SPI TCL Library

User's Guide

Table of Content

Table of Tables

Table of Pictures

References

[1] GP-22050 complete data sheet (ds_GP22050.pdf)
This document contains all the technical characteristics of the GP-22050 device.

[2] 8PI Control Panel – SPI mode of operation user’s guide

History

Version	Date	Description
1.00	15-Dec-2006	Initial revision
1.01	22-Jan-2007	Added analyser functions
1.02	27-Feb-2007	Reviewed and corrected document
1.03	30-May-2007	Review before release
1.04	05-Sep-2007	Modified SetReqClock and SetSSEdge functions description
1.05	13-Nov-2007	Update for SPI Xpress
1.06	08-Oct-2008	Added IdleBurst function
1.07	24-Sept-2009	Update for version 1.07a of 8PI Control Panel
1.08	16-Feb-2010	Review for release 1.08f.
1.09	05-July-2010	Corrected accesses length Completed some functions description
1.10	08-July-2010	Updated info about max. access length in SPI
1.11	22-Oct-2010	Removed obsolete ShiftWr and ShiftWrBurst functions
1.12	20-Jan-2012	Added IO voltage selection

1 Introduction

The objective of this document is to list and describe all the TCL procedures available in the SPI library provided to control the SPI 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

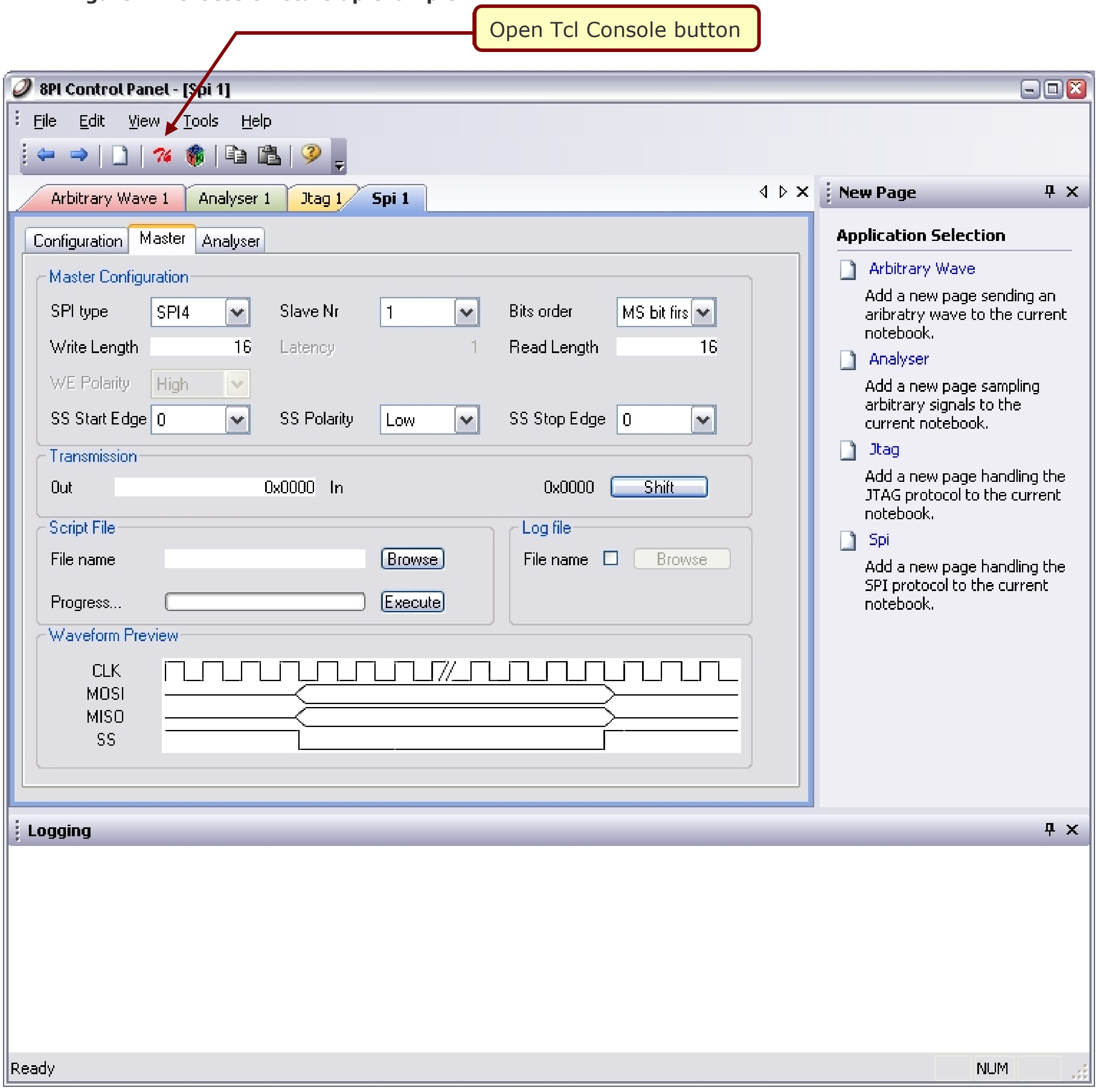
2.1 Starting a TCL Session

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



SPI type	SPI4	Slave Nr	1	Bits order	MS bit firs
Write Length	16	Latency	1	Read Length	16
WE Polarity	High				
SS Start Edge	0	SS Polarity	Low	SS Stop Edge	0

Out 0x0000 In 0x0000

File name

Progress...

File name ☐

Application Selection

Arbitrary Wave
Add a new page sending an aribratry wave to the current notebook.

Analyser
Add a new page sampling arbitrary signals to the current notebook.

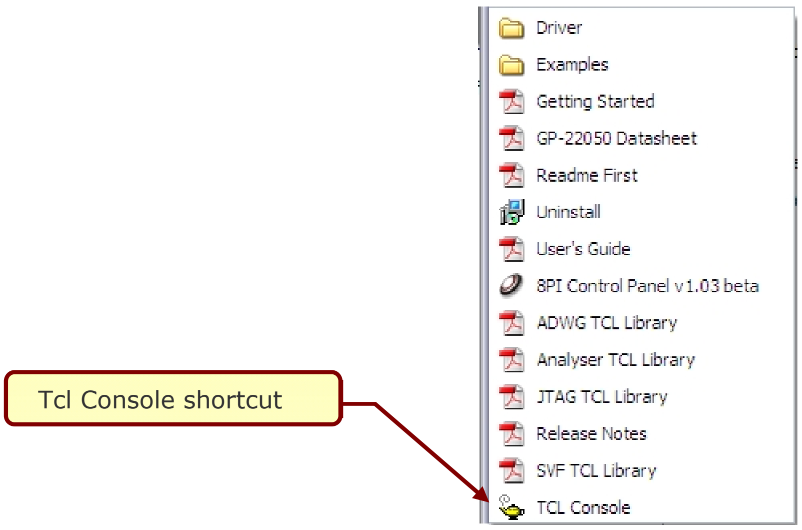
Jtag
Add a new page handling the JTAG protocol to the current notebook.

Spi
Add a new page handling the SPI protocol to the current notebook.

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

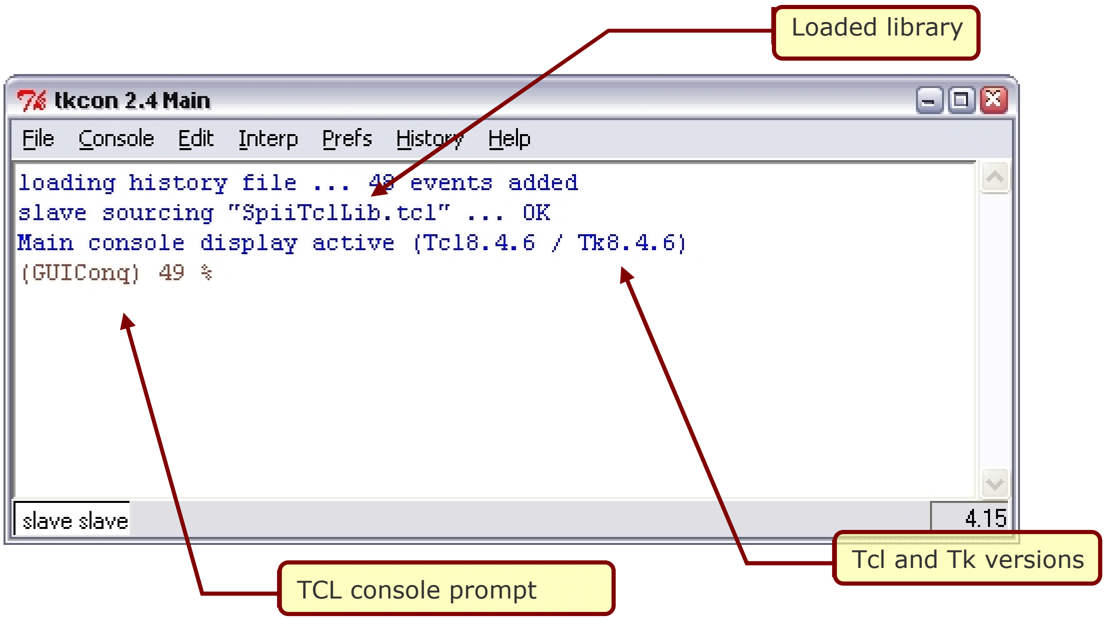
1. From the 'Byte Paradigm > 8PI Control Panel' program group, click on the 'TCL Console' shortcut. This starts the Wish84 interpreter with the tkcon console.
2. In the Tcl console, type: `% source SPIITclLib.tcl`
   This initialises your TCL session in SPI operating mode.

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



```
loading history file ... 48 events added
slave sourcing "SpiiTclLib.tcl" ... OK
Main console display active (Tcl8.4.6 / Tk8.4.6)
(GUIConq) 49 %
```

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 SPI interface signals

Please refer to SPI Xpress / GP Series devices data sheet for pin mappings.

4 SPI Library - SPITclLib

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

4.1 Quick Reference Table

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

Table 1: Quick reference table of SPI procedures

Procedures	Description
`Terminate {}`	Closes the SPI TCL session.
`IsDeviceReady {}`	Checks if the device is properly connected to the host PC.
`SelectIOVoltage {IOVoltage}`	Selects the user interface voltage.
`SetClockCont {Cont}`	Defines the serial clock operating mode: continuous clock or hole clock.
`GetClockCont {}`	Returns the clock mode currently in use.
`SetMode {Mode {Display 0}}`	Defines the SPI operating mode.
`GetMode {}`	Returns the SPI operating mode.
`SetReqClock {Freq}`	Defines the requested operating clock frequency (in Hz).
`GetReqClock {}`	Returns the current requested operating clock frequency (in Hz)
`GetSynthClock {}`	Returns the achieved operating clock frequency (in Hz)
`SetNrOfSlaves {NrSlaves {Display 0}}`	Defines the number of SPI slaves connected to the device. A maximum of 5 slaves can be connected to the same device.
`GetNrOfSlaves {}`	Returns the number of SPI slaves currently defined.
`SelectSlave {SlaveID {Display 0}}`	Defines the ID of the slave to select for the next SPI transfer.
`GetSelectedSlave {}`	Returns the ID of the slave currently selected for SPI transfers.
`SetSSEdges {SSDelayStart SSDelayStop {Display 0}}`	Defines the slave select edge positions.
`GetSSDelayStart {}`	Returns the SS start edge position.
`GetSSDelayStop {}`	Returns the SS stop edge position.
`SetSSClockMasking {Enable}`	Defines the clock masking behaviour with the slave select
`GetSSClockMasking {{Display 0}}`	Returns the clock masking mode with the slave select
`SetLatencyClockMasking {Enable}`	Defines the clock masking behaviour during SPI3 latency
`GetLatencyClockMasking {{Display 0}}`	Returns the clock masking behaviour during SPI3 latency
`SetSSActiveLevel {Level}`	Defines the active level of the SS signal(s).
`GetSSActiveLevel {{Display 0}}`	Returns the active level of the SS signal(s).
`SetBitOrder {BitOrder}`	Defines the bit ordering within each data byte.
`GetBitOrder {{Display 0}}`	Returns the bit ordering within each data byte.
`Plugin4V7V {BitOrder}`	Enable or disable to 4V to 7V plug-in.
`WriteConfFile {FileName {Message 1}}`	Writes the SPI configuration to a file.
`ReadConfFile {FileName { Message 1}}`	Reads the SPI configuration from a file.
`Idle {NrBits {Message 1}}`	Runs a given number of Idle cycles.
`ShiftWrAndRd {NrBits pDataOut {Message 1}}`	Performs a simultaneous write and read access to the selected slave (for SPI-4 only)
`ShiftWrThenRd {NrBitsOut NrBitsIn pDataOut Latency WREnHigh {Message 1}}`	Performs a write access followed by a read access to/from the selected slave (for SPI-3 only).
`InitBurst {}`	Initialise a new burst transfer
`IdleBurst {NrBits {Message 1}}`	Adds a pause (in number of Idle cycles) to the burst buffer.
`ShiftWrAndRdBurst {NrBits pDataOut {Message 1}}`	Adds a new simultaneous write and read access (to the selected slave) to the burst buffer. This is available in SPI-4 mode only.

Procedures	Description
`ShiftWrThenRdBurst {NrBitsOut NrBitsIn pDataOut Latency WREnHigh {Message 1}`	Adds a write access followed by a read access to/from the selected slave to the burst buffer. This is available in SPI-3 mode only)
`SendBurst {{Message 0}}`	Starts the a new burst transfer
`GetFirstBurstTransfer {}`	Selects the first SPI transfer to readback the captured data.
`GetNextBurstTransfer {}`	Selects the next SPI transfer to readback the captured data.
`GetBurstData {Nr}`	Returns one single data byte from one transfer.
`GetCapturedData {NrBytes}`	Returns a pointer to the full set of captured data in one transfer.
`GetLastErr {}`	Returns the last encountered error code.
`GetScriptLength {FileName}`	Returns the loaded script length in number of samples in SPI Master mode.
`ExecuteScript {FileName}`	Executes the loaded script in SPI Master mode.
`Analyse {NrBits {SPI4Mode 1} {SPI3WrLength 0} {SPI3Latency 0} {SPI3RdLength 0} {Message 0}}`	Starts the SPI analyser for a given number of bits.
`SetExportFileName {FileName}`	Specifies the output file name for the autosave feature in SPI Analyser mode.
`GetExportFileName {}`	Returns the name of output file for the autosave feature in SPI Analyser mode.
`SetExportFileType {FileType}`	Specifies the output file type for the autosave feature in SPI Analyser mode.
`GetExportFileType {}`	Returns the selected file type for the autosave feature in SPI Analyser mode.
`SetAutoSave {AutoSave}`	Enables / disables the autosave feature for the SPI Analyser.
`GetAutoSave {{Display 0}}`	Returns the Enable/Disable status of the autosave feature of the SPI Analyser.
`ExportRawDataFile {FileName}`	Exports the analysed data to a file, in raw format.
`ExportRawSPIDataFile {FileName}`	Export the analysed data to a file, in raw SPI format (values sampled at the SPI clock edges).
`ExportDecodedSPIDataFile {FileName}`	Exports the analysed data to a file, decoded format.
`SetInternalTrigger {Internal {Display 0}}`	Defines the triggering mode (internal or external).
`GetInternalTrigger {}`	Returns the current triggering mode.
`SetEdgeTrigger{Enable {Display 0}}`	Selects edge or level trigger.
`GetEdgeTrigger{{Display 0}}`	Returns edge or level trigger.
`SetCtrlTrigMask {Mask {Display 0}}`	Defines the triggering mask to select the control lines to use as trigger inputs.
`GetCtrlTrigMask {}`	Returns the triggering mask.
`SetCtrlTrigPattern {Pattern {Display 0}}`	Defines the pattern to detect on the trigger inputs to generate the trigger event.
`GetCtrlTrigPattern {}`	Returns the triggering pattern.
`SetTriggerPos {Sample}`	Defines the position of the trigger in the run – that is the number of samples before the trigger.
`GetTriggerPos {}`	Returns the programmed trigger position.
`GetOverSampling {}`	Returns the current clock oversampling.

4.2 Procedures Detailed Description

This section gives a detailed description of each procedure.

Terminate {}

parameters:	***None***
returns:	
description:	Closes the SPI TCL session. This function unloads from memory the class instance used to interact with the device and hence disconnects the SPI session from the 'Smart Router'.
conditions:	

IsDeviceReady {}

parameters:	***None***
returns:	1 if device is ready. 0 if device not ready or not connected.
description:	Returns the status of the connection with the device. Using this function is not required to be able to communicate with the device. It is provided to check status if needed. When another function requests an access to the device, the communication status is always automatically checked before starting the transfer.
see also:	

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:	

SetClockCont {Cont}

parameters:	***Cont: valid values: 0 or 1.***
returns:	
description:	Select the clock mode, between 'continuous clock' (Cont = 1) and 'hole clock' (Cont = 0). When the clock operates in *continuous* mode, it is permanently applied on the device output pin. In *hole* clock mode, the clock pulses are provided only when data is being transferred (write or read).
see also:	

GetClockCont {}

parameters:	***None***
returns:	1 if device is in 'continuous' mode; 0 if the clock is in 'hole' mode.
description:	Returns the SPI clock mode currently in use.
see also:	

SetMode {Mode {Display 0}}

parameters:	***Mode: Defines the SPI operation mode – that is the clock idle level and the edges used to send and capture data bits.***

	Idle Level	***MOSI***	***MISO***

0 (00)	***low***	***falling***	***rising***
1 (01)	***low***	***rising***	***falling***
2 (10)	***high***	***rising***	***falling***
3 (11)	***high***	***falling***	***rising***

Display: optional parameter to display additional information. By default set to 0 (disabled).

returns:

description: Defines the clock idle level and clock edges used to sample or generate data bits. When the clock mode is set to non-continuous (*hole* mode), the level of the clock when no transfer is performed can be programmed to remain high (modes 2/3) or low (modes 0/1).
The SPI clock edges used to generate the data out of the master device (MOSI) or to capture data received by the master (MISO) can also be programmed. According to the selected mode, the data are generated and captured on the rising or falling edge of the SPI clock.

see also:

GetMode {}

parameters: ***None***

returns: Integer value describing the selected mode.

	Idle Level	*MOSI*	*MISO*
0 (00)	low	falling	rising
1 (01)	low	rising	falling
2 (10)	high	rising	falling
3 (11)	high	falling	rising

description: Returns the SPI synchronisation mode currently in use.

see also:

SetReqClock {Freq}

parameters: ***Freq: integer value representing the requested frequency to be used to generate the SPI clock signal (in Hz). The value can be set between 800 Hz and 50000000 (50MHz)***

returns: An integer error code <0 if operation failed.

description: Defines the requested frequency for the SPI clock. The parameter of this function is only the requested frequency and not the real frequency that will be generated. The output clock frequency of the device can be programmed with a resolution of 4ns (integer division of a 200MHz reference clock). This means that not all frequencies can be exactly generated. The device always generates the closest frequency corresponding to an integer division of 200MHz and immediately below the requested frequency.
Example:

requested frequency	= 124000 Hz
achieved frequency	= 123992 Hz

conditions:

see also: *GetReqClock***{}**, *GetSynthClock***{}**

GetReqClock {}

parameters: ***None***

returns: A decimal value

description: Returns the requested frequency for the device operating clock (unit = Hz)

conditions:

see also: *GetSynthClock***{}**, *SetReqClock***{}**

GetSynthClock {}

parameters:	***None***
returns:	A decimal value
description:	Returns the real output clock frequency generated by the device. This frequency is the closest frequency corresponding to an integer division of 200MHz and immediately below the requested frequency.
conditions:	
see also:	*SetReqClock***{}**, *GetReqClock***{}**

SetNrOfSlaves {NrSlaves {Display 0}}

parameters:	***NrSlaves: Value defining the number of slaves connected to the Device. This value can be defined between 0 and 5. Default value is 1.*** ***Display: optional parameter used to display additional information. By default set to 0 (disabled).***
returns:	
description:	The device can control up to 5 slaves. This function defines the number of slaves connected to the master device. It configures the number *Slave Select (SS)* that must be used to activate the different slave devices. If the value is set to 0, a single slave device can be connected to the device and the *SS* line is not driven.
conditions:	
see also:	

GetNrOfSlaves {}

parameters:	***none***
returns:	Integer value
description:	Returns the current value programmed defining the number of slave devices connected to the device.

SelectSlave {SlaveID {Display 0}}

parameters:	***SlaveID: Integer value between 1 and 5. By default, ID=1 is defined.*** ***Display: optional parameter used to display additional information. By default set to 0 (disabled).***
returns:	*none*
description:	Defines the ID of the slave that must be selected for the following data transfers. Since the maximum number of slaves that can be connected to the device is limited to 5, the ID value must be defined between 1 and 5.

GetSelectedSlave {}

parameters:	***none***
returns:	Integer value between 1 and 5.
description:	Returns the selected slave.

SetSSEdges {SSDelayStart SSDelayStop {Display 0}}

parameters:	***SSDelayStart: slave select start edge delay.*** ***SSDelayStop: slave select stop edge delay.*** ***Display: optional parameter used to display additional information. By default set to 0 (disabled).***
returns:	An integer error code <0 if operation failed.

description: Defines the delays for the edges of the slave select signal. The rising and falling edges can separately be shifted from 0 to 2 quarters of clock period before and after the conventional start end end edge of the transaction. Shifting the SS edges automatically sets up the proper clock oversampling. Valid values for the parameters are ranging from -2 to 1.

For example, SetSSEdges(-2, 1) means:

- SS starts with -2/4 (-1/2) x SCLK delay, relative to the first active SCLK edge of the SPI access
- SS ends with +1/4 x SCLK delay, relative to the last active SCLK edge of the SPI access.

Please note: the following rules apply for the max. SCLK frequency:

Delay on SS (START or STOP)	**Max. SCLK frequency**
0	50 MHz
½ SCLK	25 MHz
¼ SCLK	12.5 MHz

GetSSDelayStart {}
parameters:
none*returns:* Integer value between -2 and 1.
description: Returns the slave select start edge position as defined with the function SetSSEdges

GetSSDelayStop {}
parameters: ***none***
returns: Integer value between -2 and 1.
description: Returns the slave select stop edge position as defined with the function SetSSEdges

SetSSActiveLevel {Level}
parameters: ***Level: valid values: 0 or 1. 0 for 'active low'; 1 for 'active high'.***
returns:
description: Defines the SS signal(s) active level (high or low).

GetSSActiveLevel {{Display 0}}
parameters: ***Display: optional parameter used to display additional information. By default set to 0 (disabled).***
returns: 0 or 1. 0 for 'active low'; 1 for 'active high'.
description: Returns the SS signal(s) line active level.

SetSSClockMasking {Enable}
parameters: ***Enable: valid values: 1 enables the masking of the clock while the slave select is inactive, 0 disables this feature.***
returns:
description: Defines the clock masking behaviour with the slave select.

GetSSClockMasking {{Display 0}}

parameters:	***Display: optional parameter used to display additional information. By default set to 0 (disabled).***
returns:	0 or 1. 1 for clock masking with slave select; 0 otherwise.
description:	Returns the clock masking mode with the slave select.

SetLatencyClockMasking {Enable}

parameters:	***Enable: valid values: 1 enables the masking of the clock during the SPI3 latency, i.e. while the SPI3 master switched from write to read; 0 disables this feature.***
returns:	
description:	Defines the clock masking behaviour during SPI3 latency.

GetLatencyClockMasking {{Display 0}}

parameters:	***Display: optional parameter used to display additional information. By default set to 0 (disabled).***
returns:	0 or 1. 1 for clock masking during SPI3 latency; 0 otherwise.
description:	Returns the clock masking behaviour during SPI3 latency.

SetBitOrder {BitOrder}

parameters: ***BitOrder: sets MS bit / LS bit first***

returns: *none*

description: Defines the bit ordering within each data byte

0	LS bit first: each byte is sent/read from LSb to MSb
1	MS bit first: each byte is sent/read from MSb down to LSb

Note that the least significant byte of the buffer is always sent first.

GetBitOrder {{Display 0}}

parameters: ***Display: optional parameter used to display additional information. By default set to 0 (disabled).***

returns: Value representing the bit order within each data byte.

description: Defines the bit ordering within the data bytes.

0	LS bit first
1	MS bit first

Note that the least significant byte of the buffer is always sent first.

Plugin4V7V {Enable}

parameters:	***Enable: enables or disables the 4V-7V plug-in***
returns:	*none*
description:	

WriteConfFile {FileName {Message 1}}

parameters:

FileName:	***Output file name, including path.***
Message:	***Flag controlling the generation of dialog box to report error messages.***

1	***dialog box enabled***
0	***dialog box disabled***

When dialog box are disabled, the error is only reported using the return code.

returns: *none*

description: Writes the current SPI configuration to a file. Refer to [2] for a description of the configuration file format.

ReadConfFile {FileName {Message 1}}

parameters: ***FileName:*** ***Input file name, including path.***

Message: ***Flag controlling the generation of dialog box to report error messages.***

1 ***dialog box enabled***

0 ***dialog box disabled***

When dialog box are disabled, the error is only reported using the return code.

returns: *none*

description: Reads a configuration from a file. Refer to [2] for a description of the configuration file format.

Idle {NrBits {Message 1}}

parameters: ***NrBits:*** ***Number of bits (or SPI clock cycles) to run.***

Message: ***Flag controlling the generation of dialog box to report error messages.***

1 ***dialog box enabled***

0 ***dialog box disabled***

When dialog box are disabled, the error is only reported using the return code.

returns: *Integer:* Error code: ≥0 if successful; another value if failed

description: (SPI Master) Holds the SPI interface lines in their default level during a given number of SPI clock cycles.

ShiftWrAndRd {NrBits pDataOut {Message 1}}

parameters: ***NrBits:*** ***Integer value representing the total number of bits to be transferred to/from the slave device. This value must be defined between 1 and a max value described in*** Table 2***. (32.000 bits is the absolute maximum).***

pDataOut: ***Value in hex (0x) with the data to send out.***

Message: ***Flag controlling the generation of dialog box to report error messages.***

1 ***dialog box enabled***

0 ***dialog box disabled***

When dialog box are disabled, the error is only reported using the return code.

returns: An integer error code. ≥0 if successful.

description: Performs a simultaneous write and read access to the slave device. This function can only be used for a 4-wire SPI configuration. Each time a bit is sent out, a bit is captured. If more bits must be read than written, then the output data buffer must be padded with 0 to contain the correct number of bits. The read value is stored in the TCL global variable 'ShiftDataIn'.

		SPI 4 accesses	**SPI 3 accesses**		
SS start delay	**SS end delay**	**ShiftWrAndRd max. access length**	**ShiftWrThenRd Max. WR length**	**ShiftWrThenRd Max. Latency length**	**ShiftWrThenRd Max. Rd length**
(don't care)	-1/4 SCLK	8.000 bits	1.023 bits		1.023 bits

(don't care)	+1/4 SCLK	8.000 bits	1.023 bits	400 bits	1.023 bits
-1/2 SCLK	-1/2 SCLK	16.000 bits	2.047 bits		2.047 bits
-1/4 SCLK	(don't care)	8.000 bits	1.023 bits		1.023 bits
+1/4 SCLK	(don't care)	8.000 bits	1.023 bits		1.023 bits
no delay		**32.000 bits**	**4.095 bits**		**4.095 bits**

Table 2 : Access length according to access type and SS edges positioning.

Refer to function SetSSEdges for more information about how to set the SS start and end delays.

Positioning SS at ½ or ¼ of SCLK automatically switches an oversampling mode in the device, which limits the maximum access length of each type of access.

ShiftWrThenRd {NrBitsOut NrBitsIn pDataOut Latency WREnHigh {Message 1}}

parameters: ***NrBitsOut:*** ***Integer value representing the number of bits to be written to the slave device. This value must be defined between 1 and a max. value described in*** Table 2***.***

NrBitsIn: ***Integer value representing the number of bits to be read from the slave device. This value must be defined between 0 and a max. value described in*** Table 2***.***

pDataOut: ***Value in hex (0x) with the data to send out.***

Latency: ***Integer value defining the number of clock cycles to insert between the write and read access. The value must be defined between 0 and 400.***

WrEnHigh: ***Boolean flag defining the polarity of the write enable signal.***

1 ***active high write enable***

0 ***active low write enable***

Message: ***Boolean flag controlling the generation of dialog box to report error messages.***

1 ***dialog box enabled***

0 ***dialog box disabled***

When dialog box are disabled, the error is only reported using the return code.

returns: An integer error code. ≥0 if successful.

description: Performs a write access followed by a read access to the slave device. This function can only be used for a 3-wire SPI configuration. A first write access is performed to the selected slave device. The length of the access is defined by *NrBitsOut*. Then an idle period of programmable length (*Latency*) is waited before starting the read access. This idle period is used to give time to reverse the data signal direction. The latency can be programmed to 0, but it is recommended to program it at least to 1 to avoid shorts/conflict on the data line due to the time needed by the different devices to reverse the direction.

When the idle period is completed, a read access is started. *NrBitsIn* bits are captured. The read value is stored in the TCL global variable 'ShiftDataIn'.

InitBurst {}

parameters: ***none***

returns:

description: Initialises the system for a new SPI burst transfer. Data read during a previous burst and still available in memory is discarded.

IdleBurst {NrBits {Message 1}}

parameters:	***NrBits:***	***Number of bits (or SPI clock cycles) to run.***
	Message:	***Flag controlling the generation of dialog box to report error messages.*** ***1 dialog box enabled*** ***0 dialog box disabled***
	When dialog box are disabled, the error is only reported using the return code.	
returns:	*Integer:* Error code: ≥0 if successful; another value if failed	
description:	(SPI Master) Holds the SPI interface lines in their default level during a given number of SPI clock cycles. IdleBurst is used with the burst transfer. A burst transfer executes in 2 steps. First, the burst commands are stacked into memory; second, when calling the SendBurst function, the stored burst commands are executed one by one.	

ShiftWrAndRdBurst {NrBits pDataOut {Message 1}}

parameters:	***NrBits:***	***Integer value representing the total number of bits to be transferred to/from the slave device. This value must be defined between 1 and a max value described in*** Table 2***. (32.000 bits is the absolute maximum).***
	pDataOut:	***Value in hex (0x) with the data to send out.***
	Message:	***Flag controlling the generation of dialog box to report error messages.*** ***1 dialog box enabled*** ***0 dialog box disabled***
	When dialog box are disabled, the error is only reported using the return code.	
returns:	An integer error code. ≥0 if successful.	
description:	Adds a new simultaneous write and read access (to the slave device) to the burst buffer. The burst transfer is started by calling the SendBurst function. This function can only be used for a 4-wire SPI configuration. Each time a bit is sent out, a bit is captured. If more bits must be read than written, then the output data buffer must be padded with 0 to contain the correct number of bits.	

ShiftWrThenRdBurst {NrBitsOut NrBitsIn pDataOut Latency WREnHigh {Message 1}}

parameters:	***NrBitsOut:***	***Integer value representing the number of bits to be written to the slave device. This value must be defined between 1 and a max. value described in*** Table 2***.***
	NrBitsIn:	***Integer value representing the number of bits to be read from the slave device. This value must be defined between 1 and a max. value described in*** Table 2***.***
	pDataOut:	***Value in hex (0x) with the data to send out.***
	Latency:	***Integer value defining the number of clock cycles to insert between the write and read access. The value must be defined between 0 and 400.***
	WrEnHigh:	***Boolean flag defining the polarity of the write enable signal.*** ***1*** ***active high write enable*** ***0*** ***active low write enable***
	Message:	***Boolean flag controlling the generation of dialog box to report error messages.*** ***1*** ***dialog box enabled*** ***0*** ***dialog box disabled***

When dialog box are disabled, the error is only reported using the return code.

returns: An integer error code. ≥0 if successful.

description: Adds a new write access followed by a read access (to the slave device) to the burst buffer. The burst transfer is started by calling the SendBurst function. This function can only be used for a 3-wire SPI configuration. A first write access is performed to the selected slave device. The length of the access is defined by *NrBitsOut*. Then an idle period of programmable length (*Latency*) is waited before starting the read access. This idle period is used to give time to reverse the data signal direction. The latency can be programmed to 0, but it is recommended to program it at least to 1 to avoid shorts/conflict on the data line due to the time needed by the different devices to reverse the direction.

When the idle period is completed, a read access is started. *NrBitsIn* bits are captured.

Note that the 'WrEnHigh' and the 'Latency' must be the same for all transfers in one burst.

SendBurst {}

parameters: ***none***

returns:

description: Executes the SPI transfers defined by successive calls to the ShiftWrAndRdBurst and ShiftWrThenRdBurst function.

GetFirstBurstTransfer {}

parameters: ***none***

returns: An integer error code: < 0 if operation failed; ≥ if successful.

description: After the execution of a burst transfer, this command requests the selection of the first SPI transfer to prepare the readback of the captured data. To actually read the readback data, please refer to GetBurstData and GetCapturedData; to select the following burst transfer, please refer to the GetNextBurstTransfer function.

GetNextBurstTransfer {}

parameters:	***none***
returns:	An integer error code: < 0 if operation failed; ≥ if successful.
description:	After the execution of a burst transfer, this command requests the selection of the next SPI transfer to prepare the readback of the captured data. To actually read the readback data, please refer to GetBurstData and GetCapturedData; to select the first burst transfer, please refer to the GetFirstBurstTransfer function.

GetBurstData {Nr}

parameters:	***Nr : integer used to select the byte index within the selected transfer. For instance, if one single transfer is to return 5 bytes of data, Nr can take values from 0 to 4.***
returns:	An integer error code: < 0 if operation failed; ≥ if successful.
description:	Returns one single data byte from the transfer selected with GetFirstBurstTransfer or GetNextBurstTransfer commands. The requested data is stored in the global variable *ShiftDataIn.* Please also refer to GetCapturedData

GetCapturedData {NrBytes}

parameters:	***NrBytes : integer used to specify the total number of bytes to collect from the selected transfer.***
returns:	An integer error code: < 0 if operation failed; ≥ if successful.
description:	Returns the specified number of bytes from the transfer selected with GetFirstBurstTransfer or GetNextBurstTransfer commands.

GetLastErr {}

parameters:	***none***
returns:	
description:	Returns the last error code returned by the device.

GetScriptLength {FileName}

parameters:	***FileName: String – path and file name of the script.***
returns:	An integer value representing the script length in number of SPI accesses.
description:	This function analyses the script file given as input parameter and returns its length in number of lines.

ExecuteScript {FileName}

parameters:	***FileName: String – path and file name of the script.***
returns:	*Integer :* ≥0 if execution successful; other value if execution failed.
description:	Starts the execution of the script given as input parameter in SPI Master mode.

Analyse {NrBits {SPI4Mode 1} {SPI3WrLength 0} {SPI3Latency 0} {SPI3RdLength 0} {Message 0}}

parameters:	***NrBits:***	***unsigned integer specifying the number of samples to analyse.***
	SPI4Mode:	***boolean defining the SPI interface type (SPI4 or SPI3)***
	SPI3WrLength:	***in SPI3 mode, length of the write phase in clock cycles***
	SPI3Latency:	***in SPI3 mode, length of the write-to-read latency in clock cycles.***
	SPI3RdLength:	***in SPI3 mode, length of the read phase in clock cycles.***
	Message:	***boolean value enabling / disabling the function pop-up messages; 1 to enable; 0 to disable.***

returns: Integer : ≥0 if execution successful; other value if execution failed.

description: Starts the sampling and the analysis of the specified number of samples from a SPI interface. Note that the analysis is done by oversampling the SPI interface. Hence, the "NrBits" parameters specifies a number of samples taken and not the number of SPI bits sampled.
Example: assume one wants so sample a 1MHz SPI bus during 1ms with on oversampling of 10, this would mean "NrBits" must be equal to 10000.

SetExportFileName {FileName}

parameters: ***FileName : String – path and file name of the export file***
returns:
description: Specifies the name of the export file used with the autosave feature.

GetExportFileName {Display 0}

parameters:
returns: String – path and file name of the export file.
description: Returns the name of the export file used with the autosave feature.

SetExportFileType { FileType }

parameters: ***FileType: unsigned integer representing the output file type:***
0 : Raw data file
1 : SPI raw data file
2 : Decoded data file
returns:
description: Specifies the type of the export file used with the autosave feature.
The SPI Analyser proceeds by oversampling the data from the SPI port. It can present the data in 3 formats:
0 : Raw data file: all the samples are given out.
1 : SPI raw data file : the analyser the SPI port signals sampled at the chosen SPI clock edge.
2 : Decoded data file : the SPI transaction are extracted from the bitstream (refer to [2] for a description of the corresponding syntax).

GetExportFileType {}

parameters: ***none***
returns: Integer representing the file type*:*
0 : Raw data file
1 : SPI raw data file
2 : Decoded data file
description: Returns the file type programmed with SetExportFileType() function for

the SPI Analyser autosave feature..

SetAutoSave {AutoSave}

parameters: ***AutoSave: boolean – 1 to enable; 0 to disable.***
returns:
description: Enables / disables the SPI Analyser autosave.

GetAutoSave {{Display 0}}

parameters: ***Display: optional parameter used to display additional information. By default set to 0 (disabled).***
returns: 0 or 1. 1 for auto-saving; 0 otherwise.
description: Returns the enable / disable status of the SPI Analyser autosave feature: 1 if enabled; 0 if disabled.

ExportRawDataFile {FileName}

parameters: ***FileName : output export path and file name.***
returns: An integer error code: ≥0 if export successful; other value if export failed.
description: Exports the SPI analysed data to an output file, using the 'raw data' format (please refer to [2] for a description of the SPI Analyser formats).

ExportRawSPIDataFile {FileName}

parameters: ***FileName : output export path and file name.***
returns: An integer error code: ≥0 if export successful; other value if export failed.
description: Exports the SPI analysed data to an output file, using the 'raw SPI data' format (please refer to [2] for a description of the SPI Analyser formats).

ExportDecodedSPIDataFile {FileName}

parameters: ***FileName : output export path and file name.***
returns: An integer error code: ≥0 if export successful; other value if export failed.
description: Exports the SPI analysed data to an output file, using the 'decoded SPI data' format (please refer to [2] for a description of the SPI Analyser formats).

SetInternalTrigger {Internal {Display 0}}

parameters: ***Internal: boolean selecting the trigger type – 1 for internal trigger; 0 for external trigger.***
Display: optional parameter used to display additional information. By default set to 0 (disabled).
returns:
description: To trigger the SPI Analyser, 2 types of trigger can be selected:

- an internal trigger – the SPI Analyser starts immediately when the user runs the Analyse() command;
- an external trigger, defined onto the device 6 control lines.

GetInternalTrigger {}

parameters:
returns: 1 for internal trigger; 0 for external trigger.
description: Returns the trigger type previously programmed with the SetInternalTrigger{} function.

SetEdgeTrigger {Enable {Display 0}}

parameters: ***Enable: boolean boolean selecting edge or level trigger – 1 for***

edge trigger; 0 for level trigger.
Display: optional parameter used to display additional information. By default set to 0 (disabled).

returns:
description: To Selects edge or level trigger.

GetEdgeTrigger {{Display 0}}

parameters: ***Display : optional parameter used to display additional information. By default set to 0 (disabled)..***
returns: 1 for edge trigger; 0 for level trigger.
description: Returns edge or level trigger.

SetCtrlTrigMask {Mask {Display 0}}

parameters: ***Mask: Integer. This range of the mask depends on the operating mode:***

- ***Analyser mode : value represents a 6 bit mask and ranges from 0x01 to 0x3F.***
- ***Master mode : value represents a 4 bit mask and ranges from 0x02 to 0x1E; bit 0 is used for the write enable signal and bit 5 is used for the clock.***

Display:optional parameter used to display additional information. By default set to 0 (enabled).

returns:
description: When the external triggering mode is used, the trigger mask selects the control lines to be used as trigger inputs. When a mask bit is set to 0, the corresponding control line is masked for triggering. The mask is given as a pointer to a short value equivalent to the binary value of the mask (example: mask = (Binary)011000 ➔ *pMask points to a short = 24).

GetCtrlTrigMask {}

parameters: ***none.***
returns:
description: Returns the mask applied on the control lines to detect the external trigger.

SetCtrlTrigPattern {Pattern {Display 0}}

parameters: ***Pattern: Integer. This range of the pattern depends on the operating mode:***

- ***Analyser mode : value represents a 6 bit mask and ranges from 0x01 to 0x3F.***
- ***Master mode : value represents a 4 bit mask and ranges from 0x02 to 0x1E; bit 0 is used for the write enable signal and bit 5 is used for the clock***

Display:optional parameter used to display additional information. By default set to 0 (disabled).

returns:
description: Defines the pattern to detect on the trigger inputs to generate the trigger event. The pattern is given as a pointer to a short value equivalent to the binary value of the pattern (example: pattern = (Binary)011000 ➔ *pPattern points to a short = 24).

GetCtrlTrigPattern {}

parameters:	***none.***
returns:	
description:	Returns the pattern applied on the control lines to generate a trigger event and start applying data samples on the device system connector.

SetTriggerPos {Sample}

parameters:	***Sample: integer value representing the index of the sample in the run where the trigger should be positioned.***
returns:	An integer error code: 0 is successful; another value if unsuccessful.
description:	Use this function to position the trigger after a given number of samples in the total run. Once the sampling run is over, the corresponding number of samples before the trigger is displayed, together with the rest of the run *after* the trigger.

GetTriggerPos {}

parameters:	***none.***
returns:	A integer representing the trigger position.
description:	Returns the trigger position previously programmed. The trigger position is defined with the sample index at which it is positioned.

GetOverSampling {}

parameters:	***none.***
returns:	An integer value equal to 1, 2 or 4.
description:	According to the position of the SPI Master SS edges, the device automatically selects the adequate oversampling of its internal clock with respects to the SPI clock. This function returns the selected oversampling.