

Master I²C/SMBus Engine

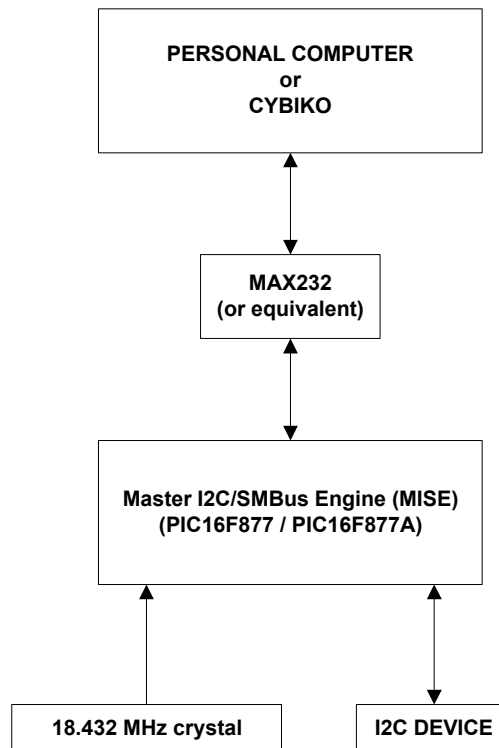


1.0 INTRODUCTION

The Master I²C/SMBus Engine (MISE) is a serial to I²C/SMBus bridge utility that allows users to quickly and easily communicate with I²C and/or SMBus devices via any serial port that handles asynchronous serial communications at 115,200 bps with a 8N1 data format (8 data bits, no parity, 1 stop bit). MISE supports ASCII-based human interactive commands in addition to a separate binary command set that allows users to build I²C/SMBus Personal Computer (PC) applications with their favorite high-level language such as C/C++, Visual Basic, etc. Since the MISE is hosted by a Microchip PIC16F877 or PIC16F877A microcontroller that has an on-board I²C/SMBus engine, the user need not get bogged down with the actual implementation of the I²C/SMBus low-level functions. The MISE can be configured to use different SCL clock frequencies, and the bus type (I²C or SMBus) as well as the slew rate control can be changed and are sticky settings so that they will be retained upon subsequent power-ups.

If you ever wanted to have an I²C/SMBus port on your PC then the Master I²C/SMBus Engine is for you!

2.0 SYSTEM BLOCK DIAGRAM



NOTE: Don't forget the details: crystal load capacitors, decoupling capacitors across the power supply pins, and pull-up resistors on the I²C bus.

The PIC16F877/PIC16F877A microcontroller must be supplied with an 18.432 MHz parallel-cut crystal and appropriate loading capacitors. Consult crystal manufacturer's data sheet for capacitor values, but 33 pF should be a good starting point.

The I²C must have appropriate pull-up resistors on the SDA and SCL lines. Consult the latest Philips I²C Bus Specification for detailed information regarding pull-up resistors, but 3.0 K ohm resistors would be a good starting point.

A MAX232 or equivalent TTL-to-RS-232 level translator must be used if the asynchronous serial port implements the RS-232 electrical specification. Don't forget to connect the serial port's ground to the Master I²C/SMBus Engine ground!

4.0 PIC16F877/PIC16F877A PIN USAGE

From a hardware perspective, the Master I²C/SMBus Engine is simple to wire up. The following list shows PICmicro pin and package-independent pin usage to aid in wiring up your Master I²C/SMBus Engine.

NOTE: V_{DD} must be 4.5V to 5.5V.
Don't forget placing decoupling capacitors as close to the PICmicro as possible.
Keep wiring neat and keep component leads short.

PIN NAME	PIN USAGE
/MCLR/V _{PP}	Connect to V_{DD} (or use your favorite reset circuit/reset supervisor)
V_{DD}	Connect BOTH V_{DD} pins to your 4.5V to 5.5V supply
V_{SS}	Connect BOTH V_{SS} pins to your power supply ground
OSC1, OSC2	Connect your 18.432 MHz parallel-cut crystal to these pins. Don't forget your loading capacitors.
SCL	I ² C Bus SCL line (don't forget your pull-up resistor to V_{DD})
SDA	I ² C Bus SDA line (don't forget your pull-up resistor to V_{DD})
TX	Master I ² C/SMBus Engine's serial port transmit line - Connect to your MAX232
RX	Master I ² C/SMBus Engine's serial port receive line - Connect to your MAX232

That's it. All other pins not named above may be left unconnected as they are configured as outputs.

Question: How will I know that the Master I²C/SMBus Engine is working?

Answer: Upon power-up of the PICmicro that hosts the Master I²C/SMBus Engine (or by a software reset command), the Master I²C/SMBus Engine will perform an ASCII dump to its serial port via the TX line (the display may vary somewhat depending on the firmware version or whether sticky settings were changed, but you should see something like the following):

```
Master I2C/SMBus Engine
v1.0 (C) 2004 Ken Pergola
```

```
ASCII Commands:
1 2 3 S R F * _ # ^ ! ?
```

```
SCL Frequency: 98 KHz
Bus Mode: I2C
Slew Rate Mode: OFF
```

4.0 SERIAL TERMINAL SETUP

Although the Master I²C/SMBus Engine was primarily designed for the Cybiko Classic handheld computer using VTterm application, it will work with most standard PC-hosted terminal emulators such as:

Mtty Serial Terminal
HyperTerminal
Etc.

NOTE: If you wire up your Master I²C/SMBus Engine such that it will be used with a PC's serial port without a null modem adapter or cable, the Cybiko will require a null-modem adapter. The converse is true as well: that is, if you wire up your Master I²C/SMBus Engine such that it will be used with a Cybiko's serial port without a null modem adapter or cable, a connection to a standard PC's serial port will require a null-modem adapter

4.1 USING A PC-HOSTED SERIAL TERMINAL

Terminal UART settings:

115,200 bps
8N1 (8 data bits, no parity, 1 stop bit)
No hardware or software flow control
Local echo off

4.2 USING THE CIBIKO WIRELESS INTER-TAINMENT SYSTEM

Cybiko application: VTterm v1.2 (written by Jeff Frohwein: <http://www.devrs.com/cybiko/download.php> - apps)

If the above link does not work, a search engine of 'vtterm' should get you on track. Download vtterm.zip. Once unzipped, the two files you need to load onto your Cybiko are VTterm.app and ComPort.dl.

If you will be using the Cybiko because of its portability, invoke VTterm (VT100 emulator for Cybiko) and ensure its settings are configured as follows: The settings screen is invoked within VTterm by hitting the Cybiko's 'Select' button.

Terminal Settings

Baud Rate	115200
Data Bits	8
Stop Bits	1.0
Parity	None
Handshake	None
Font Size	5x7
Local Echo	No

5.0 USER-INTERACTIVE ASCII COMMANDS

MASTER I ² C/SMBus ENGINE ASCII COMMAND FUNCTION LIST	ASCII COMMAND
START_CONDITION	1
REPEATED_START_CONDITION	2
STOP_CONDITION	3
SEND_AND_CHECK_SLAVE_FOR_ACK	S
RECEIVE_AND_MASTER_ACK	R
RECEIVE_AND_MASTER_NACK	F
BUS RESET	*
SET_BUS_TYPE	_
SET_SCL_FREQUENCY	#
SET_OUTPUT_SLEW_CONTROL_MODE	^
SOFTWARE_RESET	!
HELP	?

These commands allow the user to manually communicate with I²C devices interactively via a serial terminal.

MASTER I ² C/SMBus ENGINE ASCII COMMAND FUNCTION	ASCII COMMAND
START_CONDITION	1

Command function description:

Initiates a start condition on the bus.

User types the following character on the keyboard: 1

MASTER I ² C/SMBus ENGINE ASCII COMMAND FUNCTION	ASCII COMMAND
REPEATED_START_CONDITION	2

Command function description:

Initiates a repeated start condition on the bus.

User types the following character on the keyboard: 2

MASTER I ² C/SMBus ENGINE ASCII COMMAND FUNCTION	ASCII COMMAND
STOP_CONDITION	3

Command function description:

Initiates a stop condition on the bus.

User types the following character on the keyboard: 3

MASTER I ² C/SMBus ENGINE ASCII COMMAND FUNCTION	ASCII COMMAND
SEND_AND_CHECK_SLAVE_FOR_ACK	S

Command function description:

Sends a byte over the bus. An automatic check for slave ACK is performed with this command.

User types the following character on the keyboard: S

User will then be prompted to enter a byte (in hexadecimal) to send to the slave device.

EXAMPLE:

If you want to send 0xA0 to the slave device, type the following at the keyboard: SA0

NOTE: Uppercase 'S' and lowercase 's' are permissible.

MASTER I ² C/SMBus ENGINE ASCII COMMAND FUNCTION	ASCII COMMAND
RECEIVE_AND_MASTER_ACK	R

Command function description:

Receives a byte from the bus. An automatic master ACK is generated with this command.

User types the following character on the keyboard: R

The byte read from the slave device will then be displayed on the screen.

NOTE: Uppercase 'R' and lowercase 'r' are permissible.

MASTER I ² C/SMBus ENGINE ASCII COMMAND FUNCTION	ASCII COMMAND
RECEIVE_AND_MASTER_NACK	F

Command function description:

Receives a byte from the bus. An automatic master NACK is generated with this command.

User types the following character on the keyboard: F

The byte read from the slave device will then be displayed on the screen.

NOTE: Uppercase 'F' and lowercase 'f' are permissible.

MASTER I ² C/SMBus ENGINE ASCII COMMAND FUNCTION	ASCII COMMAND
BUS_RESET	*

Command function description:

Generates a bus reset by initiating a back-to-back STOP condition, START condition, and STOP condition sequence.

*User types the following character on the keyboard: **

MASTER I ² C/SMBus ENGINE ASCII COMMAND FUNCTION	ASCII COMMAND
SET_BUS_TYPE	_

Command function description:

Allows the bus type (I²C or SMBus) to be set.

User types the following character on the keyboard: _ (underscore character)

The bus mode will be displayed to the user and will toggle between 'I²C' and 'SMBus' when the user repeatedly types this command.

NOTE: This is a sticky setting that will be remembered and loaded upon Master I2C/SMBus Engine power-up.

MASTER I ² C/SMBus ENGINE ASCII COMMAND FUNCTION	ASCII COMMAND
SET_SCL_FREQUENCY	#

Command function description:

Allows the user to set the Master I²C/SMBus Engine's SCL frequency.

User types the following character on the keyboard: #

User will then be prompted to set the SCL Baud Register (in hexadecimal) – see Appendix A for list of frequency choices.

The SCL Baud Register is a 7-bit register. Any value entered exceeding 0x7F will be coerced to 7-bits.

The SCL frequency displayed to the user is an approximation and is unconditionally rounded down to the nearest KHz.

For example, with a SCL Baud Register value of 0x0A, the actual SCL frequency is 418.909 KHz, but will be displayed to the user as 418 KHz.

For the exact frequency that the Master I²C/SMBus Engine outputs, please see APPENDIX A.

NOTE: This is a sticky setting that will be remembered and loaded upon Master I2C/SMBus Engine power-up.

MASTER I ² C/SMBus ENGINE ASCII COMMAND FUNCTION	ASCII COMMAND
SET_OUTPUT_SLEW_CONTROL_MODE	^

Command function description:

Allows the user to set the Master I²C/SMBus Engine's output slew control mode.

User types the following character on the keyboard: ^

The output slew mode will be displayed to the user and will toggle between 'ON' and 'OFF' when the user repeatedly types this command.

NOTE: This is a sticky setting that will be remembered and loaded upon Master I2C/SMBus Engine power-up.

MASTER I ² C/SMBus ENGINE ASCII COMMAND FUNCTION	ASCII COMMAND
SOFTWARE_RESET	!

Command function description:

Generates a software reset on the target microcontroller that hosts the Master I²C/SMBus Engine.

The software reset will reset and re-initialize the Master I²C/SMBus Engine, and display its current settings.

(On the Cybiko, it may take up to 4 seconds for the screen display to refresh after receiving this command.)

User types the following character on the keyboard: !

MASTER I ² C/SMBus ENGINE ASCII COMMAND FUNCTION		ASCII COMMAND
HELP		?

Command function description:

Displays the Master I²C/SMBus Engine's current configuration settings and the ASCII command list.
(On the Cybiko, it may take up to 4 seconds for the screen display to refresh after receiving this command.)

User types the following character on the keyboard: ?

NOTE: Due to the PICC-Lite's limited program memory space for the PIC16F877/PIC16F877A microcontroller, there simply was no room left to display detailed the ASCII commands or help information.

6.0 LOW-LEVEL BINARY COMMANDS

MASTER I ² C/SMBus ENGINE BINARY COMMAND FUNCTION LIST	BINARY COMMAND (hex)
START_CONDITION	0x80
REPEATED_START_CONDITION	0x81
STOP_CONDITION	0x82
SEND_AND_NO_CHECK_SLAVE_FOR_ACK	0x90
SEND_AND_CHECK_SLAVE_FOR_ACK	0x91
RECEIVE_AND_NO_MASTER_ACK_NACK	0xA0
RECEIVE_AND_MASTER_ACK	0xA1
RECEIVE_AND_MASTER_NACK	0xA2
CHECK_SLAVE_FOR_ACK	0xB0
MASTER_ACK	0xC0
MASTER_NACK	0xC1
BUS_RESET	0xF0
SOFTWARE_RESET	0xF1

These non-ASCII commands allow the user to create high-level I²C device applications via the serial port. For example, the MISE ActiveX DLL (MISE.dll) wraps these functions and provides sample device drivers for various I²C devices:

- Microchip MCP23016 I/O expander
- Texas Instruments PCF8574A I/O expander (second source of Philips' part)
- Microchip 24XX00 serial EEPROM
- Dallas Semiconductor DS1775 temperature sensor
- Dallas Semiconductor DS1631/DS1631A/DS1731 temperature sensor
- Texas Instruments AD1100 16-bit A/D converter

MASTER I ² C/SMBus ENGINE BINARY COMMAND FUNCTION	BINARY COMMAND
START_CONDITION	0x80

Command function description:

Initiates a start condition on the bus.

Host sends:

Byte #1: 0x80 (COMMAND FUNCTION)

I²C/SMBus Engine response:

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE
0x00: PASS – Start condition was successful.
0x01: FAIL – Start condition failed.

MASTER I ² C/SMBus ENGINE BINARY COMMAND FUNCTION	BINARY COMMAND
REPEATED_START_CONDITION	0x81

Command function description:

Initiates a repeated start condition on the bus.

Host sends:

Byte #1: 0x81 (COMMAND FUNCTION)

I²C/SMBus Engine response:

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE
0x00: PASS – Repeated start condition was successful.
0x01: FAIL – Repeated start condition failed.

MASTER I ² C/SMBus ENGINE BINARY COMMAND FUNCTION	BINARY COMMAND
STOP_CONDITION	0x82

Command function description:

Initiates a stop condition on the bus.

Host sends:

Byte #1: 0x82 (COMMAND FUNCTION)

I²C/SMBus Engine response:

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE
0x00: PASS – Stop condition was successful.
0x01: FAIL – Stop condition failed.

MASTER I ² C/SMBus ENGINE BINARY COMMAND FUNCTION	BINARY COMMAND
SEND_AND_NO_CHECK_SLAVE_FOR_ACK	0x90

Command function description:

Sends a byte over the bus. No automatic check for slave ACK is performed with this command.

Host sends:

Byte #1: 0x90 (COMMAND FUNCTION)

Byte #2: 0xFF (where FF is the byte to send)

I²C/SMBus Engine response:

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE

0x00: PASS – Byte was sent successfully.

0x01: FAIL – Byte was not sent successfully.

MASTER I ² C/SMBus ENGINE BINARY COMMAND FUNCTION	BINARY COMMAND
SEND_AND_CHECK_SLAVE_FOR_ACK	0x91

Command function description:

Sends a byte over the bus. An automatic check for slave ACK is performed with this command.

Host sends:

Byte #1: 0x91 (COMMAND FUNCTION)

Byte #2: 0xFF (where FF is the byte to send)

I²C/SMBus Engine response:

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE

0x00: PASS – Byte was sent successfully.

0x01: FAIL – Byte was not sent successfully.

Byte #2: SLAVE ACK/NACK RESPONSE (applicable only if previous command function response code is PASS)

0x00: ACK – A slave ACK condition was detected.

0x01: NACK – A slave NACK condition was detected.

MASTER I ² C/SMBus ENGINE BINARY COMMAND FUNCTION	BINARY COMMAND
RECEIVE_AND_NO_MASTER_ACK_NACK	0xA0

Command function description:

Receives a byte from the bus. No automatic master ACK or NACK is generated with this command.

Host sends:

Byte #1: 0xA0 (COMMAND FUNCTION)

I²C/SMBus Engine response:

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE

0x00: PASS – Byte was received successfully

0x01: FAIL – Byte was not received successfully.

Byte #2: Byte received (received byte is valid only if previous command function response code is PASS)

NOTE: This command would rarely be used in normal circumstances. Normally, either the **RECEIVE_AND_MASTER_ACK** or **RECEIVE_AND_MASTER_NACK** command would be used due to their explicit and automatic generation of a master ACK and master NACK, respectively.

MASTER I ² C/SMBus ENGINE BINARY COMMAND FUNCTION	BINARY COMMAND
RECEIVE_AND_MASTER_ACK	0xA1

Command function description:

Receives a byte from the bus. An automatic master ACK is generated with this command.

Host sends:

Byte #1: 0xA1 (COMMAND FUNCTION)

I²C/SMBus Engine response:

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE

0x00: PASS – Byte was received successfully

0x01: FAIL – Byte was not received successfully.

Byte #2: Byte received (received byte is valid only if previous command function response code is PASS)

MASTER I ² C/SMBus ENGINE BINARY COMMAND FUNCTION	BINARY COMMAND
RECEIVE_AND_MASTER_NACK	0xA2

Command function description:

Receives a byte from the bus. An automatic master NACK is generated with this command.

Host sends:

Byte #1: 0xA2 (COMMAND FUNCTION)

I²C/SMBus Engine response:

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE

0x00: PASS – Byte was received successfully

0x01: FAIL – Byte was not received successfully.

Byte #2: Byte received (received byte is valid only if previous command function response code is PASS)

MASTER I ² C/SMBus ENGINE BINARY COMMAND FUNCTION	BINARY COMMAND
CHECK_SLAVE_FOR_ACK	0xB0

Command function description:

Checks the slave for ACK condition.

Host sends:

Byte #1: 0xB0 (COMMAND FUNCTION)

I²C/SMBus Engine response:

Byte #1: SLAVE ACK/NACK RESPONSE

0x00: ACK – A slave ACK condition was detected.

0x01: NACK – A slave NACK condition was detected.

MASTER I ² C/SMBus ENGINE BINARY COMMAND FUNCTION	BINARY COMMAND
MASTER_ACK	0xC0

Command function description:

Generates a master ACK condition on the bus.

Host sends:

Byte #1: 0xC0 (COMMAND FUNCTION)

I²C/SMBus Engine response:

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE

0x00: PASS – Master ACK was initiated successfully

0x01: FAIL – Master ACK was not initiated successfully.

NOTE: This command would rarely be used in normal circumstances. Normally, either the **RECEIVE_AND_MASTER_ACK** or **RECEIVE_AND_MASTER_NACK** command would be used due to their explicit and automatic generation of a master ACK and master NACK, respectively.

MASTER I ² C/SMBus ENGINE BINARY COMMAND FUNCTION	BINARY COMMAND
MASTER_NACK	0xC1

Command function description:

Generates a master NACK condition on the bus.

Host sends:

Byte #1: 0xC1 (COMMAND FUNCTION)

I²C/SMBus Engine response:

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE

0x00: PASS – Master NACK was initiated successfully

0x01: FAIL – Master NACK was not initiated successfully.

NOTE: This command would rarely be used in normal circumstances. Normally, either the **RECEIVE_AND_MASTER_ACK** or **RECEIVE_AND_MASTER_NACK** command would be used due to their explicit and automatic generation of a master ACK and master NACK, respectively.

MASTER I ² C/SMBus ENGINE BINARY COMMAND FUNCTION	BINARY COMMAND
BUS_RESET	0xF0

Command function description:

Generates a bus reset by initiating a back-to-back STOP condition, START condition, and STOP condition sequence.

Host sends:

Byte #1: 0xF0 (COMMAND FUNCTION)

I²C/SMBus Engine response:

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE

0x00: PASS – Bus reset was initiated successfully

0x01: FAIL – Bus reset was not initiated successfully.

MASTER I ² C/SMBus ENGINE BINARY COMMAND FUNCTION	BINARY COMMAND
SOFTWARE_RESET	0xF1

Command function description:

Generates a software-reset on the target microcontroller that hosts the Master I²C/SMBus Engine.

The software reset will reset and re-initialize the Master I²C/SMBus Engine, and the ASCII power-up screen will be dumped to the serial port.

Host sends:

Byte #1: 0xF1 (COMMAND FUNCTION)

I²C/SMBus Engine response:

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE

0x00: PASS – Software reset was successful.

0x01: FAIL – Software reset was not successful.

NOTE: After a PASS code is received, the microcontroller that hosts the Master I²C/SMBus Engine will output its ASCII power-up screen to the serial port. Therefore, any PC host software should take this into account and perform the following steps with regard to this command:

- 1) Issue SOFTWARE_RESET command
- 2) Wait for the PASS code from the Master I²C/SMBus Engine
- 3) Delay 500 milliseconds
- 4) Clear the PC's UART receive buffer

7.0 ActiveX DLL: MISE.dll

The Master I²C/SMBus Engine can be controlled with an ActiveX DLL that wraps the binary command set as well as provides specific device drivers for various I²C/SMBus devices. The MISE ActiveX DLL (MISE.dll) was created in Visual Basic 6.0. Users may explore it within the Visual Basic Object Browser and must set a reference to it (Project menu > References) before using it.

7.1 IN-PROCESS COMPONENT BASE ADDRESS

The base address for the MISE.dll is: 0x1000000

7.2 DLL EXCEPTION ERROR CODES (DLL version v1.0.0)

It would be a sin if I did not publish the unique exception error codes that I have created for this DLL. Don't forget that your client code must have error handling to accommodate these exceptions.

COMMUNICATION_TIMEOUT = 0x80040201
(Serial communication timeout.)

START_CONDITION_FAILURE = 0x80040202
(I2C/SMBus START condition failed.)

REPEATED_START_CONDITION_FAILURE = 0x80040203
(I2C/SMBus REPEATED START condition failed.)

STOP_CONDITION_FAILURE = 0x80040204
(I2C/SMBus STOP condition failed.)

SOFTWARE_RESET_FAILURE = 0x80040205
(Attempt to software reset the Master I2C/SMBus Engine failed.)

SEND_BYTE_FAILURE = 0x80040206
(Attempt to initiate an I2C/SMBus send byte operation failed.)

RECEIVE_BYTE_FAILURE = 0x80040207
(Attempt to initiate an I2C/SMBus receive byte operation failed.)

MASTER_ACK_FAILURE = 0x80040208
(Attempt to initiate an I2C/SMBus MASTER ACK failed.)

MASTER_NACK_FAILURE = 0x80040209
(Attempt to initiate an I2C/SMBus MASTER NACK failed.)

SLAVE_ACKNOWLEDGE_FAILURE = 0x8004020A
(Slave did not acknowledge the byte sent to it.)

ILLEGAL_DEVICE_ADDRESS = 0x8004020B
(The attempt to set the device address failed because an illegal device address for the particular device was selected.)

ILLEGAL_CONVERSION_RESOLUTION = 0x8004020C
(The attempt to set the temperature resolution failed because an illegal conversion resolution for the particular device was selected.)

ILLEGAL_FAULT_TOLERANCE_VALUE = 0x8004020D
(The attempt to set the fault tolerance count failed because an illegal fault tolerance count for the particular device was selected.)

ILLEGAL_GAIN_VALUE = 0x8004020E
(The attempt to set the gain failed because an illegal gain for the particular device was selected.)

ILLEGAL_SAMPLE_RATE_VALUE = 0x8004020F
(The attempt to set the ADC sample rate failed because an illegal sample rate for the particular device was selected.)

7.2 USING THE MISE DLL IN VISUAL BASIC 6

Before using the MISE DLL in Visual Basic, you must first set a reference to it by Selecting the **PROJECT** menu, then the **References...** menu item. Then you just browse to where the MISE.dll file is located, click **Open** in the **Add Reference** dialog box, then click **OK** in the main **References** dialog box. Now you are ready to use the DLL. Here's a quick snippet to get you started:

```
On Error GoTo ErrorHandler

' Declare an object variable reference to the serial port class
Dim SerialPort As clsSerialPort_v100

' Create instance of object
Set SerialPort = New clsSerialPort_v100

' Declare an object variable reference to the low-level I2C/SMBus Engine class
Dim I2C_SMBus As clsMaster_I2C_SMBus_v100

' Create instance of object
Set I2C_SMBus = New clsMaster_I2C_SMBus_v100

' Declare an object variable reference to the ADS1100 analog-to-digital converter class
Dim ADS1100 As clsADS1100_v100

' Create instance of object
Set ADS1100 = New clsADS1100_v100

Dim lngADC_Result As Long

' Easy way to open serial port
Call SerialPort.CommunicationPort_OpenByPortNumber(2)

' Alternate method for opening serial port:
' SerialPort.CommunicationPortNumber = 2
' Call SerialPort.CommunicationPort_OpenExistingPort

' Use the 'I2C_SMBus' object to invoke the
' Master I2C/SMBus binary command set for the low-level I2C/SMBus methods:

I2C_SMBus.BusReset

' Just some examples
' I2C_SMBus.StartCondition
' I2C_SMBus.StopCondition

' ***** Texas Instruments 16-bit analog-to-digital converter
' This snippet shows how to use one of the existing MISE device drivers

' Set the gain of the ADC input
Call ADS1100.SetGainTo_1

' Set sampling frequency
Call ADS1100.SetSampleRateTo_8_SPS

' Set continuous sampling mode
Call ADS1100.ConfigurationBit_SC_Clear

' Grab the current ADC result
lngADC_Result = ADS1100.ADC_OutputResult

' Display the result to the user in decimal and in hexadecimal
MsgBox "ADC result (decimal) = " & lngADC_Result & vbCrLf & _
      "ADC result (hex) = " & Hex$(lngADC_Result)
' *****

' Close the active serial port
Call SerialPort.CommunicationPort_Close

' Clear object variable references
Set ADS1100 = Nothing
Set SerialPort = Nothing
Set I2C_SMBus = Nothing

Exit Sub

ErrorHandler:

' Do your error processing here
'
'

' Just as an example, show the error info to user
MsgBox "An error/exception has occurred." & vbCrLf & vbCrLf & _
      "Error number: " & Hex$(Err.Number) & vbCrLf & _
      "Error description: " & Err.Description & vbCrLf & _
      "Error source: " & Err.Source
```

APPENDIX A – ACTUAL MASTER I²C/SMBus ENGINE SCL FREQUENCIES

NOTE: The SCL Baud Register is effectively a 7-bit register that allows a total of 128 different SCL frequencies to be selected.

SCL BAUD REGISTER (hex)	SCL FREQUENCY (Hz)
0	4,608,000
01	2,304,000
02	1,536,000
03	1,152,000
04	921,600
05	768,000
06	658,286
07	576,000
08	512,000
09	460,800
0A	418,909
0B	384,000
0C	354,462
0D	329,143
0E	307,200
0F	288,000
10	271,059
11	256,000
12	242,526
13	230,400
14	219,429
15	209,455
16	200,348
17	192,000
18	184,320
19	177,231
1A	170,667
1B	164,571
1C	158,897
1D	153,600
1E	148,645
1F	144,000
20	139,636
21	135,529
22	131,657
23	128,000
24	124,541
25	121,263
26	118,154
27	115,200
28	112,390
29	109,714
2A	107,163
2B	104,727
2C	102,400
2D	100,174
2E	98,043
2F	96,000
30	94,041
31	92,160
32	90,353
33	88,615
34	86,943
35	85,333
36	83,782
37	82,286
38	80,842
39	79,448
3A	78,102
3B	76,800
3C	75,541
3D	74,323
3E	73,143
3F	72,000

SCL BAUD REGISTER (hex)	SCL FREQUENCY (Hz)
40	70,892
41	69,818
42	68,776
43	67,765
44	66,783
45	65,829
46	64,901
47	64,000
48	63,123
49	62,270
4A	61,440
4B	60,632
4C	59,844
4D	59,077
4E	58,329
4F	57,600
50	56,889
51	56,195
52	55,518
53	54,857
54	54,212
55	53,581
56	52,966
57	52,364
58	51,775
59	51,200
5A	50,637
5B	50,087
5C	49,548
5D	49,021
5E	48,505
5F	48,000
60	47,505
61	47,020
62	46,545
63	46,080
64	45,624
65	45,176
66	44,738
67	44,308
68	43,886
69	43,472
6A	43,065
6B	42,667
6C	42,275
6D	41,891
6E	41,514
6F	41,143
70	40,779
71	40,421
72	40,070
73	39,724
74	39,385
75	39,051
76	38,723
77	38,400
78	38,083
79	37,770
7A	37,463
7B	37,161
7C	36,864
7D	36,571
7E	36,283
7F	36,000

APPENDIX B – DOCUMENT REVISION HISTORY

v1.0.0 – 03-18-2004 – Kenneth Michael Pergola (KMP)

Initial preliminary release of the Master I²C/SMBus Engine document.

I apologize for the sparse documentation, but I hope this project is simple enough for people to use with the info in this document. You can usually find me on the PICLIST if you have any questions. Or you can e-mail me privately at: no_spam@localnet.com. That is not a typo – that's my real e-mail address – at least for now. If you find any mistakes or find parts of this document that are confusing please let me know. Thanks, and hope you find the Master I²C/SMBus Engine as useful as I have found it to be (especially when used in conjunction with the Cybiko classic).

APPENDIX C – FAQ (FREQUENTLY ASKED QUESTIONS)

NO FAQ LIST AT THIS TIME