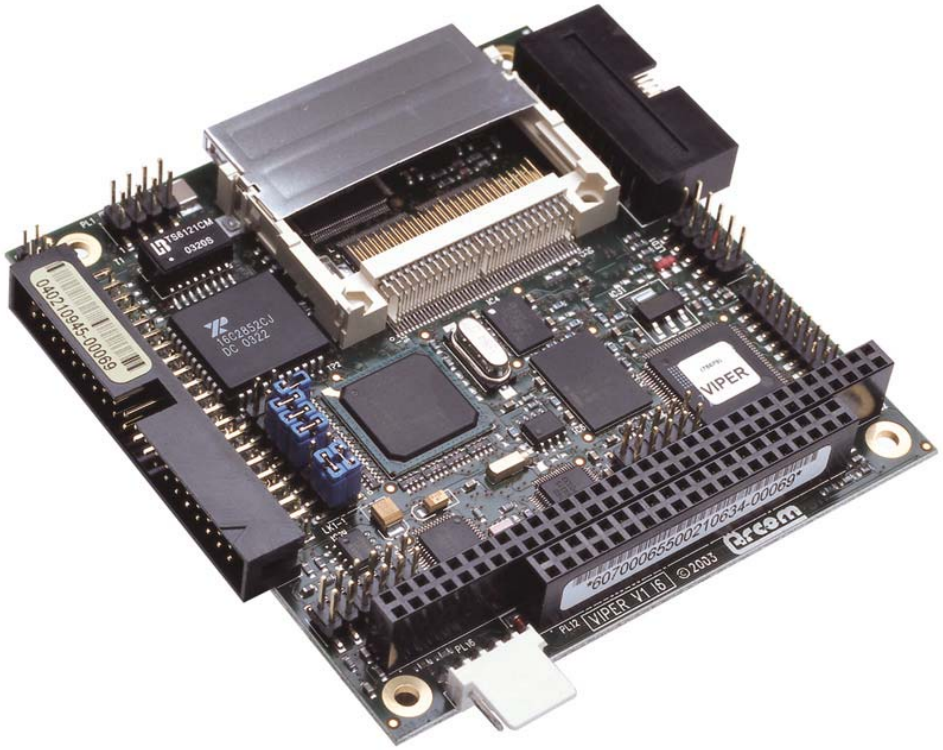
VIPER

Windows CE .NET 4.2

Technical Manual



Arcom

www.arcom.com

a spectris company

Definitions

Arcom is the trading name for Arcom Control Systems Inc and Arcom Control Systems Ltd.

Disclaimer

The information in this manual has been carefully checked and is believed to be accurate. Arcom assumes no responsibility for any infringements of patents or other rights of third parties, which may result from its use.

Arcom assumes no responsibility for any inaccuracies that may be contained in this document. Arcom makes no commitment to update or keep current the information contained in this manual.

Arcom reserves the right to make improvements to this document and/or product at any time and without notice.

Warranty

This product is supplied with a full 3 year warranty. Product warranty covers failure caused by any manufacturing defects. Arcom will make all reasonable effort to repair the product or replace it with an identical variant. Arcom reserves the right to replace the returned product with an alternative variant or an equivalent fit, form and functional product. Delivery charges will apply to all returned products. Please go to www.arcom.com/support for information about Product Return Forms.

Trademarks

Windows CE .NET is a trademark of the Microsoft Corporation.
All other trademarks recognized**.**

Revision History

Manual	*PCB*	*Date*	*Comments*
Issue A	V1 Issue 3	9th September 2003	First release
Issue B	V1 Issue 4	5th November 2003	Updated for CE 4.2 release
Issue C	V1 Issue 4	15th March 2004	Updated for CE 4.2 release 2
Issue C	V1 Issue 4	21st June 2004	Minor updates
Issue D	V1 Issue 4	11th August 2004	Minor edits, updated layout.

Arcom is a subsidiary of Spectris plc.
For contact details, see page 40.

Arcom operates a company-wide quality management system which has been certified by the British Standards Institution (BSI) as compliant with ISO9001:2000

Contents

Handling your board safely

Environmental

The VIPER enclosure is fitted with the VIPER-UPS. The battery fitted to the VIPER-UPS is a 7-cell battery pack containing Varta V500 HRT NiMH (Nickel Metal Hydride) cells. These cells contain 0% lead, 0% mercury and 0% cadmium.

Anti-static handling

The Viper and other circuit boards fitted inside the VIPER ICE contain CMOS devices. These could be damaged in the event of static electricity being discharged through them. At all times, please observe anti-static precautions when handling circuit boards. This includes storing boards in appropriate anti-static packaging and wearing a wrist strap when handling.

ElectroMagnetic Compatibility (EMC)

The VIPER is classified as a component with regard to the European Community EMC regulations and it is the user's responsibility to ensure that systems using the board are compliant with the appropriate EMC standards.

Packaging

Please ensure that should a board need to be returned to Arcom it is adequately packed preferably in the original packing material.

About this manual

This manual provides detailed information about the VIPER board supplied with the Windows CE .NET 4.2 operating system pre-loaded. It provides:

- Details about the hardware supported by the operating system.
- Information that you will find useful if you intend to develop applications for Windows CE .NET.
- Support libraries.

Related documents

More detailed information can be found in the Documentation folder on the User CD, including:

- The VIPER Board Manual (PDF).
- The VIPER CE .NET Quickstart Manual (PDF).
- Information about other items included in the Development Kit.

Conventions

Symbols

The following symbols are used in this guide:

Symbol	Explanation
	Note - information that requires your attention.
	Tip - a handy hint that may provide a useful alternative or save time.
	Caution – proceeding with a course of action may damage your equipment or result in loss of data.

Typographical conventions

Different fonts are used throughout the manual to identify different types of information, as follows:

Font	Explanation
Italics	Parts of a command that should be substituted with appropriate values.
Bold	Information that you enter yourself.
`Screen text`	Information that is displayed on screen.

Terminology

The following terms are used throughout this manual:

Term	Definition
User CD	The Arcom VIPER Windows CE .NET 4.2 Development Kit CD, supplied in the Development Kit.
VIPER CE Quickstart	The VIPER CE .NET Quickstart Manual, contained in the Documentation folder on the User CD.

Operating system support

Version

The board is supplied with a version of the Windows CE .NET 4.2 image, pre-loaded onto the on-board flash.

The image size is 21MB, and the boot time is approximately 15 seconds.

Included components

A list of included components is included in the Documentation folder on the User CD.

Hardware supported in this release

The operating system supports the following hardware:

- LCD display.
- Persistent registry.
- Flash storage. (8MB storage space available. Formatted using TFAT).
- CompactFlash storage cards.
- USB storage.
- USB human interface devices (keyboard/mouse).
- Five COM ports[1] - four RS232 and one RS422 / RS485.
- Touchscreen interface.
- 10/100 baseT Ethernet support.
- Audio.
- PC/104 interface.
- Watchdog.
- UPS power supply.
- GPIO lines.

[1] COM3 is normally used for touchscreen support.

Flat panels and video modes

The image loaded onto the VIPER is configured for the supplied NEC NL3224BC35-20 QVGA flat panel.

Support is provided for other flat panels. If you want to use a different flat panel, you must use the regpatch tool to run the supplied registry patch for the flat panel being used.

See Software tools, page 13, for more details about regpatch.

The flat panels supported are listed below.

NEC

Type	Part number	Panel size	Resolution	Standard orientation	180° rotation
TFT	NL3224BC35-20	5.5"	320 x 240	tftqvga.reg	tftqvgar.reg
TFT	NL6448BC20-08E	6.5"	640 x 480	tft6448.reg	tft6448r.reg
TFT	NL6448BC26-01	8.4"	640 x 480	tft6448.reg	tft6448r.reg

Hitachi

Type	Part number	Panel size	Resolution	Standard orientation	180° rotation
TFT	TX14D11VM1CAA	5.7"	320 x 240	tftqvga.reg	tftqvgar.reg
STN	SX14Q004-ZZA	5.7"	320 x 240	stnqvga.reg	stnqvgar.reg

Other QVGA or VGA TFT flat panels with a digital 6-bit parallel interface may also work. Please contact Arcom customer support for more details. (See Appendix A – Contacting Arcom, page 40.)

Ethernet

The current build of Windows CE .NET 4.2 supports the on-board LAN91C111 Ethernet controller.

If the CE system is not connected to a network with a DHCP server running, a static IP address should be set up.

To set up a static IP address:

1. In the **Control Panel** choose **Network and Dial-up Connections**.
2. Double-click the **LAN90001** adaptor.
3. Click the **IP Address** tab and specify the **IP Address**, **Subnet Mask** and **Default Gateway** as required.
4. Click the **Name Servers** tab, and set the **DNS** as required.
5. Save the settings.
6. Reboot the CE system.

To check the IP configuration, run **ipconfig** from the command window.

Touchscreen

The touchscreen controller application automatically runs on power up.

The touchscreen is set up and calibrated during the production process. These settings remain intact until you:

- Start from a cleared registry.
  If the registry is cleared (see Clearing the registry, page 11), the Port and Mode Detection process begins, followed immediately by the calibration process.
- Launch the calibration application.
  If TouchCal.exe is run, the touchscreen calibration process begins without running the Port and Mode Detection process.

Port and Mode Detection process

The Port and Mode Detection process detects which COM port the touchscreen is plugged into, and the baud rate being used. Follow these steps:

1 Touch the cross in the center of the bottom half of the screen.

2 When prompted, lift the pointing device off the touchscreen.

3 Touch the cross again.

4 When prompted, lift the pointing device off the touchscreen.

Touchscreen calibration process

To calibrate the touchscreen using TouchCal.exe, follow these steps:

1 Touch the cross in the top left corner of the screen.

You can move the pointing device while keeping in contact with the touchscreen, to get it into the exact position required.

2 Lift the pointing device off the touchscreen. (This is not prompted.)

3 Touch the cross in the lower right corner of the screen.

4 Lift the pointing device off the touchscreen. (This is not prompted.)

5 Touch the cross in the center of the screen.

6 Lift the pointing device off the touchscreen. (This is not prompted.)

7 Move the pointing device around the screen and check the accuracy of the pointer.

If the accuracy is not acceptable, touch the **ReCalibrate** button and start the calibration process again.

If the accuracy is acceptable touch the **Save** button to save the settings.

Registry

The operating system image supports a persistent registry. The registry link should normally be in the Retain Registry position (see the diagram below).

Saving the registry

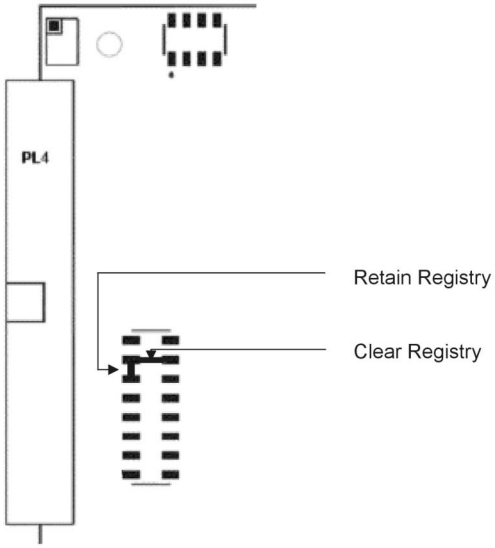
Calling the RegFlushKey function from within an application saves the registry.

Arcom's SaveReg.exe application uses RegFlushKey to save the registry. The source code for SaveReg.exe is in the Examples folder on the User CD.

Clearing the registry

The registry can be reverted to its original default values. To do this, move the registry link to the Clear Registry position while the board boots up. When the board has booted, move the link back to the Retain Registry position.

These link positions are shown in the following diagram:



With the link in the Clear Registry position, the startup folder will not be processed, and applications will not autorun.

UPS power supply

The UPS power supply control application is in the \FlashDisk\Startup folder, so that it launches automatically on power up.

For more information about this application, please refer to the UPS power supply application document in the Documentation folder on the User CD.

To enable the UPS battery, LK6 on the VIPER-UPS board must be fitted.

Real time clock

To change the date or time, double click on the clock in the task bar. The **Date/Time Properties** dialog is displayed, from where you can make the changes you require.

The time and date can be retained in the event of power loss on the main VIPER 5V supply by fitting an external battery to power the device. Details can be found in the *Power and power management* section of the VIPER Technical manual, which is in the documentation section of the User CD.

The default time zone is GMT. If you change the time zone, you must save the registry. Failure to save the time zone in the registry can result in incorrect daylight saving time adjustment dates.

Software tools

Regpatch

The persistent registry can be patched with registry files (*.reg) using the regpatch tool.

To run a registry patch:

1. Copy the filename.reg file from the registry\files section of the User CD to the root directory of the VIPER CE system.
2. Open a command prompt.
3. Enter **regpatch filename.reg**

   The registry is then patched. The following message is displayed when this is complete:

   ```
   Registry saved
   ```

   If this message does not appear, run **savereg** to save the registry changes.

4. Reboot the system.

Getflash

The entire 32MB StrataFlash can be captured to a binary file. This file can then be used to duplicate other systems.

To capture the image onto a CompactFlash card:

1. Copy getflash.exe onto the CompactFlash card, then insert it into the VIPER.
2. Open a command prompt.
3. Enter **\cfdisk\getflash \cfdisk\imagename.bin 04000000 02000000**

Strataprog

Strataprog is used to restore a backed up StrataFlash image onto a VIPER board that is running Windows CE.

To restore a StrataFlash image:

1. Copy the image file and strataprog.exe onto the CompactFlash card, then insert it into the VIPER.
2. Open a command prompt.
3. Enter **cfdisk\strataprog –b=04000000 \cfdisk\imagename.bin**

Application development

Applications for Windows CE .NET can be developed using .NET Compact Framework or eMbedded Visual C++.

Using .NET Compact Framework

For details of application development using the .NET Compact Framework, see the .NET Compact Framework section on the User CD.

Using eMbedded Visual C++

eMbedded Visual C++ 4.0

Install Microsoft eMbedded Visual C++ 4.0 according to Microsoft's instructions.

The SDKs supplied with eMbedded Visual Tools are not required, but installing the H/PC Pro SDK allows application emulation on the host system.

Emulation of Arcom hardware is not possible.

eMbedded Visual C++ 4.0 Service Pack 2

If installing Service Pack 2 was not an option on the eMbedded Visual C++ 4.0 disk, it must be downloaded and installed before applications can be developed for a Windows CE .NET 4.2 platform. It can be downloaded from:

http://msdn.microsoft.com/vstudio/device/embedded/download.asp

A link to this site is provided on the User CD.

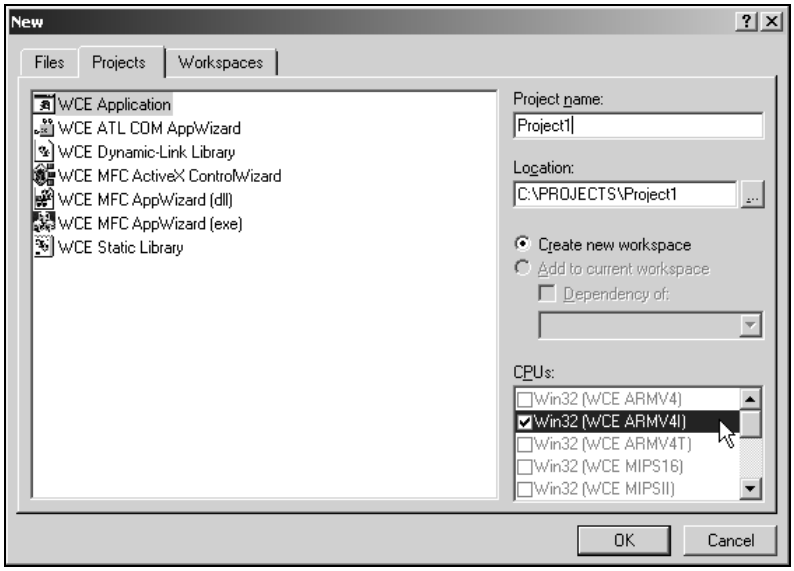
Arcom Platform SDK

The Arcom Platform SDK for Visual C++ must be installed in order to build applications for the Arcom VIPER board.

Select the Platform SDK section of User CD, then install the Visual C++ SDK.

CPU selection

When starting a new project, select the Win32 (WCE ARMV4I) processor in the **CPUs** pane of the **New** dialog, **Projects** tab.

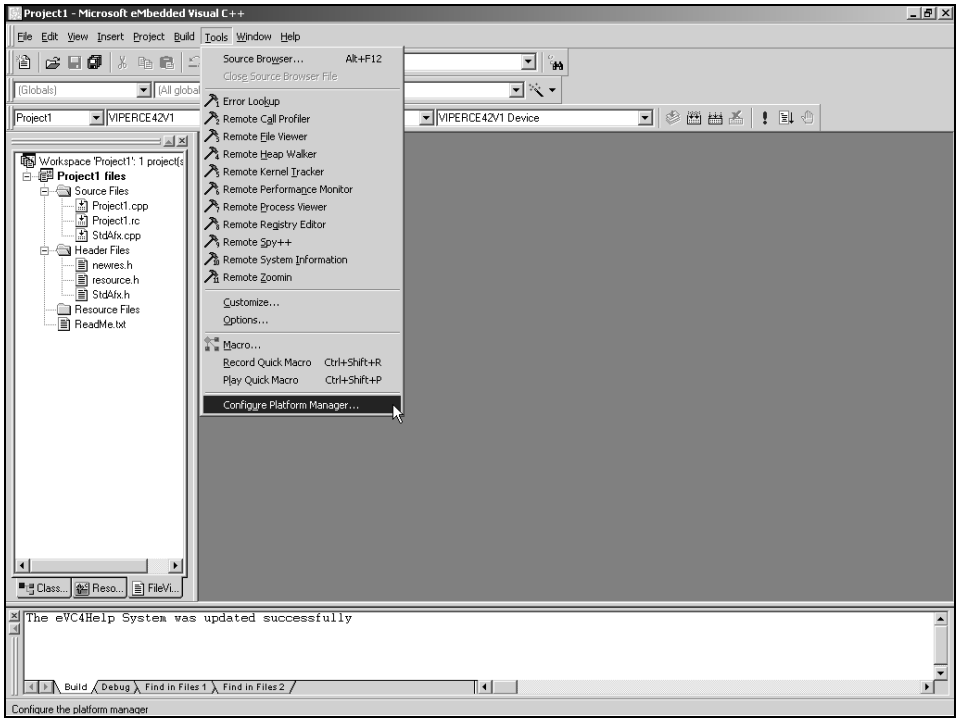


Establishing a remote debugging connection

To establish a remote debugging connection, follow these steps:

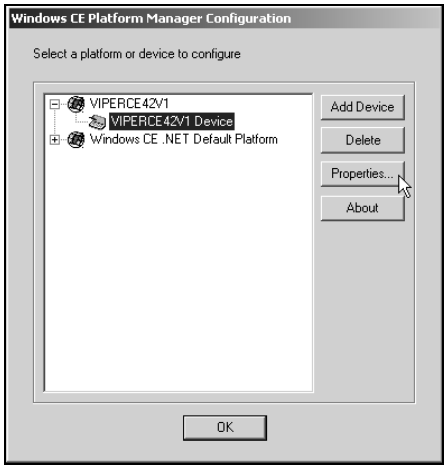
1 Connect the host system and CE system using an ActiveSync connection, as described in the VIPER Windows CE .NET 4.2 Quickstart manual.

2 Within Microsoft eMbedded Visual C++, select **Configure Platform Manager** from the **Tools** menu:



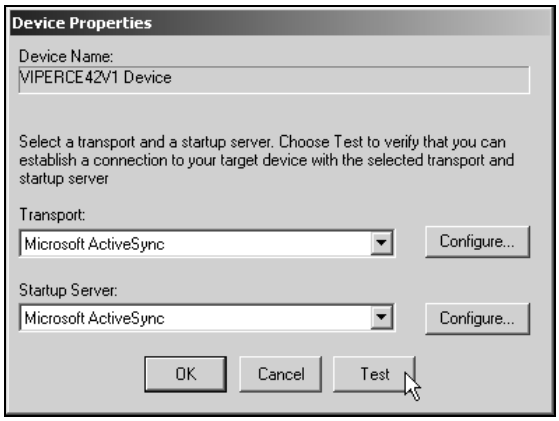
The **Windows CE Platform Manager Configuration** dialog is displayed.

3 Select the SDK to be used, highlight the default device and click on **Properties**:

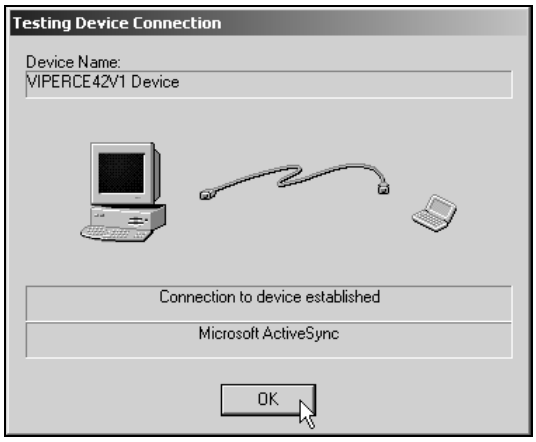


The **Device Properties** dialog is displayed for the device you chose.

4 Select **Microsoft ActiveSync** as both the **Transport** and **Startup Server** and click on **Test**.



The connection is then established:



5 When a connection is established, click on **OK** to close all windows and return to the eMbedded Visual Tools development environment.

Running applications directly from startup

The Registry link must be in the Save registry position for the startup folder to be processed. See the Registry section on page 11 for more details

In order to run files automatically when the system boots, a folder called Startup should be created in the \FlashDisk folder.

.exe files placed in this folder are run when the system starts up.

To specify command line parameters, a startup.ini file should be placed in the startup folder, based on the following sample:

```
.delay 1500
app1.exe
app2.exe /d /v:2
```

The **.delay** can be used to delay the start of an application, if required.

Any dlls that the applications require must also be copied to the \FlashDisk\User folder.

Web server

The VIPER CE system can be used as a web page server.

The location of pages to be served is determined by a registry entry at:

HKEY_LOCAL_MACHINE\Comm\HTP\VROOTS\/

The initial value of this key is \windows\www\wwwpub

To preserve pages when the system is powered down, change this registry entry to a non-volatile storage location, such as \FlashDisk\User.

An asp example is provided on the User CD.

Accessing memory and peripherals directly

Windows CE allows application programs to have direct access to the system memory. This means that developers may choose to drive their own IO devices directly rather than writing a Windows CE driver for it.

Windows CE uses a memory manager to divide up the XScale's memory space between the various applications. These mappings change during normal operation and the active application is placed into a 32MB space starting from address 0. In order to access a peripheral, the application must map the address of the peripheral into the space assigned to the application. This is done using the MmMapIoSpace function:

PVOID MmMapIoSpace(PHYSICAL_ADDRESS address, ULONG size, BOOLEAN cached)

The return value is a pointer that can be used directly in the application code.

In order to use the MmMapIoSpace function, ceddk.h must be included in the header files and ceddk.lib must be added to the list of libraries in the project settings, as shown in the following example:

```
#include <ceddk.h>
#include <stdio.h>

void main(void)
{
     PHYSICAL_ADDRESS address;
     unsigned char *pInputs;

     address.HighPart = 0;
     address.LowPart = 0x14500000;

     pInputs = (unsigned char *)MmMapIoSpace(address, 0x1000, false);
     printf("Inputs: %d\n", *pInputs);
     MmUnmapIoSpace(pInputs, 0x1000);
}
```

Please note:

- PHYSICAL_ADDRESS is a special 64-bit address used to keep compatibility with other CPU types. For XScale the HighPart should always be 0. 0x14500000 is the address of the GPIO input register on the VIPER.
- Although the register takes up only one byte, the size parameter has been set to 0x1000 as the memory manager assigns space in blocks of 0x1000.
- Use MmUnmapIoSpace at the end of the code to release the space.

For peripherals with a number of registers, avoid doing an MmMapIoSpace for each register by defining a structure that represents the device registers, then use a single MmMapIoSpace to get a pointer to the structure.

If registers are relatively close together (less than 256k apart), create one large space that covers both sets then use the same offsets within the mapped space.

For more information, refer to the GPIOTest example program on the User CD.

Using the GPIO lines

The VIPER board has eight output and eight input lines available for use by applications. The output lines are provided by GPIOs 20 through 27 from the XScale CPU itself. The input lines are provided by a single 8-bit memory mapped input register.

A simple demo program called GPIOTest (available on the User CD) shows how to use these lines directly from an application. It also serves as a more advanced example of how to access peripherals directly from an application.

The vipergpio.cpp & vipergpio.h files can be used in user applications. These need to be linked with ceddk.lib.

Support libraries

This section provides detailed information about:

- AIM104 board support. See below.
- Watchdog. See page 38.
- SRAM. See page 39.

AIM104 board support

The Arcom AIM104 DLL (aim104.dll) provides a simple set of functions that allow Arcom's range of AIM104 PC/104 expansion cards to be driven from application programs.

To use these functions, aim104.dll, gpio.dll and spinlock.dll must be in the \Flashdisk\Arcom folder on the VIPER.

The expansion cards available, and the functions that can be used with each, are explained on the following pages:

- AIM104-Relay8/IN8. See below.
- AIM104-IN16, page 25.
- AIM104 OUT-16, page 27.
- AIM104-IO32, page 29.
- AIM104-MULTI-IO, page 33.

AIM104-Relay8/IN8

The AIM104-Relay8/IN8 offers eight opto-isolated inputs and eight single pole double throw relay outputs. The AIM104 DLL considers each board to have one group of eight relays and one group of eight inputs.

In a multi-board system the first board contains relay and input group 0, the second board contains relay and input group 1, and so on. Similarly the first board contains relay or input 0 through 7, the second contains 8 through 15, and so on.

#include file:	relay8.h
Link file:	aim104.lib
VIPER registry patch:	relay8.reg

The functions available for use with the AIM104-Relay8/IN8 are explained on the following pages.

long Relay8Enable(long nGroup, long nState)

Used to enable or disable a group of relays. After a system reset all relays are disabled. Calling this function with a non-zero value for nState enables all relays in the given group; the relays immediately take up the value in the relay output register. Calling this function with a value of zero for nState disables all the relays in the given group; the relays are switched off immediately, and value in the relay output register remains unchanged. Enabling a group that is already enabled, or disabling a group that is already disabled, has no effect.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nGroup	The relay group to be enabled or disabled.
nState	Zero to disable or non-zero to enable the group.

Return value	Explanation
R8_NOERROR	Success call has completed OK.
R8_BADGROUP	nGroup value greater than the number of boards installed.
R8_HAWRWAREWRITE	Low level problem accessing the hardware.

long Relay8RelayWrite(long nRelay, long nState)

Used to set the state of an individual relay without affecting any of the other relays in the group. A non-zero value for nState turns on the given relay. A zero value turns it off. If the corresponding group is disabled at the time it remains disabled but the new value is written into the appropriate output register. Turning on a relay that is already on, or turning off one that is already off, has no effect.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nRelay	The relay group to be written to.
nState	Zero to turn off or non-zero to turn on the relay.

Return value	Explanation
R8_NOERROR	Success call has completed OK.
R8_BADRELAY	nRelay value greater than the number of available relays.
R8_HAWRWAREWRITE	Low level problem accessing the hardware.

long Relay8GroupWrite(long nGroup, long nData)

Allows all eight relays in a group to be written to simultaneously. The lower 8 bits of nData are written to the relay output register. When a bit is set, the corresponding relay is turned on. If the group is disabled at the time, the new value is written to the output register, the relays remain disabled. Only the lower 8 bits of nData are significant.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nGroup	The relay group to be written to.
nData	New value for the outputs (lower 8 bits only).

Return value	Explanation
R8_NOERROR	Success call has completed OK.
R8_BADGROUP	nGroup value greater than the number of boards installed.
R8_HAWRWAREWRITE	Low level problem accessing the hardware.

*long Relay8RelayStatus(long nRelay, long *pStatus)*

Used to test the state of a particular channel in the output register. If the value placed in the variable pointed to by pStatus is zero then the corresponding channel is off; a non-zero value indicates that the channel is on. This function returns the value in the output register even if the corresponding group is disabled at the time.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nRelay	The number of the relay to be read.
pStatus	Pointer to a variable to receive the state of the channel.

Return value	Explanation
R8_NOERROR	Success call has completed OK.
R8_BADRELAY	nRelay value greater than the number of available relays.
R8_HARDWAREREAD	Low level problem accessing the hardware.

*long Relay8GroupStatus(long nGroup, long *pStatus)*

Returns the current state of the output register. The value in the lower 8 bits placed into the variable pointed to by pStatus reflects the current state of the output register. A zero bit indicates that the corresponding channel is off. The higher order bits are always zero. The value of the output register is returned even if the corresponding group is disabled.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nGroup	The relay group to be written to.
pStatus	Pointer to a variable to receive the state of the channel.

Return value	Explanation
R8_NOERROR	Success call has completed OK.
R8_BADGROUP	nGroup value greater than the number of boards installed.
R8_HAWRWAREWRITE	Low level problem accessing the hardware.

*long Relay8ReadInput(long nInput, long *pState)*

Returns the state of an individual opto-isolated input. If the value placed in the variable pointed to by pState is non-zero, then the corresponding input is on. If the value is zero the input is off.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nInput	The input to be read.
pState	Pointer to a variable to receive the state of the channel.

Return value	Explanation
R8_NOERROR	Success call has completed OK.
R8_BADINPUT	nInput value greater than the number of available inputs.
R8_HARDWAREREAD	Low level problem accessing the hardware.

*long Relay8GroupInput(long nGroup, long *pState)*

Allows all eight opto-isolated inputs in a group to be read simultaneously. The lower 8 bits of the value placed in the variable pointed to by pState indicate the state of each of the eight channels. If a bit is zero the corresponding input is off.

This is the inverse of the actual data read back from the hardware, but having a consistent 'zero means off' improves compatibility with other routines.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nGroup	The relay group to be written to.
pState	Pointer to a variable to receive the state of the channel.

Return value	Explanation
R8_NOERROR	Success call has completed OK.
R8_BADGROUP	nGroup value greater than the number of boards installed.
R8_HARDWAREREAD	Low level problem accessing the hardware.

AIM104-IN16

The AIM104-IN16 offers 16 opto-isolated inputs in two groups of 8. The AIM104 DLL considers each board to have two groups of 8 inputs. In a multi-board system the first board contains input groups 0 and 1, the second board contains input groups 2 and 3, and so on. Similarly the first board contains individual inputs 0 through 15, the second contains 16 through 3, and so on.

#include file: in16.h
Link file: aim104.lib
VIPER registry patch: in16.reg

The functions available for use with the AIM104-IN16 are explained on the following pages.

*long IN16ReadInput(long nInput, long *pState)*

Returns the state of an individual opto-isolated input. If the value placed in the variable pointed to by pState is non-zero then the corresponding input is on. If the value is zero the input is off.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nInput	The input to be read.
pState	Pointer to a variable to receive the state of the input.

Return value	Explanation
IN16_NOERROR	Success call has completed OK.
IN16_BADINPUT	nInput value greater than the number of available inputs.
IN16_HARDWAREREAD	Low level problem accessing the hardware.

*long IN16GroupInput(long nGroup, long *pState)*

Allows a group of eight opto-isolated inputs to be read simultaneously. The lower 8 bits of the value place in the variable pointed to by pState indicate the state of each of the eight channels. If a bit is zero the corresponding input is off.

This is the inverse of the actual data read back from the hardware, but having a consistent ‘zero means off’ improves compatibility with other routines.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nGroup	The input group to be read.
pState	Pointer to a variable to receive the state of the channel.

Return value	Explanation
IN16_NOERROR	Success call has completed OK.
IN16_BADGROUP	nGroup value greater than the number of boards installed.
IN16_HARDWAREREAD	Low level problem accessing the hardware.

AIM104 OUT-16

The AIM104-OUT16 offers 16 opto-isolated outputs in two groups of 8. The AIM104 DLL considers each board to have two groups of 8 outputs. In a multi-board system the first board contains output groups 0 and 1, the second board contains input groups 2 and 3, and so on. Similarly the first board contains individual outputs 0 through 15, the second contains 16 through 31, and so on.

#include file:	out16.h
Link file:	aim104.lib
VIPER registry patch:	out16.reg

The functions available for use with the AIM104 OUT-16 are explained below.

long OUT16WriteOutput(long nOutput, long nState)

Used to set the state of an individual output without affecting any other output in the group. A non-zero value for nState turns on the given output. A zero value turns it off. If the corresponding group is disabled at the time, it remains disabled but the new value is written into the appropriate output register. Turning on an output that is already on, or turning one off that is already off, has no effect.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nOutput	The output to be written.
nState	New state of the output zero for off non-zero for on.

Return value	Explanation
OUT16_NOERROR	Success call has completed OK.
OUT16_BADINPUT	nOutput greater than the number of available outputs.
OUT16_HARDWAREWRITE	Low level problem accessing the hardware.

long OUT16GroupWrite(long nGroup, long nData)

Allows all eight outputs in a group to be written to simultaneously. The lower 8 bits of nData are written to the output register. When a bit is set, the corresponding relay is turned on. If the group is disabled at the time the new value is written to the output register the relays remain disabled. Only the lower 8 bits of nData are significant.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nGroup	The input group to be read.
nData	New data for the output group.

Return value	Explanation
OUT16_NOERROR	Success call has completed OK.
OUT16_BADGROUP	nGroup value greater than the number of boards installed.
OUT16_HARDWAREREAD	Low level problem accessing the hardware.

*long OUT16ReadStatus(long nOutput, long *pStatus)*

Used to test the state of a particular channel in the output register. If the value placed in the variable pointed to by pStatus is zero, the corresponding channel is off. A non-zero value indicates that the channel is on. This function returns the value in the output register even if the corresponding group is disabled at the time.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nOutput	The number of the output to be read.
pStatus	Pointer to a variable to receive the state of the channel.

Return value	Explanation
R8_NOERROR	Success call has completed OK.
R8_BADOUTPUT	nOutput value greater than the number of available relays.
R8_HARDWAREREAD	Low level problem accessing the hardware.

*long OUT16GroupStatus(long nGroup, long *pStatus)*

Returns the current state of the output register. The value in the lower 8 bits placed into the variable pointed to by pStatus reflect the current state of the output register. A zero bit indicates that the corresponding channel is off. The higher order bits are always zero. The value of the output register is returned even if the corresponding group is disabled.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nGroup	The relay group to be written to.
pStatus	Pointer to a variable to receive the state of the channel.

Return value	Explanation
R8_NOERROR	Success call has completed OK.
R8_BADGROUP	nGroup value greater than the number of boards installed.
R8_HARDWAREREAD	Low level problem accessing the hardware.

AIM104-IO32

The AIM104-IO32 offers 32 TTL level open collector outputs. The state of each output can also be monitored. When an output is off it may be driven by an external source allowing the line to be used for input.

The AIM104 DLL considers each board to have four groups of 8 IO lines. Each group may be accessed as inputs or outputs. In a multi-board system the first board contains groups 0 through 3, the second board contains groups 4 through 7, and so on. Similarly the first board contains individual IO lines 0 through 31, the second contains 32 through 63, and so on.

#include file: io32.h
Link file: aim104.lib
VIPER registry patch: io32.reg

The functions available for use with the AIM104-IO32 are explained on the following pages.

long IO32Enable(long nGroup, long nState)

Used to enable or disable a group of outputs. After a system reset all outputs are disabled. The hardware on the AIM104-IO32 is only capable of enabling all four groups simultaneously. However, to improve compatibility with other boards, this function still considers each board to have four groups. Enabling any group on a board enables the other three as well. Calling this function with a non-zero value for nState enables all three groups; the outputs immediately take up the value in the output registers. Calling this function with a value of zero for nState disables all three groups; the outputs are switched off immediately but the value in the output registers is not changed. Enabling a group that is already enabled or disabling a group that is already disabled has no effect.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nGroup	The IO group to be enabled or disabled.
nState	Zero to disable or non-zero to enable the group.

Return value	Explanation
IO32_NOERROR	Success call has completed OK.
IO32_BADGROUP	nGroup value greater than the available groups.
IO32_HAWRWAREWRITE	Low level problem accessing the hardware.

long IO32WriteOutput(long nOutput, long nState)

Used to set the state of an individual output without affecting any of the other output in the group. A non-zero value for nState turns on the given output; a zero value turns it off. If the corresponding group is disabled at the time, it remains disabled but the new value is written into the appropriate output register. Turning on an output that is already on, or turning off one that is already off, has no effect.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nOutput	The output to be written.
nState	New state of the output zero for off non-zero for on.

Return value	Explanation
IO32_NOERROR	Success call has completed OK.
IO32_BADINPUT	nOutput greater than the number of available outputs.
IO32_HARDWAREWRITE	Low level problem accessing the hardware.

long IO32GroupWrite(long nGroup, long nData)

Allows all eight outputs in a group to be written to simultaneously. The lower 8 bits of nData are written to the output register. When a bit is set the corresponding output is turned on. If the group is disabled at the time the new value is written to the output register, the outputs remain disabled. Only the lower 8 bits of nData are significant.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nGroup	The input group to be read.
nData	New data for the output group.

Return value	Explanation
IO32_NOERROR	Success call has completed OK.
IO32_BADGROUP	nGroup value greater than the number of boards installed.
IO32_HARDWAREREAD	Low level problem accessing the hardware.

*long IO32ReadStatus(long nOutput, long *pStatus)*

Used to test the state of a particular channel in the output register. If the value placed in the variable pointed to by pStatus is zero, the corresponding channel is off; a non-zero value indicates that the channel is on. This function returns the value in the output register even if the corresponding group is disabled at the time.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nOutput	The number of the output to be read..
pStatus	Pointer to a variable to receive the state of the channel.

Return value	Explanation
IO32_NOERROR	Success call has completed OK.
IO32_BADOUTPUT	nOutput value greater than the number of outputs.
IO32_HARDWAREREAD	Low level problem accessing the hardware.

*long IO32GroupStatus(long nGroup, long *pStatus)*

Returns the current state of the output register. The value in the lower 8 bits placed into the variable pointed to by pStatus reflects the current state of the output register. A zero bit indicates that the corresponding channel is off. The higher order bits are always zero. The value of the output register is returned even if the corresponding group is disabled.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nGroup	The relay group to be written to.
pStatus	Pointer to a variable to receive the state of the channel.

Return value	Explanation
IO32_NOERROR	Success call has completed OK.
IO32_BADGROUP	nGroup value greater than the available groups.
IO32_HARDWAREREAD	Low level problem accessing the hardware.

*long IO32ReadInput(long nInput, long *pState)*

Returns the state of an individual input. If the value placed in the variable pointed to by pState is non-zero, the corresponding input is on. If the value is zero the input is off.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nInput	The input to be read.
pState	Pointer to a variable to receive the state of the input.

Return value	Explanation
IO32_NOERROR	Success call has completed OK.
IO32_BADINPUT	nInput value greater than the number of available inputs.
IO32_HARDWAREREAD	Low level problem accessing the hardware.

*long IO32GroupInput(long nGroup, long *pState)*

Allows a group of eight inputs to be read simultaneously. The values in the lower 8 bits placed into the variable pointed to by pState indicate the state of each of the eight channels. If a bit is zero, the corresponding input is off. The upper bits of the value in pState are always zero.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nGroup	The input group to be read.
pState	Pointer to a variable to receive the state of the channel.

Return value	Explanation
IO32_NOERROR	Success call has completed OK.
IO32_BADGROUP	nGroup value greater than the number of boards installed.
IO32_HARDWAREREAD	Low level problem accessing the hardware.

AIM104-MULTI-IO

The AIM104-MULTI-IO provides 8 opto-isolated digital inputs, 2 analog outputs (voltage or current loop) and 16 single-ended or 8 differential analog inputs.

The code that implements the API must accommodate multiple boards, but you can assume that they are contiguous in memory with all boards being set for either single-ended or differential analog inputs (but not a mixture of both types).

In a multi-board system the first board contains analog output groups 0 and 1, the second board contains groups 2 and 3, and so on. Similarly the first board contains digital inputs 0 through 7, the second board contains 8 through 15, and so on.

If the boards are configured for single-ended analog inputs, the first board contains channels 0 through 15, the second board contains channels 16 through 31, and so on.

If the boards are configured for differential analog inputs, the first board contains channels 0 through 7, the second board contains channels 8 through 15, and so on.

The implementation routines must be thread and process safe. Where appropriate, implementation routines should check that the group or input number given is within the number of boards installed in the system.

#include file: multiio.h
Link file: aim104.lib
VIPER registry patch: multiio.reg

The functions available for use with the AIM104-MULTI-IO are explained on the following pages.

long MULTIIODAC(long nChan, long nData)

Used to set the value of an analog output. An nData value of 0 results in a –5V output, an nData value of 2048 results in a 0V output and an nData value of 4095 results in a +5V output.

A value of 0 or 1 on nChan uses channels 0 or 1 on the board at the base address.

A value of 2 or 3 on nChan uses channels 0 or 1 on the second board, which must be 4 addresses from the board at the base address.

A value of 4 or 5 on nChan uses channels 0 or 1 on the third board, which must be 8 addresses from the board at the base address.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nChan	The Channel to be written to.
nData	The value to be written 0 through 4095.

Return value	Explanation
MULTIIO_NOERROR	Success call has completed OK.
MULTIIO_BADCHAN	nChan greater than the number of available channels.

*long MULTIIOADC(long nChan, bool SingleEnded, long *pValue)*

In single-ended mode (SingleEnded set as TRUE):

- A value of 0 through 15 on nChan uses channels 0 through 15 on the board at the base address.
- A value of 16 through 31 on nChan uses channels 0 through 15 on the second board, which must be 4 addresses from the board at the base address.
- A value of 32 through 47 on nChan uses channels 0 through 15 on the third board, which must be 8 addresses from the board at the base address.
- An input value of –5V results in 0 being placed in the variable pointed to by pValue.
- An input value of 0V results in 2048 being placed in the variable pointed to by pValue.
- An input value of +5V results in 4095 being placed in the variable pointed to by pValue.

In differential mode (SingleEnded set as FALSE):

- A value of 0 through 7 on nChan uses channels 0 through 7 on the board at the base address.
- A value of 8 through 15 on nChan uses channels 0 through 7 on the second board, which must be 4 addresses from the board at the base address.
- A value of 16 through 23 on nChan uses channels 0 through 7 on the third board, which must be 8 addresses from the board at the base address.
- A differential input value of –5V results in 0 being placed in the variable pointed to by pValue.
- A differential input value of 0V results in 2048 being placed in the variable pointed to by pValue.
- A differential input value of +5V results in 4095 being placed in the variable pointed to by pValue.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nChan	The Channel to be read.
SingleEnded	True if using single-ended inputs. False if using differential inputs.
pValue	Pointer to a variable to receive the input.

Return value	Explanation
MULTIIO_NOERROR	Success call has completed OK.
MULTIIO_BADCHAN	nChan value greater than the number of boards installed.
MULTIIO_BADDIFCHAN	nChan value greater than the number of differential boards installed.

*long MULTIIOReadInput(long nInput, long *pStatus)*

Returns the state of a single opto-isolated digital input.

A value of 0 through 8 on nInput uses input 0 through 8 on the board at the base address.

A value of 9 through 16 on nInput uses the input 0 through 8 on the second board, which must be 4 addresses from the board at the base address.

A value of 17 through 24 on nInput uses the input 0 through 8 on the third board, which must be 8 addresses from the board at the base address.

If the value placed in the variable pointed to by pStatus is non-zero the corresponding input is on. If the value is zero, the corresponding input is off.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nInput	The number of the input to be read.
pStatus	Pointer to a variable to receive the state of the input.

Return value	Explanation
MULTIIO_NOERROR	Success call has completed OK.
MULTIIO_BADINPUT	nInput value greater than the number of inputs.

*long MULTIIOReadGroup(long nGroup, long *pStatus)*

Allows a group of eight opto-isolated inputs to be read simultaneously.

A value of 0 on nGroup reads the inputs on the board at the base address.

A value of 1 on nGroup reads the inputs on the second board, which must be 4 addresses from the board at the base address.

A value of 2 on nGroup reads the inputs on the third board, which must be 8 addresses from the board at the base address.

The value read is placed in the variable pointed to by pStatus. If a bit is set the corresponding input is on. If a bit is zero, the corresponding input is off.

This is the inverse of the actual data read back from the hardware, but having a consistent ‘zero means off’ improves compatibility with other routines.

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
nGroup	The input group to be read.
pStatus	Pointer to a variable to receive the state of the group.

Return value	Explanation
IO32_NOERROR	Success call has completed OK.
IO32_BADGROUP	nGroup value greater than the number of groups.

Watchdog

The Arcom watchdog driver provides two functions that allow the watchdog to be controlled by application programs.

To use these functions, #include “watchdog.h” and link with watchdog.lib

Once the watchdog has been enabled there is no way to disable it.

The functions available are explained below.

BOOL EnableWatchdog(void)

Enables the PXA255 watchdog (internal timer 3) to reset the processor if it times out. The timeout is set to the longest possible value (about 1150 seconds).

There are no parameters to specify. The values that are returned are explained in the following table:

Return value	Explanation
True	No errors were returned making this call.
False	Errors were returned making this call.

BOOL ToggleWatchdog(DWORD dwTimeout)

Restarts the watchdog timeout. After calling EnableWatchdog, call this function to set the required timeout. The application must call this function again within the timeout period, otherwise the processor is reset.

The parameter you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
dwTimeout	The next timeout interval in milliseconds.

Return value	Explanation
True	No errors were returned making this call.
False	Errors were returned making this call.

SRAM

The onboard SRAM can be written to and read from using two SRAM functions. The data in the SRAM can be made non-volatile by fitting an external battery to power the device in the event of power loss on the main VIPER 5V supply. Details can be found in the *Power and power management* section of the VIPER Technical manual, which is in the documentation section of the USER CD.

To use these functions, #include “sram.h” and link with sram.lib

*BOOL SRAMRead(DWORD dwAddress, DWORD dwSize, void *pBuffer)*

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
dwAddress	Location in SRAM between 0 and 256.
dwSize	Size of the data being read from SRAM.
pBuffer	pointer to a buffer to receive data read.

Return value	Explanation
True	No errors were returned making this call.
False	Errors were returned making this call.

*BOOL SRAMWrite(DWORD dwAddress, DWORD dwSize, void *pBuffer)*

The parameters you can specify and the values that are returned are explained in the following tables:

Parameter	Summary
dwAddress	Location in SRAM between 0 and 256.
dwSize	Size of the data being written to SRAM.
pBuffer	Pointer to a buffer containing data to write.

Return value	Explanation
True	No errors were returned making this call.
False	Errors were returned making this call.

Appendix A – Contacting Arcom

Arcom sales

Arcom's sales team is always available to assist you in choosing the board that best meets your requirements. Contact your local sales office or hotline.

Sales office US

Arcom
7500W 161st Street
Overland Park
Kansas
66085
USA

Tel: 913 549 1000
Fax: 913 549 1002
E-mail: us-sales@arcom.com

Sales office Europe

Arcom
Clifton Road
Cambridge
CB1 7EA
UK

Tel: 01223 411 200
Fax: 01223 410 457
E-mail: euro-sales@arcom.com

Full information about all Arcom products is available on our Web site at www.arcom.com.

While Arcom's sales team can assist you in making your decision, the final choice of boards or systems is solely and wholly the responsibility of the buyer. Arcom's entire liability in respect of the boards or systems is as set out in Arcom's standard terms and conditions of sale. If you intend to write your own low level software, you can start with the source code on the disk supplied. This is example code only to illustrate use on Arcom's products. It has not been commercially tested. No warranty is made in respect of this code and Arcom shall incur no liability whatsoever or howsoever arising from any use made of the code.

Technical support

Arcom has a team of technical support engineers who can provide assistance if you have any problems with your VIPER board.

Technical support US

Tel: 913 549 1010
Fax: 913 549 1001
E-mail: us-support@arcom.com

Technical support Europe

Tel: +44 (0)1223 412 428
Fax: +44 (0)1223 403 409
E-mail: euro-support@arcom.com

Appendix B – Reference information

Product information

Product notices, updated drivers, support material, 24hr-online ordering:

www.arcom.com

PC/104 Consortium

PC/104 specifications. Vendor information and available add on products.

www.PC/104.org

USB Information

Universal Serial Bus (USB) specification and product information

www.usb.org

CFA (CompactFlash Association)

CF+ and CompactFlash specification and product information

www.compactflash.org/

Appendix C – Acronyms and abbreviations

API	Application Program(ming) Interface
COM	Communication Port
CPU	Central Processing Unit (PXA255)
CMOS	Complementary Metal Oxide Semiconductor
EMC	Electromagnetic Compatibility
GPIO	General Purpose Input/Output
IO	Input/Output
LCD	Liquid Crystal Display
NiMH	Nickel Metal Hydride
OS	Operating System
RTC	Real Time Clock
SBC	Single Board Computer
SDRAM	Synchronous Dynamic Random Access Memory
SRAM	Static Random Access Memory
STN	Super Twisted Nematic, technology of passive matrix liquid crystal
TFT	Thin Film Transistor, a type of LCD flat-panel display screen
UPS	Uninterruptible Power Supply
USB	Universal Serial Bus
VGA	Video Graphics Adapter, display resolution 640 x 480 pixels
VIPER-ICE	VIPER-Industrial Compact Enclosure

Index

W