Open On-Chip Debugger (OpenOCD)

Edition 1.0 for OpenOCD version 1.0
4 April 2009

Short Contents

Table of Contents

OpenOCD

This manual documents edition 1.0 of the Open On-Chip Debugger (OpenOCD) version 1.0, 4 April 2009.

About

The Open On-Chip Debugger (OpenOCD) aims to provide debugging, in-system programming and boundary-scan testing for embedded target devices.

JTAG: OpenOCD uses a "hardware interface dongle" to communicate with the JTAG (IEEE 1149.1) compliant taps on your target board.

Dongles: OpenOCD currently supports many types of hardware dongles: USB based, parallel port based, and other standalone boxes that run OpenOCD internally. See the section titled: See Chapter 3 [JTAG Hardware Dongles], page 8.

GDB Debug: It allows ARM7 (ARM7TDMI and ARM720t), ARM9 (ARM920T, ARM922T, ARM926EJ–S, ARM966E–S), XScale (PXA25x, IXP42x) and Cortex-M3 (Luminary Stellaris LM3 and ST STM32) based cores to be debugged via the GDB protocol.

Flash Programing: Flash writing is supported for external CFI compatible flashes (Intel and AMD/Spansion command set) and several internal flashes (LPC2000, AT91SAM7, STR7x, STR9x, LM3, and STM32x). Preliminary support for using the LPC3180's NAND flash controller is included.

1 Developers

OpenOCD was created by Dominic Rath as part of a diploma thesis written at the University of Applied Sciences Augsburg (http://www.fh-augsburg.de). Others interested in improving the state of free and open debug and testing technology are welcome to participate.

Other developers have contributed support for additional targets and flashes as well as numerous bugfixes and enhancements. See the AUTHORS file for regular contributors.

The main OpenOCD web site is available at http://openocd.berlios.de/web/.

1.1 Coding Style

The following rules try to describe formatting and naming conventions that should be followed to make the whole OpenOCD code look more consistent. The ultimate goal of coding style should be readability, and these rules may be ignored for a particular (small) piece of code if that makes it more readable.

1.1.1 Formatting rules:

- remove any trailing white space
- use TAB characters for indentation, not spaces
- displayed TAB width is 4 characters
- make sure NOT to use DOS '\r\n' line feeds
- do not add more than 2 empty lines to source files
- do not add trailing empty lines to source files
- do not use C++ style comments (//)
- lines may be reasonably wide - there's no anachronistic 80 characters limit

1.1.2 Naming rules:

- identifiers use lower-case letters only
- identifiers consisting of multiple words use underline characters between consecutive words
- macros use upper-case letters only
- structure names shall be appended with '_s'
- typedefs shall be appended with '_t'

1.1.3 Function calls:

- function calls have no space between the functions name and the parameter list: my_func(param1, param2, ...)

2 Building

2.1 Pre-Built Tools

If you are interested in getting actual work done rather than building OpenOCD, then check if your interface supplier provides binaries for you. Chances are that that binary is from some SVN version that is more stable than SVN trunk where bleeding edge development takes place.

2.2 Packagers Please Read!

You are a **PACKAGER** of OpenOCD if you

1. **Sell dongles** and include pre-built binaries
2. **Supply tools** i.e.: A complete development solution
3. **Supply IDEs** like Eclipse, or RHIDE, etc.
4. **Build packages** i.e.: RPM files, or DEB files for a Linux Distro

As a **PACKAGER** - you are at the top of the food chain. You solve problems for downstream users. What you fix or solve - solves hundreds if not thousands of user questions. If something does not work for you please let us know. That said, would also like you to follow a few suggestions:

1. **Always build with printer ports enabled.**
2. **Try to use LIBFTDI + LIBUSB where possible. You cover more bases.**

- **Why YES to LIBFTDI + LIBUSB?**
  - **LESS** work - libusb perhaps already there
  - **LESS** work - identical code, multiple platforms
  - **MORE** dongles are supported
  - **MORE** platforms are supported
  - **MORE** complete solution
- **Why not LIBFTDI + LIBUSB** (i.e.: ftd2xx instead)?
  - **LESS** speed - some say it is slower
  - **LESS** complex to distribute (external dependencies)

2.3 Building From Source

You can download the current SVN version with an SVN client of your choice from the following repositories:

`svn://svn.berlios.de/openocd/trunk`

or

`http://svn.berlios.de/svnroot/repos/openocd/trunk`

Using the SVN command line client, you can use the following command to fetch the latest version (make sure there is no (non-svn) directory called "openocd" in the current directory):

```
svn checkout svn://svn.berlios.de/openocd/trunk openocd
```

Building OpenOCD requires a recent version of the GNU autotools (autoconf >= 2.59 and automake >= 1.9). For building on Windows, you have to use Cygwin. Make sure that your `PATH` environment variable contains no other locations with Unix utils (like UnxUtils) - these can't handle the Cygwin paths, resulting in obscure dependency errors (This is an observation I've gathered from the logs of one user - correct me if I'm wrong).

You further need the appropriate driver files, if you want to build support for a FTDI FT2232 based interface:

- **ftdi2232** libftdi (http://www.intra2net.com/opensource/ftdi/)
- **ftd2xx** libftd2xx (http://www.ftdichip.com/Drivers/D2XX.htm)
- When using the Amontec JTAGkey, you have to get the drivers from the Amontec homepage (http://www.amontec.com), as the JTAGkey uses a non-standard VID/PID.

libftdi is supported under Windows. Do not use versions earlier than 0.14.

In general, the D2XX driver provides superior performance (several times as fast), but has the draw-back of being binary-only - though that isn't that bad, as it isn't a kernel module, only a user space library.

To build OpenOCD (on both Linux and Cygwin), use the following commands:

```
./bootstrap
```

Bootstrap generates the configure script, and prepares building on your system.

```
./configure [options, see below]
```

Configure generates the Makefiles used to build OpenOCD.

```
make
make install
```

Make builds OpenOCD, and places the final executable in ./src/, the last step, "make install" is optional.

The configure script takes several options, specifying which JTAG interfaces should be included (among other things):

- '`--enable-parport`' - Enable building the PC parallel port driver.
- '`--enable-parport_ppdev`' - Enable use of ppdev (/dev/parportN) for parport.
- '`--enable-parport_giveio`' - Enable use of giveio for parport instead of ioperm.
- '`--enable-amtjtagaccel`' - Enable building the Amontec JTAG-Accelerator driver.
- '`--enable-ecosboard`' - Enable building support for eCosBoard based JTAG debugger.
- '`--enable-ioutil`' - Enable ioutil functions - useful for standalone OpenOCD implementations.
- '`--enable-httpd`' - Enable builtin httpd server - useful for standalone OpenOCD implementations.
- '`--enable-ep93xx`' - Enable building support for EP93xx based SBCs.
- '`--enable-at91rm9200`' - Enable building support for AT91RM9200 based SBCs.
- '`--enable-gw16012`' - Enable building support for the Gateworks GW16012 JTAG programmer.

- '`--enable-ft2232_ftd2xx`' - Numerous USB type ARM JTAG dongles use the FT2232C chip from this FTDICHIP.COM chip (closed source).
- '`--enable-ft2232_libftdi`' - An open source (free) alternative to FTDICHIP.COM ftd2xx solution (Linux, MacOS, Cygwin).
- '`--with-ftd2xx-win32-zipdir=PATH`' - If using FTDICHIP.COM ft2232c, point at the directory where the Win32 FTDICHIP.COM 'CDM' driver zip file was unpacked.
- '`--with-ftd2xx-linux-tardir=PATH`' - Linux only. Equivalent of '`--with-ftd2xx-win32-zipdir`', where you unpacked the TAR.GZ file.
- '`--with-ftd2xx-lib=shared|static`' - Linux only. Default: static. Specifies how the FTDICHIP.COM libftd2xx driver should be linked. Note: 'static' only works in conjunction with '`--with-ftd2xx-linux-tardir`'. The 'shared' value is supported (12/26/2008), however you must manually install the required header files and shared libraries in an appropriate place. This uses "libusb" internally.
- '`--enable-presto_libftdi`' - Enable building support for ASIX Presto programmer using the libftdi driver.
- '`--enable-presto_ftd2xx`' - Enable building support for ASIX Presto programmer using the FTD2XX driver.
- '`--enable-usbprog`' - Enable building support for the USBprog JTAG programmer.
- '`--enable-oocd_trace`' - Enable building support for the OpenOCD+trace ETM capture device.
- '`--enable-jlink`' - Enable building support for the Segger J-Link JTAG programmer.
- '`--enable-vsllink`' - Enable building support for the Versaloon-Link JTAG programmer.
- '`--enable-rlink`' - Enable building support for the Raisonance RLink JTAG programmer.
- '`--enable-arm-jtag-ew`' - Enable building support for the Olimex ARM-JTAG-EW programmer.
- '`--enable-dummy`' - Enable building the dummy port driver.

2.4 Parallel Port Dongles

If you want to access the parallel port using the PPDEV interface you have to specify both the '`--enable-parport`' AND the '`--enable-parport_ppdev`' option since the '`--enable-parport_ppdev`' option actually is an option to the parport driver (see http://forum.sparkfun.com/viewtopic.php?t=3795 for more info).

The same is true for the '`--enable-parport_giveio`' option, you have to use both the '`--enable-parport`' AND the '`--enable-parport_giveio`' option if you want to use giveio instead of ioperm parallel port access method.

2.5 FT2232C Based USB Dongles

There are 2 methods of using the FTD2232, either (1) using the FTDICHIP.COM closed source driver, or (2) the open (and free) driver libftdi. Some claim the (closed) FTDICHIP.COM solution is faster.

The FTDICHIP drivers come as either a (win32) ZIP file, or a (Linux) TAR.GZ file. You must unpack them "some where" convient. As of this writing (12/26/2008) FTDICHIP does not supply means to install these files "in an appropriate place" As a result, there are two "./configure" options that help.

Below is an example build process:

1) Check out the latest version of "openocd" from SVN.

2) Download & unpack either the Windows or Linux FTD2xx drivers (http://www.ftdichip.com/Drivers/D2XX.htm).

```
/home/duane/ftd2xx.win32    => the Cygwin/Win32 ZIP file contents.
/home/duane/libftd2xx0.4.16 => the Linux TAR.GZ file contents.
```

3) Configure with these options:

```
Cygwin FTDICHIP solution:
   ./configure --prefix=/home/duane/mytools \
               --enable-ft2232_ftd2xx \
               --with-ftd2xx-win32-zipdir=/home/duane/ftd2xx.win32

Linux FTDICHIP solution:
   ./configure --prefix=/home/duane/mytools \
               --enable-ft2232_ftd2xx \
               --with-ft2xx-linux-tardir=/home/duane/libftd2xx0.4.16

Cygwin/Linux LIBFTDI solution:
    Assumes:
    1a) For Windows: The Windows port of LIBUSB is in place.
    1b) For Linux: libusb has been built/installed and is in place.

    2) And libftdi has been built and installed
    Note: libftdi - relies upon libusb.

    ./configure --prefix=/home/duane/mytools \
                --enable-ft2232_libftdi
```

4) Then just type "make", and perhaps "make install".

2.6 Miscellaneous Configure Options

- '`--disable-option-checking`' - Ignore unrecognized '`--enable`' and '`--with`' options.
- '`--enable-gccwarnings`' - Enable extra gcc warnings during build. Default is enabled.
- '`--enable-release`' - Enable building of an OpenOCD release, generally this is for developers. It simply omits the svn version string when the openocd '`-v`' is executed.

3 JTAG Hardware Dongles

Defined: **dongle**: A small device that plugins into a computer and serves as an adapter .... [snip]

In the OpenOCD case, this generally refers to **a small adapater** one attaches to your computer via USB or the Parallel Printer Port. The execption being the Zylin ZY1000 which is a small box you attach via an ethernet cable. The Zylin ZY1000 has the advantage that it does not require any drivers to be installed on the developer PC. It also has a built in web interface. It supports RTCK/RCLK or adaptive clocking and has a built in relay to power cycle targets remotely.

3.1 Choosing a Dongle

There are three things you should keep in mind when choosing a dongle.

1. **Voltage** What voltage is your target? 1.8, 2.8, 3.3, or 5V? Does your dongle support it?
2. **Connection** Printer Ports - Does your computer have one?
3. **Connection** Is that long printer bit-bang cable practical?
4. **RTCK** Do you require RTCK? Also known as "adaptive clocking"

3.2 Stand alone Systems

ZY1000 See: http://www.zylin.com/zy1000.html Technically, not a dongle, but a stand-alone box. The ZY1000 has the advantage that it does not require any drivers installed on the developer PC. It also has a built in web interface. It supports RTCK/RCLK or adaptive clocking and has a built in relay to power cycle targets remotely.

3.3 USB FT2232 Based

There are many USB JTAG dongles on the market, many of them are based on a chip from "Future Technology Devices International" (FTDI) known as the FTDI FT2232.

See: http://www.ftdichip.com or http://www.ftdichip.com/Products/FT2232H.htm

As of 28/Nov/2008, the following are supported:

- **usbjtag**
  Link http://www.hs-augsburg.de/~hhoegl/proj/usbjtag/usbjtag.html
- **jtagkey**
  See: http://www.amontec.com/jtagkey.shtml
- **oocdlink**
  See: http://www.oocdlink.com By Joern Kaipf
- **signalyzer**
  See: http://www.signalyzer.com
- **evb_lm3s811**
  See: http://www.luminarymicro.com - The Luminary Micro Stellaris LM3S811 eval board has an FTD2232C chip built in.
- **olimex-jtag**
  See: http://www.olimex.com

- **flyswatter**
  See: http://www.tincantools.com
- **turtelizer2**
  See: http://www.ethernut.de, or http://www.ethernut.de/en/hardware/turtelizer/index.html
- **comstick**
  Link: http://www.hitex.com/index.php?id=383
- **stm32stick**
  Link http://www.hitex.com/stm32-stick
- **axm0432_jtag**
  Axiom AXM-0432 Link http://www.axman.com

3.4 USB JLINK based

There are several OEM versions of the Segger **JLINK** adapter. It is an example of a micro controller based JTAG adapter, it uses an AT91SAM764 internally.

- **ATMEL SAMICE** Only works with ATMEL chips!
  Link: http://www.atmel.com/dyn/products/tools_card.asp?tool_id=3892
- **SEGGER JLINK**
  Link: http://www.segger.com/jlink.html
- **IAR J-Link**
  Link: http://www.iar.com/website1/1.0.1.0/369/1/index.php

3.5 USB RLINK based

Raisonance has an adapter called **RLink**. It exists in a stripped-down form on the STM32 Primer, permanently attached to the JTAG lines. It also exists on the STM32 Primer2, but that is wired for SWD and not JTAG, thus not supported.

- **Raisonance RLink**
  Link: http://www.raisonance.com/products/RLink.php
- **STM32 Primer**
  Link: http://www.stm32circle.com/resources/stm32primer.php
- **STM32 Primer2**
  Link: http://www.stm32circle.com/resources/stm32primer2.php

3.6 USB Other

- **USBprog**
  Link: http://www.embedded-projects.net/usbprog - which uses an Atmel MEGA32 and a UBN9604
- **USB - Presto**
  Link: http://tools.asix.net/prg_presto.htm
- **Versaloon-Link**
  Link: http://www.simonqian.com/en/Versaloon
- **ARM-JTAG-EW**
  Link: http://www.olimex.com/dev/arm-jtag-ew.html

3.7 IBM PC Parallel Printer Port Based

The two well known "JTAG Parallel Ports" cables are the Xilnx DLC5 and the MacGraigor Wiggler. There are many clones and variations of these on the market.

- **Wiggler** - There are many clones of this.
  Link: http://www.macraigor.com/wiggler.htm
- **DLC5** - From XILINX - There are many clones of this
  Link: Search the web for: "XILINX DLC5" - it is no longer produced, PDF schematics are easily found and it is easy to make.
- **Amontec - JTAG Accelerator**
  Link: http://www.amontec.com/jtag_accelerator.shtml
- **GW16402**
  Link: http://www.gateworks.com/products/avila_accessories/gw16042.php
- **Wiggler2**
  Link: http://www.ccac.rwth-aachen.de/~michaels/index.php/hardware/armjtag
- **Wiggler_ntrst_inverted**
  Yet another variation - See the source code, src/jtag/parport.c
- **old_amt_wiggler**
  Unknown - probably not on the market today
- **arm-jtag**
  Link: Most likely http://www.olimex.com/dev/arm-jtag.html [another wiggler clone]
- **chameleon**
  Link: http://www.amontec.com/chameleon.shtml
- **Triton**
  Unknown.
- **Lattice**
  ispDownload from Lattice Semiconductor http://www.latticesemi.com/lit/docs/devtools/dlcable.
- **flashlink**
  From ST Microsystems, link: http://www.st.com/stonline/products/literature/um/7889.pdf
  Title: FlashLINK JTAG programing cable for PSD and uPSD

3.8 Other...

- **ep93xx**
  An EP93xx based Linux machine using the GPIO pins directly.
- **at91rm9200**
  Like the EP93xx - but an ATMEL AT91RM9200 based solution using the GPIO pins on the chip.

4 Running

The '`--help`' option shows:

```
bash$ openocd --help
```

```
--help       | -h       display this help
--version    | -v       display OpenOCD version
--file       | -f       use configuration file <name>
--search     | -s       dir to search for config files and scripts
--debug      | -d       set debug level <0-3>
--log_output | -l       redirect log output to file <name>
--command    | -c       run <command>
--pipe       | -p       use pipes when talking to gdb
```

By default OpenOCD reads the file configuration file "openocd.cfg" in the current directory. To specify a different (or multiple) configuration file, you can use the "-f" option. For example:

```
openocd -f config1.cfg -f config2.cfg -f config3.cfg
```

Once started, OpenOCD runs as a daemon, waiting for connections from clients (Telnet, GDB, Other).

If you are having problems, you can enable internal debug messages via the "-d" option.

Also it is possible to interleave commands w/config scripts using the '`-c`' command line switch.

To enable debug output (when reporting problems or working on OpenOCD itself), use the '`-d`' command line switch. This sets the '`debug_level`' to `"3"`, outputting the most information, including debug messages. The default setting is `"2"`, outputting only informational messages, warnings and errors. You can also change this setting from within a telnet or gdb session using '`debug_level <n>`' See [debug_level], page 45.

You can redirect all output from the daemon to a file using the '`-l <logfile>`' switch.

Search paths for config/script files can be added to OpenOCD by using the '`-s <search>`' switch. The current directory and the OpenOCD target library is in the search path by default.

For details on the '`-p`' option. See [Connecting to GDB], page 57.

Note! OpenOCD will launch the GDB & telnet server even if it can not establish a connection with the target. In general, it is possible for the JTAG controller to be unresponsive until the target is set up correctly via e.g. GDB monitor commands in a GDB init script.

5 Simple Configuration Files

5.1 Outline

There are 4 basic ways of "configurating" OpenOCD to run, they are:

1. A small openocd.cfg file which "sources" other configuration files
2. A monolithic openocd.cfg file
3. Many -f filename options on the command line
4. Your Mixed Solution

5.2 Small configuration file method

This is the preferred method. It is simple and works well for many people. The developers of OpenOCD would encourage you to use this method. If you create a new configuration please email new configurations to the development list.

Here is an example of an openocd.cfg file for an ATMEL at91sam7x256

```
source [find interface/signalyzer.cfg]

# Change the default telnet port...
telnet_port 4444
# GDB connects here
gdb_port 3333
# GDB can also flash my flash!
gdb_memory_map enable
gdb_flash_program enable

source [find target/sam7x256.cfg]
```

There are many example configuration scripts you can work with. You should look in the directory: `$(INSTALLDIR)/lib/openocd`. You should find:

1. **board** - eval board level configurations
2. **interface** - specific dongle configurations
3. **target** - the target chips
4. **tcl** - helper scripts
5. **xscale** - things specific to the xscale.

Look first in the "boards" area, then the "targets" area. Often a board configuration is a good example to work from.

5.3 Many -f filename options

Some believe this is a wonderful solution, others find it painful.

You can use a series of "-f filename" options on the command line, OpenOCD will read each filename in sequence, for example:

```
openocd -f file1.cfg -f file2.cfg -f file2.cfg
```

You can also intermix various commands with the "-c" command line option.

5.4 Monolithic file

The “Monolithic File” dispenses with all “source” statements and puts everything in one self contained (monolithic) file. This is not encouraged.

Please try to “source” various files or use the multiple -f technique.

5.5 Advice for you

Often, one uses a “mixed approach”. Where possible, please try to “source” common things, and if needed cut/paste parts of the standard distribution configuration files as needed.

REMEMBER: The “important parts” of your configuration file are:

1. **Interface** - Defines the dongle
2. **Taps** - Defines the JTAG Taps
3. **GDB Targets** - What GDB talks to
4. **Flash Programing** - Very Helpful

Some key things you should look at and understand are:

1. The reset configuration of your debug environment as a whole
2. Is there a “work area” that OpenOCD can use?
   For ARM - work areas mean up to 10x faster downloads.
3. For MMU/MPU based ARM chips (i.e.: ARM9 and later) will that work area still be available?
4. For complex targets (multiple chips) the JTAG SPEED becomes an issue.

6 Config File Guidelines

This section/chapter is aimed at developers and integrators of OpenOCD. These are guidelines for creating new boards and new target configurations as of 28/Nov/2008.

However, you, the user of OpenOCD, should be somewhat familiar with this section as it should help explain some of the internals of what you might be looking at.

The user should find the following directories under `$(INSTALLDIR)/lib/openocd` :

- **interface**
  Think JTAG Dongle. Files that configure the JTAG dongle go here.
- **board**
  Think Circuit Board, PWA, PCB, they go by many names. Board files contain initialization items that are specific to a board - for example: The SDRAM initialization sequence for the board, or the type of external flash and what address it is found at. Any initialization sequence to enable that external flash or SDRAM should be found in the board file. Boards may also contain multiple targets, i.e.: Two CPUs, or a CPU and an FPGA or CPLD.
- **target**
  Think chip. The "target" directory represents a JTAG tap (or chip) OpenOCD should control, not a board. Two common types of targets are ARM chips and FPGA or CPLD chips.

If needed... The user in their "openocd.cfg" file or the board file might override a specific feature in any of the above files by setting a variable or two before sourcing the target file. Or adding various commands specific to their situation.

6.1 Interface Config Files

The user should be able to source one of these files via a command like this:

```
  source [find interface/FOOBAR.cfg]
Or:
  openocd -f interface/FOOBAR.cfg
```

A preconfigured interface file should exist for every interface in use today, that said, perhaps some interfaces have only been used by the sole developer who created it.

FIXME/NOTE: We need to add support for a variable like Tcl variable tcl_platform(platform), it should be called jim_platform (because it is jim, not real tcl) and it should contain 1 of 3 words: "linux", "cygwin" or "mingw"

Interface files should be found in `$(INSTALLDIR)/lib/openocd/interface`

6.2 Board Config Files

Note: BOARD directory NEW as of 28/nov/2008

The user should be able to source one of these files via a command like this:

```
  source [find board/FOOBAR.cfg]
Or:
  openocd -f board/FOOBAR.cfg
```

The board file should contain one or more `source [find target/FOO.cfg]` statements along with any board specific things.

In summary the board files should contain (if present)

1. External flash configuration (i.e.: the flash on CS0)
2. SDRAM configuration (size, speed, etc.
3. Board specific IO configuration (i.e.: GPIO pins might disable a 2nd flash)
4. Multiple TARGET source statements
5. All things that are not "inside a chip"
6. Things inside a chip go in a 'target' file

6.3 Target Config Files

The user should be able to source one of these files via a command like this:

```
  source [find target/FOOBAR.cfg]
Or:
  openocd -f target/FOOBAR.cfg
```

In summary the target files should contain

1. Set defaults
2. Create taps
3. Reset configuration
4. Work areas
5. CPU/Chip/CPU-Core specific features
6. On-Chip flash

6.3.1 Important variable names

By default, the end user should never need to set these variables. However, if the user needs to override a setting they only need to set the variable in a simple way.

- **CHIPNAME**
  This gives a name to the overall chip, and is used as part of the tap identifier dotted name.
- **ENDIAN**
  By default little - unless the chip or board is not normally used that way.
- **CPUTAPID**
  When OpenOCD examines the JTAG chain, it will attempt to identify every chip. If the `-expected-id` is nonzero, OpenOCD attempts to verify the tap id number verses configuration file and may issue an error or warning like this. The hope is that this will help to pinpoint problems in OpenOCD configurations.

```
Info:   JTAG tap: sam7x256.cpu tap/device found: 0x3f0f0f0f (Manufacturer: 0x787, P
Error:  ERROR: Tap: sam7x256.cpu - Expected id: 0x12345678, Got: 0x3f0f0f0f
Error:  ERROR: expected: mfg: 0x33c, part: 0x2345, ver: 0x1
Error:  ERROR:      got: mfg: 0x787, part: 0xf0f0, ver: 0x3
```

- **_TARGETNAME**
  By convention, this variable is created by the target configuration script. The board

configuration file may make use of this variable to configure things like a "reset init" script, or other things specific to that board and that target.

If the chip has 2 targets, use the names **_TARGETNAME0**, **_TARGETNAME1**, ... etc.

Remember: The "board file" may include multiple targets.

At no time should the name "target0" (the default target name if none was specified) be used. The name "target0" is a hard coded name - the next target on the board will be some other number.

The user (or board file) should reasonably be able to:

```
source [find target/FOO.cfg]
$_TARGETNAME configure ... FOO specific parameters

source [find target/BAR.cfg]
$_TARGETNAME configure ... BAR specific parameters
```

6.3.2 Tcl Variables Guide Line

The Full Tcl/Tk language supports "namespaces" - JIM-Tcl does not.

Thus the rule we follow in OpenOCD is this: Variables that begin with a leading underscore are temporary in nature, and can be modified and used at will within a ?TARGET? configuration file.

EXAMPLE: The user should be able to do this:

```
# Board has 3 chips,
#    PXA270 #1 network side, big endian
#    PXA270 #2 video side, little endian
#    Xilinx    Glue logic
set CHIPNAME network
set ENDIAN big
source [find target/pxa270.cfg]
# variable: _TARGETNAME = network.cpu
# other commands can refer to the "network.cpu" tap.
$_TARGETNAME configure .... params for this CPU..

set ENDIAN little
set CHIPNAME video
source [find target/pxa270.cfg]
# variable: _TARGETNAME = video.cpu
# other commands can refer to the "video.cpu" tap.
$_TARGETNAME configure .... params for this CPU..

unset ENDIAN
set CHIPNAME xilinx
source [find target/spartan3.cfg]

# Since $_TARGETNAME is temporal..
#  these names still work!
network.cpu configure ... params
```

```
video.cpu   configure ... params
```

6.3.3 Default Value Boiler Plate Code

All target configuration files should start with this (or a modified form)

```
# SIMPLE example
if { [info exists CHIPNAME] } {
   set  _CHIPNAME $CHIPNAME
} else {
   set  _CHIPNAME sam7x256
}

if { [info exists ENDIAN] } {
   set  _ENDIAN $ENDIAN
} else {
   set  _ENDIAN little
}

if { [info exists CPUTAPID ] } {
   set _CPUTAPID $CPUTAPID
} else {
   set _CPUTAPID 0x3f0f0f0f
}
```

6.3.4 Creating Taps

After the "defaults" are choosen [see above] the taps are created.

SIMPLE example: such as an Atmel AT91SAM7X256

```
# for an ARM7TDMI.
set _TARGETNAME [format "%s.cpu" $_CHIPNAME]
jtag newtap $_CHIPNAME cpu -irlen 4 -ircapture 0x1 -irmask 0xf -expected-id $_CPUTAPID
```

COMPLEX example:

This is an SNIP/example for an STR912 - which has 3 internal taps. Key features shown:

1. **Unform tap names** - See: Tap Naming Convention
2. **_TARGETNAME** is created at the end where used.

```
if { [info exists FLASHTAPID ] } {
   set _FLASHTAPID $FLASHTAPID
} else {
   set _FLASHTAPID 0x25966041
}
jtag newtap $_CHIPNAME flash -irlen 8 -ircapture 0x1 -irmask 0x1 -expected-id $_FLASHT

if { [info exists CPUTAPID ] } {
   set _CPUTAPID $CPUTAPID
} else {
```

```
   set _CPUTAPID 0x25966041
}
jtag newtap $_CHIPNAME cpu   -irlen 4 -ircapture 0xf -irmask 0xe -expected-id $_CPUTAP


if { [info exists BSTAPID ] } {
   set _BSTAPID $BSTAPID
} else {
   set _BSTAPID 0x1457f041
}
jtag newtap $_CHIPNAME bs    -irlen 5 -ircapture 0x1 -irmask 0x1 -expected-id $_BSTAPI

set _TARGETNAME [format "%s.cpu" $_CHIPNAME]
```

Tap Naming Convention

See the command "jtag newtap" for detail, but in brief the names you should use are:

- **tap**
- **cpu**
- **flash**
- **bs**
- **jrc**
- **unknownN** - it happens :-(

6.3.5 Reset Configuration

Some chips have specific ways the TRST and SRST signals are managed. If these are **CHIP SPECIFIC** they go here, if they are **BOARD SPECIFIC** they go in the board file.

6.3.6 Work Areas

Work areas are small RAM areas used by OpenOCD to speed up downloads, and to download small snippets of code to program flash chips.

If the chip includes a form of "on-chip-ram" - and many do - define a reasonable work area and use the "backup" option.

PROBLEMS: On more complex chips, this "work area" may become inaccessible if/when the application code enables or disables the MMU.

6.3.7 ARM Core Specific Hacks

If the chip has a DCC, enable it. If the chip is an ARM9 with some special high speed download features - enable it.

If the chip has an ARM "vector catch" feature - by default enable it for Undefined Instructions, Data Abort, and Prefetch Abort, if the user is really writing a handler for those situations - they can easily disable it. Experiance has shown the "vector catch" is helpful - for common programing errors.

If present, the MMU, the MPU and the CACHE should be disabled.

6.3.8 Internal Flash Configuration

This applies **ONLY TO MICROCONTROLLERS** that have flash built in.

Never ever in the "target configuration file" define any type of flash that is external to the chip. (For example the BOOT flash on Chip Select 0). The BOOT flash information goes in a board file - not the TARGET (chip) file.

Examples:

- at91sam7x256 - has 256K flash YES enable it.
- str912 - has flash internal YES enable it.
- imx27 - uses boot flash on CS0 - it goes in the board file.
- pxa270 - again - CS0 flash - it goes in the board file.

7 About JIM-Tcl

OpenOCD includes a small "TCL Interpreter" known as JIM-TCL. You can learn more about JIM here: http://jim.berlios.de

- **JIM vs. Tcl**
  JIM-TCL is a stripped down version of the well known Tcl language, which can be found here: http://www.tcl.tk. JIM-Tcl has far fewer features. JIM-Tcl is a single .C file and a single .H file and impliments the basic Tcl command set along. In contrast: Tcl 8.6 is a 4.2 MB .zip file containing 1540 files.
- **Missing Features**
  Our practice has been: Add/clone the real Tcl feature if/when needed. We welcome JIM Tcl improvements, not bloat.
- **Scripts**
  OpenOCD configuration scripts are JIM Tcl Scripts. OpenOCD's command interpreter today (28/nov/2008) is a mixture of (newer) JIM-Tcl commands, and (older) the orginal command interpreter.
- **Commands**
  At the OpenOCD telnet command line (or via the GDB mon command) one can type a Tcl for() loop, set variables, etc.
- **Historical Note**
  JIM-Tcl was introduced to OpenOCD in spring 2008.
- **Need a crash course in Tcl?**
  See: See Chapter 22 [Tcl Crash Course], page 66.

8 Daemon Configuration

The commands here are commonly found in the openocd.cfg file and are used to specify what TCP/IP ports are used, and how GDB should be supported.

8.1 init

This command terminates the configuration stage and enters the normal command mode. This can be useful to add commands to the startup scripts and commands such as resetting the target, programming flash, etc. To reset the CPU upon startup, add "init" and "reset" at the end of the config script or at the end of the OpenOCD command line using the '`-c`' command line switch.

If this command does not appear in any startup/configuration file OpenOCD executes the command for you after processing all configuration files and/or command line options.

NOTE: This command normally occurs at or near the end of your openocd.cfg file to force OpenOCD to "initialize" and make the targets ready. For example: If your openocd.cfg file needs to read/write memory on your target - the init command must occur before the memory read/write commands.

8.2 TCP/IP Ports

- **telnet_port** <*number*>
  Intended for a human. Port on which to listen for incoming telnet connections.
- **tcl_port** <*number*>
  Intended as a machine interface. Port on which to listen for incoming Tcl syntax. This port is intended as a simplified RPC connection that can be used by clients to issue commands and get the output from the Tcl engine.
- **gdb_port** <*number*>
  First port on which to listen for incoming GDB connections. The GDB port for the first target will be gdb_port, the second target will listen on gdb_port + 1, and so on.

8.3 GDB Items

- **gdb_breakpoint_override** <*hard*|*soft*|*disable*>
  Force breakpoint type for gdb 'break' commands. The raison d'etre for this option is to support GDB GUI's without a hard/soft breakpoint concept where the default OpenOCD and GDB behaviour is not sufficient. Note that GDB will use hardware breakpoints if the memory map has been set up for flash regions.

  This option replaces older arm7_9 target commands that addressed the same issue.
- **gdb_detach** <*resume*|*reset*|*halt*|*nothing*>
  Configures what OpenOCD will do when GDB detaches from the daemon. Default behaviour is <*resume*>
- **gdb_memory_map** <*enable*|*disable*>
  Set to <*enable*> to cause OpenOCD to send the memory configuration to GDB when requested. GDB will then know when to set hardware breakpoints, and program flash using the GDB load command. '`gdb_flash_program enable`' must also be enabled for

flash programming to work. Default behaviour is <*enable*> See [gdb_flash_program], page 22.

- **gdb_flash_program** <*enable*|*disable*>
  Set to <*enable*> to cause OpenOCD to program the flash memory when a vFlash packet is received. Default behaviour is <*enable*>

9 Interface - Dongle Configuration

Interface commands are normally found in an interface configuration file which is sourced by your openocd.cfg file. These commands tell OpenOCD what type of JTAG dongle you have and how to talk to it.

9.1 Simple Complete Interface Examples

A Turtelizer FT2232 Based JTAG Dongle

```
#interface
interface ft2232
ft2232_device_desc "Turtelizer JTAG/RS232 Adapter A"
ft2232_layout turtelizer2
ft2232_vid_pid 0x0403 0xbdc8
```

A SEGGER Jlink

```
# jlink interface
interface jlink
```

A Raisonance RLink

```
# rlink interface
interface rlink
```

Parallel Port

```
interface parport
parport_port 0xc8b8
parport_cable wiggler
jtag_speed 0
```

ARM-JTAG-EW

```
interface arm-jtag-ew
```

9.2 Interface Command

The interface command tells OpenOCD what type of JTAG dongle you are using. Depending on the type of dongle, you may need to have one or more additional commands.

- **interface** <*name*>
  Use the interface driver <*name*> to connect to the target. Currently supported interfaces are
  - **parport**
    PC parallel port bit-banging (Wigglers, PLD download cable, ...)
  - **amt_jtagaccel**
    Amontec Chameleon in its JTAG Accelerator configuration connected to a PC's EPP mode parallel port
  - **ft2232**
    FTDI FT2232 (USB) based devices using either the open-source libftdi or the binary only FTD2XX driver. The FTD2XX is superior in performance, but not available on every platform. The libftdi uses libusb, and should be portable to all systems that provide libusb.

- **ep93xx**
  Cirrus Logic EP93xx based single-board computer bit-banging (in development)
- **presto**
  ASIX PRESTO USB JTAG programmer.
- **usbprog**
  usbprog is a freely programmable USB adapter.
- **gw16012**
  Gateworks GW16012 JTAG programmer.
- **jlink**
  Segger jlink USB adapter
- **rlink**
  Raisonance RLink USB adapter
- **vsllink**
  vsllink is part of Versaloon which is a versatile USB programmer.
- **arm-jtag-ew**
  Olimex ARM-JTAG-EW USB adapter

9.2.1 parport options

- **parport_port** <*number*>
  Either the address of the I/O port (default: 0x378 for LPT1) or the number of the '`/dev/parport`' device

  When using PPDEV to access the parallel port, use the number of the parallel port: '`parport_port 0`' (the default). If '`parport_port 0x378`' is specified you may encounter a problem.
- **parport_cable** <*name*>
  The layout of the parallel port cable used to connect to the target. Currently supported cables are
  - **wiggler** The original Wiggler layout, also supported by several clones, such as the Olimex ARM-JTAG
  - **wiggler2** Same as original wiggler except an led is fitted on D5.
  - **wiggler_ntrst_inverted** Same as original wiggler except TRST is inverted.
  - **old_amt_wiggler** The Wiggler configuration that comes with Amontec's Chameleon Programmer. The new version available from the website uses the original Wiggler layout ('*wiggler*')
  - **chameleon** The Amontec Chameleon's CPLD when operated in configuration mode. This is only used to program the Chameleon itself, not a connected target.
  - **dlc5** The Xilinx Parallel cable III.
  - **triton** The parallel port adapter found on the 'Karo Triton 1 Development Board'. This is also the layout used by the HollyGates design (see http://www.lartmaker.nl/projects/jtag/).
  - **flashlink** The ST Parallel cable.
  - **arm-jtag** Same as original wiggler except SRST and TRST connections reversed and TRST is also inverted.

  - **altium** Altium Universal JTAG cable.
- **parport_write_on_exit** <*on*|*off*>
  This will configure the parallel driver to write a known value to the parallel interface on exiting OpenOCD

9.2.2 amt_jtagaccel options

- **parport_port** <*number*>
  Either the address of the I/O port (default: 0x378 for LPT1) or the number of the '`/dev/parport`' device

9.2.3 ft2232 options

- **ft2232_device_desc** <*description*>
  The USB device description of the FTDI FT2232 device. If not specified, the FTDI default value is used. This setting is only valid if compiled with FTD2XX support.

  TODO: Confirm the following: On Windows the name needs to end with a "space A"? Or not? It has to do with the FTD2xx driver. When must this be added and when must it not be added? Why can't the code in the interface or in OpenOCD automatically add this if needed? – Duane.
- **ft2232_serial** <*serial-number*>
  The serial number of the FTDI FT2232 device. If not specified, the FTDI default values are used.
- **ft2232_layout** <*name*>
  The layout of the FT2232 GPIO signals used to control output-enables and reset signals. Valid layouts are
  - **usbjtag** "USBJTAG-1" layout described in the original OpenOCD diploma thesis
  - **jtagkey** Amontec JTAGkey and JTAGkey-Tiny
  - **signalyzer** Signalyzer
  - **olimex-jtag** Olimex ARM-USB-OCD
  - **m5960** American Microsystems M5960
  - **evb_lm3s811** Luminary Micro EVB_LM3S811 as a JTAG interface (not onboard processor), no TRST or SRST signals on external connector
  - **comstick** Hitex STR9 comstick
  - **stm32stick** Hitex STM32 Performance Stick
  - **flyswatter** Tin Can Tools Flyswatter
  - **turtelizer2** egnite Software turtelizer2
  - **oocdlink** OOCDLink
  - **axm0432_jtag** Axiom AXM-0432
- **ft2232_vid_pid** <*vid*> <*pid*>
  The vendor ID and product ID of the FTDI FT2232 device. If not specified, the FTDI default values are used. Multiple <*vid*>, <*pid*> pairs may be given, e.g.

  ```
  ft2232_vid_pid 0x0403 0xcff8 0x15ba 0x0003
  ```
- **ft2232_latency** <*ms*>
  On some systems using FT2232 based JTAG interfaces the FT_Read function call in

ft2232_read() fails to return the expected number of bytes. This can be caused by USB communication delays and has proved hard to reproduce and debug. Setting the FT2232 latency timer to a larger value increases delays for short USB packets but it also reduces the risk of timeouts before receiving the expected number of bytes. The OpenOCD default value is 2 and for some systems a value of 10 has proved useful.

9.2.4 ep93xx options

Currently, there are no options available for the ep93xx interface.

9.3 JTAG Speed

- **jtag_khz** <*reset speed kHz*>

  It is debatable if this command belongs here - or in a board configuration file. In fact, in some situations the JTAG speed is changed during the target initialisation process (i.e.: (1) slow at reset, (2) program the CPU clocks, (3) run fast)

  Speed 0 (khz) selects RTCK method. A non-zero speed is in KHZ. Hence: 3000 is 3mhz.

  Not all interfaces support "rtck". If the interface device can not support the rate asked for, or can not translate from kHz to jtag_speed, then an error is returned.

  Make sure the JTAG clock is no more than $1/6th CPU - Clock$. This is especially true for synthesized cores (-S). Also see RTCK.

  NOTE: Script writers If the target chip requires/uses RTCK - please use the command: 'jtag_rclk FREQ'. This Tcl proc (in startup.tcl) attempts to enable RTCK, if that fails it falls back to the specified frequency.

  ```
  # Fall back to 3mhz if RCLK is not supported
  jtag_rclk 3000
  ```

- **DEPRECATED jtag_speed** - please use jtag_khz above.
  Limit the maximum speed of the JTAG interface. Usually, a value of zero means maximum speed. The actual effect of this option depends on the JTAG interface used.

  The speed used during reset can be adjusted using setting jtag_speed during pre_reset and post_reset events.

  - wiggler: maximum speed / *number*
  - ft2232: 6MHz / (*number*+1)
  - amt jtagaccel: 8 / 2***number*
  - jlink: maximum speed in kHz (0-12000), 0 will use RTCK
  - rlink: 24MHz / *number*, but only for certain values of *number*

10 Reset Configuration

Every system configuration may require a different reset configuration. This can also be quite confusing. Please see the various board files for example.

10.1 jtag_nsrst_delay <*ms*>

How long (in milliseconds) OpenOCD should wait after deasserting nSRST before starting new JTAG operations.

10.2 jtag_ntrst_delay <*ms*>

Same **jtag_nsrst_delay**, but for nTRST

The jtag_n[st]rst_delay options are useful if reset circuitry (like a big resistor/capacitor, reset supervisor, or on-chip features). This keeps the signal asserted for some time after the external reset got deasserted.

10.3 reset_config

Note: To maintainers and integrators: Where exactly the "reset configuration" goes is a good question. It touches several things at once. In the end, if you have a board file - the board file should define it and assume 100% that the DONGLE supports anything. However, that does not mean the target should not also make not of something the silicon vendor has done inside the chip. *Grr.... nothing is every pretty.*

Problems:

1. Every JTAG Dongle is slightly different, some dongles implement reset differently.
2. Every board is also slightly different; some boards tie TRST and SRST together.
3. Every chip is slightly different; some chips internally tie the two signals together.
4. Some may not implement all of the signals the same way.
5. Some signals might be push-pull, others open-drain/collector.

Best Case: OpenOCD can hold the SRST (push-button-reset), then reset the TAP via TRST and send commands through the JTAG tap to halt the CPU at the reset vector before the 1st instruction is executed, and finally release the SRST signal.
Depending on your board vendor, chip vendor, etc., these signals may have slightly different names.

OpenOCD defines these signals in these terms:

- **TRST** - is Tap Reset - and should reset only the TAP.
- **SRST** - is System Reset - typically equal to a reset push button.

The Command:

- **reset_config** <*signals*> [*combination*] [*trst_type*] [*srst_type*]
  The `reset_config` command tells OpenOCD the reset configuration of your combination of Dongle, Board, and Chips. If the JTAG interface provides SRST, but the

target doesn't connect that signal properly, then OpenOCD can't use it. <*signals*> can be '`none`', '`trst_only`', '`srst_only`' or '`trst_and_srst`'.

[*combination*] is an optional value specifying broken reset signal implementations. '`srst_pulls_trst`' states that the test logic is reset together with the reset of the system (e.g. Philips LPC2000, "broken" board layout), '`trst_pulls_srst`' says that the system is reset together with the test logic (only hypothetical, I haven't seen hardware with such a bug, and can be worked around). '`combined`' implies both '`srst_pulls_trst`' and '`trst_pulls_srst`'. The default behaviour if no option given is '`separate`'.

The [*trst_type*] and [*srst_type*] parameters allow the driver type of the reset lines to be specified. Possible values are '`trst_push_pull`' (default) and '`trst_open_drain`' for the test reset signal, and '`srst_open_drain`' (default) and '`srst_push_pull`' for the system reset. These values only affect JTAG interfaces with support for different drivers, like the Amontec JTAGkey and JTAGAccelerator.

11 Tap Creation

In order for OpenOCD to control a target, a JTAG tap must be defined/created.

Commands to create taps are normally found in a configuration file and are not normally typed by a human.

When a tap is created a **dotted.name** is created for the tap. Other commands use that dotted.name to manipulate or refer to the tap.

Tap Uses:

- **Debug Target** A tap can be used by a GDB debug target
- **Flash Programing** Some chips program the flash via JTAG
- **Boundry Scan** Some chips support boundary scan.

11.1 jtag newtap

`jtag newtap CHIPNAME TAPNAME configparams ....`

- **CHIPNAME**
  is a symbolic name of the chip.
- **TAPNAME**
  is a symbol name of a tap present on the chip.
- **Required configparams**
  Every tap has 3 required configparams, and several "optional parameters", the required parameters are:
  - **-irlen NUMBER** - the length in bits of the instruction register, mostly 4 or 5 bits.
  - **-ircapture NUMBER** - the IDCODE capture command, usually 0x01.
  - **-irmask NUMBER** - the corresponding mask for the IR register. For some devices, there are bits in the IR that aren't used. This lets you mask them off when doing comparisons. In general, this should just be all ones for the size of the IR.

  An example of a FOOBAR Tap

  ```
  jtag newtap foobar tap -irlen 7 -ircapture 0x42 -irmask 0x55
  ```

  Creates the tap "foobar.tap" with the instruction register (IR) is 7 bits long, during Capture-IR 0x42 is loaded into the IR, and bits [6,4,2,0] are checked.
- **Optional configparams**
  - **-expected-id NUMBER**
    By default it is zero. If non-zero represents the expected tap ID used when the JTAG chain is examined. See below.
  - **-disable**
  - **-enable**
    By default not specified the tap is enabled. Some chips have a JTAG route controller (JRC) that is used to enable and/or disable specific JTAG taps. You can later enable or disable any JTAG tap via the command **jtag tapenable DOTTED.NAME** or **jtag tapdisable DOTTED.NAME**

Notes:

- **Technically**
  newtap is a sub command of the "jtag" command
- **Big Picture Background**
  GDB Talks to OpenOCD using the GDB protocol via TCP/IP. OpenOCD then uses the JTAG interface (the dongle) to control the JTAG chain on your board. Your board has one or more chips in a *daisy chain configuration*. Each chip may have one or more JTAG taps. GDB ends up talking via OpenOCD to one of the taps.
- **NAME Rules**
  Names follow "C" symbol name rules (start with alpha ...)
- **TAPNAME - Conventions**
  - **tap** - should be used only FPGA or CPLD like devices with a single tap.
  - **cpu** - the main CPU of the chip, alternatively **foo.arm** and **foo.dsp**
  - **flash** - if the chip has a flash tap, example: str912.flash
  - **bs** - for boundary scan if this is a seperate tap.
  - **jrc** - for JTAG route controller (example: OMAP3530 found on Beagleboards)
  - **unknownN** - where N is a number if you have no idea what the tap is for
  - **Other names** - Freescale IMX31 has a SDMA (smart dma) with a JTAG tap, that tap should be called the "sdma" tap.
  - **When in doubt** - use the chip maker's name in their data sheet.
- **DOTTED.NAME**
  CHIPNAME.TAPNAME creates the tap name, aka: the **Dotted.Name** is the **CHIPNAME** and **TAPNAME** combined with a dot (period); for example: **xilinx.tap**, **str912.flash**, **omap3530.jrc**, or **stm32.cpu** The **dotted.name** is used in numerous other places to refer to various taps.
- **ORDER**
  The order this command appears via the config files is important.
- **Multi Tap Example**
  This example is based on the ST Microsystems STR912. See the ST document titled: **STR91xFAxxx, Section 3.15 Jtag Interface, Page: 28/102, Figure 3: JTAG chaining inside the STR91xFA**.

  http://eu.st.com/stonline/products/literature/ds/13495.pdf
  checked: 28/nov/2008

  The diagram shows that the TDO pin connects to the flash tap, flash TDI connects to the CPU debug tap, CPU TDI connects to the boundary scan tap which then connects to the TDI pin.

```
# The order is...
# create tap: 'str912.flash'
jtag newtap str912 flash  ... params ...
# create tap: 'str912.cpu'
jtag newtap str912 cpu  ... params ...
# create tap: 'str912.bs'
jtag newtap str912 bs  ... params ...
```

- **Note: Deprecated** - Index Numbers
  Prior to 28/nov/2008, JTAG taps where numbered from 0..N this feature is still present, however its use is highly discouraged and should not be counted upon.
- **Multiple chips**
  If your board has multiple chips, you should be able to **source** two configuration files, in the proper order, and have the taps created in the proper order.

11.2 jtag_device - REMOVED

jtag_device *<IR length> <IR capture> <IR mask> <IDCODE instruction>*

Removed: 28/nov/2008 This command has been removed and replaced by the "jtag newtap" command. The documentation remains here so that one can easily convert the old syntax to the new syntax. About the old syntax: The old syntax is positional, i.e.: The 3rd parameter is the "irmask". The new syntax requires named prefixes, and supports additional options, for example "-expected-id 0x3f0f0f0f". Please refer to the **jtag newtap** command for details.

```
OLD: jtag_device 8 0x01 0xe3 0xfe
NEW: jtag newtap CHIPNAME TAPNAME -irlen 8 -ircapture 0x01 -irmask 0xe3
```

11.3 Enable/Disable Taps

Note: These commands are intended to be used as a machine/script interface. Humans might find the "scan_chain" command more helpful when querying the state of the JTAG taps.

By default, all taps are enabled

- **jtag tapenable** *DOTTED.NAME*
- **jtag tapdisable** *DOTTED.NAME*
- **jtag tapisenabled** *DOTTED.NAME*

These commands are used when your target has a JTAG route controller that effectively adds or removes a tap from the JTAG chain in a non-standard way.

The "standard way" to remove a tap would be to place the tap in bypass mode. But with the advent of modern chips, this is not always a good solution. Some taps operate slowly, others operate fast, and there are other JTAG clock synchronisation problems one must face. To solve that problem, the JTAG route controller was introduced. Rather than "bypass" the tap, the tap is completely removed from the circuit and skipped.

From OpenOCD's point of view, a JTAG tap is in one of 3 states:

- **Enabled - Not In ByPass** and has a variable bit length
- **Enabled - In ByPass** and has a length of exactly 1 bit.
- **Disabled** and has a length of ZERO and is removed from the circuit.

The IEEE JTAG definition has no concept of a "disabled" tap. **Historical note:** this feature was added 28/nov/2008

jtag tapisenabled DOTTED.NAME

This command returns 1 if the named tap is currently enabled, 0 if not. This command exists so that scripts that manipulate a JRC (like the OMAP3530 has) can determine if OpenOCD thinks a tap is presently enabled or disabled.

12 Target Configuration

This chapter discusses how to create a GDB debug target. Before creating a "target" a JTAG tap DOTTED.NAME must exist first.

12.1 targets [NAME]

Note: This command name is PLURAL - not singular.

With NO parameter, this plural **targets** command lists all known targets in a human friendly form.

With a parameter, this plural **targets** command sets the current target to the given name. (i.e.: If there are multiple debug targets)

Example:

```
(gdb) mon targets
      CmdName     Type     Endian    ChainPos   State
--  ---------- ---------- ---------- -------- ----------
    0: target0  arm7tdmi   little        0      halted
```

12.2 target COMMANDS

Note: This command name is SINGULAR - not plural. It is used to manipulate specific targets, to create targets and other things.

Once a target is created, a TARGETNAME (object) command is created; see below for details.

The TARGET command accepts these sub-commands:

- **create** .. parameters ..
  creates a new target, see below for details.
- **types**
  Lists all supported target types (perhaps some are not yet in this document).
- **names**
  Lists all current debug target names, for example: 'str912.cpu' or 'pxa27.cpu' example usage:

```
foreach t [target names] {
    puts [format "Target: %s\n" $t]
}
```

- **current**
  Returns the current target. OpenOCD always has, or refers to the "current target" in some way. By default, commands like: "mww" (used to write memory) operate on the current target.
- **number NUMBER**
  Internally OpenOCD maintains a list of targets - in numerical index (0..N-1) this command returns the name of the target at index N. Example usage:

```
set thename [target number $x]
puts [format "Target %d is: %s\n" $x $thename]
```

- **count**
  Returns the number of targets known to OpenOCD (see number above) Example:

```
set c [target count]
for { set x 0 } { $x < $c } { incr x } {
    # Assuming you have created this function
    print_target_details $x
}
```

12.3 TARGETNAME (object) commands

Use: Once a target is created, an "object name" that represents the target is created. By convention, the target name is identical to the tap name. In a multiple target system, one can preceed many common commands with a specific target name and effect only that target.

```
str912.cpu    mww 0x1234 0x42
omap3530.cpu  mww 0x5555 123
```

Model: The Tcl/Tk language has the concept of object commands. A good example is a on screen button, once a button is created a button has a name (a path in Tk terms) and that name is useable as a 1st class command. For example in Tk, one can create a button and later configure it like this:

```
# Create
button .foobar -background red -command { foo }
# Modify
.foobar configure -foreground blue
# Query
set x [.foobar cget -background]
# Report
puts [format "The button is %s" $x]
```

In OpenOCD's terms, the "target" is an object just like a Tcl/Tk button. Commands available as a "target object" are:

- **configure** - configure the target; see Target Config/Cget Options below
- **cget** - query the target configuration; see Target Config/Cget Options below
- **curstate** - current target state (running, halt, etc.
- **eventlist**
  Intended for a human to see/read the currently configure target events.
- **Various Memory Commands** See the "mww" command elsewhere.
  - **mww** ...
  - **mwh** ...
  - **mwb** ...
  - **mdw** ...
  - **mdh** ...
  - **mdb** ...
- **Memory To Array, Array To Memory**
  These are aimed at a machine interface to memory

- **mem2array ARRAYNAME WIDTH ADDRESS COUNT**
- **array2mem ARRAYNAME WIDTH ADDRESS COUNT**
  Where:
  ARRAYNAME is the name of an array variable
  WIDTH is 8/16/32 - indicating the memory access size
  ADDRESS is the target memory address
  COUNT is the number of elements to process

- **Used during "reset"**
  These commands are used internally by the OpenOCD scripts to deal with odd reset situations and are not documented here.
  - **arp_examine**
  - **arp_poll**
  - **arp_reset**
  - **arp_halt**
  - **arp_waitstate**
- **invoke-event EVENT-NAME**
  Invokes the specific event manually for the target

12.4 Target Events

At various times, certain things can happen, or you want them to happen.

Examples:

- What should happen when GDB connects? Should your target reset?
- When GDB tries to flash the target, do you need to enable the flash via a special command?
- During reset, do you need to write to certain memory location to reconfigure the SDRAM?

All of the above items are handled by target events.

To specify an event action, either during target creation, or later via "$_TARGETNAME configure" see this example.

Syntactially, the option is: "-event NAME BODY" where NAME is a target event name, and BODY is a Tcl procedure or string of commands to execute.

The programmers model is the "-command" option used in Tcl/Tk buttons and events. Below are two identical examples, the first creates and invokes small procedure. The second inlines the procedure.

```
proc my_attach_proc { } {
    puts "RESET...."
    reset halt
}
mychip.cpu configure -event gdb-attach my_attach_proc
mychip.cpu configure -event gdb-attach { puts "Reset..." ; reset halt }
```

12.5 Current Events

The following events are available:

- **debug-halted**
  The target has halted for debug reasons (i.e.: breakpoint)
- **debug-resumed**
  The target has resumed (i.e.: gdb said run)
- **early-halted**
  Occurs early in the halt process
- **examine-end**
  Currently not used (goal: when JTAG examine completes)
- **examine-start**
  Currently not used (goal: when JTAG examine starts)
- **gdb-attach**
  When GDB connects
- **gdb-detach**
  When GDB disconnects
- **gdb-end**
  When the taret has halted and GDB is not doing anything (see early halt)
- **gdb-flash-erase-start**
  Before the GDB flash process tries to erase the flash
- **gdb-flash-erase-end**
  After the GDB flash process has finished erasing the flash
- **gdb-flash-write-start**
  Before GDB writes to the flash
- **gdb-flash-write-end**
  After GDB writes to the flash
- **gdb-start**
  Before the taret steps, gdb is trying to start/resume the target
- **halted**
  The target has halted
- **old-gdb_program_config**
  DO NOT USE THIS: Used internally
- **old-pre_resume**
  DO NOT USE THIS: Used internally
- **reset-assert-pre**
  Before reset is asserted on the tap.
- **reset-assert-post**
  Reset is now asserted on the tap.
- **reset-deassert-pre**
  Reset is about to be released on the tap
- **reset-deassert-post**
  Reset has been released on the tap

- **reset-end**
  Currently not used.
- **reset-halt-post**
  Currently not usd
- **reset-halt-pre**
  Currently not used
- **reset-init**
  Currently not used
- **reset-start**
  Currently not used
- **reset-wait-pos**
  Currently not used
- **reset-wait-pre**
  Currently not used
- **resume-start**
  Before any target is resumed
- **resume-end**
  After all targets have resumed
- **resume-ok**
  Success
- **resumed**
  Target has resumed
- **tap-enable**
  Executed by **jtag tapenable DOTTED.NAME** command. Example:

```
jtag configure DOTTED.NAME -event tap-enable {
  puts "Enabling CPU"
  ...
}
```

- **tap-disable**
  Executed by **jtag tapdisable DOTTED.NAME** command. Example:

```
jtag configure DOTTED.NAME -event tap-disable {
  puts "Disabling CPU"
  ...
}
```

12.6 target create

target create <*NAME*> <*TYPE*> <*PARAMS ...*>

This command creates a GDB debug target that refers to a specific JTAG tap.

- **NAME**
  Is the name of the debug target. By convention it should be the tap DOTTED.NAME, this name is also used to create the target object command.

- **TYPE**
  Specifies the target type, i.e.: ARM7TDMI, or Cortex-M3. Currently supported targets are:
  - **arm7tdmi**
  - **arm720t**
  - **arm9tdmi**
  - **arm920t**
  - **arm922t**
  - **arm926ejs**
  - **arm966e**
  - **cortex_m3**
  - **feroceon**
  - **xscale**
  - **arm11**
  - **mips_m4k**
- **PARAMS**
  PARAMs are various target configuration parameters. The following ones are mandatory:
  - **-endian big|little**
  - **-chain-position DOTTED.NAME**

12.7 Target Config/Cget Options

These options can be specified when the target is created, or later via the configure option or to query the target via cget.

- **-type** - returns the target type
- **-event NAME BODY** see Target events
- **-work-area-virt** [**ADDRESS**] specify/set the work area
- **-work-area-phys** [**ADDRESS**] specify/set the work area
- **-work-area-size** [**ADDRESS**] specify/set the work area
- **-work-area-backup** [**0|1**] does the work area get backed up
- **-endian** [**big|little**]
- **-variant** [**NAME**] some chips have variants OpenOCD needs to know about
- **-chain-position DOTTED.NAME** the tap name this target refers to.

Example:

```
for { set x 0 } { $x < [target count] } { incr x } {
  set name [target number $x]
  set y [$name cget -endian]
  set z [$name cget -type]
  puts [format "Chip %d is %s, Endian: %s, type: %s" $x $y $z]
}
```

12.8 Target Variants

- **arm7tdmi**
  Unknown (please write me)
- **arm720t**
  Unknown (please write me) (similar to arm7tdmi)
- **arm9tdmi**
  Variants: '`arm920t`', '`arm922t`' and '`arm940t`' This enables the hardware single-stepping support found on these cores.
- **arm920t**
  None.
- **arm966e**
  None (this is also used as the ARM946)
- **cortex_m3**
  use variant <-*variant lm3s*> when debugging Luminary lm3s targets. This will cause OpenOCD to use a software reset rather than asserting SRST to avoid a issue with clearing the debug registers. This is fixed in Fury Rev B, DustDevil Rev B, Tempest, these revisions will be detected and the normal reset behaviour used.
- **xscale**
  Supported variants are '`ixp42x`', '`ixp45x`', '`ixp46x`','`pxa250`', '`pxa255`', '`pxa26x`'.
- **arm11**
  Supported variants are '`arm1136`', '`arm1156`', '`arm1176`'
- **mips_m4k**
  Use variant '`ejtag_srst`' when debugging targets that do not provide a functional SRST line on the EJTAG connector. This causes OpenOCD to instead use an EJTAG software reset command to reset the processor. You still need to enable '`srst`' on the reset configuration command to enable OpenOCD hardware reset functionality.

12.9 working_area - Command Removed

Please use the "$_TARGETNAME configure -work-area-... parameters instead
This documentation remains because there are existing scripts that still use this that need to be converted.

```
working_area target# address  size backup| [virtualaddress]
```

The target# is a the 0 based target numerical index.

This command specifies a working area for the debugger to use. This may be used to speed-up downloads to target memory and flash operations, or to perform otherwise unavailable operations (some coprocessor operations on ARM7/9 systems, for example). The last parameter decides whether the memory should be preserved (<*backup*>) or can simply be overwritten (<*nobackup*>). If possible, use a working_area that doesn't need to be backed up, as performing a backup slows down operation.

13 Flash programming

Note: As of 28/nov/2008 OpenOCD does not know how to program a SPI flash that a micro may boot from. Perhaps you, the reader, would like to contribute support for this.

Flash Steps:

1. Configure via the command **flash bank**
   Normally this is done in a configuration file.
2. Operate on the flash via **flash SOMECOMMAND**
   Often commands to manipulate the flash are typed by a human, or run via a script in some automated way. For example: To program the boot flash on your board.
3. GDB Flashing
   Flashing via GDB requires the flash be configured via "flash bank", and the GDB flash features be enabled. See the daemon configuration section for more details.

13.1 Flash commands

13.1.1 flash banks

flash banks
List configured flash banks
NOTE: the singular form: 'flash bank' is used to configure the flash banks.

13.1.2 flash info

flash info <*num*>
Print info about flash bank <'`num`'>

13.1.3 flash probe

flash probe <*num*>
Identify the flash, or validate the parameters of the configured flash. Operation depends on the flash type.

13.1.4 flash erase_check

flash erase_check <*num*>
Check erase state of sectors in flash bank <*num*>. This is the only operation that updates the erase state information displayed by '`flash info`'. That means you have to issue an '`erase_check`' command after erasing or programming the device to get updated information.

13.1.5 flash protect_check

flash protect_check <*num*>
Check protection state of sectors in flash bank <num>. '`flash erase_sector`' using the same syntax.

13.1.6 flash erase_sector

flash erase_sector <*num*> <*first*> <*last*>
Erase sectors at bank <*num*>, starting at sector <*first*> up to and including <*last*>. Sector numbering starts at 0. Depending on the flash type, erasing may require the protection to be disabled first (e.g. Intel Advanced Bootblock flash using the CFI driver).

13.1.7 flash erase_address

flash erase_address <*address*> <*length*>
Erase sectors starting at <*address*> for <*length*> bytes

13.1.8 flash write_bank

flash write_bank <*num*> <*file*> <*offset*>
Write the binary <*file*> to flash bank <*num*>, starting at <'`offset`'> bytes from the beginning of the bank.

13.1.9 flash write_image

flash write_image [*erase*] <*file*> [*offset*] [*type*]
Write the image <*file*> to the current target's flash bank(s). A relocation [*offset*] can be specified and the file [*type*] can be specified explicitly as '`bin`' (binary), '`ihex`' (Intel hex), '`elf`' (ELF file) or '`s19`' (Motorola s19). Flash memory will be erased prior to programming if the '`erase`' parameter is given.

13.1.10 flash protect

flash protect <*num*> <*first*> <*last*> <'`on`'|'`off`'>
Enable (*on*) or disable (*off*) protection of flash sectors <*first*> to <*last*> of '`flash bank`' <*num*>.

13.1.11 mFlash commands

- **mflash probe** Probe mflash.
- **mflash write** <*num*> <*file*> <*offset*> Write the binary <*file*> to mflash bank <*num*>, starting at <*offset*> bytes from the beginning of the bank.
- **mflash dump** <*num*> <*file*> <*offset*> <*size*> Dump <size> bytes, starting at <*offset*> bytes from the beginning of the <*num*> bank to a <*file*>.

13.2 flash bank command

The **flash bank** command is used to configure one or more flash chips (or banks in OpenOCD terms)

> **flash bank** <*driver*> <*base*> <*size*> <*chip_width*>
> <*bus_width*> <*target#*> [*driver_options ...*]

Configures a flash bank at <*base*> of <*size*> bytes and <*chip_width*> and <*bus_width*> bytes using the selected flash <driver>.

13.2.1 External Flash - cfi options

CFI flashes are external flash chips - often they are connected to a specific chip select on the CPU. By default, at hard reset, most CPUs have the ablity to "boot" from some flash chip - typically attached to the CPU's CS0 pin.

For other chip selects: OpenOCD does not know how to configure, or access a specific chip select. Instead you, the human, might need to configure additional chip selects via other commands (like: mww) , or perhaps configure a GPIO pin that controls the "write protect" pin on the flash chip.

flash bank cfi <*base*> <*size*> <*chip_width*> <*bus_width*> <*target#*> [*jedec_probe*|*x16_as_x8*]

CFI flashes require the number of the target they're connected to as an additional argument. The CFI driver makes use of a working area (specified for the target) to significantly speed up operation.

chip_width and *bus_width* are specified in bytes.

The *jedec_probe* option is used to detect certain non-CFI flash ROMs, like AM29LV010 and similar types.

x16_as_x8 ???

13.2.2 Internal Flash (Microcontrollers)

13.2.2.1 lpc2000 options

flash bank lpc2000 <*base*> <*size*> 0 0 <*target#*> <*variant*> <*clock*> [*calc_checksum*]
LPC flashes don't require the chip and bus width to be specified. Additional parameters are the <*variant*>, which may be *lpc2000_v1* (older LPC21xx and LPC22xx) or *lpc2000_v2* (LPC213x, LPC214x, LPC210[123], LPC23xx and LPC24xx), the number of the target this flash belongs to (first is 0), the frequency at which the core is currently running (in kHz - must be an integral number), and the optional keyword *calc_checksum*, telling the driver to calculate a valid checksum for the exception vector table.

13.2.2.2 at91sam7 options

flash bank at91sam7 0 0 0 0 <*target#*>
AT91SAM7 flashes only require the *target#*, all other values are looked up after reading the chip-id and type.

13.2.2.3 str7 options

flash bank str7x <*base*> <*size*> 0 0 <*target#*> <*variant*>
variant can be either STR71x, STR73x or STR75x.

13.2.2.4 str9 options

flash bank str9x <*base*> <*size*> 0 0 <*target#*>
The str9 needs the flash controller to be configured prior to Flash programming, e.g.

```
str9x flash_config 0 4 2 0 0x80000
```

This will setup the BBSR, NBBSR, BBADR and NBBADR registers respectively.

13.2.2.5 str9 options (str9xpec driver)

flash bank str9xpec <*base*> <*size*> 0 0 <*target#*>
Before using the flash commands the turbo mode must be enabled using str9xpec '`enable_turbo`' <*num*>.

Only use this driver for locking/unlocking the device or configuring the option bytes. Use the standard str9 driver for programming. See [STR9 specific commands], page 43.

13.2.2.6 Stellaris (LM3Sxxx) options

flash bank stellaris <*base*> <*size*> 0 0 <*target#*>
Stellaris flash plugin only require the *target#*.

13.2.2.7 stm32x options

flash bank stm32x <*base*> <*size*> 0 0 <*target#*>
stm32x flash plugin only require the *target#*.

13.2.2.8 aduc702x options

flash bank aduc702x 0 0 0 0 <*target#*>
The aduc702x flash plugin works with Analog Devices model numbers ADUC7019 through ADUC7028. The setup command only requires the *target#* argument (all devices in this family have the same memory layout).

13.2.3 mFlash Configuration

mflash bank <*soc*> <*base*> <*chip_width*> <*bus_width*> <*RST pin*> <*WP pin*> <*DPD pin*> <*target #*>
Configures a mflash for <*soc*> host bank at <*base*>. <*chip_width*> and <*bus_width*> are bytes order. Pin number format is dependent on host GPIO calling convention. If WP or DPD pin was not used, write -1. Currently, mflash bank support s3c2440 and pxa270.

(ex. of s3c2440) mflash <*RST pin*> is GPIO B1, <*WP pin*> and <*DPD pin*> are not used.

```
mflash bank s3c2440 0x10000000 2 2 1b -1 -1 0
```

(ex. of pxa270) mflash <*RST pin*> is GPIO 43, <*DPD pin*> is not used and <*DPD pin*> is GPIO 51.

```
mflash bank pxa270 0x08000000 2 2 43 -1 51 0
```

13.3 Microcontroller specific Flash Commands

13.3.1 AT91SAM7 specific commands

The flash configuration is deduced from the chip identification register. The flash controller handles erases automatically on a page (128/265 byte) basis, so erase is not necessary for flash programming. AT91SAM7 processors with less than 512K flash only have a single flash bank embedded on chip. AT91SAM7xx512 have two flash planes that can be erased separatly. Only an EraseAll command is supported by the controller for each flash plane and this is called with

- **flash erase** <*num*> *first_plane last_plane*
  bulk erase flash planes first_plane to last_plane.

- **at91sam7 gpnvm** <*num*> <*bit*> <'set'|'clear'>
  set or clear a gpnvm bit for the processor

13.3.2 STR9 specific commands

These are flash specific commands when using the str9xpec driver.

- **str9xpec enable_turbo** <*num*>
  enable turbo mode, will simply remove the str9 from the chain and talk directly to the embedded flash controller.
- **str9xpec disable_turbo** <*num*>
  restore the str9 into JTAG chain.
- **str9xpec lock** <*num*>
  lock str9 device. The str9 will only respond to an unlock command that will erase the device.
- **str9xpec unlock** <*num*>
  unlock str9 device.
- **str9xpec options_read** <*num*>
  read str9 option bytes.
- **str9xpec options_write** <*num*>
  write str9 option bytes.

Note: Before using the str9xpec driver here is some background info to help you better understand how the drivers works. OpenOCD has two flash drivers for the str9.

1. Standard driver '`str9x`' programmed via the str9 core. Normally used for flash programming as it is faster than the '`str9xpec`' driver.
2. Direct programming '`str9xpec`' using the flash controller. This is an ISC compilant (IEEE 1532) tap connected in series with the str9 core. The str9 core does not need to be running to program using this flash driver. Typical use for this driver is locking/unlocking the target and programming the option bytes.

Before we run any commands using the '`str9xpec`' driver we must first disable the str9 core. This example assumes the '`str9xpec`' driver has been configured for flash bank 0.

```
# assert srst, we do not want core running
# while accessing str9xpec flash driver
jtag_reset 0 1
# turn off target polling
poll off
# disable str9 core
str9xpec enable_turbo 0
# read option bytes
str9xpec options_read 0
# re-enable str9 core
str9xpec disable_turbo 0
poll on
reset halt
```

The above example will read the str9 option bytes. When performing a unlock remember that you will not be able to halt the str9 - it has been locked. Halting the core is not required

for the 'str9xpec' driver as mentioned above, just issue the commands above manually or from a telnet prompt.

13.3.3 STR9 configuration

- **str9x flash_config** <*bank*> <*BBSR*> <*NBBSR*> <*BBADR*> <*NBBADR*>
  Configure str9 flash controller.

```
e.g. str9x flash_config 0 4 2 0 0x80000
This will setup
BBSR - Boot Bank Size register
NBBSR - Non Boot Bank Size register
BBADR - Boot Bank Start Address register
NBBADR - Boot Bank Start Address register
```

13.3.4 STR9 option byte configuration

- **str9xpec options_cmap** <*num*> <'bank0'|'bank1'>
  configure str9 boot bank.
- **str9xpec options_lvdthd** <*num*> <'2.4v'|'2.7v'>
  configure str9 lvd threshold.
- **str9xpec options_lvdsel** <*num*> <'vdd'|'vdd_vddq'>
  configure str9 lvd source.
- **str9xpec options_lvdwarn** <*bank*> <'vdd'|'vdd_vddq'>
  configure str9 lvd reset warning source.

13.3.5 STM32x specific commands

These are flash specific commands when using the stm32x driver.

- **stm32x lock** <*num*>
  lock stm32 device.
- **stm32x unlock** <*num*>
  unlock stm32 device.
- **stm32x options_read** <*num*>
  read stm32 option bytes.
- **stm32x options_write** <*num*> <'SWWDG'|'HWWDG'> <'RSTSTNDBY'|'NORSTSTNDBY'> <'RSTSTOP'|'NORSTSTOP'>
  write stm32 option bytes.
- **stm32x mass_erase** <*num*>
  mass erase flash memory.

13.3.6 Stellaris specific commands

These are flash specific commands when using the Stellaris driver.

- **stellaris mass_erase** <*num*>
  mass erase flash memory.

14 General Commands

The commands documented in this chapter here are common commands that you, as a human, may want to type and see the output of. Configuration type commands are documented elsewhere.

Intent:

- **Source Of Commands**
  OpenOCD commands can occur in a configuration script (discussed elsewhere) or typed manually by a human or supplied programatically, or via one of several TCP/IP Ports.
- **From the human**
  A human should interact with the telnet interface (default port: 4444, or via GDB, default port 3333)

  To issue commands from within a GDB session, use the '`monitor`' command, e.g. use '`monitor poll`' to issue the '`poll`' command. All output is relayed through the GDB session.
- **Machine Interface** The Tcl interface's intent is to be a machine interface. The default Tcl port is 5555.

14.1 Daemon Commands

14.1.1 sleep [*msec*]

Wait for n milliseconds before resuming. Useful in connection with script files (*script* command and *target_script* configuration).

14.1.2 shutdown

Close the OpenOCD daemon, disconnecting all clients (GDB, telnet, other).

14.1.3 debug_level [*n*]

Display or adjust debug level to n<0-3>

14.1.4 fast [*enable*|*disable*]

Default disabled. Set default behaviour of OpenOCD to be "fast and dangerous". For instance ARM7/9 DCC memory downloads and fast memory access will work if the JTAG interface isn't too fast and the core doesn't run at a too low frequency. Note that this option only changes the default and that the indvidual options, like DCC memory downloads, can be enabled and disabled individually.

The target specific "dangerous" optimisation tweaking options may come and go as more robust and user friendly ways are found to ensure maximum throughput and robustness with a minimum of configuration.

Typically the "fast enable" is specified first on the command line:

```
openocd -c "fast enable" -c "interface dummy" -f target/str710.cfg
```

14.1.5 log_output <*file*>

Redirect logging to <file> (default: stderr)

14.1.6 script <*file*>

Execute commands from <file> See also: “source [find FILENAME]”

14.2 Target state handling

14.2.1 power <*on*|*off*>

Turn power switch to target on/off. No arguments: print status. Not all interfaces support this.

14.2.2 reg [‘#’|‘name’] [value]

Access a single register by its number[‘#’] or by its [‘name’]. No arguments: list all available registers for the current target. Number or name argument: display a register. Number or name and value arguments: set register value.

14.2.3 poll [‘on’|‘off’]

Poll the target for its current state. If the target is in debug mode, architecture specific information about the current state is printed. An optional parameter allows continuous polling to be enabled and disabled.

14.2.4 halt [‘ms’]

Send a halt request to the target and wait for it to halt for up to [‘ms’] milliseconds. Default [‘ms’] is 5 seconds if no arg given. Optional arg ‘ms’ is a timeout in milliseconds. Using 0 as the [‘ms’] will stop OpenOCD from waiting.

14.2.5 wait_halt [‘ms’]

Wait for the target to enter debug mode. Optional [‘ms’] is a timeout in milliseconds. Default [‘ms’] is 5 seconds if no arg is given.

14.2.6 resume [*address*]

Resume the target at its current code position, or at an optional address. OpenOCD will wait 5 seconds for the target to resume.

14.2.7 step [*address*]

Single-step the target at its current code position, or at an optional address.

14.2.8 reset [‘run’|‘halt’|‘init’]

Perform a hard-reset. The optional parameter specifies what should happen after the reset.

With no arguments a "reset run" is executed

– **run**
Let the target run.

– **halt**
Immediately halt the target (works only with certain configurations).

– **init**
Immediately halt the target, and execute the reset script (works only with certain configurations)

14.2.9 soft_reset_halt

Requesting target halt and executing a soft reset. This is often used when a target cannot be reset and halted. The target, after reset is released begins to execute code. OpenOCD attempts to stop the CPU and then sets the program counter back to the reset vector. Unfortunately the code that was executed may have left the hardware in an unknown state.

14.3 Memory access commands

14.3.1 meminfo

display available RAM memory.

14.3.2 Memory peek/poke type commands

These commands allow accesses of a specific size to the memory system. Often these are used to configure the current target in some special way. For example - one may need to write certian values to the SDRAM controller to enable SDRAM.

1. To change the current target see the “targets” (plural) command
2. In system level scripts these commands are deprecated, please use the TARGET object versions.

- **mdw** <*addr*> [*count*]
display memory words (32bit)
- **mdh** <*addr*> [*count*]
display memory half-words (16bit)
- **mdb** <*addr*> [*count*]
display memory bytes (8bit)
- **mww** <*addr*> <*value*>
write memory word (32bit)
- **mwh** <*addr*> <*value*>
write memory half-word (16bit)
- **mwb** <*addr*> <*value*>
write memory byte (8bit)

14.4 Image loading commands

14.4.1 load_image

load_image <*file*> <*address*> [‘bin’|‘ihex’|‘elf’]
Load image <*file*> to target memory at <*address*>

14.4.2 fast_load_image

fast_load_image <*file*> <*address*> [‘bin’|‘ihex’|‘elf’]
Normally you should be using **load_image** or GDB load. However, for testing purposes or when I/O overhead is significant(OpenOCD running on an embedded host), storing the image in memory and uploading the image to the target can be a way to upload e.g. multiple debug sessions when the binary does not change. Arguments are the same as **load_image**, but the image is stored in OpenOCD host memory, i.e. does not affect target. This approach is also useful when profiling target programming performance as I/O and target programming can easily be profiled separately.

14.4.3 fast_load

fast_load
Loads an image stored in memory by **fast_load_image** to the current target. Must be preceeded by fast_load_image.

14.4.4 dump_image

dump_image <*file*> <*address*> <*size*>
Dump <*size*> bytes of target memory starting at <*address*> to a (binary) <*file*>.

14.4.5 verify_image

verify_image <*file*> <*address*> [‘bin’|‘ihex’|‘elf’]
Verify <*file*> against target memory starting at <*address*>. This will first attempt a comparison using a CRC checksum, if this fails it will try a binary compare.

14.5 Breakpoint commands

- **bp** <*addr*> <*len*> [*hw*]
  set breakpoint <address> <length> [hw]
- **rbp** <*addr*>
  remove breakpoint <adress>
- **wp** <*addr*> <*len*> <*r|w|a*> [*value*] [*mask*]
  set watchpoint <address> <length> <r/w/a> [value] [mask]
- **rwp** <*addr*>
  remove watchpoint <adress>

14.6 Misc Commands

- **profile** <*seconds*> <*gmon.out*>

Profiling samples the CPU's program counter as quickly as possible, which is useful for non-intrusive stochastic profiling.

14.7 Target Specific Commands

14.8 Architecture Specific Commands

14.8.1 ARMV4/5 specific commands

These commands are specific to ARM architecture v4 and v5, like all ARM7/9 systems or Intel XScale (XScale isn't supported yet).

- **armv4_5 reg**
  Display a list of all banked core registers, fetching the current value from every core mode if necessary. OpenOCD versions before rev. 60 didn't fetch the current register value.
- **armv4_5 core_mode** [*arm*|*thumb*]
  Displays the core_mode, optionally changing it to either ARM or Thumb mode. The target is resumed in the currently set '`core_mode`'.

14.8.2 ARM7/9 specific commands

These commands are specific to ARM7 and ARM9 targets, like ARM7TDMI, ARM720t, ARM920T or ARM926EJ-S.

- **arm7_9 dbgrq** <*enable*|*disable*>
  Enable use of the DBGRQ bit to force entry into debug mode. This should be safe for all but ARM7TDMI–S cores (like Philips LPC).
- **arm7_9 fast_memory_access** <*enable*|*disable*>
  Allow OpenOCD to read and write memory without checking completion of the operation. This provides a huge speed increase, especially with USB JTAG cables (FT2232), but might be unsafe if used with targets running at very low speeds, like the 32kHz startup clock of an AT91RM9200.
- **arm7_9 dcc_downloads** <*enable*|*disable*>
  Enable the use of the debug communications channel (DCC) to write larger (>128 byte) amounts of memory. DCC downloads offer a huge speed increase, but might be potentially unsafe, especially with targets running at very low speeds. This command was introduced with OpenOCD rev. 60.

14.8.3 ARM720T specific commands

- **arm720t cp15** <*num*> [*value*]
  display/modify cp15 register <'`num`'> ['`value`'].
- **arm720t md<bhw>_phys** <*addr*> [*count*]
  Display memory at physical address addr.
- **arm720t mw<bhw>_phys** <*addr*> <*value*>
  Write memory at physical address addr.
- **arm720t virt2phys** <*va*>
  Translate a virtual address to a physical address.

14.8.4 ARM9TDMI specific commands

- **arm9tdmi vector_catch** <*all*|*none*>
  Catch arm9 interrupt vectors, can be '`all`' '`none`' or any of the following: '`reset`' '`undef`' '`swi`' '`pabt`' '`dabt`' '`reserved`' '`irq`' '`fiq`'.

Can also be used on other ARM9 based cores such as ARM966, ARM920T and ARM926EJ-S.

14.8.5 ARM966E specific commands

- **arm966e cp15** <*num*> [*value*]
  display/modify cp15 register <‘`num`’> [‘`value`’].

14.8.6 ARM920T specific commands

- **arm920t cp15** <*num*> [*value*]
  display/modify cp15 register <‘`num`’> [‘`value`’].
- **arm920t cp15i** <*num*> [*value*] [*address*]
  display/modify cp15 (interpreted access) <‘`opcode`’> [‘`value`’] [‘`address`’]
- **arm920t cache_info**
  Print information about the caches found. This allows to see whether your target is an ARM920T (2x16kByte cache) or ARM922T (2x8kByte cache).
- **arm920t md<bhw>_phys** <*addr*> [*count*]
  Display memory at physical address addr.
- **arm920t mw<bhw>_phys** <*addr*> <*value*>
  Write memory at physical address addr.
- **arm920t read_cache** <*filename*>
  Dump the content of ICache and DCache to a file.
- **arm920t read_mmu** <*filename*>
  Dump the content of the ITLB and DTLB to a file.
- **arm920t virt2phys** <*va*>
  Translate a virtual address to a physical address.

14.8.7 ARM926EJ-S specific commands

- **arm926ejs cp15** <*num*> [*value*]
  display/modify cp15 register <‘`num`’> [‘`value`’].
- **arm926ejs cache_info**
  Print information about the caches found.
- **arm926ejs md<bhw>_phys** <*addr*> [*count*]
  Display memory at physical address addr.
- **arm926ejs mw<bhw>_phys** <*addr*> <*value*>
  Write memory at physical address addr.
- **arm926ejs virt2phys** <*va*>
  Translate a virtual address to a physical address.

14.8.8 CORTEX_M3 specific commands

- **cortex_m3 maskisr** <*on*|*off*>
  Enable masking (disabling) interrupts during target step/resume.

14.9 Debug commands

The following commands give direct access to the core, and are most likely only useful while debugging OpenOCD.

- **arm7_9 write_xpsr** <*32-bit value*> <'`0=cpsr`', '`1=spsr`'>
  Immediately write either the current program status register (CPSR) or the saved program status register (SPSR), without changing the register cache (as displayed by the '`reg`' and '`armv4_5 reg`' commands).
- **arm7_9 write_xpsr_im8** <*8-bit value*> <*rotate 4-bit*> <*0=cpsr,1=spsr*>
  Write the 8-bit value rotated right by 2*rotate bits, using an immediate write operation (similar to '`write_xpsr`').
- **arm7_9 write_core_reg** <*num*> <*mode*> <*value*>
  Write a core register, without changing the register cache (as displayed by the '`reg`' and '`armv4_5 reg`' commands). The <*mode*> argument takes the encoding of the [M4:M0] bits of the PSR.

14.10 Target Requests

OpenOCD can handle certain target requests, currently debugmsg are only supported for arm7_9 and cortex_m3. See libdcc in the contrib dir for more details.

- **target_request debugmsgs** <*enable*|*disable*|*charmsg*>
  Enable/disable target debugmsgs requests. debugmsgs enable messages to be sent to the debugger while the target is running. *charmsg* receives messages if Linux kernel "Kernel low-level debugging via EmbeddedICE DCC channel" option is enabled.

15 JTAG Commands

Generally most people will not use the bulk of these commands. They are mostly used by the OpenOCD developers or those who need to directly manipulate the JTAG taps.

In general these commands control JTAG taps at a very low level. For example if you need to control a JTAG Route Controller (i.e.: the OMAP3530 on the Beagle Board has one) you might use these commands in a script or an event procedure.

15.1 Commands

- **scan_chain**
  Print current scan chain configuration.
- **jtag_reset** <*trst*> <*srst*>
  Toggle reset lines.
- **endstate** <*tap_state*>
  Finish JTAG operations in <*tap_state*>.
- **runtest** <*num_cycles*>
  Move to Run-Test/Idle, and execute <*num_cycles*>
- **statemove** [*tap_state*]
  Move to current endstate or [*tap_state*]
- **irscan** <*device*> <*instr*> [*dev2*] [*instr2*] ...
  Execute IR scan <*device*> <*instr*> [*dev2*] [*instr2*] ...
- **drscan** <*device*> [*dev2*] [*var2*] ...
  Execute DR scan <*device*> [*dev2*] [*var2*] ...
- **verify_ircapture** <'`enable`'|'`disable`'>
  Verify value captured during Capture-IR. Default is enabled.
- **var** <*name*> [*num_fields*|*del*] [*size1*] ...
  Allocate, display or delete variable <*name*> [*num_fields*|*del*] [*size1*] ...
- **field** <*var*> <*field*> [*value*|*flip*] Display/modify variable field <*var*> <*field*> [*value*|*flip*].

15.2 Tap states

Available tap_states are:

- **RESET**
- **IDLE**
- **DRSELECT**
- **DRCAPTURE**
- **DRSHIFT**
- **DREXIT1**
- **DRPAUSE**
- **DREXIT2**
- **DRUPDATE**
- **IRSELECT**

- **IRCAPTURE**
- **IRSHIFT**
- **IREXIT1**
- **IRPAUSE**
- **IREXIT2**
- **IRUPDATE**

16 TFTP

If OpenOCD runs on an embedded host(as ZY1000 does), then TFTP can be used to access files on PCs (either the developer's PC or some other PC).

The way this works on the ZY1000 is to prefix a filename by "/tftp/ip/" and append the TFTP path on the TFTP server (tftpd). E.g. "load_image /tftp/10.0.0.96/c:\temp\abc.elf" will load c:\temp\abc.elf from the developer pc (10.0.0.96) into memory as if the file was hosted on the embedded host.

In order to achieve decent performance, you must choose a TFTP server that supports a packet size bigger than the default packet size (512 bytes). There are numerous TFTP servers out there (free and commercial) and you will have to do a bit of googling to find something that fits your requirements.

17 Sample Scripts

This page shows how to use the Target Library.

The configuration script can be divided into the following sections:

- Daemon configuration
- Interface
- JTAG scan chain
- Target configuration
- Flash configuration

Detailed information about each section can be found at OpenOCD configuration.

17.1 AT91R40008 example

To start OpenOCD with a target script for the AT91R40008 CPU and reset the CPU upon startup of the OpenOCD daemon.

```
openocd -f interface/parport.cfg -f target/at91r40008.cfg -c init -c reset
```

18 GDB and OpenOCD

OpenOCD complies with the remote gdbserver protocol, and as such can be used to debug remote targets.

18.1 Connecting to GDB

Use GDB 6.7 or newer with OpenOCD if you run into trouble. For instance GDB 6.3 has a known bug that produces bogus memory access errors, which has since been fixed: look up 1836 in http://sourceware.org/cgi-bin/gnatsweb.pl?database=gdb

OpenOCD can communicate with GDB in two ways:

1. A socket (TCP/IP) connection is typically started as follows:

   ```
   target remote localhost:3333
   ```

   This would cause GDB to connect to the gdbserver on the local pc using port 3333.
2. A pipe connection is typically started as follows:

   ```
   target remote | openocd --pipe
   ```

   This would cause GDB to run OpenOCD and communicate using pipes (stdin/stdout). Using this method has the advantage of GDB starting/stopping OpenOCD for the debug session.

To see a list of available OpenOCD commands type '`monitor help`' on the GDB command line.

OpenOCD supports the gdb '`qSupported`' packet, this enables information to be sent by the GDB remote server (i.e. OpenOCD) to GDB. Typical information includes packet size and the device's memory map.

Previous versions of OpenOCD required the following GDB options to increase the packet size and speed up GDB communication:

```
set remote memory-write-packet-size 1024
set remote memory-write-packet-size fixed
set remote memory-read-packet-size 1024
set remote memory-read-packet-size fixed
```

This is now handled in the '`qSupported`' PacketSize and should not be required.

18.2 Programming using GDB

By default the target memory map is sent to GDB. This can be disabled by the following OpenOCD configuration option:

```
gdb_memory_map disable
```

For this to function correctly a valid flash configuration must also be set in OpenOCD. For faster performance you should also configure a valid working area.

Informing GDB of the memory map of the target will enable GDB to protect any flash areas of the target and use hardware breakpoints by default. This means that the

OpenOCD option '`gdb_breakpoint_override`' is not required when using a memory map. See [gdb_breakpoint_override], page 21.

To view the configured memory map in GDB, use the GDB command '`info mem`' All other unassigned addresses within GDB are treated as RAM.

GDB 6.8 and higher set any memory area not in the memory map as inaccessible. This can be changed to the old behaviour by using the following GDB command

```
set mem inaccessible-by-default off
```

If '`gdb_flash_program enable`' is also used, GDB will be able to program any flash memory using the vFlash interface.

GDB will look at the target memory map when a load command is given, if any areas to be programmed lie within the target flash area the vFlash packets will be used.

If the target needs configuring before GDB programming, an event script can be executed:

```
$_TARGETNAME configure -event EVENTNAME BODY
```

To verify any flash programming the GDB command '`compare-sections`' can be used.

19 Tcl Scripting API

19.1 API rules

The commands are stateless. E.g. the telnet command line has a concept of currently active target, the Tcl API proc's take this sort of state information as an argument to each proc.

There are three main types of return values: single value, name value pair list and lists.

Name value pair. The proc 'foo' below returns a name/value pair list.

```
 >  set foo(me)  Duane
 >  set foo(you) Oyvind
 >  set foo(mouse) Micky
 >  set foo(duck) Donald

If one does this:

 >  set foo

The result is:

    me Duane you Oyvind mouse Micky duck Donald

Thus, to get the names of the associative array is easy:

     foreach  { name value }   [set foo]   {
                puts "Name: $name, Value: $value"
     }
```

Lists returned must be relatively small. Otherwise a range should be passed in to the proc in question.

19.2 Internal low-level Commands

By low-level, the intent is a human would not directly use these commands.

Low-level commands are (should be) prefixed with "openocd_", e.g. openocd_flash_banks is the low level API upon which "flash banks" is implemented.

- **ocd_mem2array** <*varname*> <*width*> <*addr*> <*nelems*>

  Read memory and return as a Tcl array for script processing

- **ocd_array2mem** <*varname*> <*width*> <*addr*> <*nelems*>

  Convert a Tcl array to memory locations and write the values

- **ocd_flash_banks** <*driver*> <*base*> <*size*> <*chip_width*> <*bus_width*> <*target*> ['`driver options`' ...]

  Return information about the flash banks

OpenOCD commands can consist of two words, e.g. "flash banks". The startup.tcl "unknown" proc will translate this into a Tcl proc called "flash_banks".

19.3 OpenOCD specific Global Variables

19.3.1 HostOS

Real Tcl has ::tcl_platform(), and platform::identify, and many other variables. JimTCL, as implemented in OpenOCD creates $HostOS which holds one of the following values:

- **winxx** Built using Microsoft Visual Studio
- **linux** Linux is the underlying operating sytem
- **darwin** Darwin (mac-os) is the underlying operating sytem.
- **cygwin** Running under Cygwin
- **mingw32** Running under MingW32
- **other** Unknown, none of the above.

Note: 'winxx' was choosen because today (March-2009) no distinction is made between Win32 and Win64.

20 Deprecated/Removed Commands

Certain OpenOCD commands have been deprecated/removed during the various revisions.

- **arm7_9 fast_writes**
  use '`arm7_9 fast_memory_access`' command with same args. See [arm7_9 fast_memory_access], page 50.
- **arm7_9 force_hw_bkpts**
  Use '`gdb_breakpoint_override`' instead. Note that GDB will use hardware breakpoints for flash if the GDB memory map has been set up(default when flash is declared in target configuration). See [gdb_breakpoint_override], page 21.
- **arm7_9 sw_bkpts**
  On by default. See also '`gdb_breakpoint_override`'. See [gdb_breakpoint_override], page 21.
- **daemon_startup**
  this config option has been removed, simply adding '`init`' and '`reset halt`' to the end of your config script will give the same behaviour as using '`daemon_startup reset`' and '`target cortex_m3 little reset_halt 0`'.
- **dump_binary**
  use '`dump_image`' command with same args. See [dump_image], page 48.
- **flash erase**
  use '`flash erase_sector`' command with same args. See [flash erase_sector], page 40.
- **flash write**
  use '`flash write_bank`' command with same args. See [flash write_bank], page 40.
- **flash write_binary**
  use '`flash write_bank`' command with same args. See [flash write_bank], page 40.
- **flash auto_erase**
  use '`flash write_image`' command passing '`erase`' as the first parameter. See [flash write_image], page 40.
- **load_binary**
  use '`load_image`' command with same args. See [load_image], page 48.
- **run_and_halt_time**
  This command has been removed for simpler reset behaviour, it can be simulated with the following commands:

  ```
  reset run
  sleep 100
  halt
  ```

- **target** <*type*> <*endian*> <*jtag-position*>
  use the create subcommand of '`target`'.
- **target_script** <*target#*> <*eventname*> <*scriptname*>
  use <*target_name*> configure -event <*eventname*> "script <*scriptname*>"
- **working_area**
  use the '`configure`' subcommand of '`target`' to set the work-area-virt, work-area-phy, work-area-size, and work-area-backup properties of the target.

21 FAQ

1. **RTCK, also known as: Adaptive Clocking - What is it?**

   In digital circuit design it is often refered to as "clock synchronisation" the JTAG interface uses one clock (TCK or TCLK) operating at some speed, your target is operating at another. The two clocks are not synchronised, they are "asynchronous"

   In order for the two to work together they must be synchronised. Otherwise the two systems will get out of sync with each other and nothing will work. There are 2 basic options:

   1. Use a special circuit.
   2. One clock must be some multiple slower than the other.

   Does this really matter? For some chips and some situations, this is a non-issue (i.e.: A 500MHz ARM926) but for others - for example some Atmel SAM7 and SAM9 chips start operation from reset at 32kHz - program/enable the oscillators and eventually the main clock. It is in those critical times you must slow the JTAG clock to sometimes 1 to 4kHz.

   Imagine debugging a 500MHz ARM926 hand held battery powered device that "deep sleeps" at 32kHz between every keystroke. It can be painful.

   Solution #1 - A special circuit

   In order to make use of this, your JTAG dongle must support the RTCK feature. Not all dongles support this - keep reading!

   The RTCK signal often found in some ARM chips is used to help with this problem. ARM has a good description of the problem described at this link: http://www.arm.com/support/faqdev/4170.html [checked 28/nov/2008]. Link title: "How does the JTAG synchronisation logic work? / how does adaptive clocking work?".

   The nice thing about adaptive clocking is that "battery powered hand held device example" - the adaptiveness works perfectly all the time. One can set a break point or halt the system in the deep power down code, slow step out until the system speeds up.

   Solution #2 - Always works - but may be slower

   Often this is a perfectly acceptable solution.

   In most simple terms: Often the JTAG clock must be 1/10 to 1/12 of the target clock speed. But what that "magic division" is varies depending on the chips on your board. **ARM rule of thumb** Most ARM based systems require an 8:1 division. **Xilinx rule of thumb** is 1/12 the clock speed.

   Note: Many FTDI2232C based JTAG dongles are limited to 6MHz.

   You can still debug the 'low power' situations - you just need to manually adjust the clock speed at every step. While painful and tedious, it is not always practical.

   It is however easy to "code your way around it" - i.e.: Cheat a little, have a special debug mode in your application that does a "high power sleep". If you are careful - 98% of your problems can be debugged this way.

   To set the JTAG frequency use the command:

```
# Example: 1.234MHz
jtag_khz 1234
```

2. **Win32 Pathnames** Why don't backslashes work in Windows paths?

   OpenOCD uses Tcl and a backslash is an escape char. Use `{` and `}` around Windows filenames.

   ```
   > echo \a

   > echo {\a}
   \a
   > echo "\a"

   >
   ```

3. **Missing: cygwin1.dll** OpenOCD complains about a missing cygwin1.dll.

   Make sure you have Cygwin installed, or at least a version of OpenOCD that claims to come with all the necessary DLLs. When using Cygwin, try launching OpenOCD from the Cygwin shell.

4. **Breakpoint Issue** I'm trying to set a breakpoint using GDB (or a frontend like Insight or Eclipse), but OpenOCD complains that "Info: arm7_9_common.c:213 arm7_9_add_breakpoint(): sw breakpoint requested, but software breakpoints not enabled".

   GDB issues software breakpoints when a normal breakpoint is requested, or to implement source-line single-stepping. On ARMv4T systems, like ARM7TDMI, ARM720T or ARM920T, software breakpoints consume one of the two available hardware breakpoints.

5. **LPC2000 Flash** When erasing or writing LPC2000 on-chip flash, the operation fails at random.

   Make sure the core frequency specified in the '`flash lpc2000`' line matches the clock at the time you're programming the flash. If you've specified the crystal's frequency, make sure the PLL is disabled. If you've specified the full core speed (e.g. 60MHz), make sure the PLL is enabled.

6. **Amontec Chameleon** When debugging using an Amontec Chameleon in its JTAG Accelerator configuration, I keep getting "Error: amt_jtagaccel.c:184 amt_wait_scan_busy(): amt_jtagaccel timed out while waiting for end of scan, rtck was disabled".

   Make sure your PC's parallel port operates in EPP mode. You might have to try several settings in your PC BIOS (ECP, EPP, and different versions of those).

7. **Data Aborts** When debugging with OpenOCD and GDB (plain GDB, Insight, or Eclipse), I get lots of "Error: arm7_9_common.c:1771 arm7_9_read_memory(): memory read caused data abort".

   The errors are non-fatal, and are the result of GDB trying to trace stack frames beyond the last valid frame. It might be possible to prevent this by setting up a proper "initial" stack frame, if you happen to know what exactly has to be done, feel free to add this here.

   Simple: In your startup code - push 8 registers of zeros onto the stack before calling main(). What GDB is doing is "climbing" the run time stack by reading various values

on the stack using the standard call frame for the target. GDB keeps going - until one of 2 things happen **#1** an invalid frame is found, or **#2** some huge number of stackframes have been processed. By pushing zeros on the stack, GDB gracefully stops.

Debugging Interrupt Service Routines - In your ISR before you call your C code, do the same - artifically push some zeros onto the stack, remember to pop them off when the ISR is done.

Also note: If you have a multi-threaded operating system, they often do not **in the intrest of saving memory** waste these few bytes. Painful...

8. **JTAG Reset Config** I get the following message in the OpenOCD console (or log file): "Warning: arm7_9_common.c:679 arm7_9_assert_reset(): srst resets test logic, too".

   This warning doesn't indicate any serious problem, as long as you don't want to debug your core right out of reset. Your .cfg file specified '`jtag_reset trst_and_srst srst_pulls_trst`' to tell OpenOCD that either your board, your debugger or your target uC (e.g. LPC2000) can't assert the two reset signals independently. With this setup, it's not possible to halt the core right out of reset, everything else should work fine.

9. **USB Power** When using OpenOCD in conjunction with Amontec JTAGkey and the Yagarto toolchain (Eclipse, arm-elf-gcc, arm-elf-gdb), the debugging seems to be unstable. When single-stepping over large blocks of code, GDB and OpenOCD quit with an error message. Is there a stability issue with OpenOCD?

   No, this is not a stability issue concerning OpenOCD. Most users have solved this issue by simply using a self-powered USB hub, which they connect their Amontec JTAGkey to. Apparently, some computers do not provide a USB power supply stable enough for the Amontec JTAGkey to be operated.

   Laptops running on battery have this problem too...

10. **USB Power** When using the Amontec JTAGkey, sometimes OpenOCD crashes with the following error messages: "Error: ft2232.c:201 ft2232_read(): FT_Read returned: 4" and "Error: ft2232.c:365 ft2232_send_and_recv(): couldn't read from FT2232". What does that mean and what might be the reason for this?

    First of all, the reason might be the USB power supply. Try using a self-powered hub instead of a direct connection to your computer. Secondly, the error code 4 corresponds to an FT_IO_ERROR, which means that the driver for the FTDI USB chip ran into some sort of error - this points us to a USB problem.

11. **GDB Disconnects** When using the Amontec JTAGkey, sometimes OpenOCD crashes with the following error message: "Error: gdb_server.c:101 gdb_get_char(): read: 10054". What does that mean and what might be the reason for this?

    Error code 10054 corresponds to WSAECONNRESET, which means that the debugger (GDB) has closed the connection to OpenOCD. This might be a GDB issue.

12. **LPC2000 Flash** In the configuration file in the section where flash device configurations are described, there is a parameter for specifying the clock frequency for LPC2000 internal flash devices (e.g. '`flash bank lpc2000 0x0 0x40000 0 0 0 lpc2000_v1 14746 calc_checksum`'), which must be specified in kilohertz. However, I do have a quartz crystal of a frequency that contains fractions of kilohertz (e.g. 14,745,600 Hz, i.e. 14,745.600 kHz). Is it possible to specify real numbers for the clock frequency?

No. The clock frequency specified here must be given as an integral number. However, this clock frequency is used by the In-Application-Programming (IAP) routines of the LPC2000 family only, which seems to be very tolerant concerning the given clock frequency, so a slight difference between the specified clock frequency and the actual clock frequency will not cause any trouble.

13. **Command Order** Do I have to keep a specific order for the commands in the configuration file?

    Well, yes and no. Commands can be given in arbitrary order, yet the devices listed for the JTAG scan chain must be given in the right order (jtag newdevice), with the device closest to the TDO-Pin being listed first. In general, whenever objects of the same type exist which require an index number, then these objects must be given in the right order (jtag newtap, targets and flash banks - a target references a jtag newtap and a flash bank references a target).

    You can use the "scan_chain" command to verify and display the tap order.

14. **JTAG Tap Order** JTAG tap order - command order

    Many newer devices have multiple JTAG taps. For example: ST Microsystems STM32 chips have two taps, a "boundary scan tap" and "Cortex-M3" tap. Example: The STM32 reference manual, Document ID: RM0008, Section 26.5, Figure 259, page 651/681, the "TDI" pin is connected to the boundary scan tap, which then connects to the Cortex-M3 tap, which then connects to the TDO pin.

    Thus, the proper order for the STM32 chip is: (1) The Cortex-M3, then (2) The boundary scan tap. If your board includes an additional JTAG chip in the scan chain (for example a Xilinx CPLD or FPGA) you could place it before or after the STM32 chip in the chain. For example:

    - OpenOCD_TDI(output) -> STM32 TDI Pin (BS Input)
    - STM32 BS TDO (output) -> STM32 Cortex-M3 TDI (input)
    - STM32 Cortex-M3 TDO (output) -> SM32 TDO Pin
    - STM32 TDO Pin (output) -> Xilinx TDI Pin (input)
    - Xilinx TDO Pin -> OpenOCD TDO (input)

    The "jtag device" commands would thus be in the order shown below. Note:

    - jtag newtap Xilinx tap -irlen ...
    - jtag newtap stm32 cpu -irlen ...
    - jtag newtap stm32 bs -irlen ...
    - # Create the debug target and say where it is
    - target create stm32.cpu -chain-position stm32.cpu ...

15. **SYSCOMP** Sometimes my debugging session terminates with an error. When I look into the log file, I can see these error messages: Error: arm7_9_common.c:561 arm7_9_execute_sys_speed(): timeout waiting for SYSCOMP

    TODO.

22 Tcl Crash Course

Not everyone knows Tcl - this is not intended to be a replacement for learning Tcl, the intent of this chapter is to give you some idea of how the Tcl scripts work.

This chapter is written with two audiences in mind. (1) OpenOCD users who need to understand a bit more of how JIM-Tcl works so they can do something useful, and (2) those that want to add a new command to OpenOCD.

22.1 Tcl Rule #1

There is a famous joke, it goes like this:

1. Rule #1: The wife is always correct
2. Rule #2: If you think otherwise, See Rule #1

The Tcl equal is this:

1. Rule #1: Everything is a string
2. Rule #2: If you think otherwise, See Rule #1

As in the famous joke, the consequences of Rule #1 are profound. Once you understand Rule #1, you will understand Tcl.

22.2 Tcl Rule #1b

There is a second pair of rules.

1. Rule #1: Control flow does not exist. Only commands
   For example: the classic FOR loop or IF statement is not a control flow item, they are commands, there is no such thing as control flow in Tcl.
2. Rule #2: If you think otherwise, See Rule #1
   Actually what happens is this: There are commands that by convention, act like control flow key words in other languages. One of those commands is the word "for", another command is "if".

22.3 Per Rule #1 - All Results are strings

Every Tcl command results in a string. The word "result" is used deliberatly. No result is just an empty string. Remember: *Rule #1 - Everything is a string*

22.4 Tcl Quoting Operators

In life of a Tcl script, there are two important periods of time, the difference is subtle.

1. Parse Time
2. Evaluation Time

The two key items here are how "quoted things" work in Tcl. Tcl has three primary quoting constructs, the [square-brackets] the {curly-braces} and "double-quotes"

By now you should know $VARIABLES always start with a $DOLLAR sign. BTW: To set a variable, you actually use the command "set", as in "set VARNAME VALUE" much like the ancient BASIC langauge "let x = 1" statement, but without the equal sign.

- **[square-brackets]**
  [square-brackets] are command substitutions. It operates much like Unix Shell ‘back-ticks‘. The result of a [square-bracket] operation is exactly 1 string. *Remember Rule #1 - Everything is a string.* These two statements are roughly identical:

```
# bash example
X=‘date‘
echo "The Date is: $X"
# Tcl example
set X [date]
puts "The Date is: $X"
```

- **“double-quoted-things”**
  “double-quoted-things” are just simply quoted text. $VARIABLES and [square-brackets] are expanded in place - the result however is exactly 1 string. *Remember Rule #1 - Everything is a string*

```
set x "Dinner"
puts "It is now \"[date]\", $x is in 1 hour"
```

- **{Curly-Braces}**
  {Curly-Braces} are magic: $VARIABLES and [square-brackets] are parsed, but are NOT expanded or executed. {Curly-Braces} are like ’single-quote’ operators in BASH shell scripts, with the added feature: {curly-braces} can be nested, single quotes can not. {{{this is nested 3 times}}} NOTE: [date] is perhaps a bad example, as of 28/nov/2008, Jim/OpenOCD does not have a date command.

22.5 Consequences of Rule 1/2/3/4

The consequences of Rule 1 are profound.

22.5.1 Tokenisation & Execution.

Of course, whitespace, blank lines and #comment lines are handled in the normal way.

As a script is parsed, each (multi) line in the script file is tokenised and according to the quoting rules. After tokenisation, that line is immedatly executed.

Multi line statements end with one or more “still-open” {curly-braces} which - eventually - closes a few lines later.

22.5.2 Command Execution

Remember earlier: There are no “control flow” statements in Tcl. Instead there are COMMANDS that simply act like control flow operators.

Commands are executed like this:

1. Parse the next line into (argc) and (argv[]).
2. Look up (argv[0]) in a table and call its function.
3. Repeat until End Of File.

It sort of works like this:

```
for(;;){
    ReadAndParse( &argc, &argv );
```

```
        cmdPtr = LookupCommand( argv[0] );

        (*cmdPtr->Execute)( argc, argv );
    }
```

When the command "proc" is parsed (which creates a procedure function) it gets 3 parameters on the command line. **1** the name of the proc (function), **2** the list of parameters, and **3** the body of the function. Not the choice of words: LIST and BODY. The PROC command stores these items in a table somewhere so it can be found by "LookupCommand()"

22.5.3 The FOR command

The most interesting command to look at is the FOR command. In Tcl, the FOR command is normally implemented in C. Remember, FOR is a command just like any other command.

When the ascii text containing the FOR command is parsed, the parser produces 5 parameter strings, *(If in doubt: Refer to Rule #1)* they are:

0. The ascii text 'for'
1. The start text
2. The test expression
3. The next text
4. The body text

Sort of reminds you of "main(int argc, char **argv)" does it not? Remember *Rule #1 - Everything is a string.* The key point is this: Often many of those parameters are in {curly-braces} - thus the variables inside are not expanded or replaced until later.

Remember that every Tcl command looks like the classic "main(argc, argv)" function in C. In JimTCL - they actually look like this:

```
int
MyCommand( Jim_Interp *interp,
           int *argc,
           Jim_Obj * const *argvs );
```

Real Tcl is nearly identical. Although the newer versions have introduced a byte-code parser and intepreter, but at the core, it still operates in the same basic way.

22.5.4 FOR command implementation

To understand Tcl it is perhaps most helpful to see the FOR command. Remember, it is a COMMAND not a control flow structure.

In Tcl there are two underlying C helper functions.

Remember Rule #1 - You are a string.

The **first** helper parses and executes commands found in an ascii string. Commands can be seperated by semicolons, or newlines. While parsing, variables are expanded via the quoting rules.

The **second** helper evaluates an ascii string as a numerical expression and returns a value.

Here is an example of how the **FOR** command could be implemented. The pseudo code below does not show error handling.

```
void Execute_AsciiString( void *interp, const char *string );

int Evaluate_AsciiExpression( void *interp, const char *string );

int
MyForCommand( void *interp,
              int argc,
              char **argv )
{
   if( argc != 5 ){
       SetResult( interp, "WRONG number of parameters");
       return ERROR;
   }

   // argv[0] = the ascii string just like C

   // Execute the start statement.
   Execute_AsciiString( interp, argv[1] );

   // Top of loop test
   for(;;){
        i = Evaluate_AsciiExpression(interp, argv[2]);
        if( i == 0 )
            break;

        // Execute the body
        Execute_AsciiString( interp, argv[3] );

        // Execute the LOOP  part
        Execute_AsciiString( interp, argv[4] );
    }

    // Return no error
    SetResult( interp, "" );
    return SUCCESS;
}
```

Every other command IF, WHILE, FORMAT, PUTS, EXPR, everything works in the same basic way.

22.6 OpenOCD Tcl Usage

22.6.1 source and find commands

Where: In many configuration files
Example: **source [find FILENAME]**
Remember the parsing rules

1. The FIND command is in square brackets.

The FIND command is executed with the parameter FILENAME. It should find the full path to the named file. The RESULT is a string, which is substituted on the orginal command line.

2. The command source is executed with the resulting filename.
   SOURCE reads a file and executes as a script.

22.6.2 format command

Where: Generally occurs in numerous places.
Tcl has no command like **printf()**, instead it has **format**, which is really more like **sprintf()**.
Example

```
set x 6
set y 7
puts [format "The answer: %d" [expr $x * $y]]
```

1. The SET command creates 2 variables, X and Y.
2. The double [nested] EXPR command performs math
   The EXPR command produces numerical result as a string.
   Refer to Rule #1
3. The format command is executed, producing a single string
   Refer to Rule #1.
4. The PUTS command outputs the text.

22.6.3 Body or Inlined Text

Where: Various TARGET scripts.

```
#1 Good
   proc someproc {} {
       ... multiple lines of stuff ...
   }
   $_TARGETNAME configure -event FOO  someproc
#2 Good - no variables
   $_TARGETNAME confgure -event foo "this ; that;"
#3 Good Curly Braces
   $_TARGETNAME configure -event FOO {
        puts "Time: [date]"
   }
#4 DANGER DANGER DANGER
   $_TARGETNAME configure -event foo "puts \"Time: [date]\""
```

1. The $_TARGETNAME is an OpenOCD variable convention.
   $_TARGETNAME represents the last target created, the value changes each time a new target is created. Remember the parsing rules. When the ascii text is parsed, the **$_TARGETNAME** becomes a simple string, the name of the target which happens to be a TARGET (object) command.
2. The 2nd parameter to the '`-event`' parameter is a TCBODY
   There are 4 examples:
   1. The TCLBODY is a simple string that happens to be a proc name

2. The TCLBODY is several simple commands seperated by semicolons
3. The TCLBODY is a multi-line {curly-brace} quoted string
4. The TCLBODY is a string with variables that get expanded.

In the end, when the target event FOO occurs the TCLBODY is evaluated. Method **#1** and **#2** are functionally identical. For Method **#3** and **#4** it is more interesting. What is the TCLBODY?

Remember the parsing rules. In case #3, {curly-braces} mean the $VARS and [square-brackets] are expanded later, when the EVENT occurs, and the text is evaluated. In case #4, they are replaced before the "Target Object Command" is executed. This occurs at the same time $_TARGETNAME is replaced. In case #4 the date will never change. {BTW: [date] is perhaps a bad example, as of 28/nov/2008, Jim/OpenOCD does not have a date command}

22.6.4 Global Variables

Where: You might discover this when writing your own procs
In simple terms: Inside a PROC, if you need to access a global variable you must say so. See also "upvar". Example:

```
proc myproc { } {
     set y 0  #Local variable Y
     global x #Global variable X
     puts [format "X=%d, Y=%d" $x $y]
}
```

22.7 Other Tcl Hacks

Dynamic variable creation

```
# Dynamically create a bunch of variables.
for { set x 0  } { $x < 32 } { set x [expr $x + 1]} {
    # Create var name
    set vn [format "BIT%d" $x]
    # Make it a global
    global $vn
    # Set it.
    set $vn   [expr (1 << $x)]
}
```

Dynamic proc/command creation

```
# One "X" function - 5 uart functions.
foreach who {A B C D E}
   proc [format "show_uart%c" $who] { } "show_UARTx $who"
}
```

23 Target Library

OpenOCD comes with a target configuration script library. These scripts can be used as-is or serve as a starting point.

The target library is published together with the OpenOCD executable and the path to the target library is in the OpenOCD script search path. Similarly there are example scripts for configuring the JTAG interface.

The command line below uses the example parport configuration script that ship with OpenOCD, then configures the str710.cfg target and finally issues the init and reset commands. The communication speed is set to 10kHz for reset and 8MHz for post reset.

```
openocd -f interface/parport.cfg -f target/str710.cfg -c "init" -c "reset"
```

To list the target scripts available:

```
$ ls  /usr/local/lib/openocd/target

arm7_fast.cfg    lm3s6965.cfg  pxa255.cfg      stm32.cfg   xba_revA3.cfg
at91eb40a.cfg    lpc2148.cfg   pxa255_sst.cfg  str710.cfg  zy1000.cfg
at91r40008.cfg   lpc2294.cfg   sam7s256.cfg    str912.cfg
at91sam9260.cfg  nslu2.cfg     sam7x256.cfg    wi-9c.cfg
```

Appendix A The GNU Free Documentation License.

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and

that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called

an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

OpenOCD Index

G

H

I

J

L

M

O

P

R

S

T

U

V

W

Z