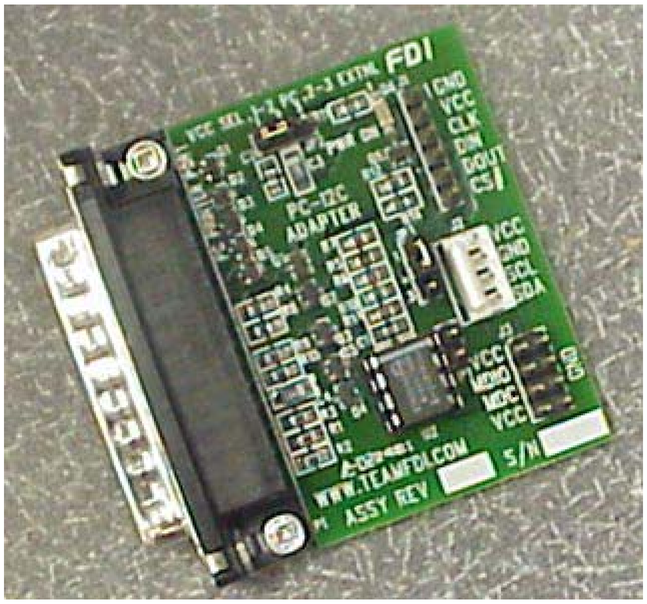
PC-I2C-KIT ©

with MDIO and SPI support

Quick Start Manual



FDI *Future Designs, Inc.*
Your Development Partner

P:\PC Products\PC-I2C\Docs\Manual\Pci2c Quick Start V3_E.doc, Revision 3.E, 3/11/2004 1:05 PM
Printed in the United States of America

Table of Contents

1. INTRODUCTION

The PC-I2C-KIT interfaces to the standard Parallel Printer Port on any IBM-compatible PC and allows bi-directional communications with I²C, MDIO, and SPI based peripherals. The software runs on Windows 9x, Windows NT4.0 (SP3 or greater), Windows 2000, or Windows XP (Home or Professional). The software allows the user to easily select a peripheral from the menu of supported devices. Users can add new devices through the use of Device Description Files (DDF). (See Appendix G for a further description of DDFs.) The kit comes complete with Software, PC-I2C board, Quick Start Manual and an 18” cable to allow direct connection to the user’s application or any of the standard I²C demo boards.

In addition, the PC-I2C Board provides a socket for direct connection of an on-board EEPROM to the I²C bus. This allows you to program I²C EEPROMs with only your computer, power supplied by a target, and the PC-I2C Adapter Board. A PCF8582 EEPROM is even supplied with PC-I2C-KIT.

2. PC-I2C-KIT CONTENTS

- PC-I2C Board with PCF8582 EEPROM
- 4-pin connecting cable (18” length)
- Software
- PC-I2C Quick Start Manual
- Registration Form

3. BEFORE YOU BEGIN

3.1 Reference Information

This manual assumes that the user has some experience with I²C, MDIO, SPI, and basic electronics. For more details on the operation of the I²C bus please refer to the appropriate Philips Semiconductors documentation. Philips Semiconductors provides technical information for the I²C bus. For more details on MDIO, refer to the IEEE web site (www.ieee.org).

Some excellent sources of information
I2C:

- 80C51 Based 8-Bit Microcontrollers Databook (Philips Document # IC20)
- I²C Peripherals Databook (Philips Document # IC12)

MDIO:

- IEEE 802.3 Working Group

SPI:

- http://www.mct.net/faq/spi.html

3.2 Requirements

- Windows 9x/NT/2000/XP
- 2MB Disk Space
- CD-ROM drive (used for installation only)
- Parallel Printer Port (LPT1 – LPT3 are supported)

3.3 Power Requirements

JP2 must be jumpered 2-3. This setting for jumper JP2 configures the PC-I2C Board for target power. The target board or adaptor must then provide power to the PC-I2C Board. The PC-I2C Board requires 1.2 – 5.5VDC at a maximum of 35 mA with typical current consumption at approximately 16 mA. This allows the PC-I2C to be used with other boards operating at 1.2 – 5.5 VDC.

Note: The PC-I2C will not work properly when other devices are connected to the parallel port. (Such as scanners, cameras, printers, or copy protection keys)

4. GENERAL OPERATION

4.1 Installing the PC-I2C Software

- Insert the CD into the CD-ROM drive
- Select PC-I2C to install.
- Follow the instructions provided by the software installation program.

4.2 Connecting the PC-I2C Board to your computer

- Ground yourself before handling the PC-I2C Board by touching the bare metal on the rear of your computer.
- It is not necessary to power-down the PC before connecting the PC-I2C Board.
- Remove the PC-I2C Board from the anti-static bag.
- Select an available Parallel Printer Port.

  Note 1: Verify that the connector is actually a Parallel Printer Port since there are some SCSI cards that utilize the same style connector.

  Note 2: It is ***strongly*** recommended that the PC-I2C Board be connected directly to your computer. Correct operation of the PC-I2C Board can not be guaranteed if it is connected through a parallel printer cable or switch box.
- Connect the PC-I2C Board to the Parallel Printer Port connector (Female DB25).

Warning: Improperly attaching the PC-I2C Board to connectors other than a Parallel Printer Port may cause damage. (A SCSI port can seriously damage the PC-I2C board.)

4.3 Setting the Bus Speed

The PC-I2C software allows you to change the Bus Speed to suit your application. This allows communications with slow peripheral devices, such as an analog to digital converter or a microcontroller without a hardware interface, and high-speed devices operating at frequencies up to 100 KHz.

The Bus Speed is set on the Main Program Window. This allows adjustment of this parameter over the supported range.

4.4 I^{2}C Operation

4.4.1 On-Board EEPROM (U2)

The on-board EEPROM socket U2 provides support for the included PCF8582 as well as most I^{2}C EEPROMs in an 8-pin DIP package. The PC-I2C Software allows programming of most I^{2}C EEPROMs at the default address.

Note: The default I^{2}C address for this on-board EEPROM device is 0xA8.

If an external I^{2}C device using this address (0xA8) is connected to the PC-I2C Board, the EEPROM in socket U2 should be removed to avoid address conflicts.

4.4.2 Connecting the PC-I2C Board to your peripheral

Included in the PC-I2C-KIT is an 18” four-conductor cable for connecting the PC-I2C Board to another I^{2}C Board. Extreme care should be taken to verify the proper power & ground connections. The PC-I2C Board or your I^{2}C Board may be damaged by improper connection of this cable. The PC-I2C Board provides a connector (J2) for use with external I^{2}C peripherals. This connector supports the common pinout used on other I^{2}C development and demonstration boards. This allows direct connection to the PC-I2C Board with the included one-to-one cable.

4.4.3 Configuring the PC-I2C Board

For normal operation, no configuration changes should be required on the PC-I2C Board.

The jumper on JP2 must be in the 2-3 position. This setting configures the PC-I2C Board for target power supply. The target board must then provide power to the PC-I2C Board via the connection. The PC-I2C Board requires 1.2 – 5.5VDC at a maximum of 35 mA with typical current consumption at approximately 16 mA. This allows the PC-I2C to be used with other boards operating at 1.2 VDC – 5.5 VDC.

4.5 MDIO Operation

4.5.1 MDIO Support

PC-I2C supports and has been tested for IEEE 802.3 serial communications using clause 22 and clause 45. Information regarding IEEE standards can be found at www.ieee.org. DDF and SDF files, outlined in Appendix G and Appendix H respectively, can be used for MDIO operations. If you purchased a PC-I2C-DEV, there are also functions available for programs using MDIO.

4.5.2 MDIO Setup of the PC-I2C Board

For normal MDIO operation, follow the General Operations in section 4 keeping in mind that the SCL line is equivalent to the MDC line and the SDA line is equivalent to the MDIO line. No configuration changes should be required on the PC-I2C Board. The PC-I2C Board also includes one connector (J3) for MDIO. This connector supports the common pinout used for communication with other MDIO boards.

The jumper on JP2 ***must*** be in the 2-3 position. This setting configures the PC-I2C Board for target power supply. The target board must then provide power to the PC-I2C Board via the connection. The PC-I2C Board requires 1.2 – 5.5VDC at a maximum of 35 mA with typical current consumption at approximately 16 mA. This allows the PC-I2C Board to be used with MDIO target boards operating at 1.2 VDC – 5.5 VDC. Since the MDIO spec is less than 5VDC, setting JP2 to PC power is not an option because it may damage the MDIO device. This is due to the PC Parallel Port's power output voltage of 5VDC. The jumper on JP2 ***must*** be in the 2-3 position.

Note: If JP2 is set to anything other than target power (2-3) in MDIO operation serious damage to the target board can occur.

4.5.3 On board I2C EEPROM

The included I2C PCF8582C EEPROM will not respond unless an I2C Start Condition is generated. Due to this the EEPROM should not interfere with normal MDIO operation because an MDIO Preamble is High where as an I2C Start Condition is Low. However, if a device does respond in such a way as to generate a start condition, the EEPROM must be removed from the board.

Note: Keep this EEPROM in case a return or repair is necessary later.

4.6 SPI Operation

4.6.1 SPI Support

PC-I2C supports general SPI serial communications. Information regarding SPI can be found in each device's datasheet. DDF and SDF files, outlined in Appendix G and Appendix H respectively, can be used for SPI operations. If you purchased a PC-I2C-DEV, there are also functions available for programs using SPI.

4.6.2 SPI Setup of the PC-I2C Board

For normal SPI operation, follow the General Operations in section 4. There are a few changes to keep in mind:

- J1 should be used for SPI operation.
- When connecting your device, the CS# line on J1 should be connected to the Chip Select (CS) line or the Chip Enable (CE) line on your target device.
- The DIN line on J1 should be connected to the Data Out (DO) line on the target device.
- The DOUT line on J1 should be connected to the Data In (DI) line on the target device.
- JP3 must be configured in the 1-2 position for SPI operation.

The jumper on JP2 *must* be in the 2-3 position. This setting configures the PC-I2C Board for target power supply. The target board must then provide power to the PC-I2C Board via the connection. The PC-I2C Board requires 1.2 – 5.5VDC at a maximum of 35 mA with typical current consumption at approximately 16 mA. This allows the PC-I2C Board to be used with SPI target boards operating at 1.2 VDC – 5.5 VDC.

Note: If JP2 is set to anything other than target power (2-3) in SPI operation damage to the target board may occur.

4.6.3 On board I2C EEPROM

The included I2C PCF8582C EEPROM will not respond unless an I2C Start Condition is generated. Due to this the EEPROM should not interfere with normal SPI operation because SPI never requires a change in the data line during a high clock. However, if a device does respond in such a way as to generate a start condition, the EEPROM must be removed from the board.

Note: Keep this EEPROM in case a return or repair is necessary later.

5. APPENDICES

5.1 Appendix A: Jumper Settings

JP1 – This jumper is for use with some I²C EEPROMs with a hardware "Write Protect" function, which requires that pin 7 of the EEPROM be connected to Ground. JP1 is not normally loaded since most I²C EEPROMs will not function correctly if pin 7 is connected to Ground. Load JP1 to use other I²C EEPROMs which support the "Write Protect" function on pin 7.

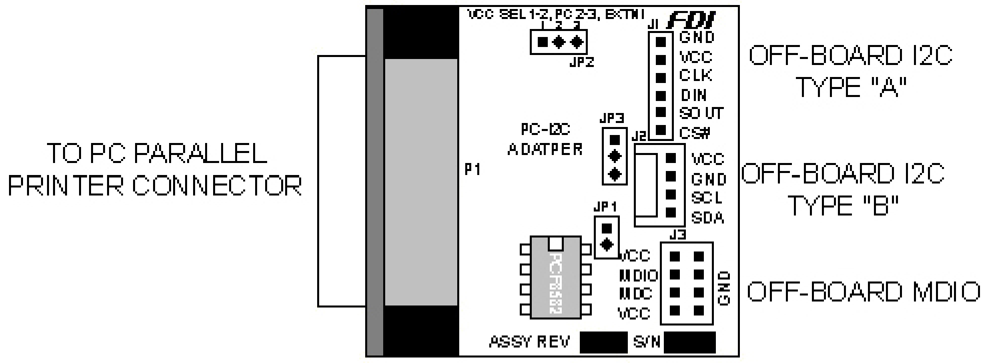
JP2 – This jumper is utilized to select the source of power for the PC-I2C Board. JP2 must be jumpered 2-3 which allows externally supplied power. The PC-I2C Board requires 1.2 – 5.5VDC at a maximum of 35 mA with typical current consumption at approximately 16 mA. This allows the PC-I2C to be used with other boards operating at 1.2 – 5.5 VDC.

Warnings:

1. If JP2 is jumpered 1-2, do NOT connect the P2-I2C Board to any other board.
2. If JP2 is jumpered 2-3, then the PC-I2C Board requires that power be supplied from a target source. When a target power supply or I²C peripheral board is used extreme care should be taken to verify the power connections before powering up the I^2C 'system'.

JP3 – This jumper must be set to 2-3 for I2C or MDIO operation. It should be set to 1-2 for SPI Operation.

5.2 Appendix B: Connector Functions and Locations



P1 – This is the connection to the PC Parallel Printer Port (LPTn)

J1 – This connector may be used in any of the supported modes of operation. When using the included cable, only I2C & MDIO modes are supported since the cable is only 4 pins. Note that for I2C/MDIO applications, J1 has reversed VCC/GND connections from J2, so extreme care should be taken when using this connector. For SPI applications, this connector MUST be used and the user must provide a custom cable for his or her target board.

J2 – This connector ONLY supports I2C & MDIO modes of operation. The included cable may be utilized, but extreme care should be taken to determine the proper power and ground connection. The VCC & GND pins are reversed from J1. Verify the power and ground connections before powering up the PC-I2C board.

J3 – This connector is utilized for certain specific MDIO applications.

5.3 Appendix C: Software Updates

PC-I2C Software updates are available directly from the FDI web site at www.teamfdi.com.

5.4 Appendix D: Technical Support

Philips Semiconductors provides technical information for the I²C bus. Technical support for most I²C peripherals and microcontrollers can be found at their corresponding vendors.

IEEE provides technical information for MDIO Clause 22 and MDIO Clause 45. Technical Support for most MDIO peripherals can be found at their corresponding vendors.

SPI support is usually found on the individual devices' vendors' websites or in the individual devices' data sheets. SPI information can be found on the web, as well. Some very helpful information can be found at:
    http://www.mct.net/faq/spi.html

Technical Support for the PC-I2C-KIT is available directly from FDI via email at support@teamfdi.com. You may also fax your technical support questions to (256) 883-1241 or call us directly at (256) 883-1240. Support questions that are emailed will usually receive faster response times than fax or phone requests.

5.5 Appendix E: Disclaimer

ESD may damage electronic components. Be sure to touch the surface of a grounded device (such as the bare metal on the rear of your computer) before handling the PC-I2C Board.

FDI is not responsible for any damage that may be caused by misuse or improper installation of the PC-I2C Board.

5.6 Appendix F: Verification Procedure

Before Beginning, make sure that:

- The jumper is set to pins 2 and 3 on JP2.
- The jumper is set to pins 2 and 3 on JP3.
- Power is supplied to one of the 4 Vcc pins on PC-I2C.
- Ground is supplied to one of the 6 GND pins on PC-I2C.
- EEPROM (Part # PCF8582C-2 or equivalent) is correctly positioned in PC-I2C board (U2).
- Nothing else is attached to the PC parallel port
- No software drivers are installed that utilize the PC Parallel Port (such as camera interfaces, scanner software, or copy protection keys)

1. Connect PC-I2C board to parallel port on PC.
2. Connect the connector cable to J1 on the PC-I2C board and the target I2C connector.
3. Start up PC-I2C software and Select PCF8582C in the Device Selection Box. Click Open.
4. Type A8 in the Slave Address field.
5. Go to "Fill Buffer" and type 55 in "Pattern". Click OK. Windows should all read "55".
6. Go to "Write" and click OK. Windows should still read "55".
7. Go to "Invert" and click OK. Windows should read "AA".
8. Go to "Read" and click OK. Windows should read "55".
9. Go to "Erase" and click OK. Windows should still read "55".
10. Go to "Read" and click OK. Windows should read "FF".

If the PC-I2C completed all steps correctly, then the PC-I2C software and board are working properly.

Note: The PC-I2C will not work properly when other devices are connected to the parallel port. (Such as scanners, cameras, printers, or copy protection keys)

5.7 Appendix G: Device Descriptor Files Specification

5.7.1 Device Descriptor File Overview

The Device Descriptor File (DDF) is an ASCII text file that completely describes the registers and memory locations present in a particular device. It also specifies the protocol that is used to communicate with the device (e.g. I2C, MDIO, or SPI). All of the DDF files are located in the Devices subdirectory. When the PC-I2C software is started, all of the DDF files are read and the information is used to generate the supported Bus Protocol and Device lists in the PC-I2C software. In order to add support for new devices to the PC-I2C software, a new DDF file must be created and placed in the Devices subdirectory. This new device will show up in the supported device list the next time the PC-I2C software is started.

The DDF file consists of identifier tokens followed by their corresponding values. The identifier tokens are grouped into one of three different descriptor blocks. These blocks are the Device Block, the Register Block, and the Memory Block. There must be a single Device Block as the first descriptor block in the file. The Device block is then followed by one or more Register Blocks and possibly a single memory Block. The format of these blocks can vary depending on the type of bus protocol used to communicate with the device. These differences are notes in the sections below. Note that all tokens are enclosed in square brackets [] and the values can be a HEX value (preceded by 0x), a decimal value (with a leading 0), or an ASCII string enclosed by double quotes.

Comments can be added to the DDF file by starting a line with two forward slashes (//). The DDF file parser will ignore any lines that start with the comment characters. A comment is required on the first line in order to ensure that a text editor was used.

5.7.2 [DEVICE]

The DEVICE Block must be the first descriptor block encountered in the DDF file. It is used to define the basic characteristics of the device. These characteristics include the Device Name, the Device Description, the Device Type, and the valid Base Addresses of the device. There can be only one DEVICE Block in the DDF file.

[NAME]

The [NAME] token is used to define the name of the device. The ASCII string following this token will be displayed in the Device list of the PC-I2C software. This string can be up to 64 characters long and can include spaces and other special characters.

[DESC]

The [DESC] token is used to define a description of the device. The ASCII string following this token will be displayed below the Device list of the PC-I2C software when this device is selected. This string can be up to 64 characters long and can include spaces and other special characters.

[TYPE]
The [TYPE] token is used to define the bus interface protocol for this device. The ASCII string following this token will determine the bus protocol. The currently supported values are:

- I2C – This token defines a standard I2C device
- MDIO_22 – This token defines an MDIO device that adheres to IEEE 802.3 Clause 22 (old MDIO)
- MDIO_45 – This token defines an MDIO device that adheres to IEEE 802.3ae Clause 45 (10G MDIO)
- SPI – This token defines a SPI device

[CS POLARITY]
The [CS POLARITY] token is used to define the value held at CS when a command is being sent for the SPI protocol. This value is ignored for all other protocols. The currently supported values are:

- High, One, 1 – This token defines a logic value of 1.
- Low, Zero, 0 – This token defines a logic value of 0.
- None, N/A – This token defines the CS line as unnecessary.

[ADDR]
The [ADDR] token is used to define the address(es) associated with this device. Its meaning changes depending on the value of the TYPE token as follows:

- I2C – The values listed after the token are the valid Slave addresses for this device. If more than one Slave address is defined, the values are separated by commas.
- MDIO_22 – The value listed after the token is the 5 bit address of the PHY to be accessed.
- MDIO_45 – This address is the two 5 bit addresses used to address the MMD.
- SPI – Not applicable.

An Example Device Block
The following example is for a Philips Semiconductor PCF8582C EEPROM device. It is an I2C device with 8 possible Slave addresses.

```
[DEVICE]
  [NAME]      “PCF8582C”
  [DESC]      “Philips PCF8582C 256 x 8 bit EEPROM”
  [TYPE]      “I2C”
  [ADDR]      0xa0,0xa2,0xa4,0xa6,0xa8,0xaa,0xac,0xae
```

5.7.3 [REGISTER]

The REGISTER block is used to define the characteristics of a single register within a device. These characteristics include the Register Name, the Register Address, the Register Size, and the Register Type. There can be as many

REGISTER Blocks as needed, in the DDF file, to fully describe every register in the device.

[REG NAME]
The [REG NAME] token is used to define the name of the register. The ASCII string following this token will be displayed in the register list of the PC-I2C Register Dialog. This string can be up to 64 characters long.

[REG ADDR]
The [REG ADDR] token is used to define the address of the register. The value following this token will be displayed along with the register name in the register list of the PC-I2C Register Dialog. This is the value that will be used as the address for reading and writing to this particular register. If the register has no address (i.e. there is only one register that can be read), then N/A should be entered for this value. (See the PCF8574A.DDF for an example.)

[REG SIZE]
The [REG SIZE] token is used to define the data size of the register. The value following this token is the data size in bytes for this register. A value of 0x01 would correspond to an 8-bit register; a value of 0x02 would correspond to a 16-bit register; etc.

[REG TYPE]
The [REG TYPE] token is used to define the access type of the register. The ASCII string following this token will determine the access type. The currently supported values are:

- **RW** – The register can be both read and written. Both the ‘R’ and ‘W’ buttons for this register will be enabled.
- **RO** – The register can only be read. The ‘R’ button will be enabled and the ‘W’ button will be disabled for this register.
- **WO** – The register can only be written. The ‘W’ button will be enabled and the ‘R’ button will be disabled for this register.
- **RSV** – The register is a “Reserved” register. Both the ‘R’ and ‘W’ buttons for this register will be enabled. Also, if the “Display Reserved Registers” option is not selected on the PC-I2C main window, all registers with a type of RSV will be hidden.

An Example Register Block
The following example is for the Control/Status register of the Philips Semiconductor PCF8583 Clock/Calendar chip. The register is 8 bits wide and is a read/write register.

```
[REGISTER]
 [REG NAME]     “Control/Status”
 [REG ADDR]     0x00
 [REG SIZE]     0x01
 [REG TYPE]     “RW”
```

5.7.4 [MEMORY]

The MEMORY block is used to define the characteristics of a memory array within a device. These characteristics include the Memory Block Name, the Memory Base Address, the Memory Size, the Memory Type, the Page Write Size, and the Write Delay time. There can be only one MEMORY Block per DDF file.

[MEM NAME]

The [MEM NAME] token is used to define the name of the memory block. The ASCII string following this token will be displayed in the tab of the PC-I2C Memory Dialog. This string can be up to 64 characters long and can include spaces and other special characters.

[MEM ADDR]

The [MEM ADDR] token is used to define the base address of the memory block. The value following this token will be used as the start of the memory block and will be the first address displayed when the Memory Dialog is opened.

[MEM SIZE]

The [MEM SIZE] token is used to define the size (in bytes) of the memory block. The value following this token will be used to determine the amount of memory that can be displayed and modified.

[MEM WIDTH]

The [MEM WIDTH] token is used to define the width (in bytes) of the memory word. The value following this token will be used to determine the method to be used when reading or writing to memory. If this value is more than 1, then the entire memory array will be used for all read, write, and erase operations. Memory is only displayed in bytes.

[MEM TYPE]

The [MEM TYPE] token is used to define the access type of the memory block. The ASCII string following this token will determine the access type. The currently supported values are:

- **RW** – The memory block can be both read and written.
- **RO** – The memory block can only be read.

[MEM PAGE]

The [MEM PAGE] token is used to define the page write size for the memory block. The value following this token determines the number of bytes that can be written to memory block in a single instruction. This value is only relevant for EEPROM/FLASH type devices that utilize the page write mode. For other devices, this value can be set to the memory block size.

[MEM WDLY]

The [MEM WDLY] token is used to define the write delay for the memory block. The value following this token determines the amount of time (in ms) to delay after a write instruction, before the next access occurs. This prevents the overlap of operations due to excessive write times for some devices.

An Example Memory Block

The following example is for a Philips Semiconductor PCF8583 Clock/Calendar chip with 240 bytes of SRAM (running from address 0x10 to 0xff). The page write is set to 8 bytes and the write delay is set to 1ms.

```
[MEMORY]
 [MEM NAME]       "Memory"
 [MEM ADDR]       0x0010
 [MEM SIZE]       0x00f0
 [MEM WIDTH]      01
 [MEM TYPE]       "RW"
 [MEM PAGE]       0x08
 [MEM WDLY]       0x01
```

5.7.5 SPI Specific [MEMORY] Tokens

These [MEMORY] Tokens are used in the DDF file for SPI only. SPI support is limited to memory devices at this time.

[MEM POSTWCLK]

The [MEM POSTWCLK] token is used to define the number of clocks to wait for a response after completing a write or erase command. The CS line is toggled and when the DIN line matches the value specified by [MEM POSTWPOL], PC-I2C will toggle the CS line and move on to the next command.

[MEM POSTWPOL]

The [MEM POSTWPOL] token is used to define the value on the DIN line that signifies that the target device is finished writing or erasing.

[MEM ENWIDTH]

The [MEM ENWIDTH] token is used to define the size (in bits) of the target device’s Enable Command. The value following this token will be used to determine the width of the command. This token also determines the number of entries allowed in the [MEM ENABLE] token.

[MEM ENABLE]

The [MEM ENABLE] token is used to define the command that allows writing to and erasing from the device. The ASCII string following this token will determine the command. The currently supported values for each bit are:

- **High, One, 1** – The current bit is a logic 1.
- **Low, Zero, 0** – The current bit is a logic 0.

[MEM DISWIDTH]

The [MEM DISWIDTH] token is used to define the size (in bits) of the target device’s Disable Command. The value following this token will be used to determine the width of the command. This token also determines the number of entries allowed in the [MEM DISABLE] token.

[MEM DISABLE]
The [MEM DISABLE] token is used to define the command that disallows writing to and erasing from the device. The ASCII string following this token will determine the command. The currently supported values for each bit are:

- **High, One, 1** – The current bit is a logic 1.
- **Low, Zero, 0** – The current bit is a logic 0.

[MEM WWIDTH]
The [MEM WWIDTH] token is used to define the size (in bits) of the target device's Write Command. The value following this token will be used to determine the width of the command. This token also determines the number of entries allowed in the [MEM WCOM] token.

[MEM WCOM]
The [MEM WCOM] token is used to define the command that writes to the device. The ASCII string following this token will determine the command. The currently supported values for each bit are:

- **High, One, 1** – The current bit is a logic 1.
- **Low, Zero, 0** – The current bit is a logic 0.
- **A#** – The current bit is address bit # (starting with address bit 0).
- **D#** – The current bit is data bit # (starting with data bit 0).

[MEM RWIDTH]
The [MEM RWIDTH] token is used to define the size (in bits) of the target device's Read Command. The value following this token will be used to determine the width of the command. This token also determines the number of entries allowed in the [MEM RCOM] token.

[MEM RCOM]
The [MEM RCOM] token is used to define the command that reads from the device. The ASCII string following this token will determine the command. The currently supported values for each bit are:

- **High, One, 1** – The current bit is a logic 1.
- **Low, Zero, 0** – The current bit is a logic 0.
- **A#** – The current bit is address bit # (starting with address bit 0).
- **D#** – The current bit is data bit # (starting with data bit 0).

[MEM EWIDTH]
The [MEM EWIDTH] token is used to define the size (in bits) of the target device's Erase Command. The value following this token will be used to determine the width of the command. This token also determines the number of entries allowed in the [MEM ECOM] token.

[MEM ECOM]

The [MEM ECOM] token is used to define the command that erases from the device. The ASCII string following this token will determine the command. The currently supported values for each bit are:

- **High, One, 1** – The current bit is a logic 1.
- **Low, Zero, 0** – The current bit is a logic 0.
- **A#** – The current bit is address bit # (starting with address bit 0).

[MEM ERALWIDTH]

The [MEM ERALWIDTH] token is used to define the size (in bits) of the target device's Erase All Command. The value following this token will be used to determine the width of the command. This token also determines the number of entries allowed in the [MEM ERAL] token.

[MEM ERAL]

The [MEM ERAL] token is used to define the command that erases all of the device. The ASCII string following this token will determine the command. The currently supported values for each bit are:

- **High, One, 1** – The current bit is a logic 1.
- **Low, Zero, 0** – The current bit is a logic 0.

An Example SPI Memory Block

The following example is for a Atmel AT93C46A EEPROM chip with 128 bytes (running from address 0x00 to 0x7f). The page write is set to 8 bytes and the write delay is set to 1ms.

```
[MEMORY]
 [MEM NAME]          "SPI Memory"
 [MEM ADDR]          0x00
 [MEM SIZE]          0x40
 [MEM WIDTH]         02
 [MEM TYPE]          "RW"
 [MEM PAGE]          0x08
 [MEM WDLY]          01
 [MEM POSTWCLK]      01000
 [MEM POSTWPOL]      High
 [MEM ENWIDTH]       010
 [MEM ENABLE]        0, 1, 0, 0, 1, 1, 0, 0, 0, 0
 [MEM DISWIDTH]      010
 [MEM DISABLE]       0, 1, 0, 0, 0, 0, 0, 0, 0, 0
 [MEM WWIDTH]        026
 [MEM WCOM]          0, 1, 0, 1, A5, A4, A3, A2, A1, A0, D15, D14, D13,
                     D12, D11, D10, D9, D8, D7, D6, D5, D4, D3, D2, D1,
                     D0
 [MEM RWIDTH]        026
 [MEM RCOM]             0, 1, 1, 0, A5, A4, A3, A2, A1, A0, D15, D14, D13,
                        D12, D11, D10, D9, D8, D7, D6, D5, D4, D3, D2, D1,
                        D0
 [MEM EWIDTH]           010
 [MEM ECOM]             0, 1, 1, 1, A5, A4, A3, A2, A1, A0
 [MEM ERALWIDTH]        010
 [MEM ERAL]             0, 1, 0, 0, 1, 0, 0, 0, 0, 0
```

5.7.6 Example Device Descriptor File

The following example is the entire DDF file for the Philips Semiconductor PCF8583 Clock/Calendar Chip with 240 bytes of SRAM. It demonstrates all three types of Descriptor Blocks.

```
// PCF8583.ddf
//
//  This is the device definition file (DDF) for the Philips
// Semiconductor PCF8583 I2C Clock Calendar Chip with 240 x 8 RAM.
//
[DEVICE]
 [NAME]       "PCF8583"
 [DESC]       "Philips PCF8583 Clock Calendar with 240 x 8 RAM"
 [TYPE]       "I2C"
 [ADDR]       0xA2

[REGISTER]
 [REG NAME]        "Control/Status"
 [REG ADDR]        0x00
 [REG SIZE]              1
 [REG TYPE]        "RW"

[REGISTER]
 [REG NAME]        "Hundredths"
 [REG ADDR]        0x01
 [REG SIZE]              1
 [REG TYPE]        "RW"

[REGISTER]
 [REG NAME]        "Seconds"
 [REG ADDR]        0x02
 [REG SIZE]              1
 [REG TYPE]        "RW"

[REGISTER]
 [REG NAME]        "Minutes"
 [REG ADDR]        0x03
 [REG SIZE]              1
 [REG TYPE]        "RW"
```

```
[REGISTER]
 [REG NAME]        "Hours"
 [REG ADDR]        0x04
 [REG SIZE]              1
 [REG TYPE]        "RW"

[REGISTER]
 [REG NAME]        "Year/Date"
 [REG ADDR]        0x05
 [REG SIZE]              1
 [REG TYPE]        "RW"

[REGISTER]
 [REG NAME]        "Weekday/Month"
 [REG ADDR]        0x06
 [REG SIZE]              1
 [REG TYPE]        "RW"

[REGISTER]
 [REG NAME]        "Timer"
 [REG ADDR]        0x07
 [REG SIZE]              1
 [REG TYPE]        "RW"

[REGISTER]
 [REG NAME]        "Alarm Control"
 [REG ADDR]        0x08
 [REG SIZE]              1
 [REG TYPE]        "RW"

[REGISTER]
 [REG NAME]        "Alarm Hundredths"
 [REG ADDR]        0x09
 [REG SIZE]              1
 [REG TYPE]        "RW"

[REGISTER]
 [REG NAME]        "Alarm Seconds"
 [REG ADDR]        0x0a
 [REG SIZE]              1
 [REG TYPE]        "RW"

[REGISTER]
 [REG NAME]        "Alarm Minutes"
 [REG ADDR]        0x0b
 [REG SIZE]              1
 [REG TYPE]        "RW"
```

```
[REGISTER]
 [REG NAME]        "Alarm Hours"
 [REG ADDR]        0x0c
 [REG SIZE]                1
 [REG TYPE]        "RW"

[REGISTER]
 [REG NAME]        "Alarm Date"
 [REG ADDR]        0x0d
 [REG SIZE]                1
 [REG TYPE]        "RW"

[REGISTER]
 [REG NAME]        "Alarm Month"
 [REG ADDR]        0x0e
 [REG SIZE]                1
 [REG TYPE]        "RW"

[REGISTER]
 [REG NAME]        "Alarm Timer"
 [REG ADDR]        0x0f
 [REG SIZE]                1
 [REG TYPE]        "RW"

[MEMORY]
 [MEM NAME]        "Memory"
 [MEM ADDR]        0x0010
 [MEM SIZE]        0x00f0
 [MEM TYPE]        "RW"
 [MEM PAGE]        0x08
 [MEM WDLY]        0x01
```

5.8 Appendix H: Sequence Descriptor Files Specification

5.8.1 Sequence Descriptor File Overview

The Sequence Description File (SDF) is an ASCII text file that describes the sequence of instructions that the user intends to send to the I2C, MDIO, or SPI bus in a high-speed environment. All of the SDF files are located in the Sequence subdirectory. When the PC-I2C software is started, all of the SDF files are read and the information is used to generate the supported Sequence list in the PC-I2C software. In order to add a sequence to the PC-I2C software, a new SDF file must be created and placed in the Sequence subdirectory. This new device will show up in the supported device list the next time the PC-I2C software is started.

The SDF file consists of instruction tokens followed by their corresponding values. Four description instructions are required and the file parser will ignore any sequence files without one of each. Note that all tokens are followed by at least one space or tab and the values can be a HEX value (preceded by 0x), a decimal value (with a leading 0), or an ASCII string enclosed by double quotes. Instruction lines must be limited to 255 characters total.

Comments can be added to the SDF file by starting a line with two forward slashes (//). The SDF file parser will ignore any lines that start with the comment characters. A comment is required on the first line in order to ensure that a text editor was used.

5.8.2 Sequence Descriptor File Tokens

The following section contains all of the valid tokens used in a Sequence Descriptor File. A short description follows an example of each.

Note: While specific positions of the NAME, DESC, USE, and ADDR tokens are not required, it is suggested that they occur as the first four instructions in order to minimize confusion.

NAME Token

```
NAME        "Any ASCII string representing the name of the Sequence "
```

The NAME token is used to set the name that will be displayed within the Sequence list in PC-I2C. The only character that cannot be used within the quotation marks is another quotation mark. This is a required token.

DESC Token

```
DESC        "Any ASCII string representing the description of the Sequence "
```

The DESC token is used to set the description that will be displayed when this Sequence's name is chosen in the Sequence list in PC-I2C. The only character that cannot be used within the quotation marks is another quotation mark. This is a required token.

USE Token

USE “The DDF file name being used for this Sequence followed by .ddf ”

The USE token is used to set the name of the DDF file that will be used when running this Sequence in PC-I2C. The name of the file is case sensitive. The file extension (“.ddf”) is required. The characters that cannot be used within the quotation marks are another quotation mark and any character which cannot be used in file naming. This is a required token.

ADDR Token

```
ADDR 0xFF 0x1F
ADDR 0255
```

The ADDR token is used to set the address of the specific device that will be used when running this Sequence in PC-I2C. This token requires input of one address for I2C and MDIO 22, and the port address followed by the device address for MDIO 45. This is a required token.

READ Token

```
READ MEM[0xFFFF]        DISPLAY(“Heading=”)
READ REG[065535]        LOG(“Heading=”)
```

The READ token reads the specified memory(MEM) or register(REG) and displays(DISPLAY) the value or logs(LOG) it to a “.log” file with the same name as the Sequence Descriptor File and a “-##” added to the name. This was chosen in order to preserve previously logged files. The register is specified by the register number in the .ddf file while the memory is specified by the address which is to be accessed. The quotation marks can be filled with whatever the user would like to appear in front of the output value. This will always be preceded by “Line #: ” where # is the line number. The base, hex or decimal, used for specifying the register or memory specifies the base of the output. Again, the total output size, including line number, cannot exceed 255 characters. With SPI, the MEM token specifies byte addressing and the REG token specifies word addressing. This is only applicable when the Memory Width is greater than 1.

WRITE Token

```
WRITE       MEM[0xFFFF]       0256
WRITE       REG[065535]       0xFF
```

The WRITE token writes the specified data, represented in hex or decimal, to the specified memory(MEM) or register(REG). The register is specified by the register number in the .ddf file while the memory is specified by the address which is to be accessed. This particular action is followed by an automatic delay of 100 milliseconds.

PAUSE

```
PAUSE    00
PAUSE    016777216
PAUSE    0x1000000
```

The PAUSE token signifies that the sequencer should pause execution for at least the specified length of time in milliseconds. Note: the specified amount of time is a minimum amount, not a definite amount. Indefinite or user ended pauses are signified by a value of zero (00). The value entered after the PAUSE token must not exceed 0x1000000.

START_LOOP Token

```
START_LOOP
START_LOOP    032767
START_LOOP    0x7FFF
```

The START_LOOP token signifies that the next line is the beginning of a loop which will run a number of times equal to the value following the START_LOOP token. Infinite loops are signified by the absence of a value, but a warning will pop up on loading PC-I2C. If both tokens, START_LOOP and END_LOOP, specify a value for the number of loops, the value at START_LOOP takes precedence. The value entered after the START_LOOP token must not exceed 0x7FFF.

END_LOOP Token

```
END_LOOP
END_LOOP  032767
END_LOOP  0x7FFF
```

The END_LOOP token signifies the end of a loop which will run a number of times equal to the value following the END_LOOP token. Infinite loops are signified by the absence of a value, but a warning will pop up on loading PC-I2C. On an infinite loop, a dialog will pop up every 10,000 loops asking whether or not to continue looping. If “No” is chosen, the sequencer will continue past the END_LOOP token. If both tokens, START_LOOP and END_LOOP, specify a value for the number of loops, the value at START_LOOP takes precedence. The value entered after the END_LOOP token must not exceed 0x7FFF.

5.8.3 Example Sequence Descriptor File

The following example is the entire file “Sequencer Demo.sdf” which uses the PCF8582C included with PC-I2C-KIT to demonstrate all the commands that can be used in the sequencer.

```
// This is a test file for testing the sequencer

//This instruction specifies the label to be displayed in the "Sequence:" combo
        box in PC-I2C
NAME "Demonstration"

//This instruction specifies the description to be displayed when this sequence is
        selected in the "Sequence:" combo box
DESC "This is a test file for demonstrating the sequencer."

//This instruction specifies the DDF (Device Descriptor File) to be used
USE "PCF8582C.ddf"

//This instruction specifies the address of the device to be used
ADDR 0xA8

//This instruction specifies a read from memory address 0x01 to be displayed to
        the screen with the message "Memory at address 0x01 is"
READ MEM[0x01] DISPLAY("Memory at address 0x01 is")

//This instruction specifies a write to memory address 0x05 of a value of 0x01
WRITE MEM[0x05] 0x01

//This instruction specifies a pause for at least 598 milliseconds.
PAUSE 0598

//This instruction specifies a read from memory address 0x05 to be logged to the
        log file with the message "But the memory located at 0x05 is"
READ MEM[0x05] LOG("But the memory located at 0x05 is")

//This instruction specifies the start of a loop with 25 iterations
START_LOOP 25

//This instruction specifies a read from memory address 0x01 to be logged to the
        log file with the message "Can the Memory at 0x01 be represented in
        decimal like this?=>"
READ MEM[01] LOG("Can the Memory at 0x01 be represented in decimal like
        this?=>")

//This instruction specifies the end of a loop
END_LOOP

//This instruction specifies the start of a loop
START_LOOP
```

```
//This instruction specifies a read from memory address 0x05 to be logged to the
        log file with the message "Type anything you want here. Anything at all. As
        long as you don't exceed a total instruction length of 256 characters."
READ MEM[0x05] LOG("Type anything you want here. Anything at all. As long
        as you don't exceed a total instruction length of 256 characters.")

//This instruction specifies the end of a loop with 5 iterations
END_LOOP 5

//This instruction specifies the start of a loop with 25 iterations
START_LOOP 25

//This instruction specifies a write to memory address 0x15 of a value of 0x01
WRITE MEM[21] 0x01

//This instruction specifies a read from memory address 0x15 to be logged to the
        log file with the message "MemoryCheck="
READ MEM[0x15] LOG("MemoryCheck=")

//This instruction specifies a read from memory address 0x15 to be logged to the
        log file with the message "MemoryCheck="
READ MEM[21] LOG("MemoryCheck=")

//This instruction specifies the end of a loop with 4 iterations but is overridden by
        the Start_Loop command's number of iterations
END_LOOP 4
```