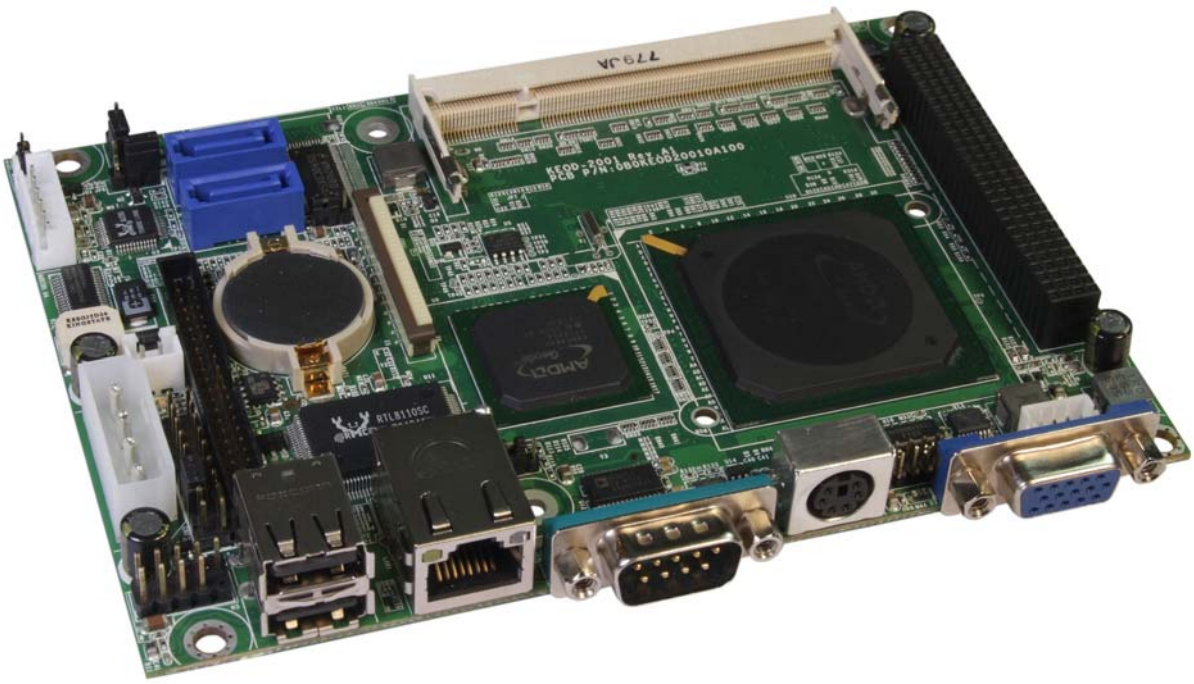
kontron

» Kontron Software Guide «



JRex*plus* LX

KTD-S0004-A

» Table of Contents «

1 User Information

1.1 About This Document

This document provides information about products from KONTRON Technology A/S and/or its subsidiaries. No warranty of suitability, purpose or fitness is implied. While every attempt has been made to ensure that the information in this document is accurate the information contained within is supplied “as-is” - no liability is taken for any inaccuracies. Manual is subject to change without prior notice.
KONTRON assumes no responsibility for the circuits, descriptions and tables indicated as far as patents or other rights of third parties are concerned.

1.2 Copyright Notice

1.3 Trademarks

Brand and product names are trademarks or registered trademarks of their respective owners.

1.4 Standards

KONTRON Technology A/S is certified to ISO 9000 standards.

1.5 Warranty

This product is warranted against defects in material and workmanship for the warranty period from the date of shipment. During the warranty period KONTRON Technology A/S will at its discretion decide to repair or replace defective products.
Within the warranty period the repair of products is free of charge as long as warranty conditions are observed.
The warranty does not apply to defects resulting from improper or inadequate maintenance or handling by the buyer, unauthorized modification or misuse, operation outside of the product’s environmental specifications or improper installation or maintenance.
KONTRON Technology A/S will not be responsible for any defects or damages to third party products that are caused by a faulty KONTRON Technology A/S product.

1.6 Life Support Policy

KONTRON Technology's products are not for use as critical components in life support devices or systems without express written approval of the general manager of KONTRON Technology A/S.
As used herein:
Life support devices or systems are devices or systems which
a) are intended for surgical implant into body or
b) support or sustain life and whose failure to perform, when properly used in accordance with instructions for use provided in the labelling, can be reasonably expected to result in significant injury to the user.
A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

1.7 Technical Support

Please consult our web site at **http://www.kontron.com/support** for the latest product documentation, utilities, drivers and support contacts. In any case you can always contact your board supplier for technical support.
Before contacting support please be prepared to provide as much information as possible:
Board identification:

- Type
- Part number (find PN on label)
- Serial number (find SN on label)

Board configuration:

- DRAM type and size
- BIOS revision (find in the BIOS Setup)
- BIOS settings different than default settings (refer to the BIOS Setup section)

System environment:

- O/S type and version
- Driver origin and version
- Attached hardware (drives, USB devices, LCD panels ...)

2 BIOS Update

The KONTRON update tool (based on the AMD utility 'FlashROM.com') is only available for the DOS operating system. First copy the files LX800UPD.EXE, UPDATE.EXT (absolutely essential) and the BIOS file to a directory. Only the following combination of command line arguments has been tested and should be used for the update process.
Note: COMMAND.COM must reside in the same directory or alternatively there has to be a valid path in the variable COMSPEC.

The syntax of the DOS update tool is:

LX800UPD **/C** **/R** <BIOS filename>

/C = destroy CMOS checksum
/R = warmstart after programming

Following combinations are valid:

LX800UPD <BIOS filename>
LX800UPD **/C** <BIOS filename>
LX800UPD **/C** **/R** <BIOS filename>

3 Graphics Interface

3.1 LCD/LVDS Technology Overview

3.1.1 Detailed Timing Descriptor (EDID or DisplayID™)

The input fields Pixel Clock, Horizontal Active, Horizontal Blank, Horizontal Sync Offset, Horizontal Sync Width, Vertical Active, Vertical Blank, Vertical Sync Offset and Vertical Sync Width must be filled in with the correct values according to the panel's data sheet. In many cases the value for Horizontal/Vertical Blank cannot be read directly from the data sheet. Instead terms such as Display Period (active pixels/lines) or Horizontal/Vertical Total appear.
In this case the following calculation can be made:

⇒ **Blank Value = Total Value – Active Value.**

Sometimes the datasheet does not specify Sync Offset and/or Sync Width. In this case the permissible values can only be determined though testing. However the rule is:

⇒ **The sum of Sync Offset and Sync Width must not exceed the value for Horizontal/Vertical Blank.**

Also datasheets are often different for displays with double pixel clock. If Pixel Clock and Horizontal Values seem to be halved, this must be corrected for input:

⇒ **The values must always be entered as though it were a panel with single pixel clock.**

Example 1:

PRIMEVIEW PM070WL4 (single pixel clock)
Data sheet specifications:

Clock Frequency [typ.]	32 MHz	
HSync Period [typ.]	1056 Clocks	(equivalent to Horizontal Total)
HSync Display Period [typ.]	800 Clocks	(equivalent to Horizontal Active)
HSync Pulse Width [typ.]	128 Clocks	
HSync Front Porch [typ.]	42 Clocks	
HSync Back Porch [typ.]	86 Clocks	
VSync Period [typ.]	525 Lines	(equivalent to Vertical Total)
VSync Display Period	480 Lines	(equivalent to Vertical Active)
VSync Pulse Width [typ.]	2 Lines	
VSync Front Porch [typ.]	10 Lines	
VSync Back Porch [typ.]	33 Lines	

Result:

Pixel Clock	32	
Horizontal Active	800	
Horizontal Blank	256	((128 + 42 + 86) → H. Pulse Width + H. Front Porch + H. Back Porch)
Horizontal Sync Offset	42	(H. Front Porch)
Horizontal Sync Width	128	(H. Pulse Width)
Vertical Active	480	
Vertical Blank	45	((2 + 10 + 33) → V. Pulse Width + V. Front Porch + V. Back Porch)
Vertical Sync Offset	10	(V. Front Porch)
Vertical Sync Width	3	(V. Pulse Width)

Example 2 (not useable on JRex*plus* LX):

SHARP LQ190E1LW01 (double pixel clock)
Data sheet specifications (no definition of Sync Offset and Sync Width):

Clock Frequency [typ.]	54 MHz	
Horizontal Period (1) [typ.]	844 Clocks	(equivalent to Horizontal Total)
Horizontal Display Period	640 Clocks	(equivalent to Horizontal Active)
Vertical Period [typ.]	1066 Lines	(equivalent to Vertical Total)
Vertical Display Period	1024 Lines	(equivalent to Vertical Active)

Result:

Pixel Clock	108	(2 x 54 MHz)
Horizontal Active	1280	(2 x 640 Clocks)
Horizontal Blank	408	((844 – 640) x 2 Clocks)
Horizontal Sync Offset	45	(normally approx. 10 – 15 % of Horizontal Blank)
Horizontal Sync Width	140	(normally approx. 30 – 70 % of Horizontal Blank)
Vertical Active	1024	
Vertical Blank	42	(1066 – 1024 Lines)
Vertical Sync Offset	1	(normally approx. 1 – 3 Lines)
Vertical Sync Width	3	(normally approx. 1 – 15 Lines)

Example 3 (not useable on JRex*plus* LX):

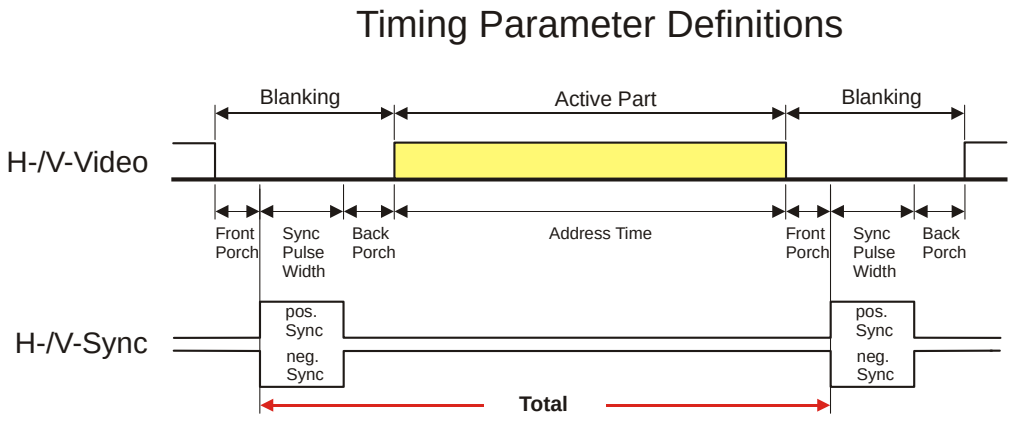
LG-PHILIPS LM170E01-TLA1 (double pixel clock)
Data sheet specifications:

Clock Frequency [typ.]	54 MHz
Hsync Period [typ.]	844 Clocks
Horiz. Valid [typ.]	640 Clocks
Horiz. Back Porch [typ.]	124 Clocks
Horiz. Front Porch [typ.]	24 Clocks
Vsync Period [typ.]	1066 Lines
Vert. Valid [typ.]	1024 Lines
Vert. Back Porch [typ.]	38 Lines
Vert. Front Porch [typ.]	1 Line

Result:

Pixel Clock	108	(2 x 54 MHz)
Horizontal Active	1280	(2 x 640 Clocks → Horizontal Addr. Time)
Horizontal Blank	408	((844 – 640) x 2 Clocks)
Horizontal Sync Offset	48	(2 x 24 Clocks → Horizontal Front Porch)
Horizontal Sync Width	112	(((408/2 – 124 – 24) x 2) → H. Blank – H. Back Porch – H. Front Porch)
Vertical Active	1024	(Vertical Addr. Time)
Vertical Blank	42	(1066 – 1024 Lines)
Vertical Sync Offset	1	(Vertical Front Porch)
Vertical Sync Width	3	(Vertical Blank – Vertical Back Porch – Vertical Front Porch)

The following picture shows the typical video timing.



3.1.2 24 Bit Color Mapping Tips

The double pixel clock or 24-bit color depth can generally be taken from the datasheet. There are two interface modes existing at 24-bit color depth: **FPDI** (Flat Panel Display Interface) or **LDI** (LVDS Display Interface). Some panels use the line SELL LVDS (SELect Lvds data order). The LVDS data assignment in the datasheet can give you an indication by the last channel (e.g. RX3/TX3 – SELL LVDS = low) wether it is a LDI panel (contains the lowest bits). Most panels have a FPDI interface.

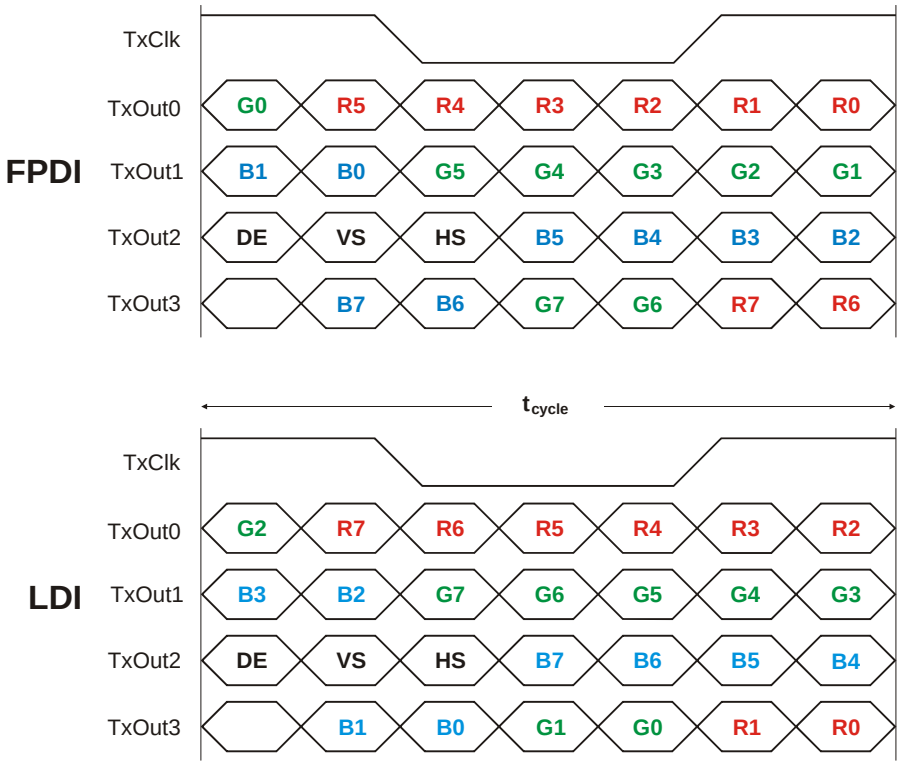
Example:

FPDI data assignment (LVDS channel 3 even or odd):

Tx/Rx27	Red 6 (e.g. even: RE6 or ER6)
Tx/Rx5	Red 7
Tx/Rx10	Green 6 (e.g. even: GE6 or EG6)
Tx/Rx11	Green 7
Tx/Rx16	Blue 6 (e.g. even: BE6 or EB6)
Tx/Rx17	Blue 7
Tx/Rx23	not used

LDI data assignment (LVDS channel 3 even or odd):

Tx/Rx27	Red 0 (e.g. even: RE0 or ER0)
Tx/Rx5	Red 1
Tx/Rx10	Green 0 (e.g. even: GE0 or EG0)
Tx/Rx11	Green 1
Tx/Rx16	Blue 0 (e.g. even: BE0 or EB0)
Tx/Rx17	Blue 1
Tx/Rx23	not used



3.2 EDID 1.3 Specification (VESA)

The EDID (Extended Display Identification Data) record has a fixed structure. The first 8 bytes contain the distinctive identification 00h, FFh, FFh, FFh, FFh, FFh, FFh, 00h. The end of the record is marked by the checksum (1 byte). The result of the addition of all bytes including the checksum has to be zero.
For a comprehensive support of the majority of available panels you don't need all fields of the EDID record. The **Detailed Timing Descriptor** (18 bytes) is the most important field. No 24bit panels (FPDI/LDI) are supported though. This means EDID should only be used for 18bit panels.
For further information please consult the official EDID specification from the VESA comitee which has to be payed.

3.3 DisplayID™ Specification (VESA)

Intended as a replacement for all previous EDID versions DisplayID™ contains many new features. It's a structure with several well defined elements (tags). Not every element that is listed in the specification has to be part of the resulting data set (basic section).
KONTRON has decided to use this selection of tags (mandatory presence).

Tag	Description
00h	Product Identification Data Block (Vendor ID, Product Code, Manufacturing Date ...)
03h	Type I Detailed Timing Data Block (Pixel Clock, Horizontal/Vertical Data ...)
0Ch	Display Device Data Block (Device Technology, Operating Mode, Color Depth ...)
0Dh	Interface Power Sequencing Data Block (Power On/Off Timing)
0Fh	Display Interface Data Block (Interface Type, Interface Attribute ...)

3.3.1 DisplayID™ Parameter Summary

Only a part of the parameters used in the DisplayID™ Windows® tool are interpreted by a specific board. The following table shows a summary of the used parameters (valid for JRex*plus* LX).

Group	Parameter	Comment
Type I Timing	**Pixel Clock**	
Type I Timing	**Horizontal Active**	
Type I Timing	**Horizontal Blank**	
Type I Timing	**Horizontal Sync Offset**	Front porch
Type I Timing	**Horizontal Sync Width**	
Type I Timing	**Vertical Active**	
Type I Timing	**Vertical Blank**	
Type I Timing	**Vertical Sync Offset**	Front porch
Type I Timing	**Vertical Sync Width**	
Display Interface 2	**Signal Polarity**	Only H-Sync and V-Sync
Display Interface 2	**DE Mode**	

3.3.2 DisplayID™ Restrictions

Depending on the graphic controller not all features can be used. The following table shows the most important restrictions.

Restrictions for JRex*plus* LX
Panels with dual or quad clock not supported (2 or 4 Pixel per Clock)
Panels with LDI 24bit color mapping not supported
Variable power sequencing not supported

DisplayID™ is not a solution for every LCD. Many displays center the picture automatically using the DE-signal. In general the picture is centered automatically in horizontal direction but some displays also center the picture vertically using this signal. Only for displays that can be driven with a **fixed mode** timing a displacement of the picture is possible. For these information please check the datasheet. With the JRex*plus* LX board the parameters for the typical detailled timing out of the datasheet might not be correct for centering the picture. In general the exact positioning of the picture has to be obtained experimental.

3.3.3 LCD Panel Selection

The choice of an LCD display is basically defined by two parameters.

Parameter	Value
Pixel per Clock (Channels)	**1**
Maximum Pixel Clock	**87 MHz**

Currently this leads to a maximum resolution of

1366 Pixel (e.g. SHARP LK315T3LA31)

With the AMD graphic driver it is not guaranteed that every resolution can be achieved. KONTRON does not guarantee the correct function of the board for untypical resolution. In principal the use of DisplayID™ allows realizing every special display resolution. For this a valid DisplayID™ dataset must be written to the onboard EEPROM. Additionally the BIOS Setup entry

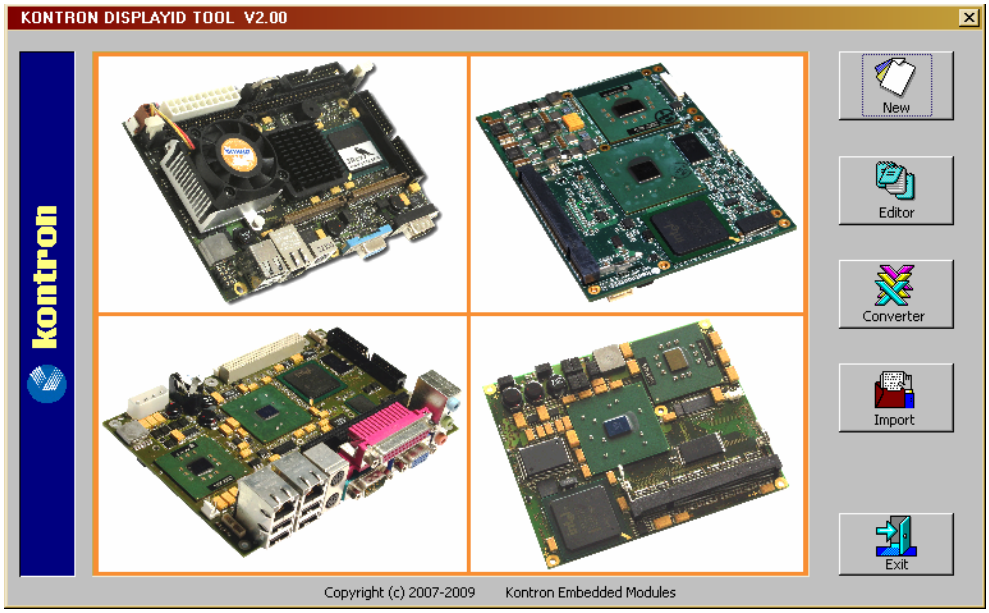
Advanced Chipset Features/LCD Resolution

must be set to **Auto**.

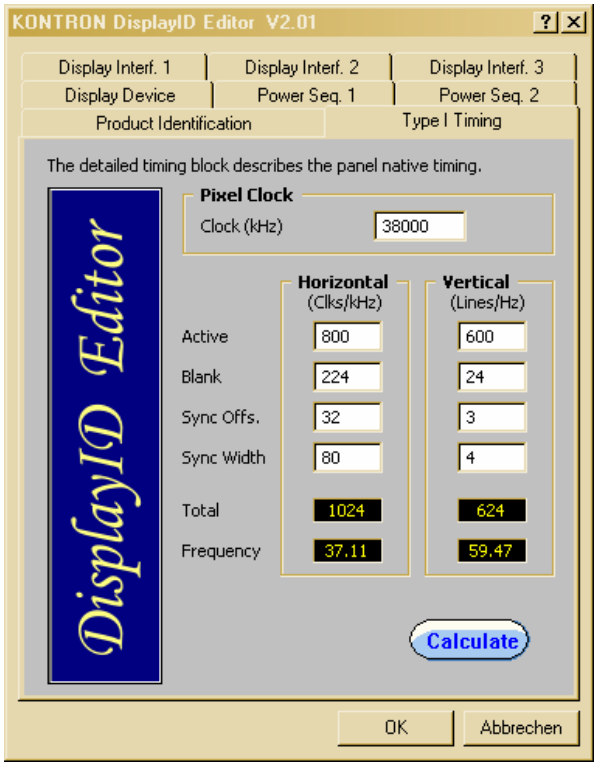
Many displays with a resolution up to XGA (1024 x 768) have a digital (TTL) interface. KONTRON offers a special adapter to connect these displays to the LVDS interface (KAB-ADAPT-LVDStoTTL with part number 61029).

3.3.4 DisplayID™ Windows® Tool

The DisplayID™ parameter can be modified with the DisplayID™ Windows® tool.



For an example the following picture shows the input fields for the **Detailed Timing** parameters.



For more information see the documentation of the DisplayID™ tool (software can be downloaded from kontron.com).

The DisplayID™ Editor saves the parameters in a intermediate file format. The file extension is 'KDD' (Kontron DisplayID™ Data). This file format cannot be used to program the onboard EEPROM.
For transfering this file format into the binary file format for the EEPROM apply the Converter.

3.3.5 Building DisplayID™ File

❶ Start the Windows® tool **DisplayID.exe**.
❷ Use the **Editor** if you want to modify an existing DisplayID™ file or select **New** to create a complete new record.
❸ Change respectively enter new parameters.
❹ Save the parameters in a file with the extension 'KDD'.
❺ Open the saved 'KDD'-file using the **Converter**.
❻ Save the binary file with the extension 'KDB' (Kontron DisplayID™ Binary).
❼ Program the onboard EEPROM using the board specific DOS update tool.

3.3.6 Erasing DisplayID™ Record

Programming the first 128 bytes in the EEPROM with the values 00h or FFh deletes a valid DisplayID™ record.

3.3.7 EEPROM Update Tool

The syntax of the DOS EEPROM update tool is:

BLX8-DID <option> <filename>

/W = read a file (must be KDB-format) and write the content to the EEPROM
/R = read the EEPROM and write the content to a file
/C = read a file and compare the content with the EEPROM
/D = clear the EEPROM content (without filename)

4 Serial-ATA Interface

4.1 Windows® XP SP2 or SP3 Boot

To install Windows® XP on a Serial-ATA harddisk without a legacy floppy drive the file TXTSETUP.OEM must be changed respectively recreated. The following modifications allows the installation from a USB floppy drive.
Example for a new TXTSETUP.OEM file:

```
[Disks]
disk1 = "VIA_SCSI", viamraid.inf

[Defaults]
scsi = VCOMBORAID_I386

[scsi]
VCOMBORAID_I386 = "VIA V-RAID Controller Series (Windows XP)"

[Files.scsi.VCOMBORAID_I386]
driver  = disk1, viamraid.sys, CFG
inf     = disk1, viamraid.inf
catalog = disk1, viamraid.cat

[config.CFG]
value = parameters\PnpInterface,5,REG_DWORD,1

[HardwareIds.scsi.VCOMBORAID_I386]
id = "PCI\VEN_1106&DEV_3349&CC_0104","viamraid"
id = "PCI\VEN_1106&DEV_6287&CC_0106","viamraid"
id = "PCI\VEN_1106&DEV_0591&CC_0104","viamraid"
id = "PCI\VEN_1106&DEV_3249&CC_0104","viamraid"
id = "PCI\VEN_1106&DEV_3149&CC_0104","viamraid"
id = "PCI\VEN_1106&DEV_3164&CC_0104","viamraid"
id = "PCI\VEN_1106&DEV_0581&CC_0104","viamraid"

;--The following lines give additional USB floppy support
id = "USB\VID_03F0&PID_2001", "usbstor" #--HP
id = "USB\VID_08BD&PID_1100", "usbstor" #--Iomega
id = "USB\VID_0409&PID_0040", "usbstor" #--NEC
id = "USB\VID_055D&PID_2020", "usbstor" #--Samsung
id = "USB\VID_0424&PID_0FDC", "usbstor" #--SMSC
id = "USB\VID_054C&PID_002C", "usbstor" #--Sony
id = "USB\VID_057B&PID_0001", "usbstor" #--Y-E Data
```

Copy all files from the directory 'i386/NT5' (VIAMRAID.CAT, VIAMRAID.INF, VIAMRAID.SYS) and in addition the new TXTSETUP.OEM file at the root of a floppy disk.

Attention: *The legacy floppy controller must be disabled (Setup entry 'Integrated Peripherals/Floppy Controller').*

If a USB floppy drive is still not detected properly the parameters can be determined as follows:
Plugin the USB floppy drive on any Desktop PC running the Windows® XP operating system and open the device manager in the system control panel. The floppy drive should be listed below the entry **Universal Serial Bus Controller**. Goto it's properties, select the details tab, then Hardware IDs where you can see **USB\VID_ ... &PID_ ....** These values must be added as a new line to the file TXTSETUP.OEM.
Presumeably this addition must correspond with the file **USBSTOR.INF**. If the floppy drive identification is not implemented in this file the installation might fail.

Manufacturer	ID in USBSTOR.INF available
HP	✔
Iomega	**Indefinite**
NEC	✔
Samsung	**Indefinite**
SMSC	✔
Sony	✔
Y-E Data	✔

5 Watchdog, Digital I/O and Backlight

The following examples (DOS programs) show the access to these three system components (C compiler: BORLAND C++).
Note: These examples can't be run on Linux and Windows®.

5.1 Watchdog Example

```
#include <dos.h>

#define JIDA16_INT                  0x15
#define WDT_INIT                    0xE000
#define WDT_TRIGGER                 0xE001
#define WDT_TIMEOUT                 50    // timeout = 10 seconds (time base = 0.2 seconds)
#define WDT_DELAY                   0     // no delay

void ActivateWatchdog (void)
{
    union REGS regs;

    regs.x.ax = WDT_INIT;
    regs.x.bx = WDT_TIMEOUT;
    regs.x.cx = WDT_DELAY;
    regs.x.dx = 0;                  // only RESET possible
    int86 (JIDA16_INT, &regs, &regs);
}

void TriggerWatchdog (void)
{
    union REGS regs;

    regs.x.ax = WDT_TRIGGER;
    int86 (JIDA16_INT, &regs, &regs);
}

void main (void)
{
    int i;

    ActivateWatchdog ();
    for (i = 0; i < 5; i++)         // wait half of expiry time (= 5 seconds)
      delay (1000);                 // wait 1 second
    TriggerWatchdog ();             // trigger the watchdog - total expiry time now 15 seconds
}
```

5.2 Digital I/O Example

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>

#define JIDA16_INT                  0x15
#define GPIO_OUTPUT                 0xEA51
#define GPIO_INPUT                  0xEA52
#define JIDA_ID                     0x4648
#define GPIO_MASK                   0x0F
#define BOARD_NUM                   1

void WriteDigitalIO (unsigned char value)       // only lower nibble valid
{
    union REGS regs;

    regs.x.ax = GPIO_OUTPUT;
    regs.x.dx = JIDA_ID;
    regs.h.cl  = BOARD_NUM;
    regs.h.ch = (value & GPIO_MASK);
    int86 (JIDA16_INT, &regs, &regs);
}

unsigned char ReadDigitalIO (void)              // only lower nibble valid
{
    union REGS regs;

    regs.x.ax = GPIO_INPUT;
    regs.x.dx = JIDA_ID;
    regs.h.cl  = BOARD_NUM;
    int86 (JIDA16_INT, &regs, &regs);
    return (regs.h.bl & GPIO_MASK);
}

void main (void)
{
    unsigned char val;
    char str [32];

    WriteDigitalIO (0x05);
    getch ();
    WriteDigitalIO (0x0A);
    getch ();
    val = ReadDigitalIO ();
    sprintf (str, "\n%02X\n", val);
    printf (str);
}
```

5.3 Backlight Example

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>

#define JIDA16_INT          0x15
#define BACKL_OUT           0xEA29
#define BACKL_IN            0xEA28
#define JIDA_ID             0x4648
#define BOARD_NUM           1

void WriteBacklight (unsigned char value)
{
    union REGS regs;

    regs.x.ax = BACKL_OUT;
    regs.x.dx = JIDA_ID;
    regs.h.cl  = BOARD_NUM;
    regs.h.ch = value;
    int86 (JIDA16_INT, &regs, &regs);
}

unsigned char ReadBacklight (void)
{
    union REGS regs;

    regs.x.ax = BACKL_IN;
    regs.x.dx = JIDA_ID;
    regs.h.cl  = BOARD_NUM;
    int86 (JIDA16_INT, &regs, &regs);
    return regs.h.ch;
}

void main (void)
{
    unsigned char val;
    char str [32];

    WriteBacklight (0x80);
    val = ReadBacklight ();
    sprintf (str, "\n%02X\n", val);
    printf (str);
}
```

6 Remote Control (JRC)

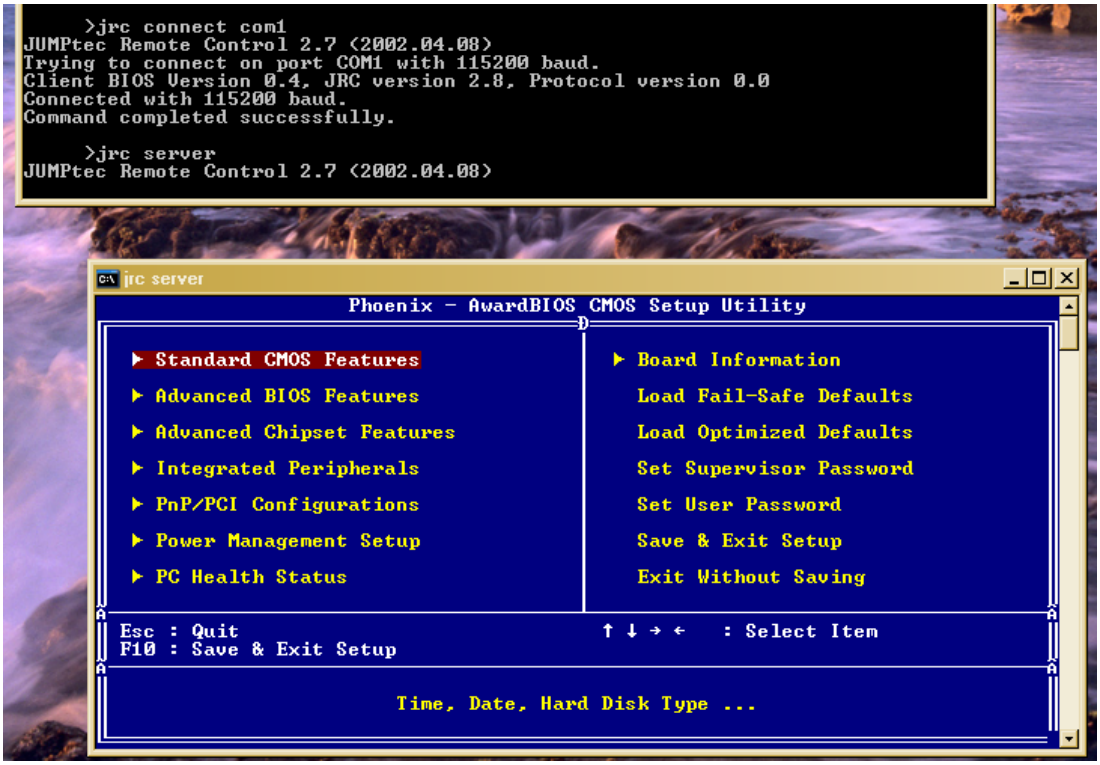
The Remote Control option provides a way to intercept and reroute certain BIOS functionality over a serial port at an early stage during the boot process. There are two distinct software components involved: the first component is part of the BIOS (client), the second component runs on a different device (host) that is connected using a serial cable to the client. This component is available as a WIN32 (JRC.EXE respectively jrc1i1xx.exe) or as a MS-DOS (JRCD.EXE repectively jrc1i2xx.exe) application.
The 'server' function is a very useful feature. The BIOS settings can be changed using this mode or the client boots from the floppy inserted in the host's floppy drive A. Exit the 'server' mode by pressing simultaneously both control keys on the host keyboard.
The following steps are necessary:
Open a command prompt window on the host
Issue a connect command with the appropriate COM port

e.g. **JRC connect COM1**

Note that the 'server' command contains an implicit connect and by default uses the port and baudrate of the last successful manual connect. Type

JRC server or **JRC server A:**

The following picture shows an example (press the 'Del' key on the client keyboard):



7 JIDA32 Interface

Most KONTRON single board computers (SBCs) are equipped with unique hardware features that cannot be accessed with standard API. The JIDA32 interface allows you to access this features in a hardware independent manner under popular 32-bit operating systems.
Not mentioned parts of the JIDA32 interface are not supported and can lead to wrong results.

7.1 Generic Part

Each SBC has a unique seven letter name that corresponds directly with the physical type of board. Examples are **PLX8**, **PDOT**, **BQBA** and **B690**. Boards are also divided into classes. The currently defined classes are **CPU**, **VGA** and **IO**. Each board has one primary class but it can also have any number of secondary classes. This allows you to talk to a class of boards that has a particular functionality without knowing the exact name of the board.

Identifier	Value
Board name	**BLX8**
Primary class	**CPU**
Secondary class	**VGA**
Boot counter	**0 ... 65535**

Note: *The boot counter is only incremented when a cold boot is performed. For a warm boot this value is not changed.*

7.2 Display Part

In this part only the backlight control can be used. The contrast control is generally not supported (modern graphic controllers don't anymore contain a STN LCD interface).

Identifier	Value
Backlight on/off	**On or Off**
Backlight brightness	**0 ... 255**
Contrast on/off	**Not supported**
Contrast value	**Not supported**
End of dark boot	**Not supported**

7.3 I2C-Bus Part

This part allows the access to serial busses. A write access is not allowed on every device though. KONTRON does not guarantee the correct function of the component respectively the warranty claim is lost in case of unallowed write cycles.

Bus Number	Technology	Type	Device Count
0	I2C (primary)	JIDA	1
1	SMBus	Generic	2
2	I2C	JILI	1

7.3.1 Bus Number 0 (JIDA)

This bus allows access to the JIDA EEPROM in which KONTRON specific manufacturing parameters are stored. A damage of these parameters leads to a loss of warranty. Due to this a write cycle may only be performed above a defined address.

Device	Address	Size	Read Access	Write Access
JIDA EEPROM	**A0h**	512 Bytes	Yes	No, forbidden

7.3.2 Bus Number 1 (Generic)

Through this bus several devices can be handled.

Device	Address	Size	Read Access	Write Access
GPIO Control	**30h**	34 Bytes	Yes	Yes
SPD EEPROM	**A0h**	256 Bytes	Yes	No, forbidden

7.3.3 BusNumber 2 (JILI)

The JILI EEPROM refers to the LCD/LVDS interface. It contains the panel specific timing parameters (e.g. a DisplayID™ record).

Device	Address	Size	Read Access	Write Access
JILI EEPROM	**A0h**	512 Bytes	Yes	Yes

7.4 CPU Performance Part

This part implements power management functions. The CPU frequency and the CPU throttling can be controlled.

Function	Supported
CPU throttling	**No**
CPU frequency	**No**

7.5 Hardware Monitor Part

The hardware monitor part contains in most cases several subsections.

Section	Sensor Count
Temperature	**2**
Fan	**Not supported**
Voltage	**6**

7.5.1 Temperature

The term On-Chip diode designates the chip temperature of the Super I/O temperature sensor (with no dependence to the CPU temperature).

Sensor	Number	Abs. Thermal Limit
On-Chip diode	**0**	
CPU diode	**1**	0 to +80^0 C

7.5.2 Voltage

The Super I/O can simultaneously monitor six analog voltage inputs.

Source	Number	Comment
CPU core	**0**	
+2.6V	**1**	DDR-SDRAM
Battery	**2**	RTC / CMOS values
+3.3V	**3**	
+5.0V	**4**	
+5.0V SB	**5**	Standby voltage

7.6 Digital I/O Part

This part defines the availability of digital input/output lines.

Type	Port	Pin Count	Position
Input	**0**	4	Bit 0 - 3
Output	**0**	4	Bit 4 - 7
Input/Output	**1 - 7**	Not supported	

7.7 Watchdog Part

The watchdog can be programmed with discrete timeout values on boards with CPLD implementation or in millisecond resolution with boards using a Super-I/O. In the discrete value scenario, if there is no exact match to the timeout value, the next higher one is used.

Type	Steps	Timeout Values	Result
Super-I/O	every 200 ms	**1 second to 254 minutes**	RESET NMI not supported

7.8 JIDA32 Windows® Programming

For further information see the actual JIDA32 documentation (JIDA32.pdf).

7.8.1 Program Language C

The demo program reads and shows the board name and the first 16 bytes of SPD EEPROM (SMBus). The program uses the static linked library JIDA.LIB.

Example:

```
#include <windows.h>
#include "jida.h"

#define I2C_BUS      1
#define DEV_ADDR   0xA0

INT WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, INT nCmdShow)
{
    BOOL    bRet;
    HJIDA   hJida = (DWORD) NULL;
    CHAR    szStr1 [128],
            szStr2 [32],
            szVal [JIDA_BOARD_MAX_SIZE_ID_STRING];
    UCHAR  uVal [32];

    if (JidaDllInitialize ())
    {
      if (JidaDllIsAvailable ())
      {
        if (JidaBoardOpen (JIDA_BOARD_CLASS_CPU, 0, JIDA_BOARDINFO_FLAGS_DEFAULT, &hJida))
        {
          bRet = JidaBoardGetName (hJida, (LPTSTR) szVal, JIDA_BOARD_MAX_SIZE_ID_STRING);
          wsprintf (szStr1, "JidaBoardGetName = %d / %s", bRet, szVal);
          lstrcpy (szStr2, "DEMO");
          MessageBox (NULL, szStr1, szStr2, MB_OK | MB_ICONEXCLAMATION);
          JidaI2CRead (hJida, I2C_BUS, DEV_ADDR, (LPBYTE) &uVal[0], 16);
          wsprintf (szStr1, "JidaI2CRead = %02X %02X %02X %02X %02X %02X %02X %02X",
                                                    uVal[0], uVal [1], uVal[2], uVal [3],
                                                    uVal[4], uVal [5], uVal[6], uVal [7]);
          MessageBox (NULL, szStr1, szStr2, MB_OK | MB_ICONEXCLAMATION);
          JidaBoardClose (hJida);
        }
      }
      JidaDllUninitialize ();
    }
    return (INT) FALSE;
}
```

7.8.2 Program Language DELPHI

The demo program activates the watchdog (timeout = 30 seconds). The keyword 'var' passes the argument by reference.

Example:

```
unit mainU;

interface

uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls;

type
    HJIDA = LongInt;

type
    TForm1 = class (TForm)
    Button1: TButton;
    procedure Button1Click (Sender: TObject);
private
    { Private-Deklarationen }
public
    hJida: HJIDA;
    { Public-Deklarationen }
end;

function JidaDllInitialize : Boolean {$IFDEF WIN32} stdcall {$ENDIF}; external 'JIDA.DLL';
function JidaDllUninitialize : Boolean {$IFDEF WIN32} stdcall {$ENDIF}; external 'JIDA.DLL';
function JidaDllIsAvailable : Boolean {$IFDEF WIN32} stdcall {$ENDIF}; external 'JIDA.DLL';
function JidaBoardOpen (pszClass:PChar; dwNum:LongInt; dwFlags:LongInt; var phJida:HJIDA) : Boolean
                        {$IFDEF WIN32} stdcall {$ENDIF}; external 'JIDA.DLL';
function JidaWDogSetConfig (hJida:HJIDA; dwType:LongInt; dwTimeout:LongInt; dwDelay:LongInt;
                        dwMode:LongInt) : Boolean {$IFDEF WIN32} stdcall {$ENDIF}; external 'JIDA.DLL';

var
    Form1: TForm1;
const
    JIDA_BOARD_CLASS_CPU = 'CPU'#0;
    JIDA_FLAGS_DEFAULT = 0;
    JIDA_TIMEOUT_VALUE = 30000;
    JIDA_DELAY_VALUE = 0;                   // Delay not supported
    JIDA_REBOOT_MODE = 0;                   // NMI not supported

implementation

{$R *.dfm}
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
{Method 1}
    if JidaDLLInitialize () then
    begin
      if JidaDllIsAvailable () then
      begin
        if JidaBoardOpen (JIDA_BOARD_CLASS_CPU, 0, JIDA_FLAGS_DEFAULT, hJida) then
          JidaWDogSetConfig (hJida, 0, JIDA_TIMEOUT_VALUE, JIDA_DELAY_VALUE, JIDA_REBOOT_MODE);
      end;
    JidaDllUninitialize ();
    end;
end;

end.
```

The associated DFM file:

```
object Form1: TForm1
    Left = 196
    Top = 107
    Width = 367
    Height = 390
    Caption = 'KONTRON JIDA32 TEST'
    Color = clBtnFace
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -11
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    OldCreateOrder = False
    PixelsPerInch = 96
    TextHeight = 13
    object Button1: TButton
      Left = 104
      Top = 128
      Width = 75
      Height = 25
      Caption = 'Test JIDA32'
      TabOrder = 0
      OnClick = Button1Click
    end
end
```

The associated DPR file:

```
program Jidatest;

uses
    Forms,
    mainU in 'mainU.pas' {Form1};

{$R *.res}

begin
    Application.Initialize;
    Application.CreateForm (TForm1, Form1);
    Application.Run;
end.
```

7.8.3 Program Language VISUAL BASIC (VB.NET)

The demo program shows the board count value and activates the watchdog (timeout = 10 seconds).

Example:

```
Public Class JidaTest

    Declare Auto Function JidaDllInitialize Lib "JIDA.DLL" () As Boolean
    Declare Auto Function JidaDllUninitialize Lib "JIDA.DLL" () As Boolean
    Declare Auto Function JidaDllIsAvailable Lib "JIDA.DLL" () As Boolean
    Declare Auto Function JidaBoardCount Lib "JIDA.DLL" _
            (ByVal classstr As String, ByVal flags As UInteger) As UInteger
    Declare Auto Function JidaBoardOpen Lib "JIDA.DLL" _
            (ByVal classstr As String, ByVal num As UInteger, ByVal flags As UInteger, _
            ByRef handle As UInteger) As Boolean
    Declare Auto Function JidaBoardClose Lib "JIDA.DLL" (ByVal handle As UInteger) As Boolean
    Declare Auto Function JidaWDogSetConfig Lib "JIDA.DLL" (ByVal handle As UInteger, _
            ByVal type As UInteger, ByVal timeout As UInteger, ByVal delay As UInteger, _
            ByVal mode As UInteger) As Boolean

    Public Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
      Const JIDA_BOARD_CLASS_CPU = "CPU"
      Const JIDA_BOARD_OPEN_FLAGS_PRIMARYONLY = 1
      Const JIDA_BOARDINFO_FLAGS_DEFAULT = 0
      Const TIMEOUT_VALUE = 10000
      Const DELAY_VALUE = 0                         'Delay not supported
      Const REBOOT_BOARD = 0                        'NMI not supported
      Dim handle As UInteger

      If JidaDllInitialize() Then
        If JidaDllIsAvailable() Then
          If JidaBoardOpen(JIDA_BOARD_CLASS_CPU, 0, JIDA_BOARDINFO_FLAGS_DEFAULT, handle) Then
            MsgBox("BoardCount = " & JidaBoardCount(JIDA_BOARD_CLASS_CPU, _
                                                    JIDA_BOARD_OPEN_FLAGS_PRIMARYONLY))
            JidaWDogSetConfig(handle, 0, TIMEOUT_VALUE, DELAY_VALUE, REBOOT_BOARD)
            JidaBoardClose(handle)
          End If
        End If
        JidaDllUninitialize()
      End If
    End Sub

End Class
```

The associated Designer file:

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> Partial Class JidaTest
    Inherits System.Windows.Forms.Form

    <System.Diagnostics.DebuggerNonUserCode()> Protected Overrides Sub Dispose(ByVal disposing As Boolean)
      Try
        If disposing AndAlso components IsNot Nothing Then
          components.Dispose()
        End If
        Finally
        MyBase.Dispose(disposing)
      End Try
    End Sub

    Private components As System.ComponentModel.IContainer

    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
      Me.Button1 = New System.Windows.Forms.Button
      Me.SuspendLayout()
      '
      'Button1
      '
      Me.Button1.Location = New System.Drawing.Point(38, 39)
      Me.Button1.Name = "Button1"
      Me.Button1.Size = New System.Drawing.Size(75, 23)
      Me.Button1.TabIndex = 0
      Me.Button1.Text = "Run Test"
      Me.Button1.UseVisualStyleBackColor = True
      '
      'JidaTest
      '
      Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
      Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
      Me.ClientSize = New System.Drawing.Size(154, 96)
      Me.Controls.Add(Me.Button1)
      Me.Name = "JidaTest"
      Me.Text = "JidaTest"
      Me.ResumeLayout(False)
    End Sub
    Friend WithEvents Button1 As System.Windows.Forms.Button

End Class
```

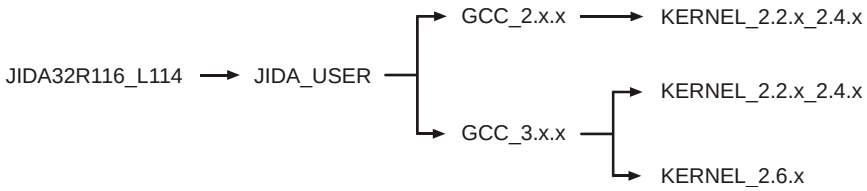
7.8.4 Module Definition File

The calling program can refer to the function by name or by ordinal value. The tool IMPDEF.EXE (e.g. BORLAND C++) make it possible to generate the DEF-file (from JIDA.DLL 06/07/2004, in newer DLLs the ordinal value can be changed).

```
EXPORTS
    JidaBoardClose                  @9
    JidaBoardCount                  @6
    JidaBoardCountA                 @50
    JidaBoardCountW                 @55
    JidaBoardGetBootCounter         @12
    JidaBoardGetBootErrorLog        @75
    JidaBoardGetInfo                @11
    JidaBoardGetInfoA               @54
    JidaBoardGetInfoW               @59
    JidaBoardGetName                @10
    JidaBoardGetNameA               @53
    JidaBoardGetNameW               @58
    JidaBoardGetOption              @14
    JidaBoardGetRunningTimeMeter    @13
    JidaBoardOpen                   @7
    JidaBoardOpenA                  @51
    JidaBoardOpenByName             @8
    JidaBoardOpenByNameA            @52
    JidaBoardOpenByNameW            @57
    JidaBoardOpenW                  @56
    JidaBoardSetOption              @15
    JidaDllGetVersion               @2
    JidaDllInitialize               @3
    JidaDllInstall                  @40
    JidaDllIsAvailable              @5
    JidaDllUninitialize             @4
    JidaFanCount                    @83
    JidaFanGetCurrent               @85
    JidaFanGetInfo                  @84
    JidaFanSetLimits                @86
    JidaI2CCount                    @29
    JidaI2CIsAvailable              @30
    JidaI2CRead                     @31
    JidaI2CReadRegister             @33
    JidaI2CType                     @63
    JidaI2CWrite                    @32
    JidaI2CWriteReadCombined        @69
    JidaI2CWriteRegister            @34
    JidaIOCount                     @35
    JidaIOGetDirection              @70
    JidaIOGetDirectionCaps          @72
    JidaIOGetNameA                  @73
    JidaIOGetNameW                  @74
```

7.9 JIDA32 Linux Programming

Please note that the JIDA32 package does not include full sources. Instead precompiled objects are provided that can be used to build a JIDA32 package for a certain environment (GCC, kernel, libc).
In order to handle GCC version incompatibilities and different kernel module build environments the package includes different branches (you can use the GCC_3.x.x subdirectory for GCC 4.x.x compiler versions).



Under each branch four subdirectories can be found:

JidaDrv: Includes the necessary source, library and build files to create the JIDA32 kernel driver module (jida.ko for kernels 2.6.x, jida.o for kernels 2.2.x and 2.4.x).

JidaLib: Includes the necessary source, library and build files to create the JIDA32 interface library (libjida.so, libjida.a).

JidaTst: Includes the necessary source and build files to create a simple JIDA32 test application (jidatst).

JWdogTst: Includes the necessary source and build files to create a simple watchdog test application (jwdogtst).

In order to build the JIDA32 kernel driver module (jida.ko/jida.o) you must install the complete kernel sources for the destination kernel under **/usr/src/linux** or alternatively provide an environment variable **KERNELDIR** that contains the path of the kernel sources.
After installation of the sources please configure, build and install the respective kernel and modules on your system. Having done so please reboot and start the system using the new kernel.

In order to build the JIDA32 kernel driver, interface library and test applications go to the
./JIDAR116_L114/JIDA_USER/'GCC_VERSION'/'KERNEL_VERSION'
directory and enter 'make all'.

In order to automatically copy the created files to their destination directories please enter 'make install' afterwards. This will copy the following files:

jida.ko to **/lib/modules/$(KERNELRELEASE)/extra/** (for kernel 2.6.x)
or
jida.o to **/lib/modules/$(KERNELRELEASE)/misc/** (for kernel 2.2.x/2.4.x)

libjida.so + libjida.a to **/usr/lib/**

jidatst + jwdogtst to **/usr/bin/**

jida.h + jwindefs.h to **/usr/include/**

You can provide a prefix for the above named directories with the **INSTALL_MOD_PATH** environment variable if you want to install the files into an alternative root file system. (Note: kernel 2.6.x only)
After successfull build and installation you should run the sample application jidatst which will display the following message:

JIDA system driver is incompatible or not installed.
Would you like to install it? (yes or no)

If you answer this question with 'yes' or 'y' the device node **/dev/jida** will be created and the driver module loaded. Afterwards some basic JIDA32 test calls will be performed which display their results on the screen. If you see this output the JIDA32 interface is operational.
If you have problems running JIDA please check if the device node **/dev/jida** is created with the correct major/minor number and is accessible by the active user. Since version JIDAR115_L113 we are using 10/250 for device nodes if a kernel of the 2.6.x branch is used. JIDA drivers for older kernels or JIDA32 revisions use 99/0 for the device node.

Note: *JIDA won't be detected automatically on kernel startup. You have to load it by yourself. You can either use the JidaDllInstall function to do it or use "**modprobe jida**" before starting your application. Most Linux distributions provide other possibilities to automatically load kernel modules. For Debian simply add a line with "**jida**" to the **/etc/modules** file.*

Note: *If you are using **udev** with a 2.6 kernel the device node will automatically be created. The default major/minor will be 10/250 which is reserved for local use. If this is conflicting with your own driver you can redefine the minor id to something else by providing the minor=x parameter when loading the module. Example: modprobe jida minor=254.*

Appendix A: Reference Documents

KONTRON Technology A/S can't guarantee the availability of internet addresses.

Document	Internet Address
Advanced Configuration and Power Interface (ACPI)	http://www.acpi.info/spec.htm
AT Attachment Storage Interface Specification (ATA)	http://t13.org
Digital Visual Interface (DVI)	http://www.ddwg.org
High Definition Audio Specification (HD Audio)	http://www.intel.com/standards/hdaudio
High Speed Serialized AT Attachment (S-ATA)	http://www.sata-io.org/developers
IEEE 802.3 Specification (Ethernet)	http://standards.ieee.org/getieee802
Low Pin Count Interface Specification (LPC-Bus)	http://developer.intel.com/design/chipsets/industry/lpc.htm
Open LVDS Display Interface Standard Spec. (Open LDI)	http://www.national.com/analog/displays/open_ldi
PCI Express Base Specification (PCI Express)	http://www.pcisig.com/specifications
SD Specification (SD Card)	http://www.sdcard.org/developers/tech/sdio/sdio_spec
System Management Bus Specification (SMBus)	http://www.smbus.org/specs
Universal Serial Bus Specification (USB)	http://www.usb.org/developers/docs

Appendix B: Document Revision History

Revision	Date	Author	Changes
S0004-A	11/04/10	M. Hüttmann	Replaced the JIDA32 Delphi example
S0004-0	12/11/09	M. Hüttmann	First release

Corporate Offices

Europe, Middle East & Africa

Oskar-von-Miller-Str. 1
85386 Eching/Munich
Germany
Tel.: +49 (0)8165/ 77 777
Fax: +49 (0)8165/ 77 219
info@kontron.com

North America

14118 Stowe Drive
Poway, CA 92064-7147
USA
Tel.: +1 888 294 4558
Fax: +1 858 677 0898
info@us.kontron.com

Asia Pacific

17 Building,Block #1,ABP
188 Southern West 4th Ring Road
Beijing 100070, P.R.China
Tel.: + 86 10 63751188
Fax: + 86 10 83682438
info@kontron.cn