

I2CC C Library

User's Guide

Byte Paradigm – info@byteparadigm.com

Table of Content

1 Introduction 4
2 I2CC C Library 5
2.1 Functions quick Reference Table 5
2.2 Functions details 6

Table of Tables

Table 1: Quick reference table of ADWG procedures (by functionality) 5
Table 2: Error codes, bits 7 to 0 10
Table 3: Warning codes, bits 15 to 8 10
Table 4: Valid codes, bits 31 to 16 10

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 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 SPI library. They are grouped by functionality as in the I2C.h header file

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

Function Prototype	Description
`CI2C(void)`	Initialises the I2C session using the I2C C library.
`void Terminate(void)`	Closes the I2C sessions. A call to this function is mandatory before closing the application.
`int SetSpeed(unsigned int Speed)`	Selects the I2C bus speed.
`int GetSpeed(void)`	Returns the selected I2C bus speed.
`int SetOversampling(unsigned int Oversampling)`	Defines the oversampling level for the I2C analyser.
`unsigned int GetOversampling(void)`	Returns the defined oversampling for the I2C analyser.
`int SetNrSamples(unsigned int NrSamples)`	Defines the number of samples to be collected when running the device in I2C analyser mode.
`unsigned int GetNrSamples(void)`	Returns the number of samples defined for the next I2C analyser run.
`void CmdBufDeleteAll(void)`	Deletes the buffer of commands used to control the I2C master.
`void CmdBufDelete(unsigned int Index)`	Deletes one command in the buffer used to control the I2C master.
`unsigned int CmdBufSize(void)`	Returns the size of the command buffer used to control the I2C master.
`void CmdBufInsert(unsigned int Index, unsigned int Cmd, unsigned int Address, unsigned int AddrType, unsigned int Length, char *pData, bool STA, bool STO)`	Inserts one command before the Index in the command buffer used to control the I2C master.
`void CmdBufAppend(unsigned int Index, unsigned int Cmd, unsigned 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.
`void CmdBufSet(unsigned int Index, unsigned int Cmd, unsigned int Address, unsigned int AddrType, unsigned int Length, char *pData, bool STA, bool STO)`	Modifies the command designated by the Index parameter.

Function Prototype	Description

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

`void Abort(void)`	Aborts the execution of a running job.
`int  GetStatus(void)`	Returns the status of a job.
`int RunMaster(void)`	Executes the set of commands programmed in the command buffer used to control the device in master mode.
`int RunAnalyser(void)`	Runs the device in analyser mode.
`int SaveConfig(char *pFileName)`	Saves the current configuration to a file.
`int LoadConfig(char *pFileName)`	Loads a configuration from a file.
`int SaveConfigAndData(char *pFileName)`	Saves the current configuration and set of data to a file.
`int LoadConfigAndData(char *pFileName)`	Loads a configuration and a set of data from a file.
`int ExportRawFile(char *pFileName)`	Exports the data sampled with the I2C analyser to a file, as raw data format.
`int ExportVCDFile(char *pFileName)`	Exports the data sampled with the I2C analyser to a file, as VCD file format.

2.2 Functions details

CI2C(void)

parameters: ***none.***
returns:
description: Initialises an I2C session using 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*.

void Terminate(void)

parameters: ***none.***
returns:
description: Terminates the I2C C session and closes the communication with the device. A call to this function is mandatory before closing the application. It is required to unregister the session from the *8PI Smart* Router.

int 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 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.

int 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 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 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 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 CmdBufInsert, CmdBufAppend and CmdBufSet commands. 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 CmdBufDeleteAll (void)

parameters: ***none.***

returns:

description: Deletes the buffer used to store I2C master commands.

void 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 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 CmdBufInsert(
unsigned int Index,
unsigned int Cmd,
unsigned int Address,
unsigned int AddrType,
unsigned int Length,
char *pData,
bool STA,
bool STO)

parameters: ***Index : position in the command buffer where the command is to be inserted.***
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 at the Index position in the command buffer. The index of all commands already in the buffer at the index location and higher is incremented.
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.

void CmdBufAppend(
unsigned int Index,
unsigned int Cmd,
unsigned int Address,
unsigned int AddrType,
unsigned int Length,
char *pData,
bool STA,
bool STO);

parameters: ***Index : position in the command buffer right after which the command is to be inserted.***
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 Index position in the command buffer. The index of all commands already in the buffer at higher locations is incremented.
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.

void CmdBufSet(
unsigned int Index,
unsigned int Cmd,
unsigned int Address,
unsigned int AddrType,
unsigned int Length,
char *pData,
bool STA,
bool STO)

parameters: ***Index : position of the command in the command buffer that is to be modified.***
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: Modifies the I2C master command located at the Index position in the command buffer. If the index does not exist, the function returns a negative error code.
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 CmdBufGet(
unsigned int Index,
char *pCmd,
char *pAddress,
char *pAddrType,
char *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 0 in case of a 7-bits address; 1 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 Abort (void)

parameters: ***none.***

returns:

description: Aborts the execution of the running job.

int 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 RunMaster(void)

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

int 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 SaveConfig (char *pFileName)

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

int LoadConfig (char *pFileName)

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

int SaveConfigAndData (char *pFileName)

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

int LoadConfigAndData (char *pFileName)

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

int 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 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.