I2C TCL Library

User's Guide

Table of Content

1 Introduction 4
2 TCL Interpreter 5
2.1 Starting a TCL Session 5
2.2 Getting Information on TCL Procedures 7
3 I2C interface signals 8
4 I2C Library – I2CTclLib 8
4.1 Quick Reference Table 8
4.2 Procedures Detailed Description 9

Table of Tables

Table 1: Quick reference table of I2C procedures 8
Table 2: Error codes, bits 7 to 0 12
Table 3: Warning codes, bits 15 to 8 12
Table 4: Valid codes, bits 31 to 16 12

Table of Pictures

Figure 1: Tcl session start-up example 5
Figure 2: Tcl console shortcut in 8PI Control Panel program group 6
Figure 3: Tcl console at startup 6

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	10-Oct-2008	Initial revision
1.01	21-Dec-2009	Updated set of functions – removed CmdBufInsert and CmdBufSet functions. Updated CmdBufAppend parameters
1.02	16-Feb-2010	Review for release 1.08f.
1.03	20-Jan-2012	Added IO voltage selection

1 Introduction

The purpose of this document is to list and describe all the TCL procedures available in the I2C library provided to control the I2C operating mode of the GP and Xpress Series devices. Each procedure functionality and parameters are described in detail.

A section is also dedicated to the TCL interpreter provided with the *8PI Control Panel* application. The way to start it and to use it is briefly described.

2 TCL Interpreter

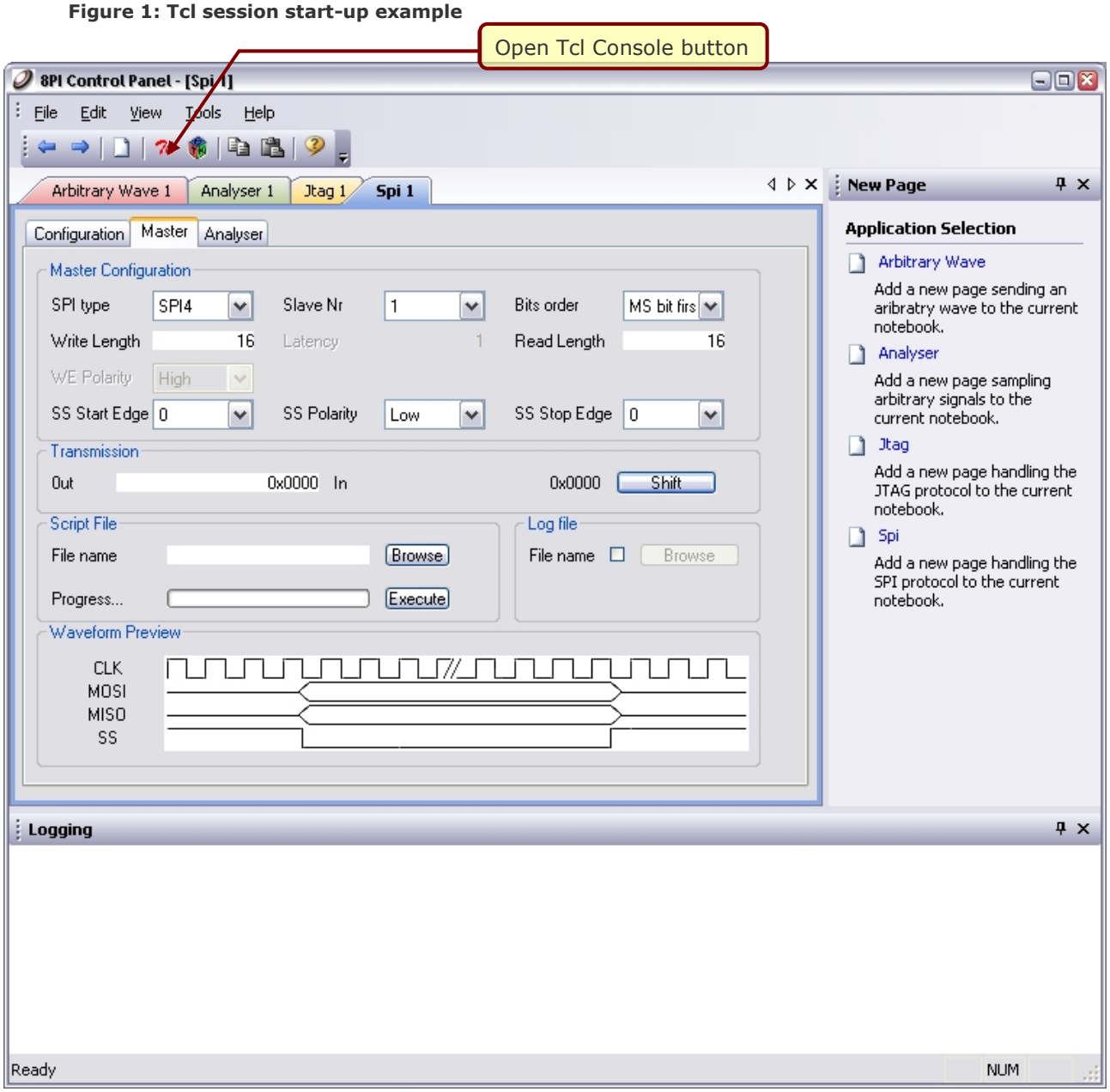
2.1 Starting a TCL Session

To start a TCL session from the 8PI Control Panel GUI:

1. From the 8PI Control Panel GUI, access the desired operating mode sheet.
2. Click on the 'Open Tcl Console' button (Figure 1).

This opens the Tcl console, loads the Tcl libraries relative to the chosen operating mode and initialises the Tcl session.

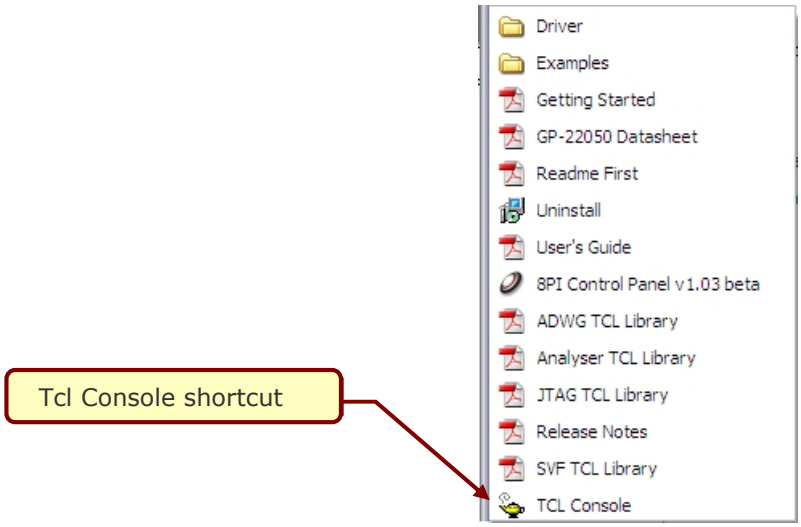
Figure 1: Tcl session start-up example



To start a stand-alone TCL session (without running GUI):

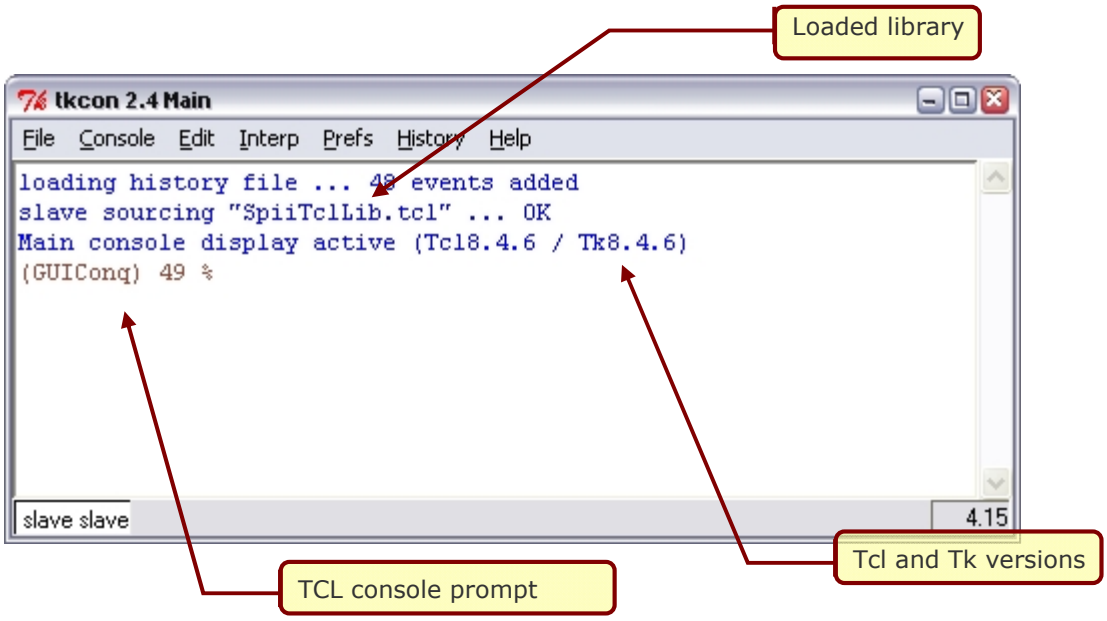
1. From the 'Byte Paradigm > 8PI Control Panel' program group, click on the 'TCL Console' shortcut. This starts the Wish84 interpreter with the tkcon console.
2. In the Tcl console, type: `% source SPIITclLib.tcl`
   This initialises your TCL session in SPI operating mode.

Figure 2: Tcl console shortcut in 8PI Control Panel program group



By default, the 8PI Control Panel software environment uses the WISH interpreter with the TKCON console (interactive mode) (Figure 3). For more information about the TKCON console, please check the following links: http://tkcon.sourceforge.net/ - http://wiki.tcl.tk/1878.
Please note that TCL is case sensitive.

Figure 3: Tcl console at startup



2.2 Getting Information on TCL Procedures

Tcl provides built-in commands for getting information about the elements loaded in memory during a Tcl session. We simply describe a few of them for those unfamiliar to the Tcl language.

To list the libraries loaded in the TCL environment:

```
% info loaded
```

To list all the procedures loaded in the TCL environment:

```
% info procs
```

To list the arguments of a given procedure:

```
% info args <procedure name>
```

To list the body of a given procedure:

```
% info body <procedure name>
```

To learn more about the TCL/TK language, numerous man pages, tutorials and references can be found at the following location: http://www.tcl.tk/doc/ .

3 I2C interface signals

Refer to [3] for pin mappings

4 I2C Library – I2CTclLib

This library contains all the procedures available to control the GP Series devices when operating in I2C mode and the I²C Xpress device.

4.1 Quick Reference Table

Table 1 gives a list of all procedures available for the I2C mode.

Table 1: Quick reference table of I2C procedures

Procedures	Description
`Terminate {}`	Closes the SPI TCL session.
`SelectIOVoltage {IOVoltage}`	Selects the user interface voltage.
`SetSpeed {Speed}`	Selects the I2C bus speed.
`SetCustomSpeed {Speed}`	Selects the I2C bus speed – custom frequency, including non standard ones.
`GetSpeed {}`	Returns the selected I2C bus speed.
`EnablePullUp {Enable}`	Enables/disables the pull-ups.
`PullUpEnabled {}`	Get the pull-up status.
`SetOversampling {Oversampling}`	Defines the oversampling level for the I2C analyser.
`GetOversampling {}`	Returns the defined oversampling for the analyser.
`SetNrSamples {NrSamples}`	Defines the number of samples to be collected when running the device in I2C analyser mode.
`GetNrSamples {}`	Returns the number of samples defined for the next I2C analyser run.
`CmdBufDeleteAll {}`	Deletes the buffer of commands used to control the I2C master.
`CmdBufDelete {Index}`	Deletes one command in the buffer used to control the I2C master.
`CmdBufSize {}`	Returns the size of the command buffer used to control the I2C master.
`CmdBufAppend {Cmd Address AddrType Length pData STA STO}`	Inserts one command after the Index in the command buffer used to control the I2C master.
`CmdBufGet {Index {Message 0}}`	Returns the parameters of the command designated by the Index parameter.
`RunMaster {}`	Executes the set of commands programmed in the command buffer used to control the device in master mode.
`RunAnalyser {}`	Runs the device in analyser mode.
`SaveConfig {pFileName}`	Saves the current configuration to a file.
`LoadConfig {pFileName}`	Loads a configuration from a file.
`SaveConfigAndData {pFileName}`	Saves the current configuration and set of data to a file.
`LoadConfigAndData {pFileName}`	Loads a configuration and a set of data from a file.
`ExportRawFile {pFileName}`	Exports the data sampled with the I2C analyser to a file, as raw data format.
`ExportVCDFile {pFileName}`	Exports the data sampled with the I2C analyser to a file, as VCD file format.

4.2 Procedures Detailed Description

This section gives a detailed description of each procedure.

Terminate {}

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.

SelectIOVoltage {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.
conditions:

SetSpeed {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.

SetCustomSpeed {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.

GetSpeed {}

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.

EnablePullUp {Enable}

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

PullUpEnabled {}

parameters: ***none.***

returns: An boolean value indicating if the pull-ups are enabled.
description: Returns the programmed I2C bus speed.

SetOversampling {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.

GetOversampling {}
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.

SetNrSamples {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.

GetNrSamples {}
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.

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

CmdBufDelete {Index}
parameters: ***Index : index of the command to be deleted.***

returns:
description: Deletes one entry (command) from the I2C master command buffer.

CmdBufSize {}

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.

CmdBufAppend {Cmd Address AddrType Length pData STA 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.

CmdBufGet {Index {Message 0}}

parameters: ***Index : position of the command returned from the command buffer.***
Message : default value: 0 – when set to 1, activates messages.

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.
The parameters are stored into the following set of global variables in the TCL/tk environment:
pCmd : command type – 1 for read / 2 for write;
pAddress : I2C device address;
pAddrType : 1 for 7-bits address; 2 for 10-bits address;
pLength : data length in bytes;
pData : set of data to be sent (write operation) or read-back data (once read executed);
pSTA : 1 if start condition; 0 if no start condition;
pSTO : 1 if stop condition; 0 if no stop condition;

pValid : 32 bits vector (coded in hex) describing the result of the operation after execution (please refer to the tables 2 to 4 below for encoding)

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

RunMaster {}

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

RunAnalyser {}

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

SaveConfig {pFileName}

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

LoadConfig {pFileName}

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

SaveConfigAndData {pFileName}

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

LoadConfigAndData {pFileName}

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

ExportRawFile {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.

ExportVCDFile {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.