Electgon
A Circuits' Particle

JTAG System

With OpenOCD Explanation

ELECTGON
www.electgon.com
contact@electgon.com

07.06.2019

Contents

Abstract

Usually data communication interfaces are classified to be serial or parallel. Talking about JTAG interface, we can mention briefly that data is transferred in JTAG serially. However, JTAG is explained better as a complete system with four-wire interface, control logic, instructions register, data registers and even programming and description languages. That is what we are going to talk about in this document.

1 Introduction

Previously, boards and PCB connections were tested using In-Circuit testing techniques. With the advance of complexity and compacted sizes of IC chips, it became more complex to test PCB connections. This led to find JTAG technique for testing the board and its populated chips. JTAG (Joint Test Action Group) and pronounced as ("J" "Tag"), came out in 1980 to test chips using the boundary scan testing. It turned out to be IEEE standard in 1990 known as IEEE-1149.1 which later was extended to include other specifications like BSDL, Programmable Devices Configuration, etc.

2 Simplified Description

JTAG Interface represents a communication technology. Rather than interfacing between two devices only, it can connect series of devices in one chain. Although JTAG interface has four wires, it depends also on some associated registers that should be added in devices compliant with JTAG technology. Since JTAG system consists of four-wires interface and configuration registers, silicon manufacturers started to use these elements (interface and registers) for extended purposes like programming and debugging. Device that are compliant with JTAG standard has four-wire interface connected to a logic called TAP (Test Access Port). Thus, a compliant device will have beside its core function a TAP logic and Boundary Scan Cells as shown in figure 1. JTAG registers are contained in cells at the boundary of the device. These cells are called then boundary scan cells. There are two modes of operation for these cells. First mode is called functional mode in which boundary scan cells have no effect and the device is operating normally according to its functionality. The other mode is called test mode in which functional core of the device is disconnected from the pins so that JTAG system can monitor and control internal nets of the device. In test mode, boundary scan cells are used to test connectivity of the device pins (check whether there are short circuit, open circuit, wrong logic, etc). And it can be used also in this mode to communicate with non-JTAG peripherals such as DDR RAM and flash.

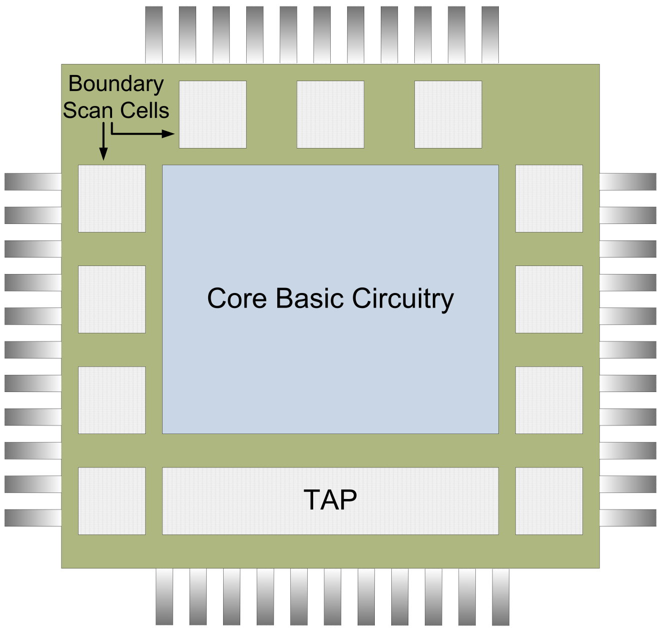


Figure 1: JTAG Compliant Architecture

For each pin of the JTAG compliant device, there will be one Boundary Scan Cell (BSC). Register inside this BSC is simply shift register that is connected with other shift registers in other BSCs as shown in figure 2.

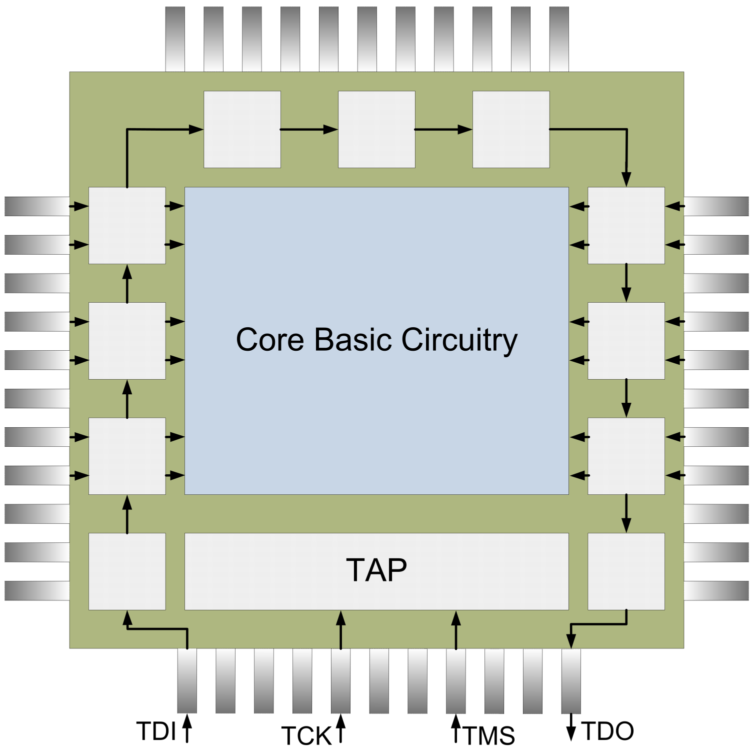


Figure 2: JTAG Pins Flow

Serial data can be provided to the first BSC and in turn it shifts it to the next BSC and so on. Last BSC will send back its output bit to the external testing machine which in turn is waiting to receive back the pattern it has sent to decide if all connections are ok or not. If for instance one pin is connected mistakenly to ground, this will lead to receiving back zeros. In that way, JTAG can be used also to test series of devices as long they are connected in one chain as shown in figure 3.

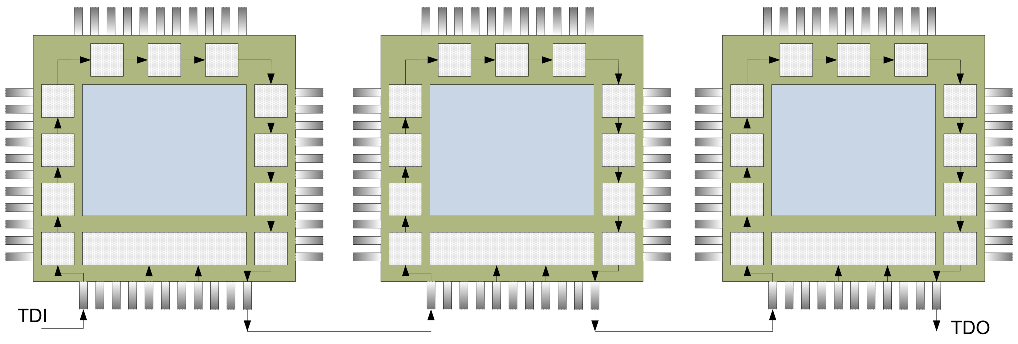


Figure 3: JTAG Chain

3 Detailed Description

Now we had overview about JTAG system. More detailed explanation may make it clearer.

3.1 Interface Pins

JTAG Interface has mainly four pins (TCK - TMS – TDI – TDO).

TCK:	Test Clock. Since JTAG is passing data serially, transmission clock is sent through this pin so that the receiving TAP can synchronize its operation with it. Limitation on the clock speed is subjected to device under debug and the debugging system.
TMS:	Test Mode Select. This Signal is used to control the TAP logic of receiving and shifting input data. It is synchronous with TCK and captured at the rising edge.
TDI:	Test Data Input. This is the data transmitted from the testing machine. It is captured at the Rising edge of TCK.
TDO:	Test Data Output. This is data shifted out from the Device under debug. It is captured at the Falling edge of TCK
TRST:	Test Reset. This signal is optional to include in the interface. It is asynchronous reset signal to reset the TAP logic. This reset is active low.

In most of the context explaining JTAG, they mention TRST as an optional reset signal. This is of course accurate but as mentioned it is resetting the TAP logic. There is also optional signal defined in IEEE 1149.1 called SRST.

SRST:	System Reset. This reset signal is resetting the whole system being debugged not only the TAP logic.

Apart from optional standard signals, There is also non-standard optional signal called return clock.

RTCK: Return Test Clock. This Signal is input signal to the debugging machine (output from device being debugged). Since the TAP clock (TCK) is usually different than Core clock, it might be needed from some debugging purposes to synchronize JTAG clock (TCK) with the core clock. So this clock is returned back from device under debug to the debugging machine in order to adapt its frequency (Adaptive clocking).

3.2 TAP contents

As defined in IEEE 1149 standard, the TAP unit shall have TAP controller, one Instruction Register and three or more Data registers. This is because JTAG is intended mainly to pass pins values. Pins values are captured in Data Registers as well as other information (will see shortly). Instruction Register contains instruction to decide which Data Register should pass its value. TAP control is organizing steps needed to pass data to/from Instruction or Data Registers. Figure 4 shows more detailed view of the JTAG system.

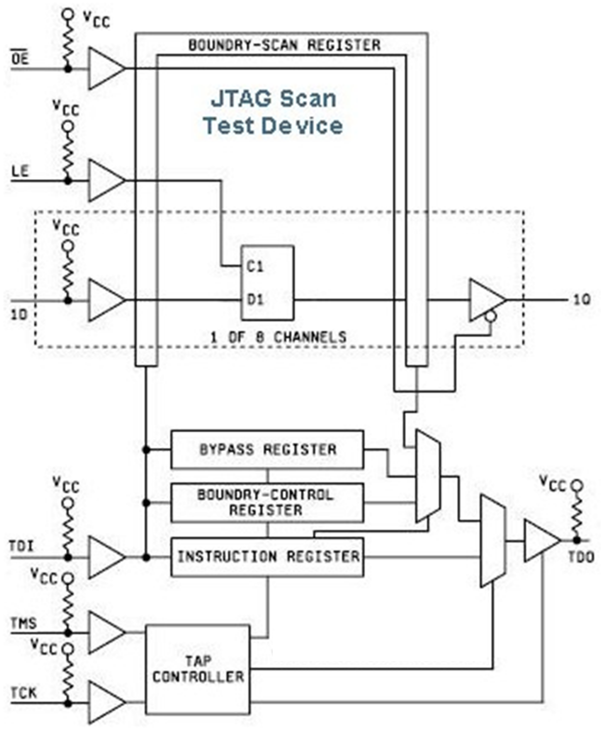


Figure 4: Detailed JTAG System[1]

3.3 TAP Controller

It is a state machine that is controlled by TMS wire. This state machine is shown in figure 5.

What can be understood then from this state machine that JTAG operation starts at Reset state then starts running the test and selecting either Instruction Register or Data Register to pass or capture data.

What is important to note here also is whatever in which state the TAP controller is, having five successive '1' in TMS, this will bring the TAP controller into the Reset state.

For a chain of JTAG devices, TAP controllers of all these devices will be synchronized in the same state. That is why operation of the JTAG starts by setting TMS to five successive '1' to ensure that all TAP Controllers will be synchronized.

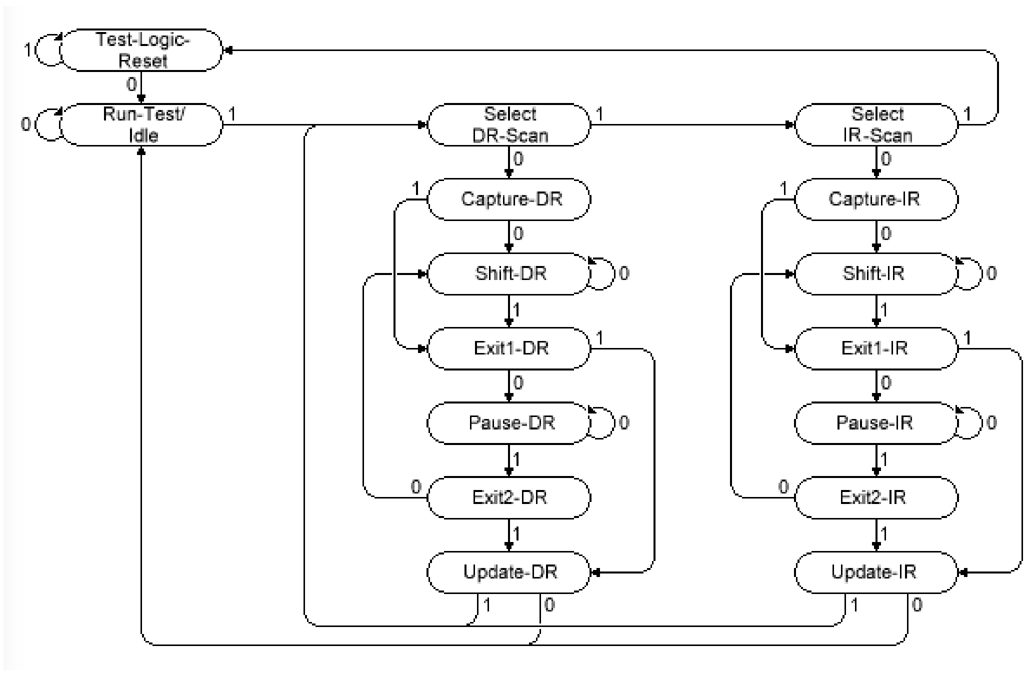


Figure 5: TAP State Machine

For some device, it can be designed to have multiple TAP. This is valid for some processor chips that have multiple cores and also some FPGA devices.

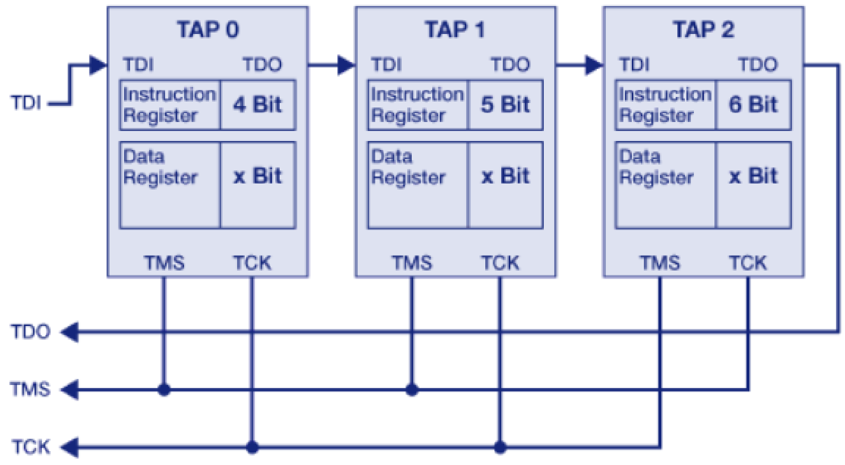


Figure 6: Multiple TAP Controller[2]

3.4 Instruction Register

As mentioned briefly previously, Instruction register holds an instruction that is going to be executed instantly. This instruction register has a specific length. There is no standard or constraints on the length of this register but there are some standard instructions that should be exist in a standard compliant JTAG system: bypass – extest – sample/reload.

When instruction register is set entirely to ones, this means Bypass instruction needed to be executed. Extest instruction has all its bit pattern zeros. No bit pattern is standardized for instruction sample/reload, it is defined according to the manufacturer.

Other instructions may be available according to the manufacturer. A common non standard command is the IDCODE command which used to direct the IC to reply with the ID of the IC chip.

3.5 Data Registers

JTAG system should have at least three data registers: Bypass register, Boundary Scan register, Device ID register. It is up to the manufacturer to add more data registers according to his requirements.

Bypass register is a single bit register which is used to bypass data directly from TDI to TDO. It is important when we have a chain of device in the JTAG loop so that we can bypass certain device by sending instruction 'Bypass' which in turn will select Bypass register. Boundary Scan register is the main register that is combined from shift register of each attached pin. Device ID register is a register that contains ID of the device.

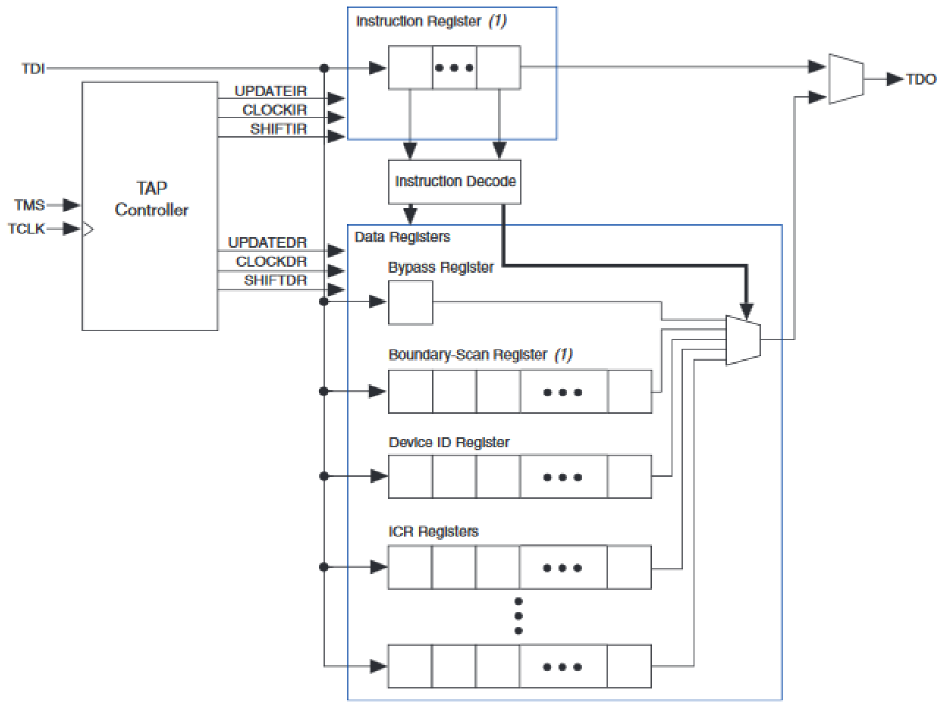


Figure 7: Data Registers Multiplexing

Boundary Scan Cell

This cell is attached to every I/O pin included in the JTAG system. The purpose of this cell is either to shift provided data at TDI pin, or to capture data from its associated I/O pin. Boundary Scan Cell is shown in figure 8.

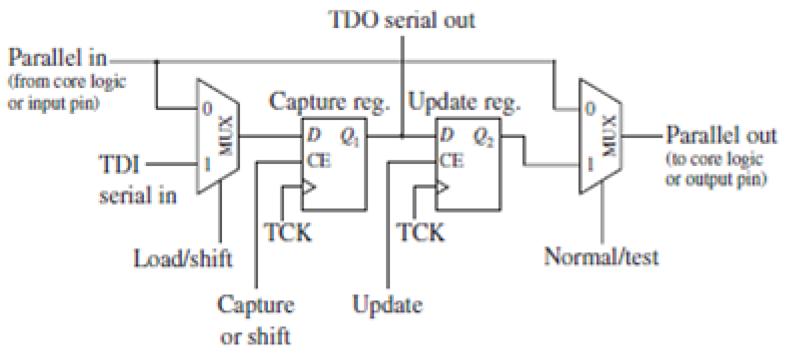


Figure 8: Boundary Scan Cell[3]

Keeping in mind that if the Pin of the IC is Input pin, the In signal is connected to input pin of the IC and the out signal is connected to core logic. Recursively, of the cell is attached to output pin of the IC, the In signal will be connected to the core logic and the Out signal will be connected to the output pin of the IC.

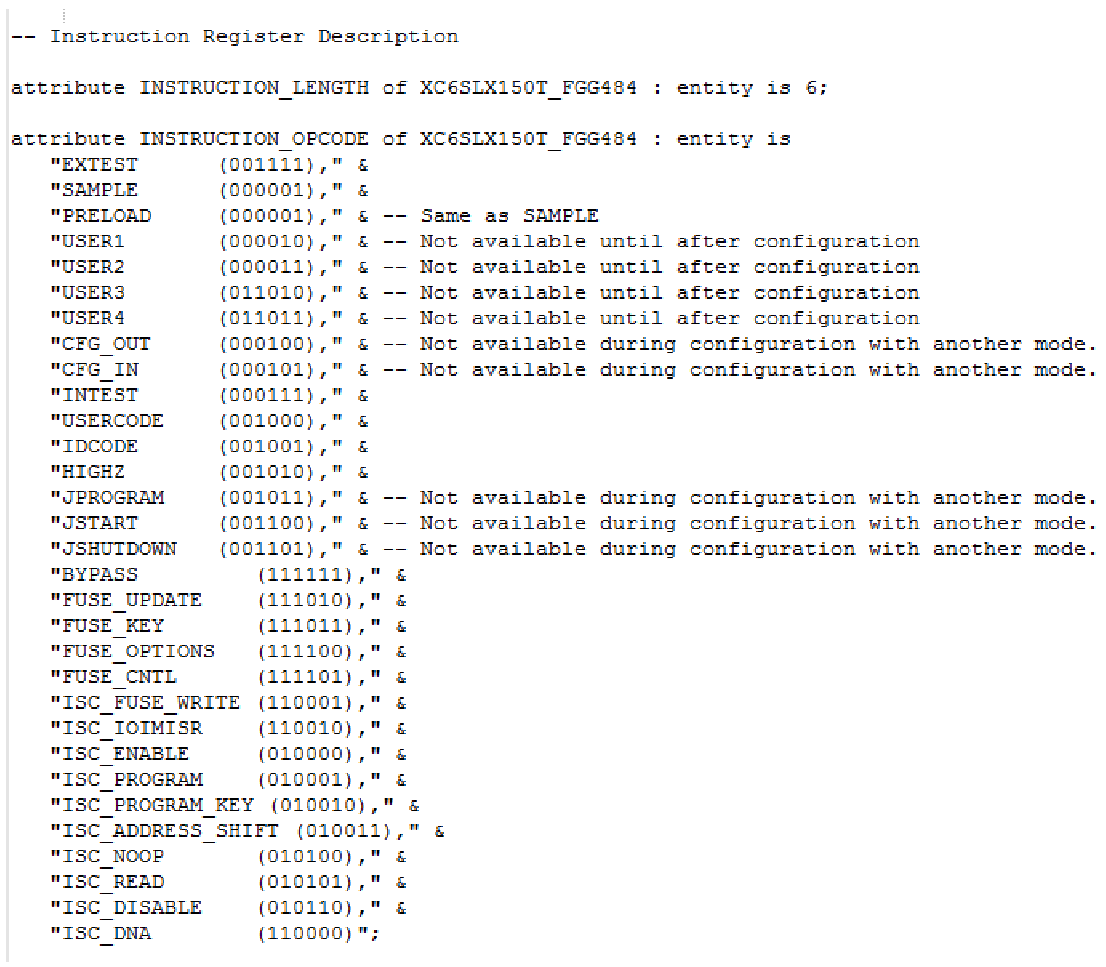
This cell will have two main modes; Normal mode in which BSC is transparent and the core logic function is operating with direct connection to the chip pins. Second mode is Test mode, in which boundary scan cell is acting according to TAP controller instructions. These instructions are capture (or load), shift, update.

Capture (or load) means to get data from TDI signal (or from I/O pin) to the next BSC. Shift means to present the captured data to the TDO signal. Update means to make this data available also to the Out signal. Note that this Out signal is connected either to the core logic or I/O pins. This helps to feed the Out signal with data provided in TDI so that we can drive the core logic (or I/O pis) with some values.

4 Boundary Scan Description Language

Now we have seen basic architecture of a JTAG system, next question could be how to interact with a JTAG of a chip. For instance a debugging system will send certain bit pattern through TDI pin, how can we define the expected return pattern through TDO pin. There must be a description then of the JTAG system of Device under debugging. Boundary Scan Description Language (BSDL) is a subset of VHDL that is used describe the JTAG system. JTAG compliant devices shall provide BSDL file which describes how the JTAG system of the device work.

For example Xilinx provides BSDL files for its chips. The following is an example of an BSDL file.



```
-- Instruction Register Description

attribute INSTRUCTION_LENGTH of XC6SLX150T_FGG484 : entity is 6;

attribute INSTRUCTION_OPCODE of XC6SLX150T_FGG484 : entity is
   "EXTEST        (001111)," &
   "SAMPLE        (000001)," &
   "PRELOAD       (000001)," & -- Same as SAMPLE
   "USER1         (000010)," & -- Not available until after configuration
   "USER2         (000011)," & -- Not available until after configuration
   "USER3         (011010)," & -- Not available until after configuration
   "USER4         (011011)," & -- Not available until after configuration
   "CFG_OUT       (000100)," & -- Not available during configuration with another mode.
   "CFG_IN        (000101)," & -- Not available during configuration with another mode.
   "INTEST        (000111)," &
   "USERCODE      (001000)," &
   "IDCODE        (001001)," &
   "HIGHZ         (001010)," &
   "JPROGRAM      (001011)," & -- Not available during configuration with another mode.
   "JSTART        (001100)," & -- Not available during configuration with another mode.
   "JSHUTDOWN     (001101)," & -- Not available during configuration with another mode.
   "BYPASS           (111111)," &
   "FUSE_UPDATE      (111010)," &
   "FUSE_KEY         (111011)," &
   "FUSE_OPTIONS     (111100)," &
   "FUSE_CNTL        (111101)," &
   "ISC_FUSE_WRITE   (110001)," &
   "ISC_IOIMISR      (110010)," &
   "ISC_ENABLE       (010000)," &
   "ISC_PROGRAM      (010001)," &
   "ISC_PROGRAM_KEY  (010010)," &
   "ISC_ADDRESS_SHIFT  (010011)," &
   "ISC_NOOP         (010100)," &
   "ISC_READ         (010101)," &
   "ISC_DISABLE      (010110)," &
   "ISC_DNA          (110000)";
```

Figure 9: BSDL File Sample

From this file we can know the length of instruction register (which is 6) also we can know which commands are supported in the available JTAG system. Using this information we can build our debugging environment. BSDL files are free and can be downloaded from any vendor resources portals.

JTAG Connectors

There is no standard for the JTAG connector used to connect debugging system and device under debugging. This means each manufacturer can configure the JTAG interface according to his needs. Below are some samples of JTAG connector for some ARM processor.

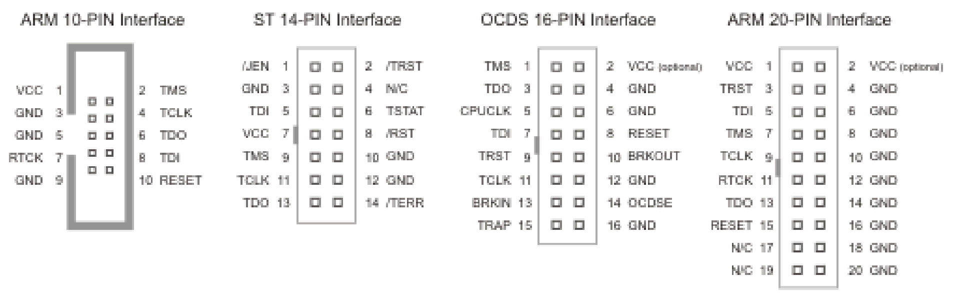


Figure 10: JTAG Connectors[4]

5 SVF Files

Previous discussion illustrated JTAG system architecture and components. Remaining now to understand how to configure JTAG system components (Instruction register, Data registers). A very simple language can be used for that; SVF (Serial Vector Format) which has few instructions that are used to either to write or read to/from JTAG registers. Specification of this language is available online and can be downloaded which declares SVF commands and how it can be used. Table 1 depicts SVF commands.

SVF Commands		
ENDDR	PIO	STATE
ENDIR	PIOMAP	TDR
FREQUENCY	RUNTEST	TIR
HDR	SDR	TRST
HIR	SIR	

Table 1: SVF Commands

For quick explanation of these commands, we have mainly instruction register and data register in the JTAG system. To write to instruction register we use SIR command, To write to data register we use SDR. The same commands can be used to read from these registers as writing to any of them is a process of shifting serial bit train in a chain. which mean when injecting new data, old data will be shifted out to TDO then we can read it.

In case of chain of JTAG system, you may need to insert a header or trailer to your bit sequence so that it can allocated in the next or previous device in the chain. For that purpose, there are HDR, HIR, TDR, TIR commands.

You can tell TAP to which state it shall go after it writes to the register. This can be instructed using ENDDR and ENDIR commands. STATE command is used to go immediately to the indicated state. FREQUENCY command is used to defined which clock frequency the TCK shall run.

For example, the following two line are used to instruct Instruction Register to be ready for writing data into a register of address E0. Next, we shift over TDI a value of ’FCAF9671’ that will be written into register of address E0 and we expect according to this writing procedure TDO pin will shift back to the user the value ’4408501’.

```
SIR 8 TDI (E0);
SDR 32 TDI (FCAF9671) TDO (44008501);
```

6 OpenOCD SVF Player

In order to provide configurations described in SVF file into the device under test, software is needed to play or interpret these SVF commands into the right synchronized hardware signals. Many software applications can be found but one of the most common applications is OpenOCD.

Many JTAG vendors or interfaces are supported in openOCD which makes it feasible to be used. Not only but also it is open source tool and support different platform.

To install openOCD you can use simply

```
$ sudo apt-get install openocd
```

But this will install openocd with default configurations.

It may be needed then to install openocd manually to customize or control with which configuration openOCD shall be installed. So to install it manually download source of openOCD

```
$ git clone git://git.code.sf.net/p/openocd/code openocd
```

If you have problems with this link, you can try another mirror link

```
$ git clone http://repo.or.cz/r/openocd.git openocd
```

After downloading install it using the following commands

```
$ cd openocd
$ ./bootstrap
$ ./configure --enable-sysfsgpio <<here you can enable target interfaces>>
$ make
$ sudo make install
```

6.1 OpenOCD Structure

It is basically built to be a debugging mechanism for embedded systems. User has to connect his device under test into a machine that is running openOCD over JTAG interface as shown in figure 11.

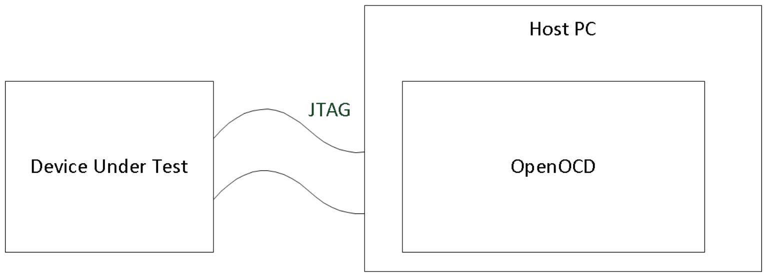


Figure 11: Simple System With OpenOCD

But since a JTAG interface is not usual to be found in a PC, a kind of adapter (or dongle) is needed to provide that interface. This Adapter is connected to the host PC via USB (for example) and connected to the device under test via JTAG interface.

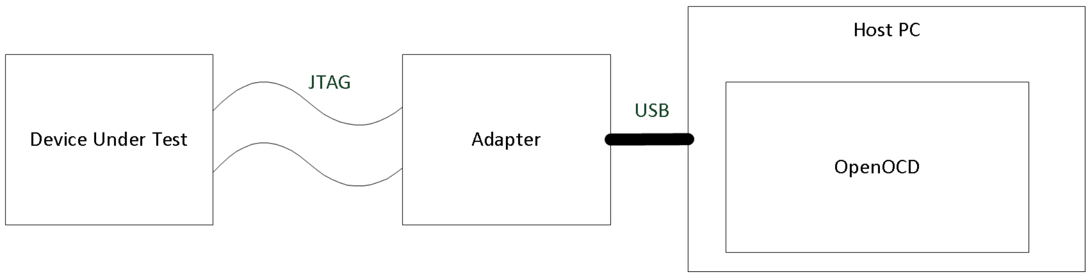


Figure 12: Adapter for OpenOCD

There are so many adapters in the market that provide this facility. That is why choosing a supporting software is important.

We will find then in openOCD different supported adapters and for each adapter. There is what is called configuration file which handles signaling and pin connections of the adapter and also its appropriate driver. For example , openOCD can be run currently on Raspberry Pi using its GPIO. This means that we can connect Raspberry Pi board directly to a device under test through GPIO pins without the need of any Adapter. In this case we need to tell openOCD which pins of the GPIO is used as a JTAG pins (TCK, TMS, TDI, TDO). Also a driver for this GPIO interface is needed to handle how TCK, TMS, TDI, etc will be synchronized and transfer logic values after it has been interpreted by openOCD.

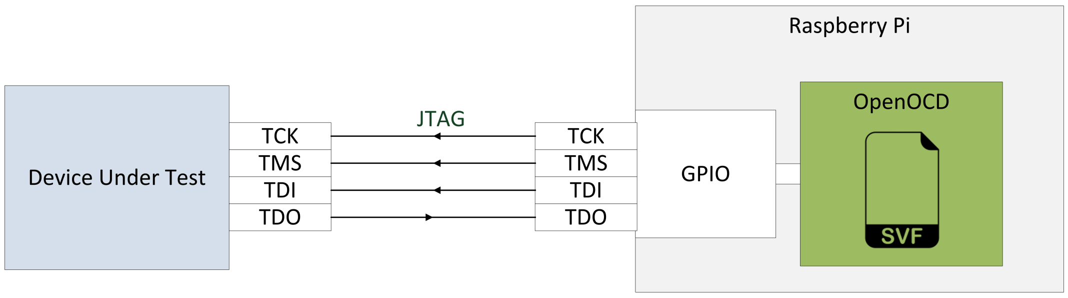


Figure 13: OpenOCD in Raspberry Pi

The main structure of openOCD may be represented as in figure

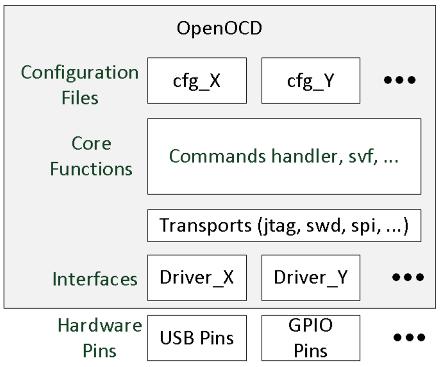


Figure 14: OpenOCD Structure

What is important to note about in this structure is the configuration files in which the user can provide some settings that will direct the drivers (i.e the interface) . For example user can mention in the configuration file to use jtag as a transport protocol over USB interface with 1 KHz speed.

6.2 Working with OpenOCD

To run OpenOCD (if its binary executable is already located in bin directory) simply use

```
$ openocd
```

But this will do nothing since we have to tell openOCD what to do. So openOCD shall be run in this format

```
$ openocd -c <command_name>
```

Here we have used the switch -c to inform openOCD to run a command. If we have multiple commands to run we can include it all in a file then inform openOCD to run a file.

```
$ openocd -f <file_name>
```

For example, after installing openOCD, you may want to know which interfaces are supported in the current installation. This can be known by running command interface_list

```
$ openocd -c interface_list
```

OpenOCD has several commands that best to be checked in their online manual. Most important commands are

Interface: in which we have to state which driver is going to be used. Last example has shown list of available interfaces. User then has to mention which interface to be used. For instance

```
$ openocd -c “interface ”openjtag
```

Init: which starts a debugging session. Without this command nothing can be meaningful.

```
$ openocd -c init
```

6.3 OpenOCD Configuration Files

As just mentioned, user can consolidate his commands into one file. The following can be a sample of a valid file

```
## OpenOCD Sample file
## that contains set of commands
interface sysfsgpio
echo “Starting OpenOCD …”.
init
debug_level -1
```

What has to be clarified now is openOCD understands tcl language which means in addition to predefined openOCD commands, user can provide his file with other tcl commands to perform further operations.

```
## OpenOCD configuration file
puts “Starting OpenOCD …”.
source [find interface/UserFile.cfg ]
exit
```

The same applies for ready configuration files in openOCD. This means that user can run configuration files directly if it is ready for performing right operations.

```
$ openocd -f <path/to/cfg/file>
```

6.4 Running SVF Files

OpenOCD is able to play SVF files. There is one command in openOCD that can be used for that.

```
$ openocd –c “svf path/to/svf/”file
```

Bibliography

[1] http://www.interfacebus.com

[2] http://www2.lauterbach.com/pdf/training_jtag.pdf

[3] Digital Systems Design Using Verilog, 1st Edition. Byeong Kil Lee, Lizy K John, Charles Roth.

[4] http://infocenter.arm.com