USB-MPC©

with MDIO and I2C support

User's Manual

FDI Future Designs, Inc.
Your Development Partner

Printed in the United States of America

Revision 0.3 08232005

Table of Contents

1. INTRODUCTION

The USB-MPC interfaces to the standard USB port on any IBM-compatible PC and allows bi-directional communications with I²C and MDIO based peripherals. The software runs on Windows 2000 (service pack 3), 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 5.5 Appendix E: Device Descriptor Files Specification for a further description of DDFs.) The kit comes complete with Software, USB-MPC device, User's Manual, and an 18" cable to allow direct connection to the user's application or any of the standard I²C demo boards.

2. USB-MPC CONTENTS

- USB-MPC device
- 4-pin I²C connecting cable (18" length)
- 10-pin MDIO connecting cable (18" length)
- Software

3. BEFORE YOU BEGIN

3.1 Reference Information

This manual assumes that the user has some experience with I²C, MDIO, 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

3.2 Requirements

- Windows 2000/XP
- 2MB Disk Space
- CD-ROM drive (used for installation only)
- USB 1.1/2.0 Port

3.3 Power Requirements

The USB-MPC device is powered via the USB cable. Internal power is directly derived from the USB bus. The USB-MPC device does not provide power (out) to the I2C or MDIO connectors. Power must be supplied via the target interface cable, from the target, to the I2C or MDIO connectors for proper operation. The USB-MPC supports operation with target devices from 1.8V (min) to 3.3V (max). The USB-MPC is NOT compatible with 5V devices and extreme care must be followed to avoid permanent damage to the USB-MPC.

4. GENERAL OPERATION

4.1 Installing the USB-MPC Software

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

Warning: You MUST install the software on the CD *before* plugging the USB-MPC device into a USB Port.

4.2 Connecting the USB-MPC device to your computer

- It is not necessary to power-down the PC before connecting the USB-MPC device.
- Select an available USB Port
- Connect the USB-MPC device to the USB Port
- Follow the instructions that appear for installing the device drivers.

Warning: Do NOT have devices connected to the MDIO port, Multi-I/O port, or I2C port when plugging into a USB connection.

4.3 Setting the Bus Speed

The USB-MPC 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 400 KHz.

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

4.4 I²C Operation

4.4.1 On-Board EEPROM

The USB-MPC contains an EEPROM that is required for proper USB operation. This EEPROM will conflict with all I2C devices using slave address 0xA0. For proper operation, do not try to program I2C devices at this address.

> Note: The default I²C address for the on-board EEPROM device is 0xA0. This address is reserved and the USB-MPC software will not allow you to read or write to this address.

4.4.2 Connecting the USB-MPC device to your peripheral

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

4.4.3 Configuring the USB-MPC device

For normal operation, no configuration changes should be required on the USB-MPC device.

4.5 MDIO Operation

4.5.1 MDIO Support

USB-MPC supports and has been tested for IEEE 802.3 serial communications using Clause 22 and Clause 45. DDF and SDF files, outlined in *5.5 Appendix E: Device Descriptor Files Specification* and *5.8 Appendix F: Sequence Descriptor Files Specification* respectively, can be used for MDIO operations.

Information regarding IEEE standards can be found at www.ieee.org.

4.5.2 MDIO Setup of the USB-MPC device

No configuration changes should be required on the USB-MPC device. The USB-MPC device also includes one connector for MDIO. This connector supports the common pinout used for communication with other MDIO boards.

5. APPENDICES

5.1 Appendix A: Connector Functions and Locations

I2C

Inter-IC Communication Bus; this connector provide an interface that is compatible with the Philips I2C Demonstration Board. The I2C interface is compatible with target devices operating at 3.3V. The maximum voltage supported is 3.5V. Applying voltages higher than the maximum will result in permanent damage to the USB-MPC and void the warranty.

I2C pinout;

Pin	Name	Function
1	VI2C	Reference power input from target
2	GND	Reference ground from target
3	SCL	I2C Clock output
4	SDA	I2C Data input/output

MDIO

Management Data Input/Output; this connector provides an interface that is compatible with IEEE RFC802.3ae MDIO specification for communications with network devices and supports both Clause 22 and Clause 45. The MDIO interface is compatible with target devices operating from 1.8V to 3.3V. The maximum voltage supported is 3.5V. Applying voltages higher than the maximum will result in permanent damage to the USB-MPC and void the warranty.

MDIO pinout;

Pin	Name	Function
1	N/C	No connection
2	N/C	No connection
3	GND	Reference ground from target
4	VMDIO	Reference power from target
5	GND	Reference ground from target
6	MDC	MDIO Clock output
7	GND	Reference ground from target
8	MDIO	MDIO Data input/output
9	GND	Reference ground from target
10	VMDIO	Reference power from target

Multi I/O

This connector provides multiple interface options supported by the USB-MPC hardware. At this time, MDIO & I2C are the only interfaces supported by the hardware and software. See the sections above for details on MDIO & I2C. Future hardware or software updates may support additional interfaces such as SPI, SMI, etc.

Multi I/O Pinout;

Pin	Name	Function
1	N/C	No connection
2	VCC	Power output from USB-MPC, 3.3V, max current 20mA
3	GND	Reference ground
4	VIO	Reference power input from target
5	GND	Reference ground
6	MDC	MDIO Clock output
7	GND	Reference ground
8	MDIO	MDIO Data input/output
9	GND	Reference ground
10	VIO	Reference power input from target
11	TBD	Undefined
12	TBD	Undefined
13	TBD	Undefined
14	TBD	Undefined
15	SCL	I2C Clock output
16	TBD	Undefined
17	SDA	I2C Data Input/output
18	TBD	Undefined
19	GND	Reference ground
20	TBD	Undefined

5.2 Appendix B: Software Updates

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

5.3 Appendix C: 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 Clause 45. Technical Support for most MDIO peripherals can be found at their corresponding vendors.

Technical Support for the USB-MPC 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.

5.4 Appendix D: Disclaimer

FDI is not responsible for any damage that may be caused by misuse or improper installation of the USB-MPC device.

5.5 Appendix E: Device Descriptor Files Specification

5.5.1 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, or MDIO). All of the DDF files are located in the Devices subdirectory. When the USB-MPC 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 USB-MPC software. In order to add support for new devices to the USB-MPC 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 USB-MPC 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 USB-MPC 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 USB-MPC 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
- MDIO_45 – This token defines an MDIO device that adheres to IEEE 802.3 Clause 45

[ADDR]

The [ADDR] token is used to define the address(es) associated with this device. It's 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 – Optional. The value listed after the token is the 5 bit address of the PHY to be accessed.
- MDIO_45 – Optional. Two values listed after the token define the port and device addresses. E.g. "[ADDR] 0 3" would be a port 0 and device 3.

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]    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 USB-MPC 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 USB-MPC 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 USB-MPC 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 USB-MPC

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 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 F: 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 or MDIO bus in a high-speed environment. All of the SDF files are located in the Sequence subdirectory. When the USB-MPC software is started, all of the SDF files are read and the information is used to generate the supported Sequence list in the USB-MPC software. In order to add a sequence to the USB-MPC 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 USB-MPC 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 USB-MPC. 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 USB-MPC. 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 USB-MPC. 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
ADDR 0255

The ADDR token is used to set the address of the specific device that will be used when running this Sequence in USB-MPC. This token requires input of one address for I2C and MDIO 22. For MDIO 45, two numbers are required, the first for the port and the second for the device. 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 USB-MPC. 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 USB-MPC. 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 USB-MPC 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 USB-MPC
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
```