Parallel I2C
Communications Adapter
Windows 95 / 98® Driver Manual

PICADLL/95

Issue 1.4

22/7/99

Welcome to the Calibre I²C for Windows 95 / 98 driver. This driver is designed to enable users to run the parallel I²C Bus operations via the Calibre parallel port I²C adapter (PICA90 or PICA93).

Please note that in this manual sometimes refers to the I²C Communications Adapter User Manual. It will prove helpful if you have this to hand.

If you have any queries relating to this or any other I²C product supplied by Calibre please visit our web site www.calibreuk.com***.***

For technical support please e-mail techsupport@calibreuk.com ***or send your queries by fax to (44) 1274 730960, for the attention of our I²C Technical Support Department.***

For technical support please e-mail techsupport@www.calibreuk.com or send your queries by fax to (44) 01274 730960, for the attention of our I²C Technical Support Department.

COPYRIGHT

Copyright (c) 1999 Calibre UK Ltd
Cornwall House, Cornwall Terrace
Bradford, BD8 7JS. UK.
E-mail: sales@calibreuk.com
Web site www.calibreuk.com
All World-wide Rights Reserved
Issue 1.4 22/7/99

Contents

INTRODUCTION

1.1. General Introduction

This document details the installation and use of the parallel port I^2C communications adapter (PICA) Windows 95 / 98 ® driver.

This driver is suitable for both the PICA90 and the PICA93LV.

1.2. System Requirements

This driver is suitable for Windows 95 / 98 ®

1.3.. Configuring the Adapter

NOTE: MANY COMPONENTS ON THE ADAPTER CARD ARE STATIC SENSITIVE. OBSERVE NORMAL STATIC SENSITIVE PRECAUTIONS WHEN HANDLING THE CARD!

See the manual supplied with the adapter for details on how to configure the adapter.

1.4. Installing the Adapter

Install the adapter into the PC in accordance with the adapter manual.

INSTALLING THE DRIVER INTO YOUR SYSTEM

2.1. File Location

Unlock the files in accordance with the instructions provided with the licence.

1)Copy the .VXD files to the C:\WINDOWS\SYSTEM\VMM32

2)Copy the .LIB into your compilers \lib

3)Copy the .DLL into \WINDOWS\SYSTEM

4)Copy the .H file into your project directory

5)Visual Basic users include the .BAS file in your project

2.2. Registering the Driver

From the DOS prompt run the PREG_95.exe program .

Guidance relating to the registry can be found on www.calibreuk.com.

2.3. Windows LIB/DLL Functions

2.3.1. Files

PREG_95.exe	Driver registration application
WRtDev0.VXD	BlueWater Systems WinRT® driver
\cpp\PICA_C95.H	"C++" function prototypes
\cpp\PICA_C95.LIB	I^2C "C++" library
\cpp\PICA_C95.DLL	"Dynamic link library"
\vb50\PICA_95.BAS	"Visual Basic 5.0 declarations"
\vb50\PICA_95.H	"C" function prototypes
\vb50\PICA_95.LIB	I^2C "C" library
\vb50\PICA_95.DLL	"Dynamic link library"

2.4. Introduction

Each utility is documented in a standard format which lists its name, usage, function and effect on the adapter is given. The adapter should be setup prior to any data transfer.

2.5. C++ functions - Function Prototypes

To ensure the prototypes are added correctly copy the file PICA_C95.H) into the directory containing your project and add the line:

```
#include "PICA_C95.H
```

The following functions are implemented in the windows libraries:-

```
extern __declspec(dllimport) int  setup (int, int, int, int);
extern __declspec(dllimport) int   sendaddress (int, int );
extern __declspec(dllimport) int   restart (int, int);
extern __declspec(dllimport) int   getstatus (void);
extern __declspec(dllimport) int   writebyte (int );
extern __declspec(dllimport) int   readbyte (int );
extern __declspec(dllimport) int   sendstop (void);
extern __declspec(dllimport) int   recover (void);
extern __declspec(dllimport) int  slavelastbyte (void);
extern __declspec(dllimport) int   dllversion (void);
```

These are implemented in the CPP library

2.6. VB5.0 functions - Function Declarations

To ensure the prototypes are added correctly copy the file PICA_95.BAS into the directory containing your project and add the file to your project line:

The following functions are implemented in the windows libraries:-

```
Public Declare Function setup Lib "pica_95.dll" (ByVal baseaddress As Integer, ByVal ownaddress As Integer, ByVal sclk As Integer, ByVal statuswait As Integer) As Integer
Public Declare Function sendaddress Lib "pica_95.dll" (ByVal slaveaddress As Integer, ByVal setnack As Integer) As Integer
Public Declare Function restart Lib "pica_95.dll" (ByVal slaveaddress As Integer, ByVal setnack As Integer) As Integer
Public Declare Function writebyte Lib "pica_95" (ByVal wrdata As Integer) As Integer
Public Declare Function readbyte Lib "pica_95.dll" (ByVal setnack As Integer) As Integer
Public Declare Function sendstop Lib "pica_95.dll" () As Integer
Public Declare Function getstatus Lib "pica_95.dll" () As Integer
Public Declare Function recover Lib "pica_95.dll" () As Integer
Public Declare Function slavelastbyte Lib "pica_95.dll" () As Integer
Public Declare Function dllversion Lib "pica_95.dll" () As Integer
```

2.7. Function Description

2.7.1. setup

Function specification: int setup(int baseaddress, int ownaddress, int sclk, int statuswait)

Parameters are:

int baseaddress
This has no function and any value may be passed to setup, The baseaddress has been retained to enable applications written for the ISA Bus ICA90 to be ported with a minimum of changes.

int ownaddress
This is the I2C address to which the adapter is to respond in slave mode. This forms the upper 7 bits of the 8 bit address, the lowest bit being the read(1) or write(0) bit. This means that if ownaddress = 57H the card will respond to a write address of AEH and a read address of AFH.

int sclk
This is the clock rate (bit rate for the I^2C serial bus) when operating as a master.

Value of sclk	Approximate SCL-KHz
0	90
1	45
2	11
3	1.5

int statuswait
This is a period of time (in micro seconds) to wait for the required bus status. If this time-out expires the I^2C functions will exit returning an error code.

Parameters returned: If the software fails to find the driver error code 9001H is returned otherwise the status is returned.

Prerequisites: None.

Functional description: This function characterises the PC and initialise adapter ready for I^2C transfers.

2.7.2. sendaddress

Function specification: Int sendaddress(int slaveaddress, int setnack)

Parameters are:

int slaveaddress
This is the address to be accessed via the I2C, e.g. A0H.

int setnack
This controls whether the Parallel I2C Communications Adapter transmits an Acknowledge down the I^2C Bus on reception of a byte. The last byte received during a transfer must not be acknowledged, in all other cases acknowledge must be enabled. If setnack = 0 then acknowledge is enabled, if setnack = 1 then acknowledge is disabled. Therefore, if a read (odd numbered) address is being sent AND only 1 Byte is to be read, setnack should be set to = 1; in all other cases it must be clear = 0.

Parameters returned: ***int ErrCode***.
If the software fails to open a handle to the driver 0x9000 is returned.
If the transfer time out occurs error code 8001H is returned otherwise the status is returned.

Prerequisites: The adapter must be configured for PC and application by running **setup**.

Functional description: The function waits for the bus to be free. Then sends the slave address with the appropriate acknowledge.

The acknowledge is set ready for the data transfer after the address and hence in read mode (odd address being sent) if only one byte is to be read the setnack parameter must equal 1. If more than one byte is to be read or if in write mode (even address being sent) then setnack must equal 0.
The function waits for the address to be sent. Should a time-out occur during the sending of an address then an error code 8001H is returned, otherwise the status is returned.

2.7.3. writebyte

Function specification: Int writebyte(int wrData)

Parameters are: ***int wrData***
This is the byte of data to be written.

Parameters returned: ***int ErrCode***.
If the software fails to open a handle to the driver 0x9000 is returned.
If the transfer time out occurs error code 8004H is returned otherwise the status is returned.

Prerequisites: Adapter must be configured using **setup**, start and write address sent by **sendaddress**.

Functional description: The function writes the data to the adapter and then waits for it to be sent. Should a time-out occur during the sending of the data then error code 8004H is returned, otherwise the status is returned.
Writebyte is compatible with both master write and slave write modes.

2.7.4. readbyte

Function specification: Int readbyte(int setnack)

Parameters are: ***int setnack***
This controls whether the adapter transmits an acknowledge down the I^2C bus on reception of a byte. The last byte received during a transfer must not be acknowledged, in all other cases acknowledge must be enabled. If setnack = 0 then acknowledge is enabled, if setnack = 1 then acknowledge is disabled. Therefore, if the LAST BUT ONE byte is to be read, setnack should be set to =1; in all other cases it is to be set = 0 (in the case of reading 1 byte only, the acknowledge will have been disabled by sendaddress and so should now be enabled again after reading the data, hence setnack = 0 for reading a single byte of data).
The first read from the adapter following a write to it will result in the data that was written being returned. This data MUST be read and discarded before real data can be read, DO NOT count this extra read when considering whether or not to acknowledge.

Parameters returned: ***int I2CData***
If the software fails to open a handle to the driver 0x9000 is returned.
The data read, if a time-out occurs the ErrCode 8005H is returned.

Prerequisites: Adapter must be configured using **setup**, start and read address sent by **sendaddress**.

Functional description: If setnack is 1 the function writes 40H to the control register to establish the correct acknowledge procedure.
The data is read from the adapter.
Readbyte is compatible with both master read and slave read modes.

2.7.5. sendstop

Function specification: Int sendstop( )

Parameters are: None.

Parameters returned: ***int ErrCode***.
If the software fails to open a handle to the driver 0x9000 is returned.
If the transfer time out occurs error code 8002H is returned otherwise the status is returned.

Prerequisites: Adapter must be configured using **setup.** Should normally only be used at the end of a transmission. Correct acknowledge sequence must have been applied if the transmission was a read.

Functional description: Instruct the adapter to send a stop code and wait for it to be sent.
Should a time-out occur during the sending of a stop then an error code 8002H is returned, otherwise the status is returned.

2.7.6. restart

Function specification: Int restart(int slaveaddress, int setnack)

Parameters are: ***int slaveaddress***
The address to be accessed via the I2C, e.g. A1H.
int setnack
This controls whether the Parallel I2C Communications Adapter transmits an Acknowledge down the I^2C Bus on reception of a byte. The last byte received during a transfer must not be acknowledged, in all other cases acknowledge must be enabled. If setnack = 0 then acknowledge is enabled, if setnack = 1 then acknowledge is disabled. Therefore, if a read (odd numbered) address is being sent AND only 1 Byte is to be read, setnack should be set to = 1; in all other cases it must be clear = 0.

Parameters returned: ***int ErrCode***
If the software fails to open a handle to the driver 0x9000 is returned.
If the transfer time out occurs error code 8003H is returned otherwise the status is returned.

Prerequisites: Adapter must be configured using **setup.** A start and slave address must have previously been sent using **sendaddress**.
Usually a data pointer would already have been written using **writebyte.**

Functional description: Sends a start code and the slave address specified and presets the acknowledge status depending on the value of setnack.
The acknowledge is set ready for the data transfer after the address and hence in read mode (odd address being sent) if only one byte is to be read the setnack parameter must equal 1. If more than one byte is to be read or if in write mode (even address being sent) then setnack must equal 0.
The function waits for the address to be sent. Should a time-out occur during the sending of an address then an error code 8003H is returned, otherwise the status is returned.

2.7.7. getstatus

Function specification: Int getstatus(void)

Parameters are: None.

Parameters returned: ***int I2Cstatus***

If the software fails to open a handle to the driver 0x9000 is returned otherwise the current value of the bus status.

Prerequisites — Adapter must be configured using **setup.**

Functional description — The function reads status word from the adapter and returns it.

2.7.8. recover

Function specification — Int recover(void)

Parameters are: — None.

Parameters returned — ***int ErrCode.***
If the software fails to open a handle to the driver 0x9000 is returned.
If the bus recovery failed error code 8006H is returned otherwise the status is returned.

Prerequisites — Adapter must be configured using **setup.**

Functional description — This function issues two consecutive stop commands on the bus, with a delay in between. It then clears the adapter registers and reads the status. This should normally set the adapter into a known idle state when a bus error or other problem has occurred.
If the status does not indicate bus free or the bus Error bit is still set then 8006H is returned otherwise the status is returned.

2.7.9. slavelastbyte

Function specification — void slavelastbyte( )

Parameters returned — If the software fails to open a handle to the driver 0x9000 is returned otherwise the function returns 0.

Prerequisites — Adapter must be configured using **setup.** This function would normally only be called following the end of a transmission in slave write mode - when the adapter is being read as a slave, by another master, *not when writing to a slave using the adapter.*

Functional description — This function is used when the adapter is a slave being read by a master elsewhere on the bus - the adapter is in slave write mode. The function must be called immediately after the master indicates the last byte has been read (by not acknowledging that byte). This function is required to clear the I^2C data lines so that the master can send a stop signal.

THE MOST COMMONLY ASKED I2C QUESTIONS

3.1. General Questions

Question **Will my adapter run I2C clock speeds greater than 90KHz?**
Answer At the moment your adapter is limited by the Bus Controller chip fitted, to a maximum of 90KHz as a master and 100KHz as a slave.

Question **Will my adapter work under Windows NT4* or Windows 95*?**
Answer The adapter will work under both these systems without problems. If you need to access the Windows NT and Windows 95 DLLs please contact our sales team for further information.
* All trade marks acknowledged

Question **I get corrupted transfers why is this?**
Answer The most likely reason for corrupted transfers is either incorrect bus termination or excessive capacitance - see the manual for details.

Question **Do you have software to talk to my.....?**
Answer Unfortunately there are too many I2C devices for us to be able to offer complete solutions - although we can supply a windows based application called WINI2C which is designed for those just starting I2C or wishing to perform simple I2C tasks, please contact our sales team or look on our web site, www.calibreuk.com for further information.

Question **I am trying to read from a device, the first time my software works fine but when I try again I can't get anything what's wrong?**
Answer Please check that you are changing the value of Setnack in accordance with the manual, it is likely that you have not made Setnack 1 for the last **AND** last but one bytes being read.

3.2. DOS Software Questions

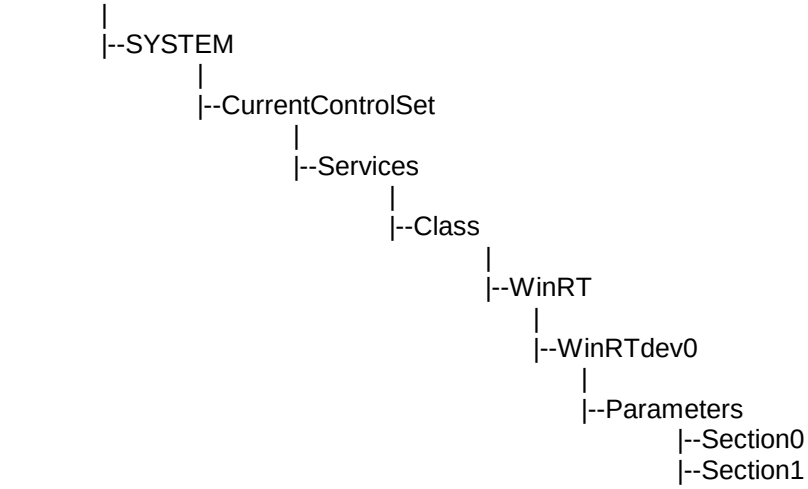
Question **My I2C adapter locks up with a constant status - why?**
Answer If you are using either the 'C' or Basic library functions supplied with the adapter on a fast PC it is possible that the PC is polling the status register too quickly, the simplest way to prevent this is to add a small delay prior to reading the status in the getstatus routine. Alternatively use the windows DLL supplied as these automatically allow for speed of the PC at run-time.

3.3. WINDOWS 95 and NT Questions

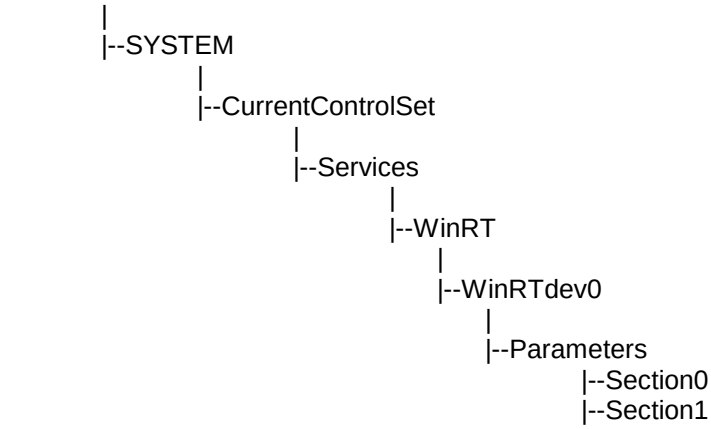
Question **My software cannot find the adapter. Your Windows software reports that it cannot configure the adapter. Why is this?**
Answer Have you registered the device driver as detailed in the software manual? If so check that the address links (see adapter manual for details) are correct for the location at which you registered the driver.

Question **I think I have registered the driver how can I find out if I have?**
Answer You need to inspect the registry as follows

Windows 95 START - Run regedit



Windows NT START - Run regedit



Question **I am using your Windows 95 / NT DLL and I am always getting a time out error code. Why?**

Answer Check the Syntax of the setup function, this problem is most usually caused by swapping the Status Wait and Sclk parameters.

The correct syntax is i2cstatus = setup (baseaddress, ownaddress, sclk, statuswait)

Question **I have read the manual and still cannot get the communications to run. What do I do next?**

Answer Check that you have fully implemented the protocol between the adapter and the other I2C devices see the device manufacturers data sheet for details.

Check that the software you have written is logically and syntactically correct - this is probably the most common cause of software faults we have to deal with.

Send us the following details:-

1)The link settings of the adapter.

2)A sketch of the relevant I2C hardware including the location of bus termination.

3)The type and speed of processor within your PC and which operating system, you are running.

4)Brief software listings, or which Calibre software you are running.

5)The serial number of your I2C adapter, or when you purchased it.

PLEASE EMAIL YOUR QUERY TO: techsupport@calibreuk.com

OR FAX YOUR QUERY TO: 44-1274-730960

We will endeavour to help you.