Hpe JTAG

Jump-Start Tutorial

Version 2009-29-09

Table of Contents

Table of Figures

1 Introduction

Hpe®_JTAG is a boundary scan tool for prototype board setup and debugging. It can also be used to automatically test limited lot productions. Batch mode is supported. Hpe®_JTAG is not intended for production testing and has certain restrictions in that context. A python scripting interfaces allows extending and automating this tool, even integrating customized GUIs. Gleichmann Electronics Research's' philosophy is not to hinder the user's creativity, so everything you can do via JTAG should be possible with this tool. Since python is an object oriented scripting language one can build and maintain powerful applications.

There is an Hpe®_JTAG simulator, so that users can get acquainted with the tool even without real hardware. The simulator is described under section "Simulator Tutorial" of the user manual. There is also a time restricted demo version of the tool. Section "Restricted Hpe_JTAG demo version" of the user manual describes its limitations. Whenever the Hpe®_JTAG user manual is referenced in this document you can quickly find the appropriate section by copying the section name, opening the user manual with "Help\Open Hpe_desk JTAG documentation" and using Ctrl-f to locate the section name.

If you do not mind reading a few pages this "Jump Start Tutorial" is probably the fastest way to learn about the tools' capabilities. It is intended to give an overview and guide you through the rest of the documentation. Chapter 2 "Getting started" should enable you to use Hpe®_JTAG for interactive PCB debugging. Chapter 3 "Automating and Extending Hpe®_JTAG" will get you started in writing your own python scripts to automate repetitive tasks or writing your own extensions e.g. for flash programming or even creating your own GUIs.

If you prefer a more interactive approach there is also an online tutorial available under "Help\Open JTAG Tutorial". It was created with the Hpe®_desk version of the GUI, so it might not match your GUI 100%, but you will get the idea.

2 Getting started

You can get started with Hpe®_JTAG with just a few mouse clicks and quickly answer questions like

- Are certain devices present in the JTAG chain?
- Are configurations bits of my SRAM FPGA set properly?
- Are clocks toggling?
- Are resets active
- Are buses stuck to certain values?
- Are peripherals on my PCB working as expected (e.g. an LED), if directly stimulated using boundary scan?

You can answer these questions, even if your FPGA or any other IEEE1149.1 compliant component cannot be configured properly, as long as you are able to establish a JTAG communication with these components. In case you are evaluating Hpe®_JTAG: you can work through this section with just the Hpe®_JTAG simulator.

See section "Software installation" of the user manual for details on how to install Hpe®_JTAG and the appropriate device drivers. Connect any supported JTAG cable to the JTAG interface of your target board and a (USB) port of the PC that hosts Hpe®_JTAG. Than power your board up. If necessary select the appropriate JTAG interface under "Settings\Hpe_desk Options\Hardware Settings". For more information please refer to section “Selecting the hardware interface” and “Checking the hardware interface functionality” of the use manual.

For the purpose of explaining the basic functionality of Hpe®_JTAG we will select the “Simulated Interface” for now. The simulator tutorial is explained under section “Simulator Tutorial”. Simply click on “Simulated Circuit Help” to open this section. The simulator tutorial will walk you through the following topics:

- Non-intrusive signal observation using “SAMPLE”
- Testing connections on your board with “EXTEST”
- Testing ICs using “INTEST”

“SAMPLE” and “EXTEST” (together with “BYPASS”) are mandatory instructions for all IEEE1149.1 compliant devices. “INTEST” is an optional instruction, which is supported by Hpe®_JTAG, if supported by the device. For more information please refer to section “Running JTAG instructions” of the user manual.

Click on „Initialize <Interface>“ to establish JTAG communication.

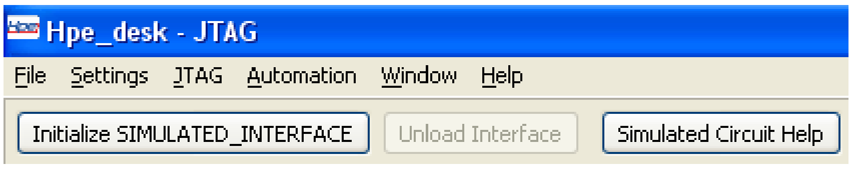
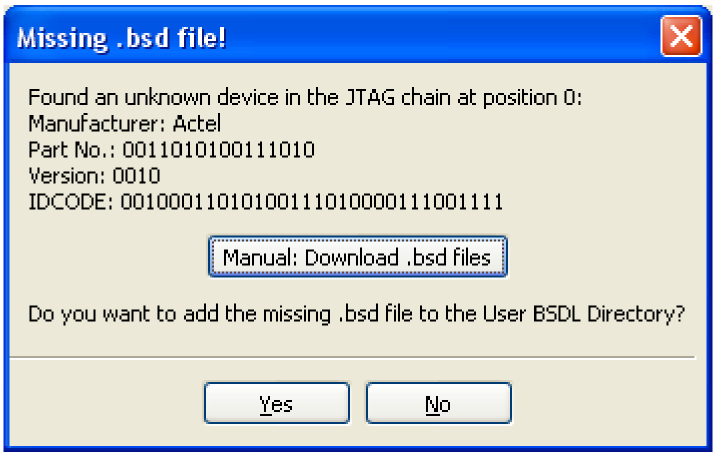

Hpe_desk - JTAG
File Settings JTAG Automation Window Help
Initialize SIMULATED_INTERFACE
Unload Interface
Simulated Circuit Help

Figure 1: Click “Initialize <interface>” to establish JTAG communication

For all devices that are unknown to Hpe®_JTAG, Hpe®_JTAG will ask you to import the corresponding BSDL file.


Missing .bsd file!

Found an unknown device in the JTAG chain at position 0:
Manufacturer: Actel
Part No.: 0011010100111010
Version: 0010
IDCODE: 00100011010100111010000111001111

Manual: Download .bsd files

Do you want to add the missing .bsd file to the User BSDL Directory?

Yes No

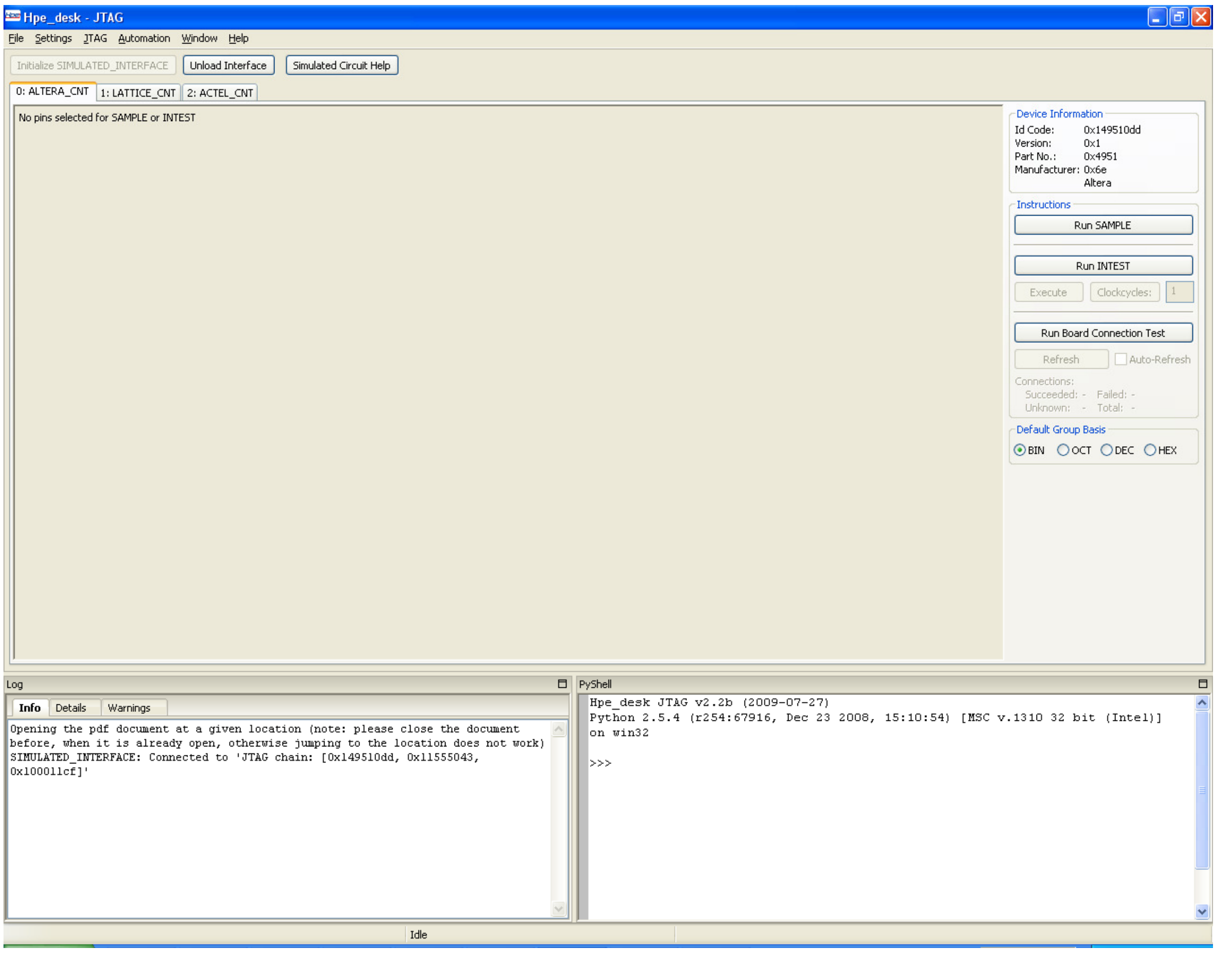
Figure 2: Hpe®_JTAG requires BSDL files to establish JTAG communication

BSDL (Boundary Scan Description Language) is a subset of VHDL (Very High Speed Integrated Circuit Hardware Description Language) that describes how JTAG communication can be established with a device. For a device to be IEEE1149.1 compliant it has to have a BSDL file. In most cases BSDL files can be downloaded from the device vendors' webpage. This is a one time effort for new devices. BSDL file import can be automated using "jtag.bsdlAddFile(file_name)". For more information please refer to section "Downloading BSDL files" and "BSDL file handling" of the user manual.

„Initialize <Interface>" is establishing a JTAG communication and reads out some general device information for each device in the chain, which is displayed in the upper right corner of the GUI under "Device Information". Each device in the chain is represented as a register card with its name and position in the chain being displayed on each tab.

The lower left section of the GUI is used to display messages. The lower right section is a python shell. There is always two ways of doing things in Hpe®_JTAG: either graphically in the GUI or via the python scripting interface. In the python shell users can try out commands and than cut & paste these commands from the shell to a script. This is a very powerful feature for sophisticated users and allows them to automate repetitive tasks and extend the functionality of the tool.

It is our philosophy that only signals relevant to the user should be displayed. Hence nothing is displayed as long as the user has not selected relevant signals, which is why the GUI is mainly grey in the beginning.



No pins selected for SAMPLE or INTEST

Device Information
Id Code: 0x149510dd
Version: 0x1
Part No.: 0x4951
Manufacturer: 0x6e
Altera

Connections:
Succeeded: - Failed: -
Unknown: - Total: -

Default Group Basis: [✓ BIN] OCT DEC HEX

```
Opening the pdf document at a given location (note: please close the document
before, when it is already open, otherwise jumping to the location does not work)
SIMULATED_INTERFACE: Connected to 'JTAG chain: [0x149510dd, 0x11555043,
0x100011cf]'
```

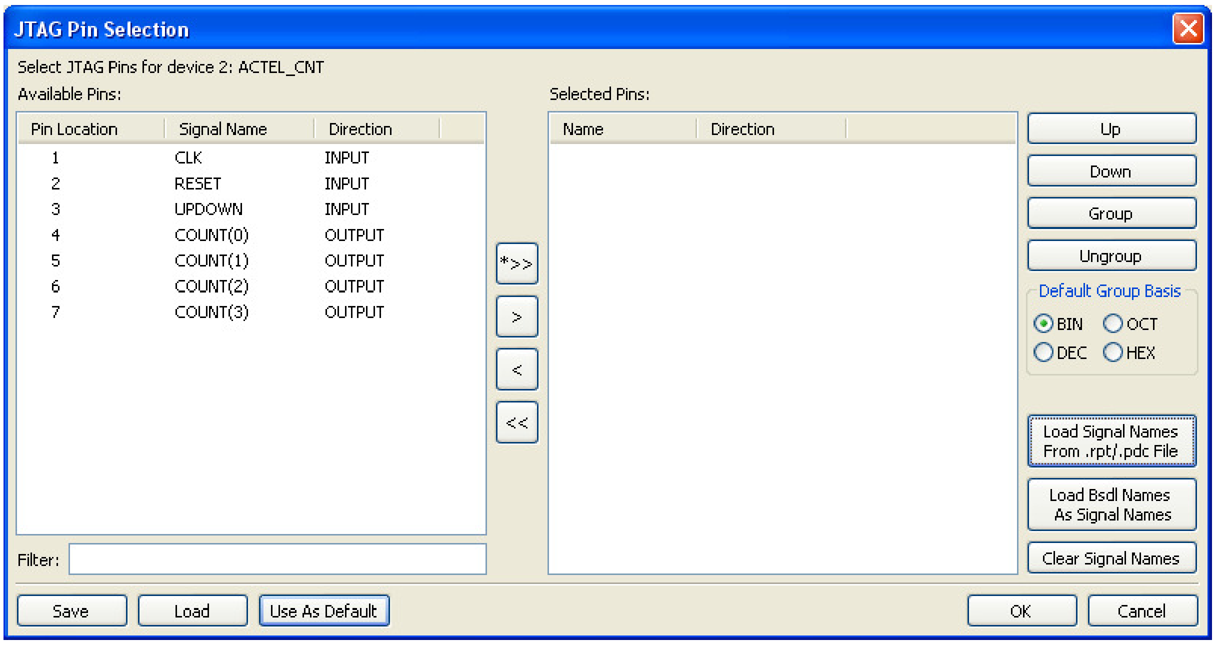
```
Hpe_desk JTAG v2.2b (2009-07-27)
Python 2.5.4 (r254:67916, Dec 23 2008, 15:10:54) [MSC v.1310 32 bit (Intel)]
on win32

>>>
```

Figure 3: Hpe®_JTAG GUI

Observing IOs using "SAMPLE"

Click on "Edit JTAG pin selection" or on "Sample" to enter the pin selection window. Signals can be quickly found by either their pin location or a signal name. Signal names can be defined by the user or loaded from the BSDL-file or a pin-file that has been generated during synthesis. Please note that the suffix of the pin-file that is expected depends on the FPGA vendor. For instance <name>.pin is expected for Altera, <name>.pad file for Lattice and a <name>.rpt or <name>.pdc for Actel devices. The devices, BSDL- and pin-files that are used by the simulator do not correspond to any real devices or files. In the example of the tutorial you will see a window similar to the one below:


JTAG Pin Selection

Select JTAG Pins for device 2: ACTEL_CNT

Available Pins:

Pin Location	Signal Name	Direction
1	CLK	INPUT
2	RESET	INPUT
3	UPDOWN	INPUT
4	COUNT(0)	OUTPUT
5	COUNT(1)	OUTPUT
6	COUNT(2)	OUTPUT
7	COUNT(3)	OUTPUT

Selected Pins:

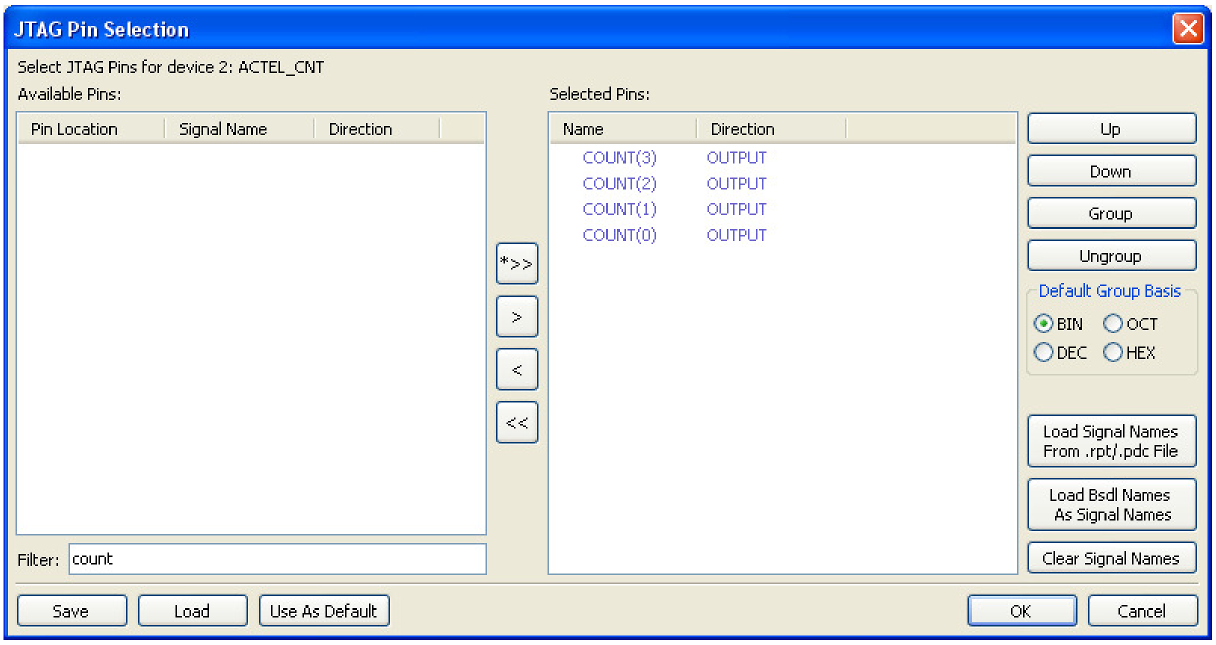
Name	Direction

Default Group Basis: [✓ BIN] OCT DEC HEX

Filter:

Figure 4: Pin selection window

By default buses are grouped. One can use the "Ungroup" button to show individual signals of a bus. In the context of the simulator you probably do not need all this, but imagine that you are trying to debug an FPGA with several hundred user IOs and you are looking for a 64-bit bus that is controlled by an enable-signal, a clock and a reset. All you remember is that you named these signals OE, CLK, RST and DATA[63:0] in your top-level HDL entity, but of course you do not remember each and every pin location of these signals. Normally it would be a tedious task to locate these signals, but with Hpe®_JTAG you will complete this task with a few mouse clicks.



JTAG Pin Selection

Select JTAG Pins for device 2: ACTEL_CNT

Available Pins:

Pin Location	Signal Name	Direction

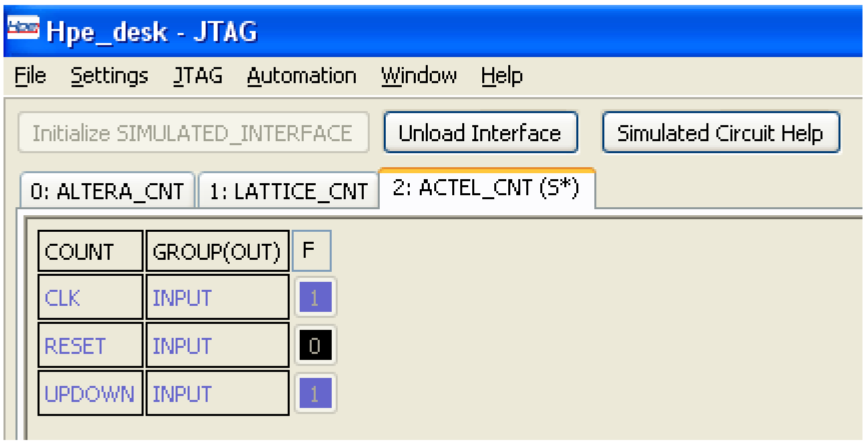
Filter: count

Selected Pins:

Name	Direction
COUNT(3)	OUTPUT
COUNT(2)	OUTPUT
COUNT(1)	OUTPUT
COUNT(0)	OUTPUT

Default Group Basis: [✓ BIN] OCT DEC HEX

Figure 5: Using filters to find signals quickly

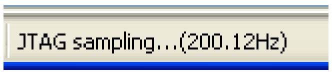


Hpe_desk - JTAG

COUNT	GROUP(OUT)	F
CLK	INPUT	1
RESET	INPUT	0
UPDOWN	INPUT	1

Figure 6: Buses can be displayed grouped or ungrouped

If you observe signals with frequencies of several hundred MHz using boundary scan, it is like taking random screen-shots of these signals. It is in the nature of boundary scan that it is relatively slow, but in most cases you will use it to detect signal activity, observe static signals (like reset vectors or configuration bits) or test the connectivity of your board. At the bottom of the GUI Hpe®_JTAG will show the sampling frequency with which signals are updated. It depends on many factors like the length of the boundary scan register, your JTAG cable and even your PC and its current CPU load. In this example the complete boundary scan register is sampled more than 200 times per second.



JTAG sampling...(200.12Hz)

Figure 7: Hpe®_JTAG: sampling frequency

Controlling IOs using “EXTEST” (Board Connection Test)

Hpe®_JTAG does not require netlists to know about the topology of a board. Instead a proprietary format “board connection description” (.bcd) can be used to tell Hpe®_JTAG how scan cells are connected on the board.

In our example we want to control the four outputs of one of the counters. Imagine that they drive LEDs on the board and you want to turn them on or off. Execute “JTAG\Generate and Load .bcd from JTAG pin selection” to generate a .bcd file. You can toggle the imaginary LEDs by clicking on the “0” or “1” values in the GUI.

.bcd have a simple syntax that is explained in the comment section of each .bcd file. After the comment section the current JTAG pin selection is listed:

```
-- single ended connections for device 0 (ACTEL_CNT):
output "COUNT(3)" 0, 7; --  (COUNT(3))
output "COUNT(2)" 0, 6; --  (COUNT(2))
output "COUNT(1)" 0, 5; --  (COUNT(1))
output "COUNT(0)" 0, 4; --  (COUNT(0))
```

Listing 1: Board Connection Description (BCD) file

.bcd files can be generated as follows:

- Automatically from the current JTAG pin selection
- Graphically
- With conversion scripts from a netlist
- With a text-editor

For more information on the BCD format please refer to section “Board Connection Description” of the user manual.

Now work through the rest of the simulator tutorial described in section “Simulator Tutorial” of the user manual. Upon completion you will know all you need to know to interactively debug boards using “SAMPLE” or “EXTEST” and even ICs using “INTEST”.

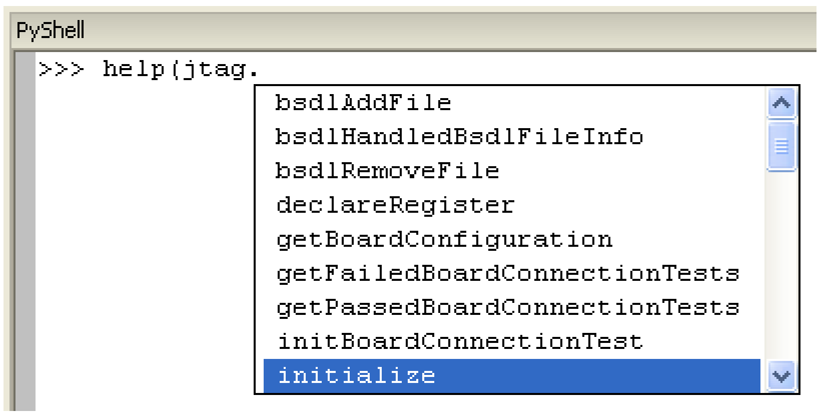
3 Automating and Extending Hpe®_JTAG

As mentioned earlier, there is always two ways of doing things in Hpe®_JTAG: either graphically in the GUI or through the python scripting interface. If you are new to python a good way of getting started is simply by typing “help” in the python shell. This will point you to “Help\Python Quick start” in the Help menu for a list of recommended reading on this powerful scripting language.

```
>>> help
Type help() for interactive help, or help(object) for help about object.
See menu Help -> Python Quickstart for an introduction about Hpe_desk python
scripting!
```

Figure 8: Getting started with python

The Hpe®_JTAG API is well documented under “Help\Browse Scripting API Documentation”. A list of JTAG related commands can also be obtained by just typing “jtag.” in the python shell. If you want to read the command line help of these commands just type “help(jtag.” This will give you a list of JTAG related commands. For instance instead of clicking on “Initialize <Interface>” you can use “jtag.initialize” and get its command line help by typing “help(jtag.” and double clicking “initialize” in the list of options.



```
>>> help(jtag.
                bsdlAddFile
                bsdlHandledBsdlFileInfo
                bsdlRemoveFile
                declareRegister
                getBoardConfiguration
                getFailedBoardConnectionTests
                getPassedBoardConnectionTests
                initBoardConnectionTest
                initialize
```

Figure 9: There is a command help integrated in the python shell

Type “jtag.initialize()” to use the default values. You will get the same result as you got when clicking the “Initialize <Interface>“ button.

Now cut & paste “jtag.initialize()” to your favourite text editor and save it as “test.py”. You can now execute it under “Automation\Execute Python Script …”. In order to make the effects of any command or chain of commands visible in the GUI you need to execute “jtag.updateGUI()”. Additional output can be displayed in the Info window by using “util.Msg()” or “util.MsgImportant()”.

You can use “jtag.initialize() to make sure that certain devices are present in the JTAG chain. Below is an example on how to perform a simple infrastructure test.

```
util.Msg("\nAttempting to initialize JTAG communication")
jtag.initialize("DIGILENT_INTERFACE", "LFEC2_50E_XXF672")
util.MsgImportant("\nCompleted JTAG initialization")
jtag.updateGUI()
```

Listing 2: Simple infrastructures test

The user manual dedicates a whole chapter to the python scripting interface. Please refer to chapter “Scripting” of the user manual, if you want to get the most out of this powerful feature.

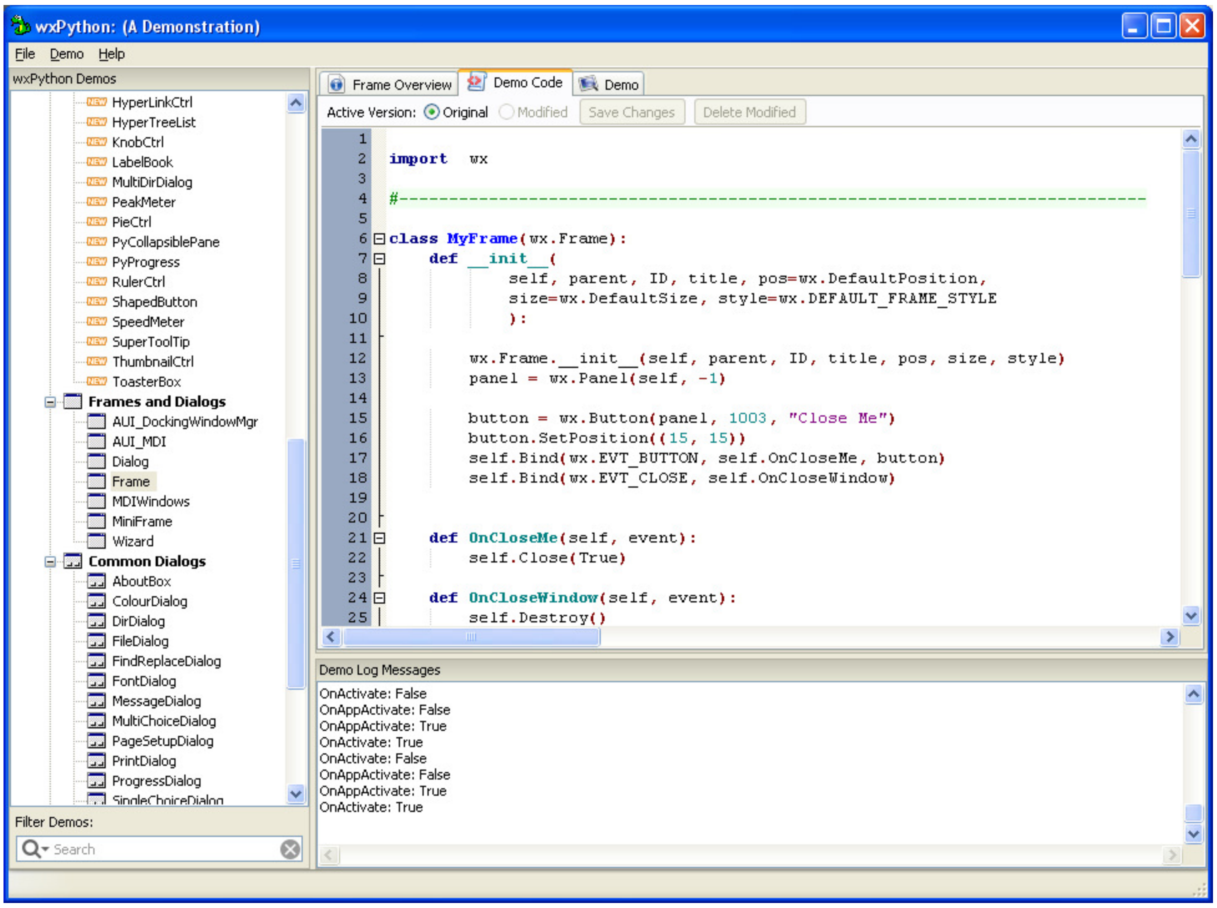
Many example scripts can be found in your installation directory under “.\scripts\JTAG”. They are explained in section “Example Scripts”, e.g. section “Board Connection Test Script” shows how to run a connection test on the board simulated by the Hpe®_JTAG simulator.

The subdirectory “.\scripts\JTAG\JtagExampleLib” contains more complex examples to generate waveforms for flash programming or bus interfaces such as I2C or SPI. These application examples are explained under “Help\Application Notes”.

If boundary scan with Hpe®_JTAG is just part of what you are trying to automate you may want to run Hpe®_JTAG in batch mode as described in section “Batch processing” of the user manual.

Creating your own GUIs

seIt is easy to create you own GUIs using wxPython. In the user manual see section “Introduction to GUI scripting” to get started. Then start the wxPython Demo and go to “Frames and Dialogs\Frame”. Click on “Demo” to see what your GUI will be looking like and click on “Demo code” to see the corresponding code examples. The example from the user manual is more or less just cutting and pasting from the wxpython demo with a few modifications.



```

import  wx

#---------------------------------------------------------------------------

class MyFrame(wx.Frame):
    def __init__(
            self, parent, ID, title, pos=wx.DefaultPosition,
            size=wx.DefaultSize, style=wx.DEFAULT_FRAME_STYLE
            ):

        wx.Frame.__init__(self, parent, ID, title, pos, size, style)
        panel = wx.Panel(self, -1)

        button = wx.Button(panel, 1003, "Close Me")
        button.SetPosition((15, 15))
        self.Bind(wx.EVT_BUTTON, self.OnCloseMe, button)
        self.Bind(wx.EVT_CLOSE, self.OnCloseWindow)


    def OnCloseMe(self, event):
        self.Close(True)

    def OnCloseWindow(self, event):
        self.Destroy()
```

```
OnActivate: False
OnAppActivate: False
OnAppActivate: True
OnActivate: True
OnActivate: False
OnAppActivate: False
OnAppActivate: True
OnActivate: True
```

Figure 10: GUIs can be quickly build by cutting and pasting from the wxPython demo

Implementing low-level JTAG accesses

With the python scripting interface you can declare, read from or write to any JTAG register. For instance open the BSDL file of the first device in your JTAG chain and search for “IDCODE”. You will find the section that defines the JTAG instructions for this device, which might look like this:

```
      attribute INSTRUCTION_LENGTH of <device> : entity is    8;
      attribute INSTRUCTION_OPCODE of <device> : entity is
                  "BYPASS (11111111), "&
                  "IDCODE (00001111), "&
                  "EXTEST (00000000), "&
                  "SAMPLE (00000001), "&
      etc.
```

Listing 3 Relevant section in a BSDL-file

In this example one could read out the device ID (see upper right corner after “tag.initialize()”) by executing:

```
      jtag.declareRegister(0,15,32)
      util.Msg("\nDevice ID: %s" % (hex(jtag.readRegister(0,15))))
      jtag.updateGUI
```

Listing 4 Reading out the ID register with low level JTAG commands

This example demonstrates a very powerful feature very clearly: with Hpe®_JTAG you can do anything that is accessible via JTAG. These commands might be very useful, if you are implementing your own JTAG controller and JTAG registers or if you have deep knowledge about the Silicon you are trying to test or debug. For instance you could use these commands to read out the configuration register of the A/D converters of Actel Fusion FPGAs.