```
VX3230 => help help
help [command ...]
- show help information (for 'command')
'help' prints online help for the monitor commands.
Without arguments, it prints a short usage message for all commands.
To get detailed help information for specific commands you can type
'help' with one or more command names as arguments.
```

6.4 Memory Commands

6.4.1 base - print or set address offset

You can use the **base** command (short: **ba**) to print or set a "base address" that is used as address offset for all memory commands; the default value of the base address is 0, so all addresses you enter are used unmodified. However, when you repeatedly have to access a certain memory region (like the internal memory of some embedded PowerPC processors) it can be very convenient to set the base address to the start of this area and then use only the offsets:

```
VX3230 => help base
base
- print address offset for memory commands
base off
- set address offset for memory commands to 'off'
VX3230 =>
```

6.4.2 crc32 - checksum calculation

The **crc32** command (short: **crc**) can be used to caculate a CRC32 checksum over a range of memory:

```
VX3230 => crc 100004 3FC
CRC32 for 00100004 ... 001003ff ==> 8a231c50
VX3230 =>
When used with 3 arguments, the command stores the calculated checksum at the given
address:
VX3230 => crc 100004 3FC 100000
CRC32 for 00100004 ... 001003ff ==> 8a231c50
VX3230 => md 100000 4
00100000: 8a231c50 deadbeef deadbeef deadbeef    .#.P............
VX3230 =>
```

As you can see, the CRC32 checksum was not only printed, but also stored at address 0x100000.

6.4.3 cmp - memory compare

With the **cmp** command you can test if the contents of two memory areas is identical or not. The command will either test the whole area as specified by the 3rd (length) argument, or stop at the first difference.

```
VX3230 =>help cmp
cmp [.b, .w, .l] addr1 addr2 count
- compare memory
VX3230 =>
```

Please note that the *count* argument specifies the number of data items to process, i. e. the number of long words or words or bytes to compare.

6.4.4 cp - memory copy

The **cp** is used to copy memory areas.

```
VX3230 =>help cp
cp [.b, .w, .l] source target count
- copy memory
VX3230 =>
```

6.4.5 loop - infinite loop on address range

```
VX3230 =>help loop
loop [.b, .w, .l] address number_of_objects
- loop on a set of addresses
VX3230  =>
```

The **loop** command reads in a tight loop from a range of memory. This is intended as a special form of a memory test, since this command tries to read the memory as fast as possible.

This command will never terminate. There is no way to stop it but to reset the board!

6.4.6 md - memory display

The **md** can be used to display memory contents both as hexadecimal and ASCII data.

```
VX3230 =>help md
md [.b, .w, .l] address [# of objects]
- memory display
VX3230 =>
```

The last displayed memory address and the value of the count argument are remembered, so when you enter **md** again without arguments it will automatically continue at the next address, and use the same *count* again.

6.4.7 mm - memory modify (auto-incrementing)

```
VX3230 =>help mm
mm [.b, .w, .l] address
- memory modify, auto increment address
VX3230 =>
```

The **mm** is a method to interactively modify memory contents. It will display the address and current contents and then prompt for user input. If you enter a legal hexadecimal number, this new value will be written to the address. Then the next address will be prompted. If you don't enter any value and just press ENTER, then the contents of this address will remain unchanged. The command stops as soon as you enter any data that is not a hex number (like .).

6.4.8 mtest - simple RAM test

```
VX3230 => help mtest
mtest [start [end [pattern]]]
    - simple RAM read/write test

VX3230 =>
```

Defaut start address = 0x00200000 and end address = 0x00400000 with pattern:

```
0x00000001,     /* single bit */
0x00000003,     /* two adjacent bits */
0x00000007,     /* three adjacent bits */
0x0000000F,     /* four adjacent bits */
0x00000005,     /* two non-adjacent bits */
0x00000015,     /* three non-adjacent bits */
0x00000055,     /* four non-adjacent bits */
0xaaaaaaaa,     /* alternating 1/0 */
```

6.4.9 mw - memory write (fill)

```
VX3230 =>help mw
mw [.b, .w, .l] address value [count]
- write memory
VX3230  =>
```

The **mw** is a way to initialize (fill) memory with some value. When called without a *count* argument, the value will be written only to the specified address. When used with a *count*, then a whole memory areas will be initialized with this value.

6.4.10 nm - memory modify (constant address)

The **nm** command (non-incrementing memory modify) can be used to interactively write different data several times to the same address. This can be useful for instance to access and modify device registers:

```
 VX3230 =>help nm
nm [.b, .w, .l] address
- memory modify, read and keep address
VX3230  =>
```

6.5 Flash Memory Commands

6.5.1 cp - memory command

```
VX3230 =>help cp
cp [.b, .w, .l] source target count
- copy memory
VX3230 =>
```

The **cp** command "knows" about flash memory areas and will automatically invoke the necessary flash programming algorithm when the target area is in flash memory.

⚠ Writing to flash memory may fail when the target area has not been erased (see **erase** below), or if it is write-protected (see **protect** below).

⚠ Remember that the *count* argument specifies the number of items to copy. If you have a "length" instead (= byte count) you should use cp.b or you will have to calculate the correct number of items.

6.5.2 erase - erase FLASH memory

The **erase** command (short: **era**) is used to erase the contents of one or more sectors of the flash memory. It is one of the more complex commands; the help output shows this.

```
VX3230 =>help era
erase start end
- erase FLASH from addr ’start’ to addr ’end’
erase start +len
- erase FLASH from addr ’start’ to the end of sect w/addr ’start’+’len’-1
erase N:SF[-SL]
- erase sectors SF-SL in FLASH bank # N
erase bank N
- erase FLASH bank # N
erase all
- erase all FLASH banks
VX3230 =>
```

6.5.3 flinfo - print FLASH memory information

The command **flinfo** (short: **fli**) can be used to get information about the available flash memory (see Flash Memory Commands below).

```
VX3230 =>help flinfo
flinfo
- print information for all FLASH memory banks
flinfo N
- print information for FLASH memory bank # N
VX3230 =>
```

6.5.4 protect - enable or disable FLASH write protect

The **protect** command is another complex one. It is used to set certain parts of the flash memory to read-only mode or to make them writable again. Flash memory that is "protected" (= read-only) cannot be written (with the **cp** command) or erased (with the **erase** command). Protected areas are marked as (RO) (for "read-only") in the output of the flinfo command.

The actual level of protection depends on the flash chips used on your hardware, and on the implementation of the flash device driver for this board. In most cases U-Boot provides just a simple software-protection, i. e. it prevents you from erasing or overwriting important stuff by accident (like the U-Boot code itself or U-Boot's environment variables), but it cannot prevent you from circumventing these restrictions - a nasty user who is loading and running his own flash driver code cannot and will not be stopped by this mechanism. Also, in most cases this protection is only effective while running U-Boot, i. e. any operating system will not know about "protected" flash areas and will happily erase these if requested to do so.

```
VX3230 =>help protect
protect on start end
- protect FLASH from addr ’start’ to addr ’end’
protect on start +len
- protect FLASH from addr ’start’ to end of sect w/addr ’start’+’len’-1
protect on N:SF[-SL]
- protect sectors SF-SL in FLASH bank # N
protect on bank N
- protect FLASH bank # N
protect on all
- protect all FLASH banks
protect off start end
- make FLASH from addr ’start’ to addr ’end’ writable
protect off start +len
- make FLASH from addr ’start’ to end of sect w/addr ’start’+’len’-1
wrtable
protect off N:SF[-SL]
- make sectors SF-SL writable in FLASH bank # N
protect off bank N
- make FLASH bank # N writable
protect off all
- make all FLASH banks writable
VX3230 =>
```

6.6 Execution Control Commands

6.6.1 autoscr - run script from memory

```
VX3230 =>help autoscr
autoscr [addr] - run script starting at addr - A valid autoscr header must be
present
VX3230 =>
```

With the **autoscr** command you can run "shell" scripts under U-Boot: You create a U-Boot script image by simply writing the commands you want to run into a text file; then you will have to use the mkimage tool to convert this text file into a U-Boot image (using the image type script).

This image can be loaded like any other image file, and with **autoscr** you can run the commands in such an image.

6.6.2 bootelf - boot from an ELF image in memory

```
VX3230 =>help bootelf
bootelf [address] - load address of the ELF image.
VX3230 =>
```

6.6.3 bootm - boot application image from memory

```
VX3230 =>help bootm
bootm [addr [arg ...]]
- boot application image stored in memory
passing arguments 'arg ...'; when booting a Linux kernel,
'arg' can be the address of an initrd image
When booting a Linux kernel which requires a flat device-tree
a third argument is required which is the address of the
device-tree blob. To boot that kernel without an initrd image,
use a '-' for the second argument. If you do not pass a third
a bd_info struct will be passed instead
VX3230 =>
```

The **bootm** command is used to start operating system images. From the image header it gets information about the type of the operating system, the file compression method used (if any), the load and entry point addresses, etc. The command will then load the image to the required memory address, uncompressing it on the fly if necessary. Depending on the OS it will pass the required boot arguments and start the OS at it's entry point.

The first argument to **bootm** is the memory address (in RAM, ROM or flash memory) where the image is stored, followed by optional arguments that depend on the OS.

Linux requires the flattened device tree blob to be passed at boot time, and bootm expects its third argument to be the address of the blob in memory. Second argument to **bootm** depends on whether an initrd initial ramdisk image is to be used. If the kernel should be booted without the initial ramdisk, the second argument should be given as "-", otherwise it is interpreted as the start address of initrd (in RAM, ROM or flash memory).

6.6.4 go - start application at address “addr”

```
VX3230 =>help go
go addr [arg ...]
- start application at address ’addr’
passing ’arg’ as arguments
VX3230 =>
```

U-Boot has support for so-called standalone applications. These are programs that do not require the complex environment of an operating system to run. Instead they can be loaded and executed by U-Boot directly, utilizing U-Boot's service functions like console I/O or malloc() and free().

This can be used to dynamically load and run special extensions to U-Boot like special hardware test routines or bootstrap code to load an OS image from some filesystem.

The go command is used to start such standalone applications. The optional arguments are passed to the application without modification.

6.7 Network Commands

6.7.1 bootp - boot image via network using BOOTP/TFTP protocol

```
VX3230 =>help bootp
bootp [loadAddress] [bootfilename]
VX3230 =>
```

The user needs to configure a **bootp** server and a **tftp** server.
For the **bootp** server, use a **bootpd** daemon running on Linux (Debian Sarge 3.0 in following example) configured as follow:

In the file **/etc/inetd.conf**:

```
bootps dgram udp wait root /usr/sbin/bootpd
bootpd -i -t 120
```

In the file **/etc/bootptap**:

```
.default:\
        :td=/tftpboot:hd=/tftpboot:bf=uImage:\
        :sm=255.255.255.0:\
        :hn:to=-18000:
minotaure:ha=0000de403614:tc=.default:ip=192.168.0.10
```

When the **bootp** and **tftp** servers are configured, the user can run the bootp command under U-Boot firmware:

```
VX3230 =>bootp
Ethernet status port 0: Link up, Full Duplex, Speed 100 Mbps
BOOTP broadcast 1
Using mv_enet0 device
TFTP from server 192.168.0.1; our IP address is 192.168.0.10
Filename '/tftpboot/uImage'.
Load address: 0x800000
Loading:
##################################################################
##################################################################
##################################################################
##################################################################
##################################################################
###########################################################
done
Bytes transferred = 1959130 (1de4da hex)
VX3230 =>
```

6.7.2 loadb - load binary file over serial line (kermit mode)

```
VX3230 =>help loadb
loadb [ off ] [ baud ]
- load binary file over serial line with offset 'off' and baudrate 'baud'
VX3230 =>
```

6.7.3 loads - load S-Record file over serial line

```
VX3230 =>help loads
loads [ off ] [ baud ]
- load S-Record file over serial line with offset ’off’ and baudrate
’baud’
VX3230 =>
```

6.7.4 loady - load binary file over serial line (ymodem mode)

```
VX3230 =>help loady
loady [ off ] [ baud ]
- load binary file over serial line with offset ’off’ and baudrate ’baud’
VX3230 =>
```

6.7.5 mii - MII utility command

```
VX3230 => help mii
mii device                     - list available devices
mii device <devname>           - set current device
mii info   <addr>              - display MII PHY info
mii read   <addr> <reg>        - read  MII PHY <addr> register <reg>
mii write  <addr> <reg> <data> - write MII PHY <addr> register <reg>
mii dump   <addr> <reg>        - pretty-print <addr> <reg> (0-5 only)
Addr and/or reg may be ranges, e.g. 2-7.
```

6.7.6 nfs - boot image via network using NFS protocol

```
VX3230 =>help nfs
nfs [loadAddress] [host ip addr:bootfilename]
VX3230 =>
```

6.7.7 ping - send ICMP ECHO_REQUEST to network host

```
VX3230 =>help ping
ping pingAddress
VX3230 =>
```

6.7.8 rarpboot - boot image via network using RARP/TFTP protocol

```
VX3230 =>help rarp
rarpboot [loadAddress] [bootfilename]
VX3230 =>
```

6.7.9 tftpboot - boot image via network using TFTP protocol

```
VX3230 =>help tftpboot
tftpboot [loadAddress] [[hostIPaddr:]bootfilename]
VX3230 =>
```

6.8 Environment Variables Commands

6.8.1 bootd - boot default, i.e., run 'bootcmd'

The **bootd** (short: **boot**) executes the default **boot** command, i. e. what happens when you don't interrupt the initial countdown. This is a synonym for the run **bootcmd** command.

6.8.2 printenv - print environment variables

```
VX3230 =>help printenv
printenv
- print values of all environment variables
printenv name ...
- print value of environment variable ’name’
VX3230 =>
```

The **printenv** command prints one, several or all variables of the U-Boot environment. When arguments are given, these are interpreted as the names of environment variables which will be printed with their values.

Without arguments, **printenv** prints all a list with all variables in the environment and their values, plus some statistics about the current usage and the total size of the memory available for the environment.

6.8.3 run - run commands in an environment variable

```
VX3230 =>help run
run var [...]
- run the commands in the environment variable(s) ’var’
VX3230 =>
```

You can use U-Boot environment variables to store commands and even sequences of commands. To execute such a command, you use the **run** command.

You can call **run** with several variables as arguments, in which case these commands will be executed in sequence.

Note If you execute several variables with one call to **run**, any failing command will cause "run" to terminate, i. e. the remaining variables are not executed.

6.8.4 saveenv - save environment variables to persistent storage

```
VX3230 =>help saveenv
saveenv - No help available.
VX3230 =>
```

All changes you make to the U-Boot environment are made in RAM only. They are lost as soon as you reboot the system. If you want to make your changes permanent you have to use the **saveenv** command to write a copy of the environment settings to persistent storage, from where they are automatically loaded during startup.

6.8.5 setenv - set environement variables

```
VX3230 =>help setenv
setenv name value ...
- set environment variable ’name’ to ’value ...’
setenv name
- delete environment variable ’name’
VX3230 =>
```

To modify the U-Boot environment you have to use the **setenv** command. When called with exactly one argument, it will delete any variable of that name from U-Boot's environment, if such a variable exists. Any storage occupied for such a variable will be automatically reclaimed.

A common mistake is to write:

```
setenv name=value
```

instead of:

```
setenv name value
```

There will be no error message displayed, which lets you believe everything went OK, but it didn't: instead of setting the variable name to the value value, you tried to delete a variable with the name name=value - this is probably not what you intended!
Always remember that name and value have to be separated by space and/or tab characters!

6.9 Filesystem Support

6.9.1 bootline - Boot Parameters

The VX3230 U-Boot firmware includes commands to manage bootrom parameters exactly like under VxWorks bootrom prompt. This is done by the VX3230 U-Boot command "bootline"

For example, to display bootrom paramaters:

```
VX3230 => bootline print

boot device                 : motetsec
unit number                 : 0
processor number            : 0
host name                   : sunblade
file name                   : /home/Vx-6.6SMP/vxworks-
6.6/target/config/VX3230/vxWorks
inet on ethernet (e)        : 192.54.144.163:0xffffff00
host inet (h)               : 192.54.144.248
user (u)                    : vxworks
ftp password (pw)           : target
flags (f)                   : 0x8
target name (tn)            : vxworks1
```

To setup bootrom parameters:

```
VX3230 => bootline set

'.' = clear field;  '-' = go to previous field;  ^D = quit

boot device                 : motetsec0
processor number            : 0
host name                   : sunblade
file name                   : /home/Vx-6.6SMP/vxworks-
6.6/target/config/VX3230/vxWorks
inet on ethernet (e)        : 192.54.144.163:0xffffff00
inet on backplane (b)       :
host inet (h)               : 192.54.144.248
gateway inet (g)            :
user (u)                    : vxworks
ftp password (pw) (blank = use rsh): target
flags (f)                   : 0x8
target name (tn)            : vxworks1
startup script (s)          :
other (o)                   :
```

6.9.2 bootvx - boot VxWorks from an ELF image

```
VX3230 => help bootvx
bootvx  [address] - load address of vxWorks ELF image.
```

6.9.3 ext2ls - list files in a directory (default/)

```
VX3230 => help ext2ls
ext2ls <interface> <dev[:part]> [directory]
    - list files from 'dev' on 'interface' in a 'directory'
```

6.9.4 ext2load - load binary file from a filesystem image

```
VX3230 => help ext2load
ext2load <interface> <dev[:part]> [addr] [filename] [bytes]
    - load binary file ’filename’ from ’dev’ on ’interface’
      to address ’addr’ from ext2 filesystem
```

For example, in order to boot a Linux image from a SATA HDD, you have to execute following commands:

```
VX3230 => ext2load sata 0:1 0x1000000 /uImage
VX3230 => ext2load sata 0:1 0x2000000 /initrd.gz.uboot
VX3230 => bootm 0x1000000 0x2000000 -
```

where:

- sata = device SATA,
- 0:1 = port number(0=port SATA0, 1=port SATA1):partition number
- 0x1000000 = target address in SDRAM where the image file will be
- /uImage = name of the binary file on the SATA filesystem under / directory

6.9.5 fatinfo - print information about filesystem

```
VX3230 => help fatinfo
fatinfo <interface> <dev[:part]>
    - print information about filesystem from ’dev’ on ’interface’
```

6.9.6 fatload - load binary file from a dos filesystem

```
VX3230 => help fatload
fatload <interface> <dev[:part]>  <addr> <filename> [bytes]
    - load binary file ’filename’ from ’dev’ on ’interface’
      to address ’addr’ from dos filesystem
```

6.9.7 fatls - list file in a directory (default/)

```
VX3230 => help fatls
fatls <interface> <dev[:part]> [directory]
    - list files from ’dev’ on ’interface’ in a ’directory’
```

6.9.8 fdt - flattened device tree utility commands

```
VX3230 => help fdt
fdt addr   <addr> [<length>]            - Set the fdt location to <addr> (if <addr>
== ’-’ than the local device tree is used)
fdt boardsetup                          - Do board-specific set up
fdt move    <fdt> <newaddr> <length>    - Copy the fdt to <addr> and make it active
fdt print  <path> [<prop>]              - Recursive print starting at <path>
fdt list   <path> [<prop>]              - Print one level starting at <path>
fdt set     <path> <prop> [<val>]       - Set <property> [to <val>]
fdt mknode <path> <node>                - Create a new node after <path>
fdt rm      <path> [<prop>]             - Delete the node or <property>
fdt header                              - Display header info
fdt bootcpu <id>                        - Set boot cpuid
fdt memory <addr> <size>                - Add/Update memory node
fdt rsvmem print                        - Show current mem reserves
fdt rsvmem add <addr> <size>            - Add a mem reserve
fdt rsvmem delete <index>               - Delete a mem reserves
fdt chosen - Add/update the /chosen branch in the tree
NOTE: If the path or property you are setting/printing has a ’#’ character
     or spaces, you MUST escape it with a \ character or quote it with ”.

VX3230 =>
```

6.9.9 fsinfo - print information about filesystems

```
VX3230 =>help fsinfo
fsinfo - print information about filesystems
VX3230 =>
```

6.9.10 imxtract - extract a part of a multi-image

```
VX3230 => help imxtract
imxtract addr part [dest]
    - extract <part> from legacy image at <addr> and copy to <dest>
```

6.10 Special Commands

6.10.1 i2c - I2C sub-system

```
VX3230 => help i2c
i2c dev [dev] - show or set current I2C bus
i2c speed [speed] - show or set I2C bus speed
i2c md chip address[.0, .1, .2] [# of objects] - read from I2C device
i2c mm chip address[.0, .1, .2] - write to I2C device (auto-incrementing)
i2c mw chip address[.0, .1, .2] value [count] - write to I2C device (fill)
i2c nm chip address[.0, .1, .2] - write to I2C device (constant address)
i2c crc32 chip address[.0, .1, .2] count - compute CRC32 checksum
i2c probe - show devices on the I2C bus
i2c loop chip address[.0, .1, .2] [# of objects] - looping read of device
```

Those commands are new I2C commands for manage I2C sub-system:

```
i2c md => imd
i2c mm => imm
i2c probe => iprobe
i2c nm => inm
i2c mw => imw
i2c crc32 => icrc32
```

6.10.2 icrc32 - checksum calculation

```
VX3230 =>help icrc32
icrc32 chip address [.0, .1, .2] count
- compute CRC32 checksum
VX3230 =>
```

6.10.3 iloop - infinite loop on range address

```
VX3230 =>help iloop
iloop chip address[.0, .1, .2] [# of objects]
- loop, reading a set of addresses
VX3230 =>
```

6.10.4 imd - I2C memory display

```
VX3230 =>help imd
imd chip address[.0, .1, .2] [# of objects]
- i2c memory display
VX3230 =>
```

6.10.5 imm - I2C memory modify (auto-incrementing)

```
VX3230 =>help imm
imm chip address[.0, .1, .2]
- memory modify, auto increment address
VX3230 =>
```

6.10.6 imw - I2C memory write (fill)

```
 VX3230 =>help imw
imw chip address[.0, .1, .2] value [count]
- memory write (fill)
VX3230 =>
```

6.10.7 inm - I2C memory modify (constant address)

```
VX3230 =>help inm
inm chip address[.0, .1, .2]
- memory modify, read and keep address
VX3230 =>
```

6.10.8 iprobe - probe to discover valid I2C chip addresses

```
VX3230 =>help iprobe
iprobe
-discover valid I2C chip addresses
VX3230 =>
```

The iprobe command returns valid I2C chip addresses in hexadecimal format. These addresses are used as the chip address argument of the commands: icr32, iloop, imd, imm, imw, inm described in previous sections (6.10.2 to 6.10.7)

Example of iprobe command output:

```
VX3230 =>iprobe
Valid chip addresses: 48
VX3230 =>
```

6.10.9 pci - list and access PCI configuration space

```
VX3230 =>help pci
pci [bus] [long]
- short or long list of PCI devices on bus ’bus’
pci header b.d.f
- show header of PCI device ’bus.device.function’
pci display[.b, .w, .l] b.d.f [address] [# of objects]
- display PCI configuration space (CFG)
pci next[.b, .w, .l] b.d.f address
- modify, read and keep CFG address
pci modify[.b, .w, .l] b.d.f address
- modify, auto increment CFG address
pci write[.b, .w, .l] b.d.f address value
- write to CFG address
```

6.10.10 sata - SATA sub-system

```
VX3230 => help sata
sata info - show available SATA devices
sata device [dev] - show or set current device
sata part [dev] - print partition table
sata read addr blk# cnt
sata write addr blk# cnt
VX3230 =>
```

Example

```
VX3230 => sata info

SATA device 1: Model: WDC WD2500YD-01NVB1 Firm: 10.02E01 Ser#:      WD-WCANK6722813
            Type: Hard Disk
            Supports 48-bit addressing
            Capacity: 239372.4 MB = 233.7 GB (490234752 x 512)
            Speed: 1.5Gbit/s

VX3230 => sata part 1

Partition Map for SATA device 1  --   Partition Type: DOS

Partition     Start Sector     Num Sectors     Type
    1              63          401562    83
    2          401625       489821850    8e
VX3230 => sata device

SATA device 0: Model:  Firm:  Ser#:
            Type: # 1F #
            Capacity: not available
            Speed: 1.5Gbit/s
VX3230 => sata device 1

SATA device 1: Model: WDC WD2500YD-01NVB1 Firm: 10.02E01 Ser#:      WD-WCANK6722813
            Type: Hard Disk
            Supports 48-bit addressing
            Capacity: 239372.4 MB = 233.7 GB (490234752 x 512)
            Speed: 1.5Gbit/s
... is now current device
VX3230 => sata device

SATA device 1: Model: WDC WD2500YD-01NVB1 Firm: 10.02E01 Ser#:      WD-WCANK6722813
            Type: Hard Disk
            Supports 48-bit addressing
            Capacity: 239372.4 MB = 233.7 GB (490234752 x 512)
            Speed: 1.5Gbit/s

VX3230 => sata part


Partition Map for SATA device 1  --   Partition Type: DOS

Partition     Start Sector     Num Sectors     Type
    1                63        401562    83
    2            401625     489821850    8e
VX3230 =>
```

6.10.11 usb - USB sub-system

```
VX3230 => help usb
usb reset - reset (rescan) USB controller
usb stop [f]  - stop USB [f]=force stop
usb tree  - show USB device tree
usb info [dev] - show available USB devices
usb storage  - show details of USB storage devices
usb dev [dev] - show or set current USB storage device
usb part [dev] - print partition table of one or all USB storage devices
usb read addr blk# cnt - read ‘cnt’ blocks starting at block ‘blk#’
    to memory address ‘addr’
VX3230 =>
```

Example:

```
VX3230 =>  usb start
(Re)start USB...
USB:   scanning bus for devices... 3 USB Device(s) found
       scanning bus for storage devices... 1 Storage Device(s) found
VX3230 => usb tree

Device Tree:
  1  Hub (12MBit/s, 0mA)
  |   OHCI Root Hub
  |
  +-2  Mass Storage (12MBit/s, 100mA)
       SMART eUSB 20090303072543AF

  3  Hub (12MBit/s, 0mA)
      OHCI Root Hub

VX3230 => usb info 1
config for device 1
1: Hub,  USB Revision 1.10
 -  OHCI Root Hub
 - Class: Hub
 - PacketSize: 8  Configurations: 1
 - Vendor: 0x0000  Product 0x0000 Version 0.0
   Configuration: 1
   - Interfaces: 1 Self Powered 0mA
     Interface: 0
     - Alternate Setting 0, Endpoints: 1
     - Class Hub
     - Endpoint 1 In Interrupt MaxPacket 2 Interval 255ms

VX3230 => usb info 2
config for device 2
2: Mass Storage,  USB Revision 2.0
 - SMART eUSB 20090303072543AF
 - Class: (from Interface) Mass Storage
 - PacketSize: 64  Configurations: 1
 - Vendor: 0x0e39  Product 0x2b00 Version 135.77
   Configuration: 1
   - Interfaces: 1 Bus Powered 100mA
     Interface: 0
     - Alternate Setting 0, Endpoints: 2
     - Class Mass Storage, Transp. SCSI, Bulk only
     - Endpoint 1 In Bulk MaxPacket 64
     - Endpoint 2 Out Bulk MaxPacket 64
```

6.10.12 usbboot - boot from USB device

```
VX3230 => help usbboot
usbboot loadAddr dev:part
VX3230 =>
```

6.11 Miscellaneous Commands

6.11.1 ads - Voltage Sensors

```
VX3230 => help ads
ads         - Read voltage

VX3230 => ads
ADS: Voltage sensors
3V3 : 3333 mV
2V5 : 2490 mV
1V8 : 1794 mV
1V2 : 1196 mV
1V0 : 1019 mV
VX3230 =>
```

6.11.2 date - get/set/reset date & time

```
VX3230 =>help date
date [MMDDhhmm[[CC]YY][.ss]]
date reset
- without arguments: print date & time
- with numeric argument: set the system date & time
- with ’reset’ argument: reset the RTC
VX3230 =>
```

6.11.3 dtt - Digital Thermometer and Thermostat

```
VX3230 =>help dtt
Read temperature from digital thermometer and thermostat.
VX3230 =>
```

Example

```
VX3230 => dtt

DTT: CPU sensors
DTT1: 34 C
DTT2: 46 C (55)

DTT: BOARD sensors
DTT1: 30 C
DTT2: 26 C
DTT3: 32 C
VX3230 =>
```

6.11.4 echo - echo args to console

```
VX3230 =>help echo
echo [args..]
- echo args to console; \c suppresses newline
VX3230 =>
```

The **echo** command echoes the arguments to the console.

6.11.5 exit script

```
VX3230 => help exit
exit     - exit functionality
```

6.11.6 itest - return true/false on integer compare

```
VX3230 =>help itest
itest [.b, .w, .l, .s] [*]value1 <op> [*]value2
VX3230 =>
```

6.11.7 reset - perform reset of the CPU

The **reset** command reboots the system.

6.11.8 sleep - delay execution for some time

```
VX3230 =>help sleep
sleep N
- delay execution for N seconds (N is _decimal_ !!!)
VX3230 =>
```

The **sleep** command pauses execution for the number of seconds given as the argument.

6.11.9 test - minimal test like /bin/sh

```
VX3230 => help test
test [args..]
    - test functionality

VX3230 =>
```

6.11.10 version - print monitor version

You can print the version and build date of the U-Boot image running on your system using the **version** command (short: **vers**)

6.11.11 ? - alias for 'help'

You can use ? as a short form for the help command.? - alias for 'help'

```
VX3230 => ?
?       - alias for ’help’
autoscr - run script from memory
base    - print or set address offset
bdinfo  - print Board Info structure
boot    - boot default, i.e., run ’bootcmd’
bootd   - boot default, i.e., run ’bootcmd’
bootelf - Boot from an ELF image in memory
bootline - print/set/default vxworks bootrom parameters
bootm   - boot application image from memory
bootp   - boot image via network using BootP/TFTP protocol
bootvx  - Boot vxWorks from an ELF image
cmp     - memory compare
coninfo - print console devices and information
cp      - memory copy
crc32   - checksum calculation
date    - get/set/reset date & time
dcache  - enable or disable data cache
diag    - perform board diagnostics
dtt     - Digital Thermometer and Thermostat
echo    - echo args to console
erase   - erase FLASH memory
exit    - exit script
ext2load- load binary file from a Ext2 filesystem
ext2ls  - list files in a directory (default /)
fatinfo - print information about filesystem
fatload - load binary file from a dos filesystem
fatls   - list files in a directory (default /)
fdt     - flattened device tree utility commands
flinfo  - print FLASH memory information
go      - start application at address ’addr’
help    - print online help
i2c     - I2C sub-system
icache  - enable or disable instruction cache
icrc32  - checksum calculation
iloop   - infinite loop on address range
imd     - i2c memory display
iminfo  - print header information for application image
imls    - list all images found in flash
imm     - i2c memory modify (auto-incrementing)
imw     - memory write (fill)
imxtract- extract a part of a multi-image
inm     - memory modify (constant address)
iprobe  - probe to discover valid I2C chip addresses
itest   - return true/false on integer compare
loadb   - load binary file over serial line (kermit mode)
loads   - load S-Record file over serial line
loady   - load binary file over serial line (ymodem mode)
loop    - infinite loop on address range
md      - memory display
mii     - MII utility commands
mm      - memory modify (auto-incrementing)
mtest   - simple RAM test
mw      - memory write (fill)
nfs     - boot image via network using NFS protocol
nm      - memory modify (constant address)
```

```
pci     - list and access PCI Configuration Space
ping    - send ICMP ECHO_REQUEST to network host
printenv- print environment variables
protect - enable or disable FLASH write protection
rarpboot- boot image via network using RARP/TFTP protocol
reginfo - print register information
reset   - Perform RESET of the CPU
run     - run commands in an environment variable
sata    - SATA sub system
saveenv - save environment variables to persistent storage
setenv  - set environment variables
sleep   - delay execution for some time
test    - minimal test like /bin/sh
tftpboot- boot image via network using TFTP protocol
uboot   - manage U-Boot updates
usb     - USB sub-system
usbboot - boot from USB device
version - print monitor version
vpdutil - VPD EEPROM utility
VX3230 =>
```

Chapter 7 - Additional Information

7.1 USB Restriction with U-Boot 1.3.3 ID 09268

7.1.1 Generality

U-Boot 1.3.3 does not support USB EHCI. It supports only USB OHCI with USB Mass storage devices of class "Bulk Only".

Since U-Boot does not support interrupts, only one USB at a time can be accessed. This prevents copy from one USB device to another USB device. To make a copy from a USB device to an other USB device, the only way is to use the memory as a temporary storage media and then copy from memory to the second USB storage device.

7.1.2 USB Flash Intel SSD U130

Intel SSD Flash U130 plugged on the VX3230 board requires high current consumption compared to an USB key Mass storage device. Therefore, it is not allowed to use this device with other USB devices under U-Boot.

The work-around to access this device is to unplug all other USB devices and run an **usb start** command.

7.1.3 USB CD/DVD drives self-powered

USB CD/DVD drives self-powered need a special attention under U-Boot on VX3230 boards.

The power must be turned OFF during POST and only turned ON when U-Boot prompt **VX3230 =>** appears.

Then, make the device ready by running an **usb start** command.

If the message "*Device NOT ready*" appears, re-do an **usb start** command.

7.1.4 CRP 3751 - Several models of DVD-ROM are not reliable under U-Boot

The accessibility to the DVD-ROM player under U-Boot is not reliable. The behavior depends a lot on the brand of the device.

It appears that the PLEXTOR model is the most reliable among all the tested models:

- DVD player SAMSUNG never detected. **usbboot** command works in a random way.
- DVD player LG not seen at all under U-Boot.
- DVD player SONY **usb start** command failed in case of presence of both the FLASH USB device and the SONY DVD player device.
- DVD player PLEXTOR no problem identified.

7.1.5 CRP 3753 - PBIT results not accessible under Operating System

There is no coherency between U-Boot and OS context concerning the results of the PBIT.

Under U-Boot, the PBIT can be run and the results are accessible.

On the contrary, under the OS, the PBIT results are not accessible.

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@kontron.com
support-kom-sa@kontron.com

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