1.0 REVISION LEVEL

1.1 FUNCTIONAL DESCRIPTION REVISION HISTORY

ISSUE	ECO #	REVISION DESCRIPTION	AUTHOR	APPROVED
01	G20080526C	Initial Release	Marco Florcruz	Vincent Vivar
				Alberto Malvar
02		Updated for FW v00.16.00	Marco Florcruz	Vincent Vivar

1.2 PRODUCT REVISION HISTORY

MODEL CODE: 73-769-001

SPEC NO.	REV.	DATE	CUST MODEL NO. OR P/N & REV.	MODEL REV.	REMARKS
41966007110	0A				
41966007110	01				
41966007110	03				
41966007110	05				

1.3 DESIGN AUTHORITY: SOFTWARE GROUP - AIL PHILIPPINE BRANCH

2.0 GENERAL DESCRIPTION AND CONTENTS

2.1 TITLE

Software Release Notes For 73-769-001 Module DLL Functional Description File

2.2 TABLE OF CONTENTS

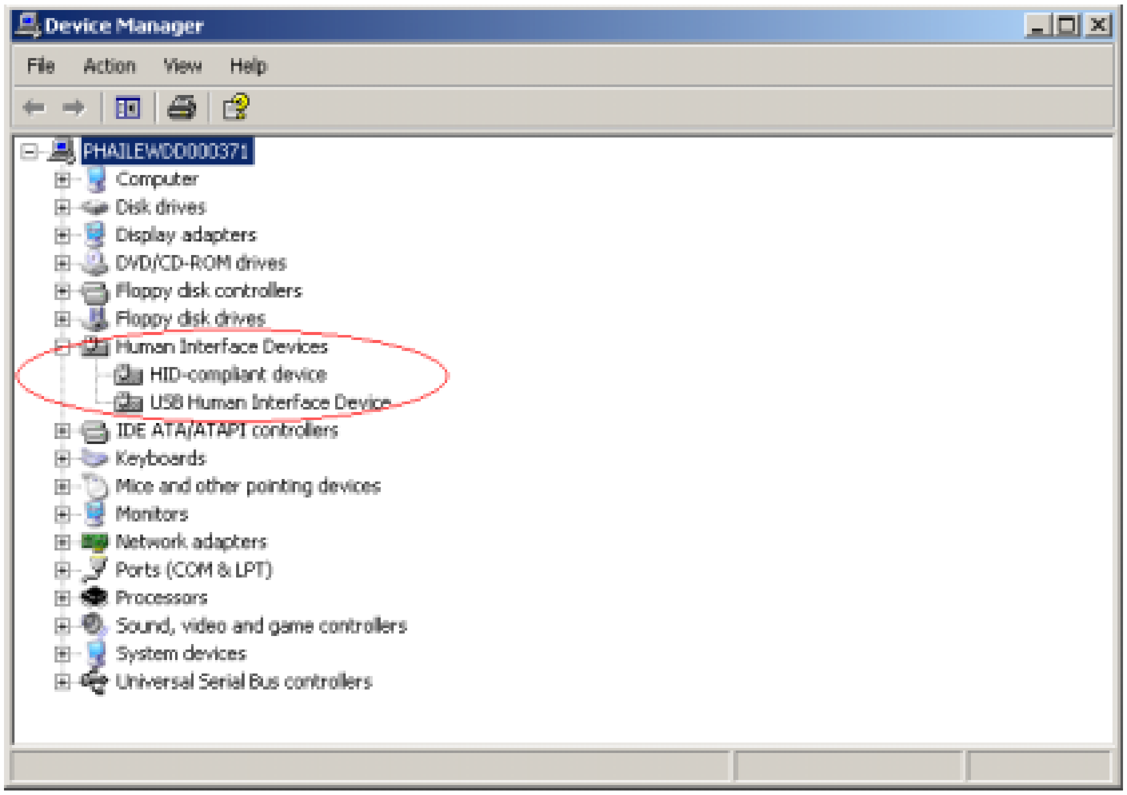
2.3 GENERAL DESCRIPTION AND CONTENTS

2.3.1 System Requirements

- PC-compatible system
- USB port (1.1 or 2.0 compatible)
- Microsoft Windows 98(SE), ME, 2000 or XP

2.3.2 Installation

The USB-to-I2C adapter is an HID class device and requires no driver installation to use. The built in HID driver of the operating is used for the driver. Once the adapter is plugged on the USB port, detection can be confirmed in the Device Manager. Upon successful detection of the adapter, an HID-compliant device will be added as seen in Figure 1.



PHAILEWDD000371
- Computer
- Disk drives
- Display adapters
- DVD/CD-ROM drives
- Floppy disk controllers
- Floppy disk drives
- Human Interface Devices
  - HID-compliant device
  - USB Human Interface Device
- IDE ATA/ATAPI controllers
- Keyboards
- Mice and other pointing devices
- Monitors
- Network adapters
- Ports (COM & LPT)
- Processors
- Sound, video and game controllers
- System devices
- Universal Serial Bus controllers

Figure 1 Device Manager when Adapter is Detected

2.3.3 About the DLL

There are two available DLL files available for use with the USB-to-I2C adapter, each using a different function calling convention. Aside from the function calling convention, there are no other differences between the two DLL. Similar functions are available for both DLL. The two DLL files are:
- iMpDll.dll – Uses C Calling; Calling function cleans the stack
- ail_HID_std.dll – Uses Standard Calling; Called function cleans the stack

To be able to use applications using the DLL, the DLL should be copied in the System32 folder located at the Windows directory (Default: Windows\System32).

2.3.4 Available Functions

Function Name:	**MPUReset**
Description:	Resets the USB-to-I2C Adapter. When sent, the USB-to-I2C will be detected to be Detached, and then Re-attached. All buffers and states are cleared. Since reset will involve detachment and re-attachment of the adapter, communication will be unavailable for a few seconds. It is recommended not to perform communication for a few secs (around 5 seconds) before retrying. No return code should be expected on this version.
Prototype:	
C/C++:	`unsigned char MPUReset(void);`
Visual Basic:	`MPUReset() As Byte`
Input Parameters:	None
Output Parameters:	None

Function Name:	**MPUSBGetDLLVersion**
Description:	Returns current DLL version, which is a non-zero value. A zero will be returned on an error.
Prototype:	
C/C++:	`unsigned short MPUSBGetDLLVersion(void);`
Visual Basic:	`MPUSBGetDLLVersion() As Integer`
Input Parameters:	None
Output Parameters:	Returns unsigned int interpreted as follows: MSB – Major version LSB – Minor Version MSB and LSB are interpreted as Hexadecimal values

Function Name:	**MPUSBGetDeviceVersion**
Description:	Returns USB-to-I2C Adapter firmware version, which is a non-zero value. A zero will be returned on an error.
Prototype:	
C/C++:	`unsigned short MPUSBGetDeviceVersion(void);`
Visual Basic:	`MPUSBGetDeviceVersion() As Integer`
Input Parameters:	None
Output Parameters:	Returns unsigned int interpreted as follows: MSB – Minor Version LSB – Major version MSB and LSB are interpreted as Hexadecimal values

Function Name:	**MPUGetI2CFrequency**
Description:	Returns I2C frequency of USB-to-I2C Adapter is currently operating at, which is a non-zero value. A zero will be returned on an error. I2C frequency accuracy is supported to be within 2% at below 100kHz. I2C frequency accuracy above 100kHz is not guaranteed. If accurate frequency is needed, it is recommended to verify the actual frequency using test equipments.
Prototype:	
C/C++:	unsigned integer MPUGetI2CFrequency (void);
Visual Basic:	MPUGetI2CFrequency`() As Integer`
Input Parameters:	None
Output Parameters:	Returns I2C frequency in Hz. Possible values from 10-400

Function Name:	**MPUSetI2CFrequency**
Description:	Sets current I2C frequency of USB-to-I2C Adapter to desired frequency. Allowed frequency is from 10 kHz to 400 kHz. Attempting to set I2C frequency below/above allowable frequencies will set frequency to minimum/maximum allowable frequency. It will return the actual frequency set, which is a non-zero value. A zero will be returned on an error.

Note that while setting frequency above 100 kHz is allowed, operating at above 100 kHz is not supported.
I2C frequency accuracy is supported to be within 2% at below 100kHz. I2C frequency accuracy above 100kHz is not guaranteed. If accurate frequency is needed, it is recommended to verify the actual frequency using test equipments.

Prototype:

C/C++:
```
unsigned integer MPUSetI2CFrequency(unsigned int
frequency);
```
Visual Basic:
```
MPUSetI2CFrequency(ByVal frequency As Integer) As Integer
```
Input Parameters: frequency - Desired I2C frequency, value accepted from 10-400

Output Parameters: Returns actual I2C frequency set

Function Name: **MPUI2CWrite**

Description: Performs I2C write to a destination address. The number of bytes to be sent must be stated, as well as the pointer to the memory location of the first byte to be sent. It must also be stated whether to include a stop bit will be sent at the end of transmission or not.

S	Address	W	A	nBytes of Data	A	P?

Figure 2 I2C Write

Prototype:

C/C++:
```
unsigned char MPUI2CWrite(unsigned char address, unsigned
short nBytes, *unsigned char WriteData, unsigned short
SendStop);
```
Visual Basic:
```
MPUI2CWrite(ByVal address As Byte, ByVal nBytes As
Integer, ByRef WriteData As Byte, ByVal SendStop As Bool)
As Byte
```
Input Parameters:
address – Destination I2C address (7-bit)
nBytes – Number of bytes to send (maximum of 60 bytes)
WriteData – Memory address of location of 1st byte to be sent
SendStop – Set to ‘1’ to transmit stop bit; Set to ‘0’ to not transmit stop bit

Output Parameters: See Section 2.3.5 Error Codes

Function Name: **MPUI2CRead**

Description: Performs I2C read to a destination address. The number of bytes to be received must be stated, as well as the pointer to the memory location where the first byte received will be stored. It must also be stated whether to include a stop bit will be sent at the end of transmission or not.

S	Address	R	A	nBytes of Data	A	P?

Figure 3 I2C Read

Prototype:

C/C++:
```
unsigned char MPUI2CRead(unsigned char address, unsigned
short nBytes, *unsigned char ReadData, unsigned short
SendStop);
```
Visual Basic:
```
MPUI2CRead(ByVal address As Byte, ByVal nBytes As Integer,
ByRef ReadData As Byte, ByVal SendStop As Bool) As Byte
```
Input Parameters:
address – Destination I2C address (7-bit)
nBytes – Number of bytes to send (maximum of 60 bytes)
Readata – Memory address of location where 1st byte to be received will be stored
SendStop – Set to ‘1’ to transmit stop bit; Set to ‘0’ to not transmit stop bit

Output Parameters: See Section 2.3.5 Error Codes

Function Name: **MPUI2CWriteArray**

Description: Performs I2C write array to a destination address, is an I2C write wherein the 1st byte sent is the sub-address (i.e. in PMBus, the subaddress will be the PMBus command), followed a certain number of bytes, with a stop bit at the end.

S	Address	W	A	Sub-Address	A	nBytes of Data	A	P

Figure 4 I2C Write Array

Prototype:

C/C++:
```
unsigned char MPUI2CWriteArray(unsigned char address,
unsigned char subaddress, unsigned short nBytes, *unsigned
char WriteData);
```

Visual Basic:
```
MPUI2CwriteArray(ByVal address As Byte, ByVal subaddress
As Byte, ByVal nBytes As Integer, ByRef WriteData As Byte)
As Byte
```

Input Parameters:
address – Destination I2C address
subaddress – Destination sub-address
nBytes – Number of bytes to send (maximum of 60 bytes)
WriteData – Memory address of location of 1st byte to be sent

Output Parameters: See Section 2.3.5 Error Codes

Function Name: **MPUI2CReadArray**

Description: Performs I2C read array to a destination address, wherein an I2C write sending one byte of sub-address (i.e. in PMBus, the subaddress will be the PMBus command) is performed (without stop bit), followed by an I2C read of a certain number of bytes. A stop bit is then sent at the end.

S	Address	W	A	Sub-Address	A	Address	R	A	nBytes of Data	A	P

Figure 5 I2C Read Array

Prototype:

C/C++:
```
unsigned char MPUI2CReadArray(unsigned char address,
unsigned char subaddress, unsigned short nBytes, *unsigned
char ReadData);
```

Visual Basic:
```
MPUI2CReadArray (ByVal address As Byte, ByVal subaddress As
Byte, ByVal nBytes As Integer, ByRef ReadData As Byte) As
Byte
```

Input Parameters:
address – Source I2C address
subaddress – Source sub-address
nBytes – Number of bytes to receive (maximum of 60 bytes)
ReadData – Memory address of location where 1st byte to be received will be stored

Output Parameters: See Section 2.3.5 Error Codes

2.3.5 Error Codes

The following are the error codes returned on some of the functions described in Section 2.3.4:

0x00	No Error
0x01	Address NACK
0x02	Data NACK
0x11	Buffer Limit (Hardware buffer)
0x12	Bus Collision (Occurs at sending Start or Stop bit)
0x13	Write Collision (Occurs of Adapter is writing data during I2C Write)
0x14	Idle Timeout
0x15	Start Timeout
0x16	Restart Timeout
0x17	Stop Timeout
0x18	Read Timeout
0x19	ACK Timeout
0x81	Buffer Overflow (Software buffer)
0xFF	USB Hardware not detected