I2CC C Library

User's Guide

Revision 1.06 – 23 January 2012

Byte Paradigm – info@byteparadigm.com

Table of Contents

Table of Tables

References

[1] GP-22050 data sheet (ds_GP22050.pdf)
[2] I2C Xpress data sheet (ds_I2CXpress.pdf)
[3] 8PI Control Panel user’s guide (ug_8PIControlPanel.pdf)

History

Version	Date	Description
1.00	9-oct-2008	Initial revision
1.01	12-nov-2009	Changed the name of the function used to initialize I2C session in C
1.02	21-dec-2009	Updated set of functions – removed CmdBufInsert and CmdBufSet functions. Updated CmdBufAppend parameters
1.03	16-feb-2010	Review for release 1.08f.
1.04	10-May-2010	Some function prototypes updates
1.05	09-Aug-2010	Added function prefixes and multi-device support
1.06	20-Jan-2012	Added IO voltage selection

1 Introduction

The I2C C library is a specialised C library used with the GP Series devices in I2C mode of operation and with the I2C Xpress device. It provides a set of 'pure C' functions to configure and control the chosen device from within a C/C++ compatible environment. As opposed to the corresponding C++ libraries, this library offers a 'pure C' interface with each function, which is often easier to integrate from within any external environment.

This library calls other libraries and functions to manage the low level transfer of data between the host PC and the device. Schematically, any session using the I2C C library starts by connecting itself to the *8PI Smart Router©* application delivered with the *8PI Control Panel*. This application manages the different client connections to the device and handles priorities between the processes and applications. On the other side, it is responsible for the actual data transfers onto the USB connection.

For advanced information and support, please submit your requests to: support@byteparadigm.com.

2 I2CC C Library

2.1 Functions quick Reference Table

Table 1 gives a list of the functions available in the I2C library. They are grouped by functionality as in the I2C.h header file

Table 1: Quick reference table of I2C procedures (by functionality)

Function Prototype	Description
`int i2c_I2CC(void)`	Initialises the I2C session using the I2C C library.
`int i2c_CreateInstance(void)`	Creates and additional instance of the library.
`void i2c_SelectInstance(int Handle)`	Selects a library instance.
`int i2c_SelectDevice(char *pSerNum)`	Selects a device using the serial number.
`void i2c_SelectIOVoltage(int IOVoltage)`	Selects the user interface voltage.
`void i2c_Terminate(void)`	Closes an instance of the I2C library. A call to this function is mandatory for each instance of the library before closing the application.
`int i2c_SetSpeed(unsigned int Speed)`	Selects the I2C bus speed among the I2C standard frequencies.
`int i2c_SetCustomSpeed(unsigned int Speed)`	Selects the I2C bus speed – custom frequency, including non standard ones
`int i2c_GetSpeed(void)`	Returns the selected I2C bus speed.
`void i2c_EnablePullUp(bool Enable)`	Enables/disables the pull-ups.
`bool i2c_PullUpEnabled(void)`	Get the pull-up status.

`int i2c_SetOversampling(`
`unsigned int Oversampling)`
	Defines the oversampling level for the I2C analyser.

`unsigned int i2c_GetOversampling(void)`	Returns the defined oversampling for the I2C analyser.
`int i2c_SetNrSamples(unsigned int NrSamples)`	Defines the number of samples to be collected when running the device in I2C analyser mode.
`unsigned int i2c_GetNrSamples(void)`	Returns the number of samples defined for the next I2C analyser run.
`void i2c_CmdBufDeleteAll(void)`	Deletes the buffer of commands used to control the I2C master.
`void i2c_CmdBufDelete(unsigned int Index)`	Deletes one command in the buffer used to control the I2C master.
`unsigned int i2c_CmdBufSize(void)`	Returns the size of the command buffer used to control the I2C master.

`void i2c_CmdBufAppend(`
`unsigned int Cmd,`
`int Address,`
`unsigned int AddrType,`
`unsigned int Length,`
`char *pData,`
`bool STA,`
`bool STO)`
	Inserts one command after the Index in the command buffer used to control the I2C master.

`int i2c_CmdBufGet(`
`unsigned int Index,`
`unsigned int *pCmd,`
`unsigned int *pAddress,`
`unsigned int *pAddrType,`
`unsigned int *pLength,`
`char *pData,`
`char *pSTA,`
`char *pSTO,`
`char *pValid);`
	Returns the parameters of the command designated by the Index parameter.

Function Prototype	Description
`void i2c_Abort(void)`	Aborts the execution of a running job.
`int i2c_GetStatus(void)`	Returns the status of a job.
`int i2c_RunMaster(void)`	Executes the set of commands programmed in the command buffer used to control the device in master mode.
`int i2c_RunMasterH(int Handle)`	Executes the set of commands programmed in the command buffer used to control the device in master mode. The commands are sent to the device linked to the specified handle.
`int i2c_RunAnalyser(void)`	Runs the device in analyser mode.
`int i2c_RunAnalyserH(int Handle)`	Runs the device (linked to the specified handle) in analyser mode.
`int i2c_SaveConfig(char *pFileName)`	Saves the current configuration to a file.
`int i2c_LoadConfig(char *pFileName)`	Loads a configuration from a file.
`int i2c_SaveConfigAndData(char *pFileName)`	Saves the current configuration and set of data to a file.
`int i2c_LoadConfigAndData(char *pFileName)`	Loads a configuration and a set of data from a file.
`int i2c_ExportRawFile(char *pFileName)`	Exports the data sampled with the I2C analyser to a file, as raw data format.
`int i2c_ExportVCDFile(char *pFileName)`	Exports the data sampled with the I2C analyser to a file, as VCD file format.

2.2 Functions details

int i2c_I2CC(void)

parameters:	***none.***
returns:	A handle to the initialised library instance.
description:	Initialises an I2C session and creates a first instance of the I2CC library. This function must be called at the start of any session using the I2CC library. It enables the control of the device by registering the C session as a client to the *8PI Smart Router*.

int i2c_CreateInstance(void)

parameters:	***none***
returns:	A handle to the created library instance.
description:	Creates an additional instance of the library. This in needed when using more than one device at the same time. Each instance of the library can be linked to a different device with the i2c_SelectDevice function.

void i2c_SelectInstance(int Handle)

parameters:	***Handle: Handle to a instance of a previously create library instance.***
returns:	
description:	Selects the library instance corresponding to the supplied handle. The selected instance must first be created with i2c_CreateInstance.

int i2c_SelectDevice(char *pSerNum)

parameters:	***pSerNum: ascii encoded string containing an 11 character serial number***
returns:	-1 when the selection fails, a positive of zero value is returned on success
description:	Selects the device based on its serial number. The selected device is associated with the currently selected library instance.

void i2c_SelectIOVoltage(int *IOVoltage)

parameters:	***IOVoltage: integer value representing the IO voltage, the IO voltage can be internally generated of user applied. The voltage level is defined in millivolts.***
returns:	
description:	This function only takes the following predefined values: 3300, 2500, 1800, 1500 and 1200. The nearest value must be selected when the user applies a different external voltage level. For example, set IOVoltage to 2500 when 2.7V is applied. The default value is 3300.

void i2c_Terminate(void)

parameters:	***none.***
returns:	
description:	Closes an instance of the I2CC library and closes the communication with the device. A call to this function is mandatory for each instance of the library before closing the application.

int i2c_SetSpeed(unsigned int Speed)

parameters:	***Speed : frequency of the I2C bus in kHz. Valid values: 10, 100, 400, 1000.***
returns:	A negative error code if the speed is not valid.
description:	Selects the I2C bus speed. If the speed is not valid, an error code is returned and the I2C bus frequency remains unchanged.

int i2c_SetCustomSpeed(unsigned int Speed)

parameters:	***Speed : frequency of the I2C bus in kHz. The value can be set between 800 Hz and 10000000 (10MHz).***
returns:	A negative error code if the speed is not valid.
description:	Selects the I2C bus speed. If the speed is not valid, an error code is returned and the I2C bus frequency remains unchanged.

int i2c_GetSpeed(void)

parameters:	***none.***
returns:	An integer value equal to the frequency of the I2C bus in kHz (10,100, 400 or 1000).
description:	Returns the programmed I2C bus speed.

void i2c_EnablePullUp(bool Enable)

parameters:	***Enable : 1 enables the pull-ups, 0 disables the pull-ups.***
returns:	
description:	Enables or disables the I2C pull-ups

bool i2c_PullUpEnabled(void)

parameters:	***none.***
returns:	An boolean value indicating if the pull-ups are enabled.
description:	Returns the programmed I2C bus speed.

int i2c_SetOversampling(unsigned int Oversampling)

parameters:	***Oversampling :*** valid values: 4 and 8.
returns:	A negative error code if the requested oversampling is not valid.
description:	Defines the oversampling for the device used in I2C analyser mode.

According to the selected bus speed, the device automatically sets the sampling frequency.

unsigned int i2c_GetOversampling(void)

parameters:	***none.***
returns:	The oversampling used by the I2C analyser – values: 4 or 8.
description:	The I2C analyser must oversample the I2C to extract data and detect transition. The oversampling rate is the number of samples per I2C clock period.

int i2c_SetNrSamples(unsigned int NrSamples)

parameters:	***NrSamples : an integer value defining the number of samples that should be collected with the I2C analyser.***
returns:	A negative error code if the value exceeds the maximum 25 Msamples.
description:	Defines the number of samples to be collected during the next I2C analyser run.

unsigned int i2c_GetNrSamples(void)

parameters:	***none.***
returns:	An integer equal to the programmed number of samples to be collected during the next I2C analyser run.
description:	Returns the parameter programmed with SetNrSamples.

About CmdBuf functions*

Every I2C transfer is stored in a separate buffer called the "command buffer" or short "CmdBuf".

In master mode, these buffers are created and filled with the CmdBufAppend command. The stored commands are executed when the RunMaster command is called.

In analyser mode, the command buffers are created automatically when the RunAnalyser command is called. This command samples an I2C bus and decodes the captured I2C transfers. Every captured transfer is stored in a separate "command buffer" (CmdBuf). The transfer information can then be retrieved with the CmdBufGet command.

void i2c_CmdBufDeleteAll(void)

parameters:	***none.***
returns:	
description:	Deletes the buffer used to store I2C master commands.

void i2c_CmdBufDelete (unsigned int Index)

parameters:	***Index : index of the command to be deleted.***
returns:	
description:	Deletes one entry (command) from the I2C master command buffer, at the index location.

unsigned int i2c_CmdBufSize(void)

parameters:	***none.***
returns:	An unsigned integer equal to the size of the buffer.
description:	Returns the command buffer size - that is the number of command stored into the command buffer.

void i2c_CmdBufAppend(

```
        unsigned int  Cmd,
        unsigned int  Address,
        unsigned int  AddrType,
        unsigned int  Length,
        char          *pData,
        bool          STA,
        bool          STO);
```

parameters: ***Cmd : command type: valid values: 1 for read, 2 for write.***
Address : I2C device address
AddrType : valid values: 1 for 7-bits address; 2 for 10-bits address.
Length : data length in bytes
****pData : pointer to the set of data to be sent (write operation) or pointer to the memory address where read-back data is stored.***
STA : valid values: 1 : generate start condition; 0 : no start condition.
STO: valid values: 1 : generate stop condition; 0 : no stop condition.

returns:

description: Inserts a I2C master command right after the last position in the command buffer. In case of a write operation, *pData points to the data to be written during the operation. In case of a read operation, *pData points to the memory space where the data read on the I2C bus is going to be stored.

```
int i2c_CmdBufGet(
        unsigned int Index,
        unsigned int *pCmd,
        unsigned int *pAddress,
        unsigned int *pAddrType,
        unsigned int *pLength,
        char *pData,
        char *pSTA,
        char *pSTO,
        char *pValid);
```

parameters: ***Index : position of the command returned from the command buffer.***
****pCmd : points to the command type: 1 for read, 2 for write.***
****pAddress : points to the I2C device address communicated with the command.***
****pAddrType :points to 1 in case of a 7-bits address; 2 in case of a 10-bits address.***
Length : data length in bytes
****pData : pointer to the set of data to be sent (write operation) or pointer to the memory address where data read from the I2C bus is stored (valid only after command execution)***
****pSTA : points to 1 if a START condition is to be executed with the command. Points to a 0 value otherwise.***
****pSTO: points to 1 if a STOP condition is to be executed with the command. Points to a 0 value otherwise.***
****pValid : pointer to a 32 bit vector describing the result of the operation after execution (please refer to encoding in*** Table 2 ***to*** Table 4***).***

returns: A negative integer if the Index does not exist in the command buffer.

description: Returns the parameters of the command designated by the Index parameter. In case of a I2C read access, *pData points to the data read on the I2C only after execution of the command; if the command was not executed yet, the returned value should be ignored. In case of a I2C write access, *pData points to the data to be written on the I2C bus.

Error code	Description
0x--------00	No error
0x--------01	Stop condition following start condition without data
0x--------02	Start condition has aborted a transfer
0x--------04	Stop condition has aborted a transfer

Table 2: Error codes, bits 7 to 0

Error code	Description
0x------00--	No warning
0x------01--	Address was NACKed by target device
0x------02--	Data NACKed during I2C write
0x------04--	Incomplete I2C transfer

Table 3: Warning codes, bits 15 to 8

Valid code	Description
0x0000----	No valid information
0x0001----	Start condition valid
0x0002----	Address valid
0x0004----	Address type valid
0x0008----	Read/write valid
0x0010----	Address acknowledge valid
0x0020----	Data length valid
0x0040----	Data valid
0x0080----	Data acknowledge valid
0x0100----	Stop condition valid
0x1000----	Transfer valid

Table 4: Valid codes, bits 31 to 16

void i2c_Abort(void)

parameters:	***none.***
returns:	
description:	Aborts the execution of the running job.

int i2c_GetStatus(void)

parameters:	***none.***
returns:	Returns 0 if Idle; 1 if a command buffer is being executed.
description:	Returns the status of a job.

int i2c_RunMaster(void)

parameters:	***none.***
returns:	An integer error code <0 if operation failed
description:	Executes a programmed command buffer in I2C master mode.

int i2c_RunMasterH(int Handle)

parameters:	***Handle: handle of a library instance***
returns:	An integer error code <0 if operation failed
description:	Executes a programmed command buffer in I2C master mode. The device linked to the specified handle is used.

int i2c_RunAnalyser(void)

parameters:	***none.***
returns:	An integer error code <0 if operation failed
description:	Runs a capture in I2C analyser mode, according to the programmed parameters.

int i2c_RunAnalyserH(int Handle)

parameters:	***Handle: handle of a library instance***
returns:	An integer error code <0 if operation failed
description:	Runs a capture in I2C analyser mode, according to the programmed parameters. The device linked to the specified handle is used for capturing.

int i2c_SaveConfig(char *pFileName)

parameters:	****pFileName : destination file name.***
returns:	
description:	Saves I2C Master / Analyser configuration to a file.

int i2c_LoadConfig(char *pFileName)

parameters:	****pFileName : source file name.***
returns:	
description:	Loads I2C Master / Analyser configuration from a file.

int i2c_SaveConfigAndData(char *pFileName)

parameters:	****pFileName : destination file name.***
returns:	
description:	Saves I2C Master / Analyser configuration and data to a file.

int i2c_LoadConfigAndData(char *pFileName)

parameters:	****pFileName : source file name.***
returns:	
description:	Loads I2C Master / Analyser configuration and data from a file.

int i2c_ExportRawFile(char *pFileName)

parameters:	****pFileName: specifies the function output file name.***
returns:	0 when the file save completed successfully.
description:	Saves the samples collected with the I2C analyser to a file, raw format.

int i2c_ExportVCDFile(char *pFileName)

parameters:	****pFileName: specifies the function output file name.***
returns:	0 when the file save completed successfully.
description:	Saves the last sampling run to a file, formatted as value change dump (VCD) vector information.