JTAG TCL Library

User's Guide

Byte Paradigm – info@byteparadigm.com

Table of Content

Table of Tables

Table of Pictures

References

[]

History

Version	Date	Description
1.00	19-Oct-2005	Initial revision
1.01	02-Nov-2005	Reviewed
1.02	15-Feb-2006	Updated to comply with software version 1.03
1.03	13-Nov-2007	Update for GP Series introduction

1 Introduction

The objective of this document is to list and describe all the TCL procedures available in the JTAG library provided to control the JTAG operating mode of the GP Series device. Each procedure functionality and parameters are described in detail. Scripts examples, which use some of these procedures, are also provided to help the users build their own test environments.

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

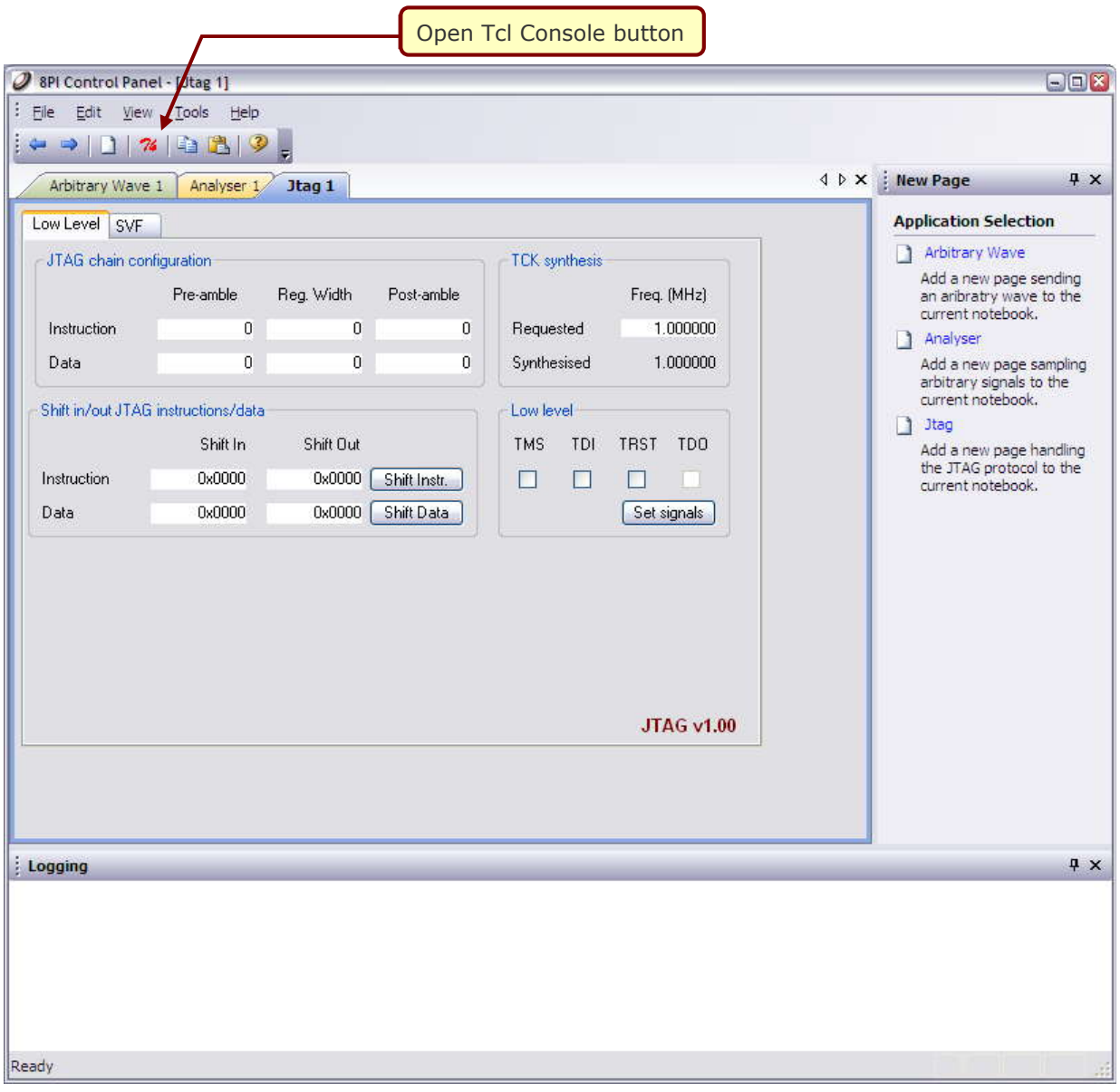
2.1 Starting a TCL Session

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.

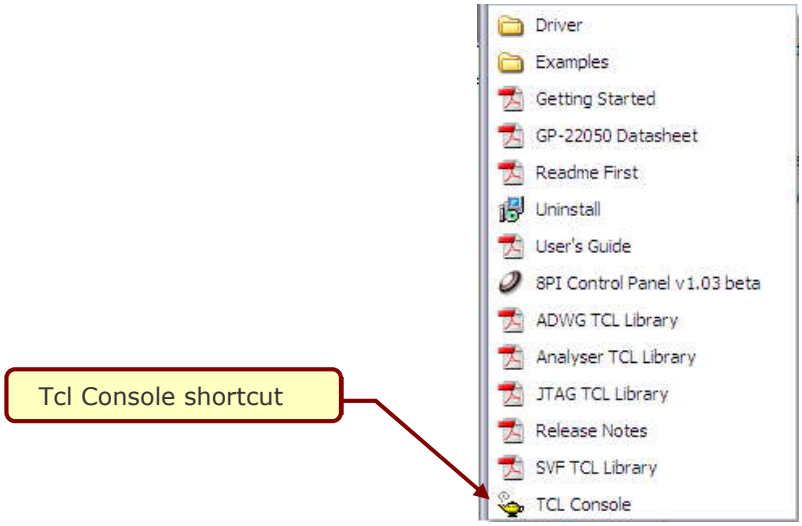
Figure 1: Tcl session start-up example



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

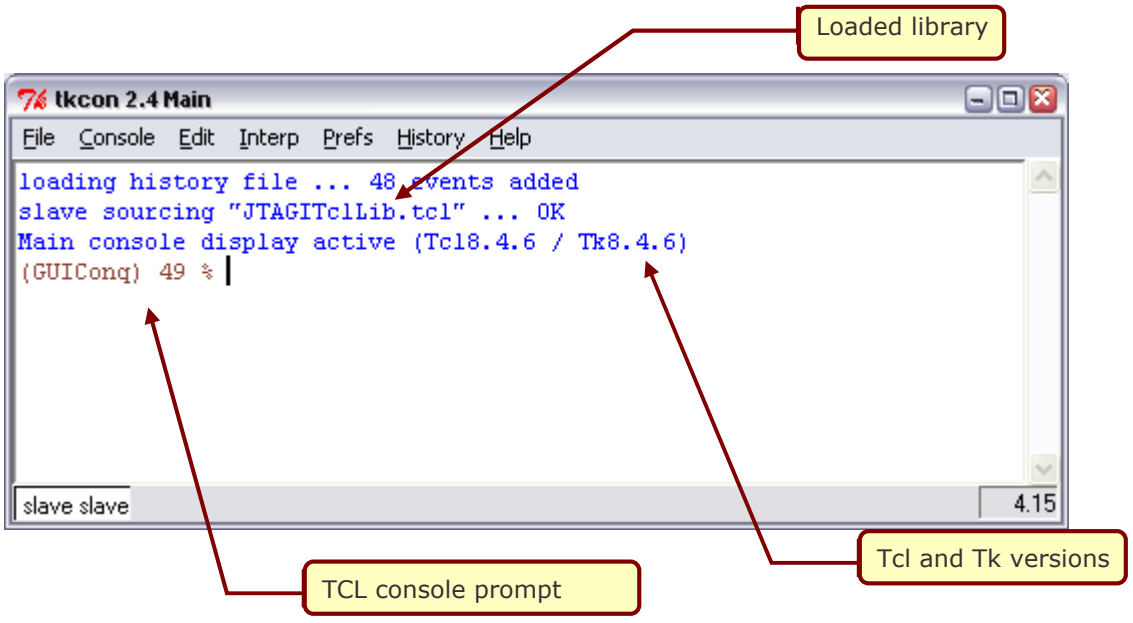
1. From the 'Byte Paradigm > 8PI Control Panel' program group, click on the 'TCL Console' shortcut. This starts the Wish84 interpreter with the tkcon console.
2. In the Tcl console, type: `% source JTAGITclLib.tcl`
   This initialises your TCL session in Analyser operating mode.

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



By default, the GP Series device 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 start-up



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 JTAG Library - JTAGITclLib

This library contains all the procedures available to control the GP Series device when operating in JTAG mode. Using these procedures in scripts or command lines allows the user to control any JTAG compliant Test Access Port (TAP) controller

3.1 Quick Reference Table

Table 1 gives a list of all procedures available for the JTAG mode. They are grouped by type and level of abstraction.

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

Procedures	Description
Initialisation	
`Init {}`	JTAG mode initialisation. This procedure is called automatically when the Tcl shell is started.
`ResetJtagFSM {}`	Resets the TAP controller to the RESET state.
Signal Level	
`SetConstantOutput {TMS TDI TRST}`	Forces a signal to 1.
`SetSignals {TMS TDI TRST}`	Sets the JTAG signals at once.
State Level	
`State {State {Repeat 1}}`	Moves to the specified state and loop if needed.
`GetCurrentState {}`	Gets the current TAP controller state.
`Idle {}`	Moves the TAP controller to the IDLE state.
Instruction/Data Level	
`Shift {BufDir ShiftType pDataOut {pDataIn -1} {EndState IDLE} {Display 0}}`	Shifts an instruction or data.
`SetInstrWidth {Width}`	Sets the width of the instruction register.
`SetDataWidth {Width}`	Sets the width of the data register.
`GetInstrWidth {}`	Gets the width of the instruction register.
`GetDataWidth {}`	Gets the width of the data register.
`SetInstrPreAmble {Length {pAmble 0x0}}`	Configures the instruction pre-amble.
`SetInstrPostAmble {Length {pAmble 0x0}}`	Configures the instruction post-amble.
`SetDataPreAmble {Length {pAmble 0x0}}`	Configures the data pre-amble.
`SetDataPostAmble {Length {pAmble 0x0}}`	Configures the data post-amble.
`GetInstrPreAmble {{pAmble 0x0}}`	Gets the instruction pre-amble.
`GetInstrPostAmble {{pAmble 0x0}}`	Gets the instruction post-amble.
`GetDataPreAmble {{pAmble 0x0}}`	Gets the data pre-amble.
`GetDataPostAmble {{pAmble 0x0}}`	Gets the data post-amble.
Miscellaneous Procedures	
`SetTCKFreq {Freq}`	Specifies the frequency of the TCK clock.
`GetTCKFreq {}`	Returns the programmed TCK clock frequency.
`SetContTCK {Cont}`	Specifies the clock 'continuity' (continuous or 'hole' clock)

3.2 Procedures Detailed Description

This section gives a detailed description of each procedure available to control the GP Series device JTAG operating mode.

3.2.1 Initialisation

Init {}

parameters: None
returns:
description: Downloads the configuration into the GP Series device and prepares it to operate in JTAG mode.
conditions: None
see also:

ResetJtagFSM {}

parameters: None
returns:
description: Resets the TAP controller to the RESET state. The TAP controller is reset by generating five pulses on TCK with TMS equal to one.
conditions: None
see also:

3.2.2 Signal Level

SetConstantOutput {TMS TDI TRST}

parameters:
TMS: TMS value to be applied
TDI: TDI value to be applied
TRST: TRST value to be applied
returns:
description: This procedure allows to force one or more of the three JTAG signals to a logic one. The specified values are ORed with JTAG signal. If for example, SetConstantOutput {0 0 1} is specified, TRST will be 1 whatever value is specified elsewhere. TMS and TDI are not affected. This procedure does not enable to force a signal to 0.
conditions: None
see also:

SetSignals {TMS TDI TRST}

parameters:
TMS: TMS value to be applied
TDI: TDI value to be applied
TRST: TRST value to be applied
returns: 1 if the TDO signal is high
0 if the TDO signal is low
description: Sets the three JTAG signals (TMS, TDI and TRST) at once and generates a pulse on TCK. TDO is read back and is assigned to the return value.
conditions: None
see also:

3.2.3 State Level

State {State {Repeat 1} }

parameters: *State*: Target state for the TAP controller state machine. The state can be entered as a full state name or an encoding, as described hereafter:

State Name	*or Encoding*
RESET	0

IDLE	1
DRSELECT	2
DRCAPTURE	3
DRSHIFT	4
DREXIT1	5
DRPAUSE	6
DREXIT2	7
DRUPDATE	8
IRSELECT	9
IRCAPTURE	10
IRSHIFT	11
IREXIT1	12
IRPAUSE	13
IREXIT2	14
IRUPDATE	15

Repeat: Optional. Number of times the state machine must cycle in the target state.

returns:
description: The TAP controller state machine moves to the specified state. Once in that state the *Repeat* number of clock cycles are generated to loop in the destination state.
conditions: None
see also: *GetCurrentState{ }*

GetCurrentState { }

parameters: None
returns: int The code of the current state.
description: Retrieves the code of the current state. For the state encoding see the previous procedure.
conditions: None
see also: *State{State {Repeat 1}}*

Idle { }

parameters: None
returns:
description: Moves the TAP controller to the IDLE state.
conditions: None
see also:

3.2.4 Instruction/Data Level

Shift { BufDir ShiftType pDataOut {pDataIn -1} {EndState IDLE} {Display 0}}

parameters:

BufDir: Specifies the direction of the shift operation. The allowed values are:

Allowed Value	*Description*
DIR_READ	shift to TDI
DIR_WRITE	shift from TDO
DIR_READWRITE	shift to TDI and from TDO

ShiftType: Specifies if an instruction or a data is shifted. The allowed values are:

Allowed Value	*Description*
SHIFT_INSTR	shift an instruction
SHIFT_DATA	shift data

pDataOut: String containing the data to be sent on TDI.
pDataIn: Optional. String allocated to receive the data coming out of TDO.
EndState: Optional. Specifies the state the TAP controller ends in after the shift operation. EndState must be specified using the state encoding. For the state encoding see the description of "NextState" above.
Display: Optional. Display the result of the shift operation in the tcl shell if set.

returns:
description: Shifts an instruction or data to the TAP controller and retrieves the TDO output on request.
conditions: None
see also:

SetInstrWidth {Width }

parameters: *Width:* Width of the instruction register.
returns:
description: Specifies the width of the instruction register. The specified value is used for all subsequent instruction shift operations until otherwise specified by a new call to the SetInstrWidth procedure.
conditions: None
see also: *GetInstrWidth{}*

SetDataWidth {Width }

parameters: *Width:* Width of the data register.
returns:
description: Specifies the width of the data register. The specified value is used for all subsequent data shift operations until otherwise specified by a new call to the SetDataWidth procedure.
conditions: None
see also: *GetDataWidth{}*

GetInstrWidth { }

parameters: *None*
returns: int Width of the instruction register
description: Retrieves the width of the instruction register as specified with the SetInstrWidth procedure.
conditions: None
see also: *SetInstrWidth{Width}*

GetDataWidth { }

parameters: *None*
returns: int Width of the data register
description: Retrieves the width of the data register as specified with the SetDataWidth procedure.
conditions: None
see also: *SetDataWidth{Width}*

SetInstrPreAmble {Length {pAmble 0x0}}

parameters: *Length:* Instruction pre-amble length.
pAmpble: Optional. *Reserved for future extensions.*
returns:

description: Specifies the length of the instruction pre-amble. The instruction pre-amble is equal to the sum of all the instruction register's lengths of the TAP controllers following the device under test in the JTAG scan chain.
conditions: None
see also: GetInstrPreAmble *{}*

SetInstrPostAmble {Length {pAmble 0x0}}

parameters: *Length:* Instruction post-amble length.
pAmpble: Optional. *Reserved for future extensions.*
returns:
description: Specifies the length of the instruction post-amble. The instruction post-amble is equal to the sum of all the instruction register's lengths of the TAP controllers preceding the device under test in the JTAG scan chain.
conditions: None
see also: GetInstrPostAmble *{}*

SetDataPreAmble {Length {pAmble 0x0}}

parameters: *Length:* Data pre-amble length.
pAmpble: Optional. *Reserved for future extensions.*
returns:
description: Specifies the length of the data pre-amble. The data pre-amble is equal to the sum of all the data register's lengths of the TAP controllers following the device under test in the JTAG scan chain.
conditions: None
see also: GetDataPreAmble *{}*

SetDataPostAmble {Length {pAmble 0x0}}

parameters: *Length:* Data post-amble length.
pAmpble: Optional. *Reserved for future extensions.*
returns:
description: Specifies the length of the data post-amble. The data post-amble is equal to the sum of all the data register's lengths of the TAP controllers preceding the device under test in the JTAG scan chain.
conditions: None
see also: GetDataPostAmble *{}*

GetInstrPreAmble {{pAmble 0x0}}

parameters: *pAmpble:* Optional. *Reserved for future extensions.*
returns: int Instruction pre-amble length as specified with the SetInstrPreAmble procedure.
description: Retrieves the length of the instruction pre-amble. The instruction pre-amble is equal to the sum of all the instruction register's lengths of the TAP controllers following the device under test in the JTAG scan chain.
conditions: None
see also: SetInstrPreAmble *{Length {pAmble 0x0}}*

GetInstrPostAmble {{pAmble 0x0}}

parameters: *pAmpble:* Optional. *Reserved for future extensions.*
returns: int Instruction post-amble length as specified with the SetInstrPostAmble procedure.
description: Retrieves the length of the instruction post-amble. The instruction post-amble is equal to the sum of all the instruction register's lengths of the TAP

controllers preceding the device under test in the JTAG scan chain.
conditions: None
see also: SetInstrPostAmble *{Length {pAmble 0x0}}*

GetDataPreAmble {{pAmble 0x0}}
parameters: *pAmpble:* Optional. *Reserved for future extensions.*
returns: int Data pre-amble length as specified with the SetDataPreAmble procedure.
description: Retrieves the length of the data pre-amble. The data pre-amble is equal to the sum of all the instruction register's lengths of the TAP controllers following the device under test in the JTAG scan chain.
conditions: None
see also: SetDataPreAmble *{Length {pAmble 0x0}}*

GetDataPostAmble {{pAmble 0x0}}
parameters: *pAmpble:* Optional. *Reserved for future extensions.*
returns: int Data post-amble length as specified with the SetDataPostAmble procedure.
description: Retrieves the length of the data post-amble. The data post-amble is equal to the sum of all the instruction register's lengths of the TAP controllers preceding the device under test in the JTAG scan chain.
conditions: None
see also: SetDataPostAmble *{Length {pAmble 0x0}}*

3.2.5 Miscellaneous Procedures

SetTCKFreq {Freq}
parameters: *Freq:* Frequency of the TCK clock in kHz.
returns: int Synthesised frequency in kHz.
description: Specifies the frequency of the TCK clock in kHz. This procedure computes the achievable frequency and assigns it to its return value.
conditions: None
see also: GetTCKFreq *{}*

GetTCKFreq { }
parameters: *None*
returns: int Synthesised frequency in kHz.
description: Retrieves the synthesised frequency of the TCK clock in kHz.
conditions: None
see also: SetTCKFreq *{Freq}*

SetContTCK {Cont }
parameters: *Cont:* Clock continuity. Hole clock otherwise if 0, continuous otherwise.
returns:
description: Specifies the clock continuity. A free running clock is generated when a continuous clock is selected. If hole clock mode is selected, a clock is generated only when a data bit or an instruction bit is shifted.
conditions: None
see also:

3.3 Reference Sequences and Scripts

Examples of the TCL scripts are provided with the TCL library. They can be accessed in the *Examples* group located under the *8PI Control Panel* program group created in your start menu during the installation of the application and driver on your computer.

Figure 4: Program group with TCL script examples