ZY1000 User's Guide

Ultimate Solutions, Inc.

Table of Contents

About

The ZY1000 is a product of Ultimate Solutions, Inc. It offers a standalone hardware debugger solution where no drivers or software is needed on the developer’s PC. An important part of the ZY1000 solution is the OpenOCD library which, essentially, provides the support for the target hardware.

1. Before you start

Congratulations on the purchase of your new ZY1000!

There are some important sources of information for setting up and using your ZY1000:

- The Quick Start Guide. The ZY1000 ships with a printed Quick Start manual that helps you set up the ZY1000. The manual in PDF format is also available at the following address: http://www.ultsol.com/pdfs/ZY1000_Quick_Start_Guide.pdf
- This manual.
- The ZY1000 web interface. Take a moment to navigate through the ZY1000 menus to have a look at the key features and explanations on how to use them. For example, you'll find a way to display the RS-232 output from your target in your web browser! In the web interface you'll find help text and links to manuals in the Documentation column on the right.
- Contact Ultimate Solutions if you have any questions. We'll be happy to answer your questions and offer guidance.

Ultimate Solutions, Inc.

10 Clever Drive

Tewksbury, MA 01876 USA

Phone: 978.455.3383

Email: info@ultsol.com

http://www.ultsol.com

2. ZY1000 setup

By default the ZY1000 is configured to use DHCP, but you can also set a fixed IP address via a serial cable terminal program or via the web interface. The ZY1000 will broadcast its presence every two seconds for the first minute after boot (UDP port 4950). This can be used to find the IP address of the ZY1000.

The simplest way to discover the IP address, is to simply run the applet from the ZY1000 Support Page:

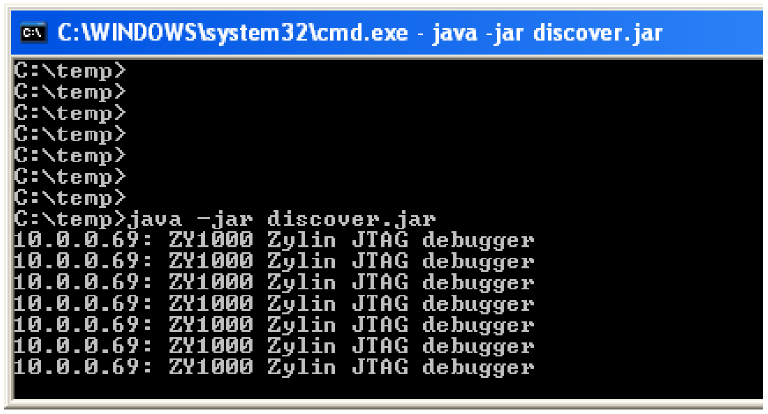
http://www.ultsol.com/index.php/component/content/article/8/216-zylin-zy1000-sprt

This applet will need to listen to your LAN and determine the IP address. To do so, you will have to grant access to the applet on this page.

2.1. Offline application

You can also download an application and run it on your own computer.

Download http://www.ultsol.com/ZY1000/discover.jar and run it from the command prompt and then power cycle the ZY1000. You can then see that the ZY1000 is on IP address 10.0.0.69. Note that the ZY1000 stops broadcasting after 60 seconds. The discover.jar works on all operating systems with Java installed.



```
C:\temp>
C:\temp>
C:\temp>
C:\temp>
C:\temp>
C:\temp>
C:\temp>
C:\temp>java -jar discover.jar
10.0.0.69: ZY1000 Zylin JTAG debugger
10.0.0.69: ZY1000 Zylin JTAG debugger
10.0.0.69: ZY1000 Zylin JTAG debugger
10.0.0.69: ZY1000 Zylin JTAG debugger
10.0.0.69: ZY1000 Zylin JTAG debugger
10.0.0.69: ZY1000 Zylin JTAG debugger
10.0.0.69: ZY1000 Zylin JTAG debugger
```

2.2. How to upgrade ZY1000 firmware

1. Download the firmware file which is suffixed with '.phi'. The latest firmware can be found at http://www.ultsol.com/index.php/component/content/article/8/217-zylin-zy1000-firmware
2. Connect your web browser to the ZY1000 web server on the ZY1000 debugger.
3. Go to the Setup ZY1000->ZY1000 Firmware page.
4. Press 'Browse' and find the firmware file you just downloaded in Step 1.
5. Press 'Upload' to upload the firmware.
6. Wait approximately 30 seconds for the new firmware to be programmed and the ZY1000 to reboot.

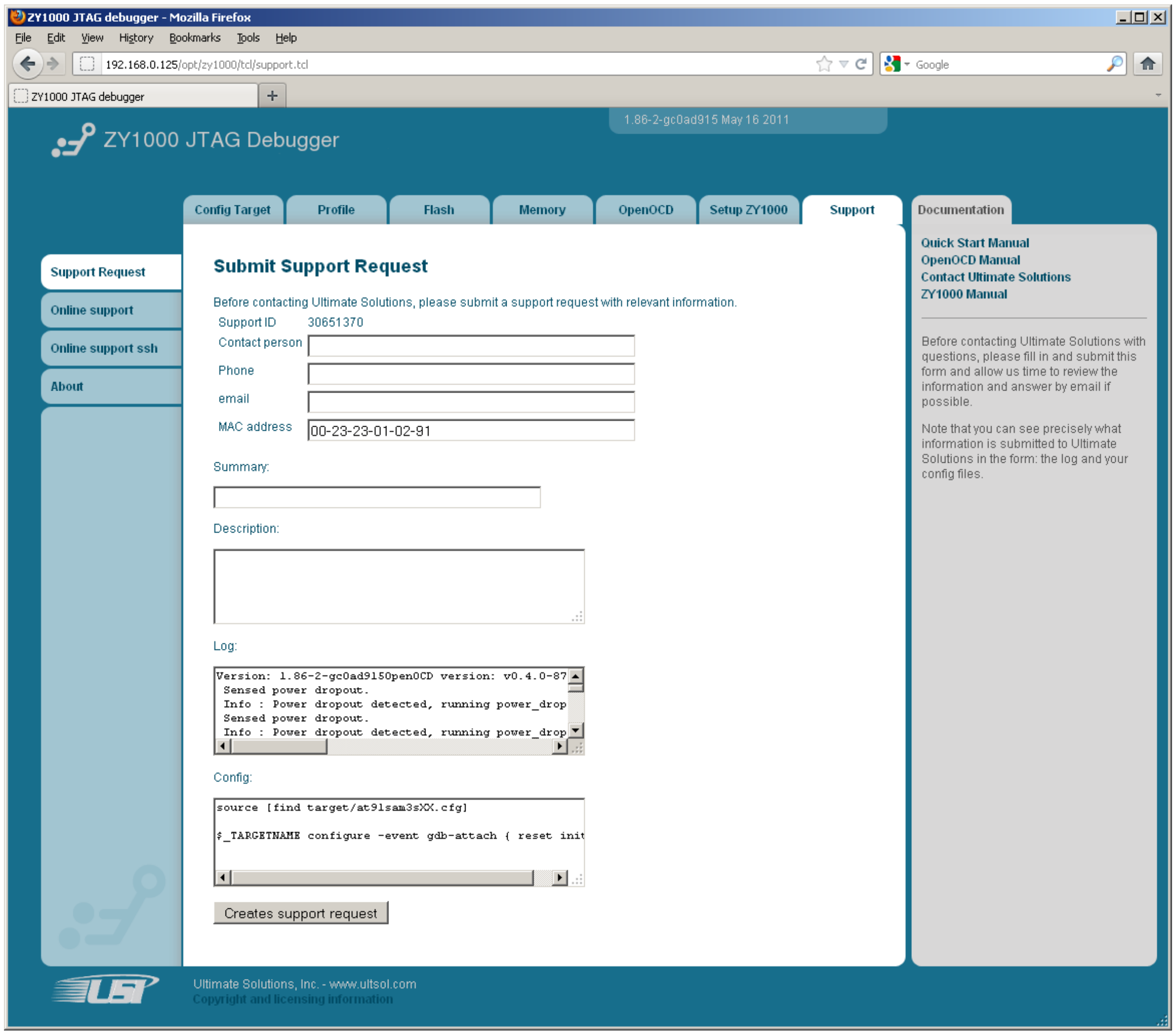
2.3. Contacting Ultimate Solutions

The ZY1000 has a support request menu that will help you put together a complete report that you can copy and paste into an email to support@ultsol.com. This report will aid us in helping you resolve any issues that you may be having.

2.3.1. Step to produce report

We recommend the following steps to produce the best possible information for us to assist you.

1. (Optional). Add a “debug_level 3” to the top of your target configuration script. This will produce a more verbose log.
2. Reload the config script / reboot the ZY1000 to start with a clean log.
3. Perform steps to reproduce the problem.
4. Enter Setup ZY1000->Support Request menu and fill it out.
5. Copy and paste the report and email it to support@ultsol.com with any relevant attachments.


ZY1000 JTAG debugger - Mozilla Firefox
File Edit View History Bookmarks Tools Help
192.168.0.125/opt/zy1000/tcl/support.tcl
Google
ZY1000 JTAG debugger
1.86-2-gc0ad915 May 16 2011
ZY1000 JTAG Debugger
Config Target
Profile
Flash
Memory
OpenOCD
Setup ZY1000
Support
Documentation
Support Request
Online support
Online support ssh
About
Quick Start Manual
OpenOCD Manual
Contact Ultimate Solutions
ZY1000 Manual
Creates support request
Ultimate Solutions, Inc. - www.ultsol.com
Copyright and licensing information

Submit Support Request

Before contacting Ultimate Solutions, please submit a support request with relevant information.

Support ID	30651370
Contact person	
Phone	
email	
MAC address	00-23-23-01-02-91

Summary:

Description:

Log:

```
Version: 1.86-2-gc0ad915OpenOCD version: v0.4.0-87
 Sensed power dropout.
 Info : Power dropout detected, running power_drop
 Sensed power dropout.
 Info : Power dropout detected, running power_drop
```

Config:

```
source [find target/at91sam3sXX.cfg]

$_TARGETNAME configure -event gdb-attach { reset init
```

Before contacting Ultimate Solutions with questions, please fill in and submit this form and allow us time to review the information and answer by email if possible.

Note that you can see precisely what information is submitted to Ultimate Solutions in the form: the log and your config files.

2.4. Advanced ZY1000 diagnostics

The ZY1000 settings are stored on a flash filesystem under /config/settings. Restoring factory settings basically deletes the /config/settings directory tree. From telnet, you can create, delete, and list files in /config/settings. Type “help ls/rm/append_file/trunk” for more information.

2.4.1. Enabling logging to serial port

You can enable logging output to the serial port from the bootloader. This should be used as a last resort diagnostic option if the web interface should be unavailable to download a log. Use the bootloader to enable logging by setting /config/settings/logserial to a value of 1.

```
Press <i> to set static IP address
Press <enter> to start Ymodem upload of firmware
Press <space> for advanced help
Press <F> format flash
Press <E> to start Ymodem upload of firmware to a specified file name
Press <Y> to start single shot update of bootloader
Press <P> set parameter
Press <D> show parameter
Press <S> to select the application to launch
Press <T> to run the ramtest application
Enter file name: /config/settings/logserial
Enter parameter: 1
```

2.4.2. Disabling current target configuration

It is possible to write startup scripts in TCL that do not terminate (accidentally or purposely). This will make the web interface unavailable to correct the problem. Use the bootloader to disable the startup configuration by setting /config/settings/openocd.cfg to an empty value.

```
Press <i> to set static IP address
Press <enter> to start Ymodem upload of firmware
Press <space> for advanced help
Press <F> format flash
Press <E> to start Ymodem upload of firmware to a specified file name
Press <Y> to start single shot update of bootloader
Press <P> set parameter
Press <D> show parameter
Press <S> to select the application to launch
Press <T> to run the ramtest application
Enter file name: /config/settings/opeoncd.cfg
Enter parameter: dummy
```

2.4.3. Advanced bootloader options

The bootloader can be used to perform a number of operations that are normally performed from the web interface. Each command is available via a keypress, e.g., <p>. When entering a text string, you can press backspace to delete a character and control-c, if you want to abort input entry and reboot the ZY1000.

- <i> - To set the static IP address

  Here you can configure the IP address, mask, and gateway. The format is address, mask, and gateway (optional): (x.x.x.x,y.y.y.y[,z.z.z.z]).

- <enter> - To start Ymodem to upload firmware.

  If you are unable to upload a new firmware file via the web interface, you can use Ymodem to upload the firmware file.
- <space> - Lists help for the advanced option.
- <F> - Formats the /config flash drive.

  This will remove the firmware and all the ZY1000 settings. The MAC address and bootloader are in another area of flash separate from /config. After you have formatted /config you must upload a firmware file using Ymodem.
- <Y> - To update the bootloader.

  It is possible, but not advised unless absolutely necessary to update the bootloader from Ymode. The MAC address is also erased at the same time the bootloader is upgraded. Once the bootloader has finished uploading, the MAC address must be set.
- <P> - Set parameter.

  This command allows one to write a file to the /config filesystem. One will be prompted for the filename, followed by the contents of the file. ZY1000 parameters are normally stored in the directory: /config/settings. Each file in the directory contains the setting(s) for a particular parameter and the name of the file is the name of the setting.
- <S> - Not used with the ZY1000.
- <T> - Not used with the ZY1000.
- <E> - Not used with the ZY1000.

2.4.4. Configuration parameters

Restoring the factory settings, recursively deletes all the files in the directory /config/settings. This directory contains all the ZY1000 parameters. The following is brief list of some of the directories and files found in the directory /config/settings.

- /config/settings/xxxx - The ZY1000 configuration parameters are stored in individual text files.
- /config/settings/logserial – Setting the contexts of this file to 1, will cause detailed dump of the log file to be written to the serial port. It is useful if logs cannot be downloaded using the ZY1000 web interface.
- /config/ip – Contains the static IP address in the format ip, mask, gateway: (x.x.x.x,y.y.y.y.z.z.z.z)
- /config/settings/openocd.cfg – Contains the TCL commands that are executed upon startup. Typically, this file contains a single line: script target/xxx.cfg”. The string ‘xxx’ corresponds a particular configuration file. By leaving this file empty or containing a syntax error will disable the current active target configuration.
- /config/target – This directory contains the modified versions of target configuration scripts. The originals are stored in /rom/target. If a target configuration script exists in /config/targar, it is loaded instead of the one found in /rom/target. By deleting the files in the directory /config/target will effectively restore the directory /rom/target.

3. Performance profiling

When optimizing an application for performance, it is crucial to know where the CPU spends most of it's time. If an operation that takes 80% of the time is improved by 50%, it yields a 40% improvement in performance. Whereas if an operation that takes 5% of the time is improved by 90%, it yields a 4.5% performance improvement.

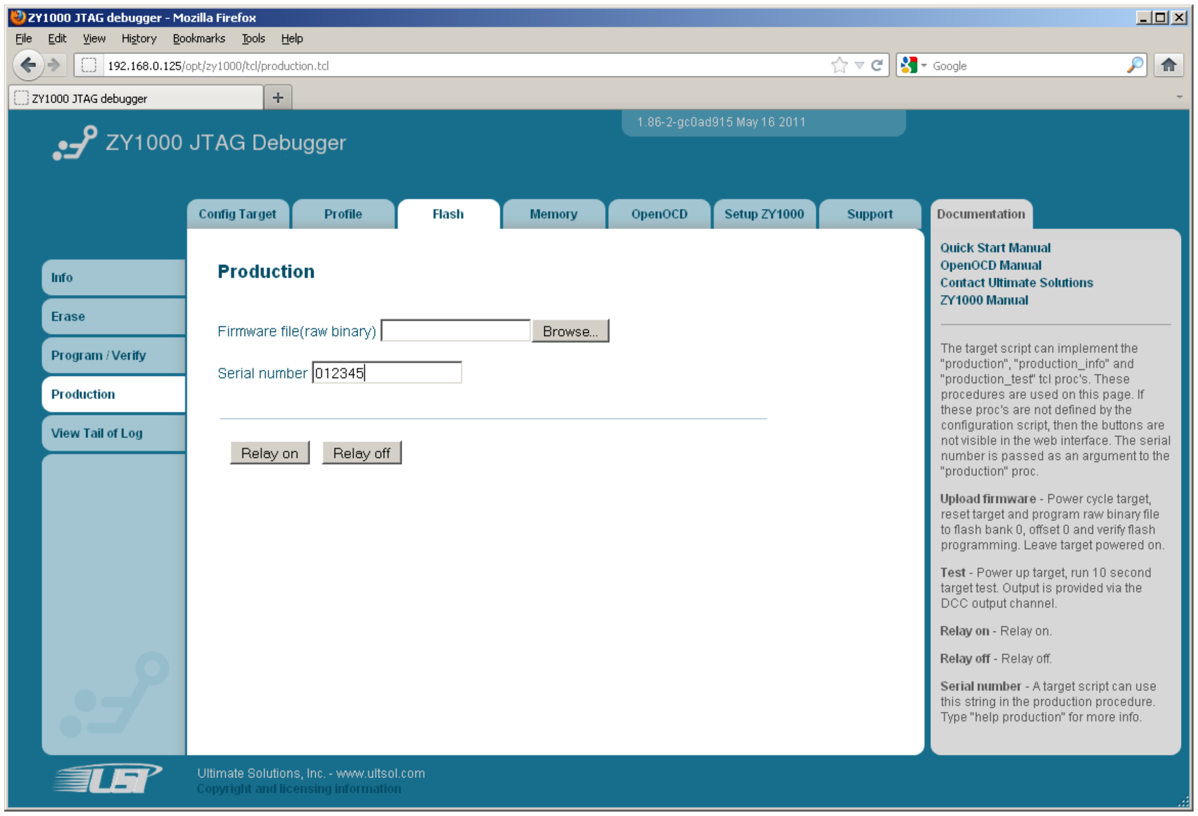
What the ZY1000 offers is an easy way to make a quick measurement, which is useful to ensure that the optimizations are at least focusing on the right part of the code. No changes in the binary are necessary. To perform a measurement, navigate to the Profiling page. Here you will upload your ELF-format binary file and choose how many seconds you want to same the program counter for. The program counter is sampled by halting and resuming the CPU. When the sampling period is complete, the result is displayed in the web browser.

The process is not entirely benign as the CPU is halted and resumed throughout the process. There are methods that will allow you to sample many more program counter samplings. The advantage of this approach is that it works on all CPU's allow you to get a better understanding of where in the code the CPU cycles are being spent.

4. Small scale production

The ZY1000 offers various methods to support small scale production, e.g., 10s to 100s of units. In a production scenario there are usually a few common operations:

- Enter or generate serial numbers, e.g., MAC addresses
- Program application(s) to flash
- Run a short quality test, e.g., go or no-go test.


ZY1000 JTAG debugger - Mozilla Firefox
File Edit View History Bookmarks Tools Help
192.168.0.125/opt/zy1000/tcl/production.tcl
Google
ZY1000 JTAG debugger
1.86-2-gc0ad915 May 16 2011
ZY1000 JTAG Debugger
Config Target
Profile
Flash
Memory
OpenOCD
Setup ZY1000
Support
Documentation
Info
Erase
Program / Verify
Production
View Tail of Log
Quick Start Manual
OpenOCD Manual
Contact Ultimate Solutions
ZY1000 Manual
Browse...
Relay on
Relay off
Ultimate Solutions, Inc. - www.ultsol.com
Copyright and licensing information

Production

Firmware file(raw binary)

Serial number 012345

The target script can implement the "production", "production_info" and "production_test" tcl proc's. These procedures are used on this page. If these proc's are not defined by the configuration script, then the buttons are not visible in the web interface. The serial number is passed as an argument to the "production" proc.

Upload firmware - Power cycle target, reset target and program raw binary file to flash bank 0, offset 0 and verify flash programming. Leave target powered on.

Test - Power up target, run 10 second target test. Output is provided via the DCC output channel.

Relay on - Relay on.

Relay off - Relay off.

Serial number - A target script can use this string in the production procedure. Type "help production" for more info.

4.1. Writing a production scripts

Writing a production script is relatively straight forward. An example below is shown how to implement the three required TCL procedures in the configuration.

```
proc production_info {} {
    return "Serial number is official MAC number. Format XXXXXXXXXXXX"
}

# There is no return value from this procedure. If it is
# successful it does not throw an exception
#
# Progress messages are output via puts
proc production {firmwarefile serialnumber} {
    if {[string length $serialnumber]!=12} {
        echo "Invalid serial number"
        return
    }
    echo "Power cycling target"
    power off
    sleep 3000
    power on
    sleep 1000




    reset init
    flash write_image erase $firmwarefile 0x1000000 bin
    verify_image $firmwarefile 0x1000000 bin

    # Big endian... weee!!!!
    echo "Setting MAC number to $serialnumber

    flash fillw [expr 0x1030000-0x8] "0x[string range $serialnumber 2 3] \
                                    [string range $serialnumber 0 1]0000” 1


    flash fillw [expr 0x1030000-0x4] "0x[string range $serialnumber 10 11] \
                                    [string range $serialnumber 4 5]” 1
    echo "Production successful"
}


proc production_test {} {
    power on
    sleep 1000
    target_request debugmsgs enable
    reset run
    sleep 25000
    target_request debugmsgs disable
    return "See IP address above..."
}
```

4.2. Autonomous ZY1000 production operation

The ZY1000 is normally attached to a computer, but perhaps it would be better to just require only the ZY1000. The ZY1000 can support autonomous operation by writing a configuration script along the following lines:

- Add a handler that executes upon detecting the target power. This handler runs the production or main-line sequence.
- Once the production sequence is complete, this script uses the LED's on the ZY1000 to indicate either failure or success.

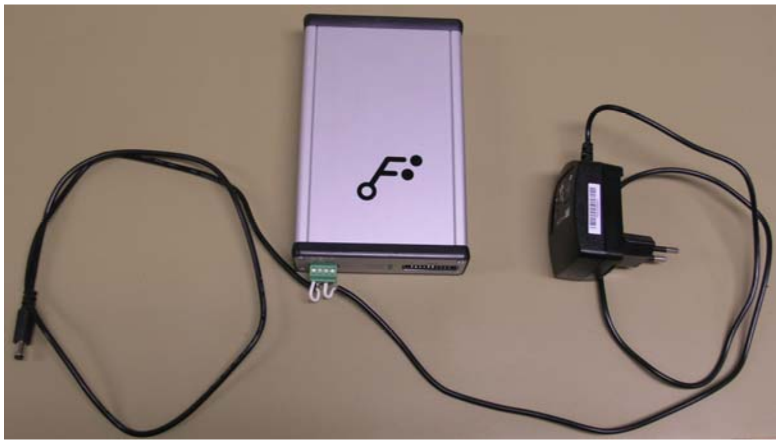
There are approximately 20M bytes of flash available to store the target image on the ZY1000. The "Power" light on the ZY1000 can be controlled from the script. By convention, flashing quickly indicates an error, where as slowly flashing indicates successful completion. The "Target" LED can also be repurposed. Slow flashing indicates an operation in progress, the LED being off indicates no target power, and being on indicates power detected.

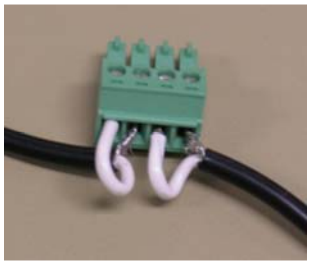
5. ZY1000 remote power cycling

The ZY1000 has the ability to remotely power cycle the target. This is done by means of a built-in low voltage, low power capable relay that can be controlled via the web interface, a GDB init sequence, or telnet. For example in GDB, one could do the following:

```
echo Turn target power on
monitor power on
echo Turn target power off
monitor power off
```

No soldering is required to set this up. The quickest way to do this is remove the DC barrel connector from a wall mount power supply. After the connector has been removed, strip the wires and attaché the wires on the green Phoenix connector on the ZY1000. Additional Phoenix connectors are available at most major electronic parts supplies.

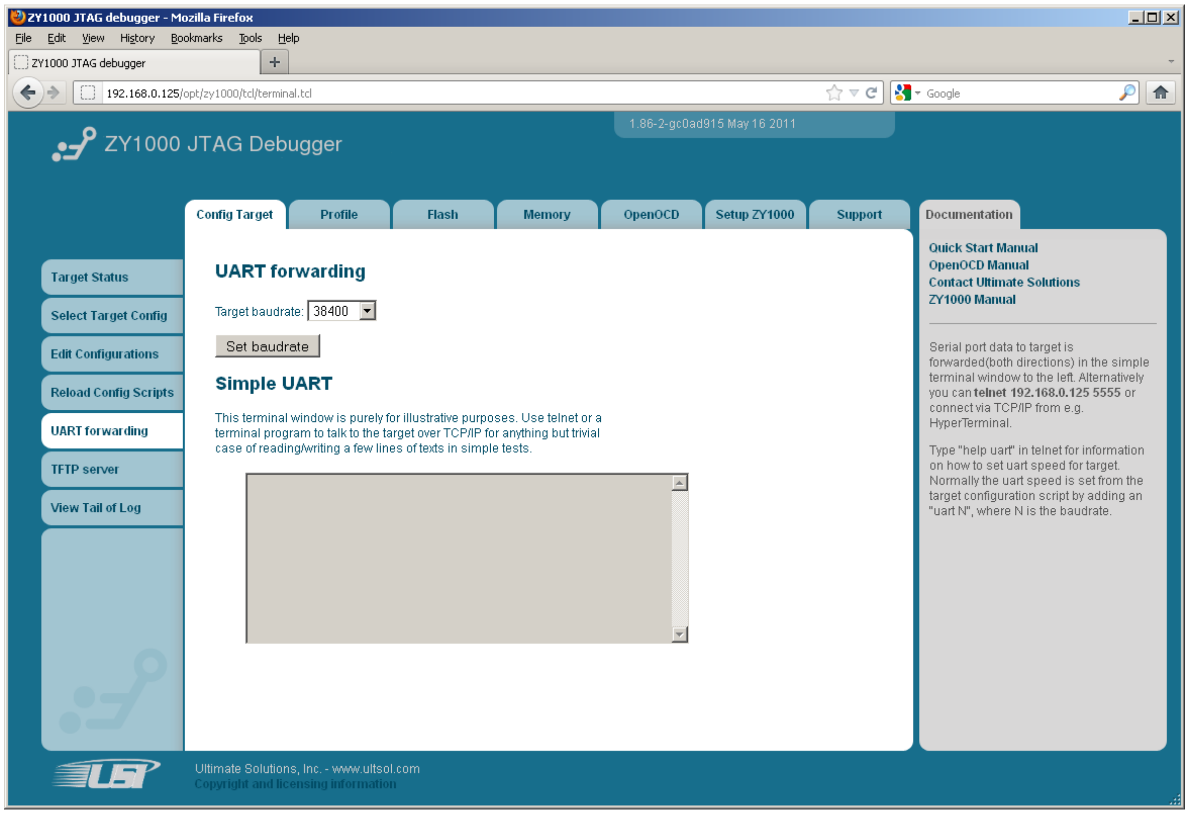




6. Remote RS232 forwarding

The ZY1000 has a serial port that can be used to access the ZY1000 bootloader, as well as to set the static IP address. However, it can also be used to forward a target's serial port data over TCP/IP. This type of situation is commonly called COM port redirection (see: http://en.wikipedia.org/wiki/COM_port_redirector for more information).

To connect to your target, you will need a null modem cable and the appropriate gender changer for your COM port. To get started, connect the cables and navigate to the ZY1000 UART forwarding web page:



UART forwarding

Target baudrate: 38400

Simple UART

This terminal window is purely for illustrative purposes. Use telnet or a terminal program to talk to the target over TCP/IP for anything but trivial case of reading/writing a few lines of texts in simple tests.

Serial port data to target is forwarded(both directions) in the simple terminal window to the left. Alternatively you can **telnet 192.168.0.125 5555** or connect via TCP/IP from e.g. HyperTerminal.

Type "help uart" in telnet for information on how to set uart speed for target. Normally the uart speed is set from the target configuration script by adding an "uart N", where N is the baudrate.

6.1. Advanced COM redirection

A more advanced scenario is to build an open-source COM redirector for ZY1000 Linux and install a virtual COM Windows device driver.

7. Remote online support

The simplest way to grant Ultimate Solutions full access to your ZY1000 and the target for support purposes is to use the Online Support menu. By granting Ultimate Solutions full support access to your ZY1000 and target, Ultimate Solutions will be able to assist you more effectively. There are of course security considerations with doing this. More information about this topic is available in the advanced section.

In cases where the ZY1000 does not have a working configuration script or there is a problem with the ZY1000's support for a particular target, this method will allow Ultimate Solutions to examine the problem and discover what the cause is with a minimum amount of effort and without having someone from Ultimate Solutions on-site.

7.1. Granting Ultimate Solutions access to target

Another method to grant Ultimate Solutions access, and by fair the easiest is to use port forwarding over ssh. The customer will create a secure tunnel from Ultimate Solutions' server to the ZY1000 unit.

7.2. A few words about security

Like most embedded devices the ZY1000 is not designed to be exposed to the Internet directly. Furthermore, the ZY1000 does not even pretend to by secure in any way, e.g., it doesn't even require authentication for access! The ZY1000 is intended to be placed on a 'friendly' LAN without any authentication, authorization, or other security measures.

If you want to give Ultimate Solutions access to a ZY1000 and your company has a restrictive security policy, then the best option is to place the ZY1000 outside the company firewall together with a computer that can dial into the Ultimate Solutions support server. The Ultimate Solutions support server provides ssh access to an ssh 'jail' where the only possible operations are port forwarding for the ZY1000 and termination of the session.

7.3. ZY1000 firmware support

This feature is only present in the ZY1000 firmware 1.67 or newer.

7.4. Hooking up the target

The first step is to connect up to the target. Ideally, the power relay and serial port forwarding should be set up.

7.5. Advanced Ultimate Solutions Online support topics

7.5.1. Set up SSH tunnel

You can connect the ZY1000 unit to Ultimate Solutions support server via ssh reverse port forwarding. If you have ssh installed on your system, you can issue the following command line. The advantage of using ssh instead of the Java applet in the ZY1000 is improved performance and robustness. Under Windows, ssh is provided in the following packages: PuTTY, MinGW, msysgit, and ssh in Cygwin to name a few.

If you want to use PuTTY, then replace the string “ssh” with PuTTY in the follow command line. The following example uses a ZY1000 with the IP address of 10.0.0.167.

```
ssh -p 45729 -l support support.zylin.com -R 7777:10.0.0.167:7777
  -R 7778:10.0.0.167:80 -R 7779:10.0.0.167:23 -R 7780:10.0.0.167:3333
  -R 7781:10.0.0.167:1234 -R 7782:10.0.0.167:69 -R 7783:10.0.0.167:70
  -R 7784:10.0.0.167:5555 -R 7785:10.0.0.167:22 -R 7786:10.0.0.167:9999
```

The login and password for this connection are “support” and “support”, respectively. The ZY1000 will display the command line to use on the “Support / Online support ssh” web page.

7.5.2. Using ZY1000 as an OpenOCD interface

The ZY1000 supports a proprietary JTAG over TCP/IP protocol. Ultimate Solutions uses this to build and run OpenOCD on a development PC system. From the development PC, OpenOCD communicates over TCP/IP using this proprietary protocol. To build and run OpenOCD to communicate directly with the ZY1000 use the following configuration line prior to building OpenOCD:

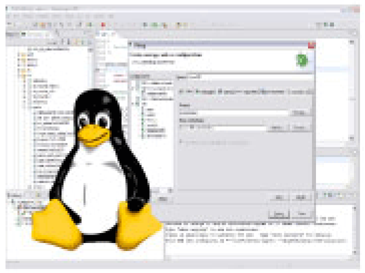
```
./configure --enable-zy1000 --enable-maintainer-mode
make
sudo make install
openocd -f zy1000server.cfg
```

In the file zy1000server.cfg, the IP address 10.0.0.138 should be replaced with the IP address of the ZY1000 unit you are using.

```
interface ZY1000
zy1000_server 10.0.0.138
```

8. GDB GUI – Eclipse

GDB itself does not have a GUI. However, Ultimate Solutions offers LinuxScope-JTD, a debugger for the Eclipse IDE. Please click below for more information:



http://www.ultsol.com/index.php/products/cross-compiler-debuggers/36-ultimate-solutions-linuxscope-jtd-jtag-target-debugger-

9. JTAG cable pinout

The ZY1000 is built with an ARM-MULTI JTAG 20-pin connector. There are many different physical JTAG connectors. Some have more pins than the ARM 20-pin connector, others have less. In either case, it is possible to connect the ZY1000 to a target that has an ARM JTAG connection that doesn't have 20 pins. By knowing the pin out of the target's JTAG connector and the ZY1000's JTAG connector, one can connect the pins using individual wires or a custom cable.

9.1. Adaptive clocking RTCK / RCLK

The ZY1000 supports adaptive clocks, also known as, RTCK or RCLK. By using the commands 'jtag khz 0', 'jtag rclk N', where N is a fallback frequency, one can enable the adaptive clocking feature. There are some situations where one can change 'jtag rclk N' to 'jtag khz N' and see improved performance or more robust behavior. The JTAG 20 cable that comes packaged with the ZY1000 is sufficient for just about any target the supports adaptive clocking. Adaptive clocking can be useful for targets that start with an RC oscillator and switch to a PLL at some point during the target processor's initialization.

As an alternative to adaptive clock, you should consider using the 'reset-start' and 'reset-init events. With the 'reset-start' event, one can set the jtag khz to a low value that will work with RC oscillators. Then after setting up the PLL in 'reset-init', you can use jtag khz to set a higher clock rate. From a performance point of view, adaptive clocking generally yields lower rates than is possible then by setting jtag khz in the 'reset-start' or 'reset-init' events.

While adaptive clocking sounds great in theory, i.e., automatically getting the correct JTAG clock frequency, is not necessarily a cure-all. Complications arise when the JTAG interface is implemented in a way that is 'tamper-proof' against reverse engineering or in a situation where the JTAG pins can be turned into GPIO pins. Unless you have to use adaptive clocking with variable frequencies, you may want to consider using a fixed frequency.

10. Field deployment

The ZY1000 is intended to be used in the field as well as at the developer's desk. Here the advantages of Ethernet connectivity and the ZY1000's ability to run the Linux OS become more apparent.

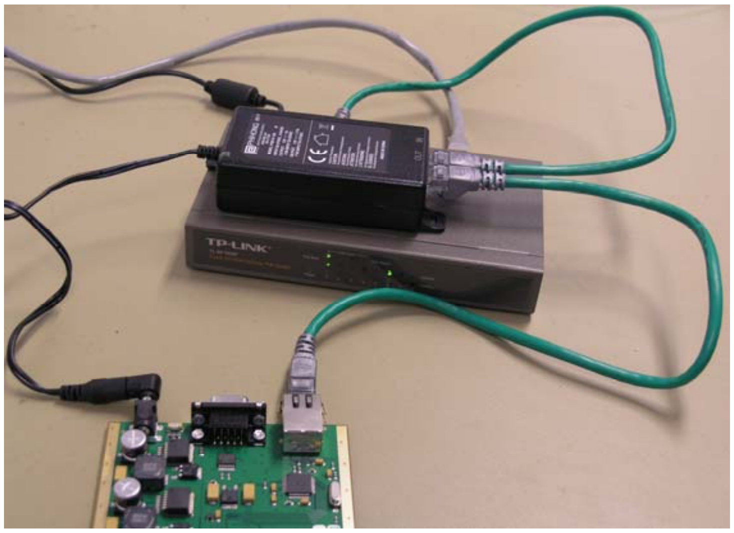
A typical example would be the remote debugging of the target. Here the power relay can be useful to power cycle the target. Note that the power relay is short-circuited when the ZY1000 is powered off. Combined with the ZY1000's JTAG interface ability to act as 'disconnected' when the ZY1000 is powered off, makes it more practical to leave the ZY1000 inside the housing of deployed hardware.

Beyond providing Ethernet connectivity, the ZY1000 also supports scripting capabilities or even running customer Linux applications. The connectivity from the developer PC to the ZY1000 can be bi-directional when using Linux. The ZY1000 can either connect to a server, e.g., via ssh, or the developer PC can connect to the ZY1000. The advantage of the ZY1000 connecting to a server would typically be to traverse firewalls or NAT's.

10.1. Power over Ethernet

The ZY1000 can be used with a Power over Ethner or PoE. There are many supplies of these types of splitters. These PoE splitters can be used to power the ZY1000 and possibly the target as well. The ZY1000 requires up to 5W and uses a barrel connector, center positive, 2.1/5.5mm, between 8 and 40 V.

In the following figure a PoE splitter from Phiphong (http://www.phihongusa.com/) is shown.



11. Schematics for ZY1000 JTAG stage

11.1. Schematics and board layout

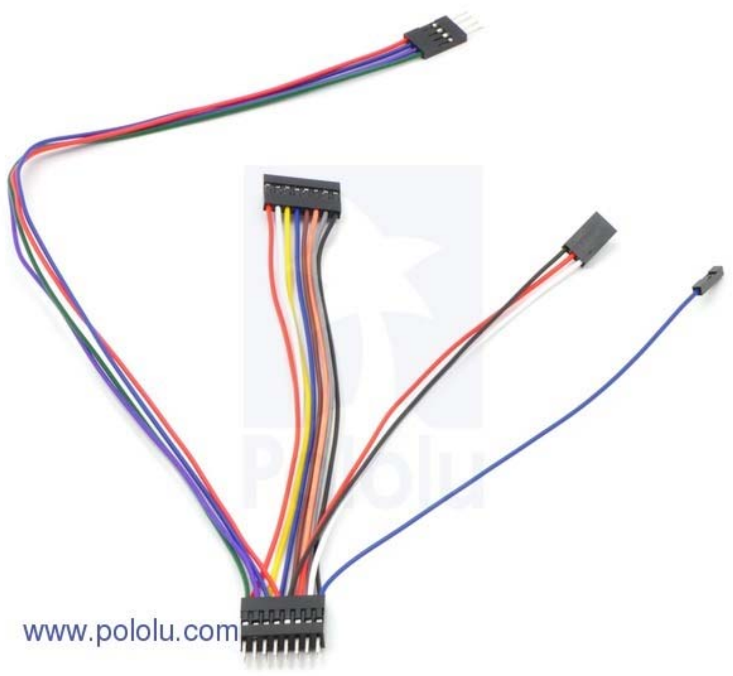
In the event you need schematics for the ZY1000 JTAG stage and/or how the top and bottom of the PCB board are laid out, please contact Ultimate Solutions.

11.2. Building custom cables

Given that there is no physical JTAG standard connectors for evaluation kits, let alone space constrained application PCB's, developers will often have to create their own custom cables. As one might expect, everybody seems to solve this problem in a different manner depending on experience, tools, and materials they happen to have on hand or can easily acquire.

One solution frequently used requires no soldering and can be used by embedded software engineers with a little bit of hardware experience, and are comfortable with using pre-crimped cables. This particular solution works well for 0.1" connectors. For smaller connectors, soldering is usually required.

The supplier of pre-crimped cables and connectors is Pololu (http://www.pololu.com). As seen in the figure below, the set of pre-crimped cables allows developers to easily create, e.g., an ARM JTAG 20-ping connector to a Texas Instrument 14-pin connector. It's just a matter of swapping a few color-coded wires.



When creating a custom ARM JTAG 20-pin connector to target cable, keep in mind that the ZY1000 only uses GND, TRST, SRST, TMS, TDI, TDO, TCLK, and RCLK. The target may or may not use all of the signals.

12. Troubleshooting

The following gives some tips and tricks for using the ZY1000 and GDB.

12.1. Problems with starting a CDT debug session

If CDT debugging works on the first try, GREAT! Embedded debugging can be quite complete with its own set of unique quirks that have to be understood and dealt with for each setup. Unfortunately, no two setups are exactly alike. First, CDT is the wrong place to try to debug the GDB and hardware debugger setup. It is not yield intuitive and detailed feedback in response to each careful step in setting up a debugging environment for embedded systems.

The sequence below is just a suggestion on how to debug. There are, of course, many other ways to do so. Only by advancing to the next level, when each level of problems are solved, can you get better feedback on the problem(s) you might be encountering.

- First, check the cables and voltages. Many times it's a simple case of mis-wiring.
- Check the schematic for how the reset (TRST and SRST) are wired. Also read the datasheet for the CPU. Often, SRST and TRST are wire together, i.e., srts_pulls_trst). TRST will reset the debugger circuitry and result in problems such as it being impossible to putting the CPU into the halted state. Note that TRST and SRST can be tied together on the PCB, but also inside the CPU proper. This can be by design, e.g., to implement copyright protection schemes.
- Start by using the ZY1000 telnet. This stage involves the command line version of GDB. If you are having trouble figuring out what is going on, set "debup_level 3" in your ZY1000 configuration script and go over the debugging messages in the low.
- Issue a "reset run". This command should be able to enumerate the JTAG toolchain.
- Issue a "reset init". This will reset and halt the target. This should be followed by run commands to initialize the target, i.e., to setup the memory map and / or required peripherals. If you only reset and halt the target, i.e., "reset halt", you will probably not be able to issue a GDB load or program the flash.
- Verify that you can read memory (see the OpenOCD command: mdw) and modify memory (see the OpenOCD command: rdw). If you have working area etup in your configuration script, make sure to veify that you can read and write to the working memory after a 'reset init'.
- If necessary, test the flash. This can be accomplished by issuing a "reset init", followed by a "flash erase" and filling the flash with data, using "flash write image" or "flash fill". Typing "help flash" in OpenOCD will also provide additional information.
- If you have flash configured, you can program flash by issuing a GDB "load" command, provided that the address of the .elf image is identical to the flash chip. If you have an .elf image that is copied form flash to RAM upon startup, you'll need to add an offset to the GDB load command. Typing "help load" in GDB will also provide some additional information.

- At this point you have successfully setup the execution environment using the GDB command line version.
- From hereon, you can set breakpoints and begin execution of the target. GDB will use the hardware breakpoints for the memory regions set up as ‘read-only’. Typing “info mem” at the command problem will provide some additional information.
- If you have not setup the flash regions in your “reset init” script, then GDB will use software breakpoints by default, which will fail for flash regions. The solution in this situation is to add a “gdb_override_breakpoint” command in your “reset init” script to force hardware breakpoints.
- At this point you should try to single step from the GDB command line.
- Next try setting a breakpoint and issuing the “continue” command and verify that the CPU halts as expected at the breakpoint address.
- At this point you have demonstrated enough of the connectivity and functionality that you can attempt to setup a debugging session with the Embedded CDT.

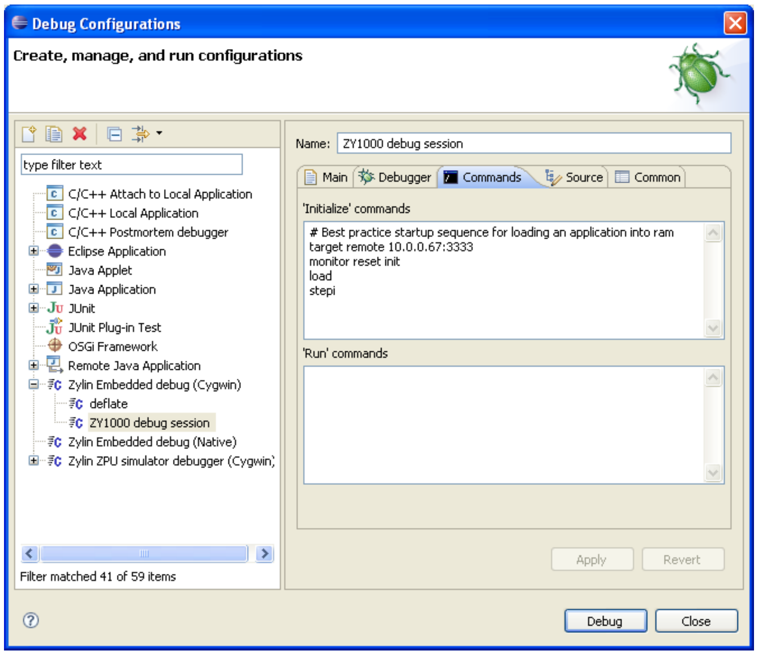
12.2. GDB init sequence

If you have a working ZY1000 “reset init” sequence, then the following GDB script should be quite straightforward. It is generally better to keep your GDB initialization script terse and leave configuration within the ZY1000 configuration script.

```
# Connect to ZY1000
#
# By design ZY1000 will not interact with the target upon connect.
# This is in order to e.g. support connecting to powered down or
# unresponsive targets. The GDB protocol requires reading the
# CPU registers upon connect and leaving the CPU in the halted state.
#
# Here ZY1000 uses two white lies: even if GDB is halted, the
# target could be running, halted, powered down, etc. Instead of
# returning the value of the registers upon connect, dummy values
# are returned. The "stepi" below will sync up GDB to the target
# state.
target remote 10.0.0.69:3333
# reset target and run initialisation script.
# all commands to ZY1000 are prefixed with "monitor".
monitor reset init
# load application into RAM
#
# If the target configuration script has been set up with flash
# and the address of the image is in flash, then ZY1000 will program
# the application into flash, erasing any data in the sectors
# where the application is to be programmed.
load
# set program counter to start of application.
# We use ZY1000 to do this because it is more robust
# than trying to do so with GDB. GDB can often get
# confused when trying to evaluate stack frames outside
# C code.
```

```
monitor reg pc 0x20000000
# To sync things up, we step a single instruction. The ZY1000
# knows not to single step immediately after a GDB connect.
stepi
```

A question that might occur to you: What belongs in the “reset init” script and what bellows in the GDB initialization script? The answer to this is to a degree a matter of preference. Perhaps one thing to keep in mind is that scripts executing on the ZY1000 will execute faster than a long sequence of monitor commands.



Name: ZY1000 debug session

'Initialize' commands

```
# Best practice startup sequence for loading an application into ram
target remote 10.0.0.67:3333
monitor reset init
load
stepi
```

'Run' commands

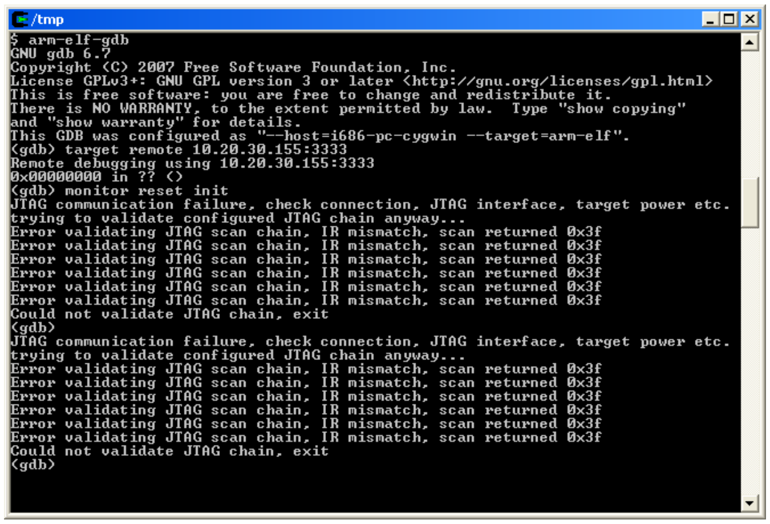
12.3. Speeding up GDB load

The best way to speed up GDB load is to setup a working area memory. When issuing “reset init”, you’ll also receive some warning if you, for example, haven’t enabled the ARM 7/9 fast memory access.

12.4. GDB low level debugging

When debugging low-level or early startup code, it might be a good idea NOT to use a graphical GDB GUI like Embedded CDT until you have stepped into the execution of the C source code.

If you use the command line version of GDB, you have a much more precise control over what commands are sent to the target. A GUI can to be helpful when starting to read variables on the stack. When debugging code that is setting up the stack, then obviously that is the last thing you would want.



```
$ arm-elf-gdb
GNU gdb 6.7
Copyright (C) 2007 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=arm-elf".
(gdb) target remote 10.20.30.155:3333
Remote debugging using 10.20.30.155:3333
0x00000000 in ?? ()
(gdb) monitor reset init
JTAG communication failure, check connection, JTAG interface, target power etc.
trying to validate configured JTAG chain anyway...
Error validating JTAG scan chain, IR mismatch, scan returned 0x3f
Error validating JTAG scan chain, IR mismatch, scan returned 0x3f
Error validating JTAG scan chain, IR mismatch, scan returned 0x3f
Error validating JTAG scan chain, IR mismatch, scan returned 0x3f
Error validating JTAG scan chain, IR mismatch, scan returned 0x3f
Error validating JTAG scan chain, IR mismatch, scan returned 0x3f
Could not validate JTAG chain, exit
(gdb)
JTAG communication failure, check connection, JTAG interface, target power etc.
trying to validate configured JTAG chain anyway...
Error validating JTAG scan chain, IR mismatch, scan returned 0x3f
Error validating JTAG scan chain, IR mismatch, scan returned 0x3f
Error validating JTAG scan chain, IR mismatch, scan returned 0x3f
Error validating JTAG scan chain, IR mismatch, scan returned 0x3f
Error validating JTAG scan chain, IR mismatch, scan returned 0x3f
Error validating JTAG scan chain, IR mismatch, scan returned 0x3f
Could not validate JTAG chain, exit
(gdb)
```

12.5. GDB embedded versus PC application debugging

There are some important differences to keep in mind when debugging embedded applications using GDB versus debugging PC applications.

For embedded, GDB does not launch the application, nor does GDB necessarily even load the application. GDB attaches to the target. The target can be in any state: powered down, running, or halted. Upon attaching to the target, GDB will issue a halt command, which may or may not succeed, after which GDB will enter the halted state.

- Caution #1: Even if GDB is halted, that does not mean that the target is halted. You can use the GDB “monitor” commands to, for example, reset the target into the halted state.
- Caution #2: You should be somewhat skeptical of the CPU register contents GDB reports immediately after attaching to the target since GDB may be out of sync with the target. Use the “monitor reg XX”, where XX is the register number, to read the actual value of the register. The command “monitor reg XX” will return an error message if the target is in the running state or otherwisde unresponsive.

For an application running out of flash, you would typically specify the executable on the GDB command line and the executable would only be used by GDB to read out symbols. The actual application would be programmed in flash by a JTAG debugger beforehand or the ZY1000 can also redirect the GDB “load” command to a flash programming command (including erasing any existing application).

The difference between loading symbols and loading applications is illustrated by the GDB “load” versus “symbol” command. The former loads the application AND symbols, whereas, the latter only loads the symbols. It is possible to load multiple symbol files into GDB at the same time, e.g., using a bootloader with the application.

12.6. Target configuration script

Before you can use the ZY1000 to debug your board or program flash, you need a target configuration script. The purpose of the target configuration script is to specify to the ZY1000 what target is connected and how to setup a minimal configuration for that target that will enable uploading code to the program flash, uploading code to RAM, and debug the target.

To get a target script correct can be anything from trivial to quite difficult. This is because there can be a high level of necessary complexity to setting up the target. The good news is that once done, you’ll probably never have to touch that script again.

The ZY1000 package provides a large lot of target configuration scripts that can be selected from a drop-down menu. You should start with a target script that closely resembles what you need and modify it to suit your purposes.

12.7. That pesky init sequence

If you don’t know how to write an initialization sequence for your target, then there are a couple of resources that you can resort to:

- Do you have the source code (machine code possibly) to set up the target? If so, you can try translating the machine code to an initialization sequence. Most often this will distill down to a dozen or so “mww” commands in the reset event sequence. If this can be done easily, then it probably the preferable means of doing things.
- Do you have an application binary that sets up the RAM that you can program to flash? If you are able to program that application to flash, then you can use the “resume” and “halt” commands in your reset initialization sequence. This will let the CPU run for a short time, enough to set up RAM. Afterwards, you should be able to upload RAM using, e.g., GDB “load”. Note that making a minimal reset initialization scrip that sets up flash can be A LOT easier than setting up the RAM.
- Read the datasheets. If you read through the datasheets you should be find all the information required to setup your target. It may not be obvious, but it’s probably there. This can be quite a time consuming process and it is fairly likely that someone has done this for you.
- A bit more esoteric approach. If you have the internal memory for your CPU, then you may be able to use the binaries of an application to set things up. The early startup code for an application is almost certainly position independent. This should allow you to load the beginning of the application into internal RAM and execute the first few instructions. You’ll have to do a bit of manual disassembly to determine how much of the beginning of the application you’ll need.

13. Security on the ZY1000

WARNING: The ZY1000 product is inherently insecure. It indiscriminately allows access via the LAN and the ZY1000 application runs as the root user. It is on the far end of the convenience end of the security versus convenience spectrum. This warned, if you do need to access the ZY1000 remotely and you know what you are doing, there are some recommended options. The ZY1000 ships with the ssh server enabled. By exposing only the ssh port and using a secure password, the communication to and from the ZY1000 unit will be secured.

14. Telnet and SSH access to the ZY1000

The ZY1000 has unsecured telnet enabled on port 9999 by default. To set the password of the ZY1000, log on via unsecured telnet and set the root password. Note that the ZY1000 ships without a root password set, so the only option to set the password is either via the serial console and telnet.

Afterwards you can use ssh or scp to copy files to and from the ZY1000.

Afterwards you can use ssh and scp to copy files to/from ZY1000.

```
jim"titan:~$ telnet 10.0.0.69 9999
Trying 10.0.0.69...
Connected to
10.0.0.69. Escape
character is ’^]’.


BusyBox v1.16.2 (2010-10-04 09:24:27 CEST) hush - the humble shell
Enter ’help’ for a list of built-in commands.

root:/> passwd root
Changing password for
root New password:
Retype password:
passwd: warning: can’t lock ’/etc/passwd’: Invalid argument
Password for root changed by
root root:/> exit
Connection closed by foreign host.
jim"titan:~$ ssh -l root 10.0.0.69
The authenticity of host ’10.0.0.69 (10.0.0.69)’ can’t be established.
RSA key fingerprint is
ef:91:63:54:20:ac:3b:c6:33:e5:fe:bc:1b:25:10:66. Are you sure you want
to continue connecting (yes/no)? yes
Warning: Permanently added ’10.0.0.69’ (RSA) to the list of known
hosts. root"10.0.0.69’s password:
ZY1000 Debugger

For further information check:
http://www.ultsol.com



BusyBox v1.16.2 (2010-10-04 09:24:27 CEST) hush - the humble shell
Enter ’help’ for a list of built-in

commands. root:~>
```

15. Linux startup script of the ZY1000

The ZY1000 runs Busybox (http://www.busybox.net). If you need to add commands to the startup, you can add them to the end of the file: /etc/rc. For example, you can mount a file share upon startup by adding it to /etc/rc.

The ZY1000 has the “vi” editor installed. This is a text editor you will find pretty much on any Unix or Linux system. Initially, it can be a bit daunting to first use, basic “vi” text editing skills will come in handy when working with Linux systems.

16. Accessing Windows file share from the ZY1000

The ZY1000 Linux distribution ships with CFIS enabled. You can mount a Windows share on the ZY1000 with the following commands:

```
mkdir -p /mnt/win ;
mountpoint -q /mnt/win ||
mount -t cifs //server-name/share-name /mnt/win \\
    -o username=shareuser,password=sharepassword
```

More information about Windows (CIFS) shares and the available options to mount cifs can be found by Google'ing "man mount cifs".

17. Building OpenOCD for the ZY1000

While the easiest way to upgrade the ZY1000 is to download the latest firmware from Ultimate Solutions, some users may want to build OpenOCD for the ZY1000 themselves. One advantage of the ZY1000 running Linux, is that building OpenOCD for the ZY1000 is straight-forward once the Altera Nios Linux development system is set up. Simply get the latest version of OpenOCD from the git repository and build it. Afterwards, you can copy it to the ZY1000 using whatever means you are comfortable with, e.g., scp, ftp, or NFS.

The following is the command sequence that should be used when building the ZY1000 for the Nios Linux platform.

```
cd openocd
sh bootstrap
./openocd/configure --enable-ioutil --enable-zy1000 \\
--host=nios2-linux-gnu --enable-zy1000-master --prefix=/opt/zy1000 \\
--enable-maintainer-mode
make -s DESTDIR=/tmp/result install
```

Copy the resulting files in /tmp/result to /opt on the ZY1000 unit.

18. Building other applications that run on the ZY1000

If you want to build other Linux applications to run on the ZY1000 and those applications are autotools, this is relatively straightforward once you have build Nios Linux locally. For example, it can be useful to run gdb on the ZY1000 to do debugging without having GDB installed on the development machine. In this situation, CFLAGS and LDFLAGS are used to point to the include and library files of Nios Linux.

The following describes the building of gdb for the ZY1000.

```
../gdb-7.2/configure --prefix=/opt --host=nios2-linux-gnu \\
--target=arm-eabi \\
CFLAGS="-I/home/oyvind/nios2-linux/uClinux-dist/staging/usr/include/ \\
-g -O2" \\
LDFLAGS="-L/home/jim/nios2-linux/uClinux-dist/staging/usr/lib"
make -s DESTDIR=/tmp/result install
nios2-linux-gnu-strip  /tmp/result/opt/bin/arm-eabi-gdb
scp /tmp/result/opt/bin/arm-eabi-gdb root"10.0.0.69:/opt/zy1000/bin
telnet 10.0.0.69 9999
root:/> /opt/zy1000/bin/arm-eabi-gdb
GNU gdb (GDB) 7.2
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
 <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
    Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=nios2-linux-gnu --target=arm-eabi".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
(gdb) target remote localhost:3333
Remote debugging using localhost:3333
0x00000000 in ?? ()
(gdb) monitor reset init
jtag_speed 1876 => JTAG clk=31 kHz
31 kHz
...
```

19. ZY1000 FPGA source code licensing

The ZY1000 FPGA code is copyright by Zylin and Ultimate Solutions, Inc. It also contains intellectual property from Altera Corp. Contact Ultimate Solutions for licensing options of the FPGA source code and access to the PCB schematics if you are interested in developing some other product based upon the ZY1000 hardware.

20. Building the ZY1000 Linux distribution

WARNING! If you do decide to start building and installing another version of Nios Linux than what ships with the ZY1000, you could erase the bootloader with the ZY1000 flash if you are not careful. If you happen to have an Altera USB Blaster, you can contact Ultimate Solutions for recovery instructions.

The ZY1000 uses the Nios Linux distribution. The uClinux-dist and linux-2.6 for's are available at http://sopc.et.ntust.edu.tw. The ZY1000 source code is published to the zy1000/master branch of the uClinux-dist and linux-2.6 repository. You can read more about Nios Linux at http://www.nioswiki.com/linux.

21. JTAG over TCP / IP

What the JTAG over TCP/IP protocol essentially does is to peek and poke the FPGA's hardware accelerated JTAG module. The performance in this case relies on the client queueing a large number of pokes and peeks, without waiting for the result. Operations that are performance critical can often be implemented efficiently in this manner. Downloading large amounts of data to a target is usually implemented by running a small application that listens on the DCC (debug communications channel) and then writes the data to flash or RAM. This can be implemented as an open loop algorithm with possibly a verification after the data has been downloaded and processed.

21.1. Protocol

Connect the JTAG over TCP/IP protocol server on port 7777 of the ZY1000 unit. Send a 32-bit little-endian command word, followed by any arguments as 32-bit little-endian words and the result, if any, is returned as a 32-bit little-endian integer. The upper 8 bits of the command word is the actual command.

Command	Command sequence	Description
POKE	FPGA register offset | 0x00000000, then 32-bit value to poke.	Pokes the FPGA register at 'off-set' with 32-bit data.
PEEK	FPGA register offset | 0x08000000	Returns value of FPGA register as 32-bit little-endian
SLEEP	0x01000000, then microseconds to sleep.	Introducing pauses must happen on the ZY1000.
WAITIDLE	0x02000000	Tells the FPGA to wait for the fifo to clear

Sleeping on the client is not robust to introduce a pause, as there is no way to know when the received commands are processed. Also sending pokes and a short sleep in open loop is fast.

21.2. JTAG over TCP / IP registers

Offset	Type	Name	Description
0x0008	Write	JTAG _move	Bit[31:16] Not used Bit[15] must be 0 Bit[14:8] nbr _of_cycles Bit[7:4] repeat _state Bit[3:0] end_state JTAG move steps: 1 Go to ’repeat state’ and run ’nbr _of_cycles’ clock cycles 2 Go to ’end state’ Note! FPGA will handle fifo full events, there is no need to check fifo full flag before writing to jtag _move register Only stable states can be used in the JTAG _move register.
0x000c	Read/ Write	JTAG _data	Bit[31:0] Jtag data register Wait until fifo empty high before reading
0x0010	Read	JTAG _status	Bit[31:11] Not used Bit[10] Error occurred (RCLK - return clock timeout) Bit[9] Fifo full Bit[8] Fifo empty Bit[7] TARGET_VOLTAGE_DROPOUT Target voltage dropout bit. Set whenever target voltage is low or have been low after last sample/hold Bit[6] SRST _ASSERTED Set whenever SRST is active or have been active after last sample/hold Bit[5:0] Not used

0x0010	Write	JTAG control set	Writing a ’1’ will set the bit in that position Writing a ’0’ have no effect Bit[31:10] Not used Bit[9] Reserved Bit[8] RCLK - return clock mode Bit[7] Sample and clear TARGET VOLTAGE DROPOUT Bit[6] Sample and clear SRST ASSERTED Bit[5] Not used Bit[4] Power LED on - default on. Bit[3] Disconnect power to target Bit[2] Switch supply voltage to VCC Bit[1] Drive nTRST low (reset TAP) Bit[0] Enable drive nSRST low (reset CPU)
0x0014	Write	JTAG control clear	Writing a ’1’ will clear the bit in that position Writing a ’0’ have no effect Bit[31:11] Not used Bit[10] Clear error register Bit[9] Not used Bit[8] RCLK - return clock disable Bit[7:5] Not used Bit[4] Power LED off Bit[3] Connect power to target Bit[2] switch supply voltage to VTref Bit[1] Drive nTRST high (TAP not reset) Bit[0] Disable drive nSRST low (CPU not reset) This will 3-state nSRST output
0x001c	Write	JTAG clk	Bit[31:16] Not used Bit[15:0] clock div Jtag clock frequency is cpu clk (60 MHz) divided by the value in jtag clk register. Valid values: even numbers from 2 to 65534.
0x0020	Write	JTAG state write	Bit[31:4] Not used Bit[3:0] internal tap controller state Make sure that event fifo is empty before manual moves of internal tap state machine. Note: Only valid for stable end states

0x0028	Write	JTAG man write	Bit[31:4] Not used Bit[3] mode (0=jtag, 1=swd) Bit[2] swd oe n (0=drive tms/swdio, 1=tri-state) Bit[1] tdi Bit[0] tms / swdio (output) Each access will result in one cycle of tck = rising and one falling edge. Tdi, tms will be given the values from register, on falling edge. For jtag mode, Tdo will be shifted into jtag data register on negative tck flank as with a jtag move access. For swd mode, swdio will be shifted into jtag data register on negative tcl flank. Will stall cpu until read for another access.

21.3. Clocking data in / out

A bit is ready to be clocked in or out as soon as the shift TAP state is entered, i.e., a bit is clocked in or out when leaving the shift state as well as when staying in the shift state. Each shift-to-shift transition results in one falling edge and in one rising edge of the JTAG clock. Note that read data will be aligned from most significant byte if the data is shorter than 32-bits.

21.4. State moves

Navigating between stable JTAG states can either be done by the hardware or manually. When the hardware performs the state moves, via JTAG _move, each state move will take seven (7) clock cycles except for the shift states. Navigating between stable JTAG states manually can be necessary when the target hardware has specific requirements on the path taken through the state machine (a dubious use of the JTAG protocol) to operate correctly. Using JTAG _move is generally faster as the ZY1000 CPU will be able to run in parallel to the JTAG master hardware.

21.5. Clocking out / in data

The ZY1000 JTAG hardware can clock out data very efficiently as the JTAG hardware allows the CPU to fill the JTAG master command FIFO in an open loop. The ZY1000 CPU will stall when the JTAG master FIFO is full.

To place a clock in or out operation in the FIFO, first write the data to be written (LSB is clocked out first, MSB is clocked in first) to JTAG_data, then write the command to JTAG_MOVE:

1. Write to JTAG_data
2. Write to JTAG_move. This will put a request in the FIFO, the ZY1000 will stall when the FIFO is full. Not that the final resting state can not be a shift state as this would commit the JTAG master hardware to clock out or in one more but when leaving the shift state.

To read the data clocked in:

1. Wait for FIFO empty using the WAITIDLE command. It is possible to implement polling as well, but this will be orders of magnitude slower.
2. Read JTAG_data

A subtle point about the JTAG state machine definition is that the data is clocked in or out when leaving the shift state, so one has to be careful not to end up in the shift state when data is clocked out.

The number of cycles, 'nbr_of_cycles', in JTAG_move merits a bit more explanation:

- nbr_of_cycles == 0: Go to end_staate (no tck cycle afterwards). If already in end_state, no action.
- nbr_of_cycles == 1. Go to repeat_state (no cycle) go to end_state.
- nbr_of_cycles == 2. Go to repeat_state (one cycle) go to end_state.

- ‘nbr_of_cycles == n. Go to repeat state (n – 1 cycle) go to end_state.

When going to end_state, one tck-cycle will be executed regardless if end_state and repeat state is equal.

21.6. JTAG _move sample

The following sequence will reset the TAP, set a 4-bit IR to IDCODE and get a 32-bit result from DR. Note that the transitions between states take seven (7) cycles. If you want to do this in the minimum number of cycles, as opposed to as quickly as possible, you could use the manual path move mode. The JTAG master hardware uses a few more cycles, but runs in parallel to the ZY1000 CPU so throughput is better.

```
# Here ZY1000 register base = 0x08000000
# Reset TAP controller, clock out more than 5
TMS=1 poke 0x08000008 0x700
# Navigate to IR set a 4 bit value=5, navigate to shift-DR
poke 0x0800000c 0x5
poke 0x08000008 0x4b3
# Clock in 32 bit of data, go to run-test-idle
state poke 0x08000008 0x2038
# Things are so slow, we don’t have to wait for FIFO to
empty peek 0x0800000c
```

21.7. JTAG _move write

Sometimes it is necessary to manually navigate a path through the JTAG state machine. This can be necessary when some TAP uses side effects of specific paths (dubious practice) to set or clear flags for instance.

To do a manual path move:

1. Issue WAITIDLE
2. Write to JTAG_man_write as many times a necessary
3. Write to JTAG_state_write to tell the JTAG controller what the end state is. By design the JTAG controller makes no assumptions about the end state here as this can be used in non-JTAG scenarios such as switching from JTAG to SWD.

21.8. TAP state numbering

‘0’	Test Logic Reset *
‘1’	select dr scan
‘2’	capture dr
‘3’	Shift-DR *
‘4’	Exit 1-dr
‘5’	Pause-DR *
‘6’	exit2-dr
‘7’	update dr
‘8’	Run-Test/Idle
‘9’	select-ir-scan
‘a’	capture-ir
‘b’	Shift-IR
‘c’	exit1-ir
‘d’	Pause-IR *
‘e’	exit2-ir
‘f’	update-ir

* = stable end states

21.9. Sample code

OpenOCD uses the ZY1000’s JTAG over TCP/IP protocol, so there is example code inside the file: src/jtag/zy1000/zy1000.c. This file is copyrighted by Zylin AS and Ultimate Solutions, Inc. Contact Ultimate Solutions for a license to include this file in a non-GPL application. A license to use the file as an example to write your own code is hereby granted.

22. Remote access to the ZY1000

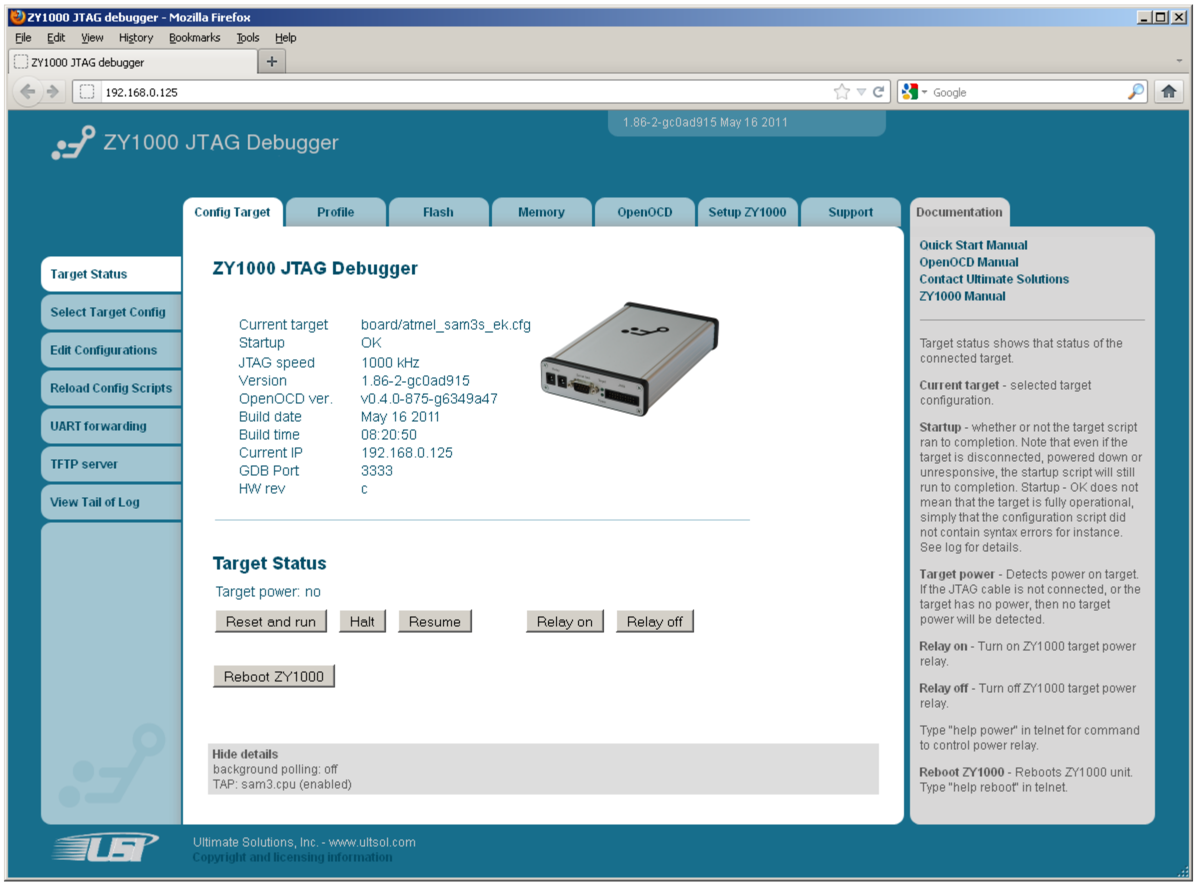
22.1. Secure remote access

If you need to access your ZY1000 remotely in a secure manner, it is recommended that you set up an ssh server or dial into the ZY1000 via ssh. The ZY1000 can run an ssh server in the case where it runs the Linux version of its' firmware. The follow table is a list of ports, defined locally, that are mapped to the ZY1000 ports.

Local port	Remote port	Description
7777	7777	JTAG over TCP/IP
7778	80	Web interface
7779	23	Telnet
7780	3333	GDB server
7781	1234	Resets ZY1000 upon receiving a connect
7782	69	tftp server
7783	70	internal - tftp profile
7784	5555	serial port forwarding
7784	22	ssh port - used when ZY1000 runs Linux

22.2. Accessing the ZY1000 remotely

Once you have setup the necessary configuration, you can access the ZY1000 by accessing ports on your local machine. To test the configuration, you can access the GUI interface by using the following URL: http://localhost.:7778.



ZY1000 JTAG Debugger

Current target	board/atmel_sam3s_ek.cfg
Startup	OK
JTAG speed	1000 kHz
Version	1.86-2-gc0ad915
OpenOCD ver.	v0.4.0-875-g6349a47
Build date	May 16 2011
Build time	08:20:50
Current IP	192.168.0.125
GDB Port	3333
HW rev	c

Target Status

Target power: no

Hide details
background polling: off
TAP: sam3.cpu (enabled)

Target status shows that status of the connected target.

Current target - selected target configuration.

Startup - whether or not the target script ran to completion. Note that even if the target is disconnected, powered down or unresponsive, the startup script will still run to completion. Startup - OK does not mean that the target is fully operational, simply that the configuration script did not contain syntax errors for instance. See log for details.

Target power - Detects power on target. If the JTAG cable is not connected, or the target has no power, then no target power will be detected.

Relay on - Turn on ZY1000 target power relay.

Relay off - Turn off ZY1000 target power relay.

Type "help power" in telnet for command to control power relay.

Reboot ZY1000 - Reboots ZY1000 unit. Type "help reboot" in telnet.

Ultimate Solutions, Inc. - www.ultsol.com
Copyright and licensing information

23. UrJTAG on the ZY1000

The ZY1000 can be used with UrJTAG. UrJTAG is an open source JTAG tool that is oriented towards boundary scan and low level SVF programming and programming. Typical tasks would be to measure and manipulate the pads of a BGA package or program an SVF file into an FPGA or CPLD.

UrJTAG can be run either on your development PC or on the ZY10000. The advantage of running it on the ZY1000 is that you do not need to install UrJTAG on your development PC. However, running UrJTAG on your development PC will yield significantly faster operation for cases where the CPU processing power is a factor. For example, SVF parsing is slower on the ZY1000. An advantage of the ZY1000 is that is has a TCP/IP interface and therefore requires no drivers to be installed.

23.1. Example of programming an FPGA with an SVF file from ZY1000

1. Copy any bsdl files you need to e.g. /bsdl on the ZY1000 via e.g. ftp.
2. Copy the urjtag.txt file below to on ZY1000 and test.svf file to e.g. "/"
3. Telnet into the ZY1000 (telnet 10.0.0.167 9999) and run the command below to program the SVF into the FPGA.

```
HOME=/tmp /opt/zy1000/bin/jtag -n urjtag.txt
```

23.1.1. urjtag.txt file

```
# You may need to upload bsdl files for your
# part, e.g. to /bsdl via ftp
bsdl path /bsdl
cable ZY1000 server=127.0.0.1
pod reset=0
detect
get signal IO_A7
scan
svf /test.svf stop progress ref_freq=1000000
```

23.1.2. Output from telnet session

```
Initializing Zylin ZY1000 JTAG probe
IR length: 6
Chain length: 1
Device Id: 00000010011000011000000010010011 (0x02618093) Filename:
                /bsdl/xc3s200an_ft256.bsd
IO_A7 = 1
IO_E2: 0 > 1
Parsing 43030/43035 ( 99%)
Scanned device output matched expected TDO values.
```

23.2. Example of programming an FPGA with an SVF file from a PC

Here, some extra bsdl files from Linux are placed into the “bsdl” folder on the developer’s PC, relative to the current working directory.

To program an SVF file, run the following UrJTAG command:

```
jtag -n urjtag.txt
```

23.2.1. Output

```
Initializing Zylin ZY1000 JTAG probe
IR length: 6
Chain length: 1
Device Id: 00000010011000011000000010010011 (0x02618093) Filename:
                bsdl/xc3s200an_ft256.bsd
IO_A7 = 1
Parsing 43030/43035 ( 99%)
```

23.2.2. urjtag.txt command file

```
bsdl path bsdl
cable ZY1000
server=10.0.0.167 pod
reset=0
detect
get signal
IO_A7 scan
svf ../XilinxArmJtagOnly/McsJtag.svf stop progress ref_freq=100000
```

24. Using the ZY1000 for remote development and maintenance

As the ZY1000 has a standalone and Ethernet networking capability, it can be especially effective in solving remote development and maintenance issues. This chapter briefly describes some of the problems and their solution with the ZY1000.

24.1. Traversal of NAT, DMZ’s and firewalls

The ZY1000 can run Linux. Therefore, it can dial into a server. NAT and firewalls often allow outgoing, but not incoming connections. When a company’s policies are restrictive, it may be necessary to place the ZY1000 in a DMZ (demilitarized zone) where no connections with the inside LAN and outlying outgoing connections are possible.

24.2. Untrained staff at remote site

When staff at a remote site is untrained, they will be able to navigate to the ZY1000 web menus an click “connect” on the ZY1000 support menu. Staff at the “other end” that needs access to the ZY1000 remotely can then, similarly, navigate to a Ultimate Solutions web page that allows connecting to the ZY1000.

24.3. Crashes in the field after a long period of operation

If a problem arises after a long period in the field, then with a ZY1000 at the remote site, you will be able to inspect the problem by possibly dumping memory, and / or logs.

24.4. Field versus lab behavior

Developers will, of course, address all problems that can be addressed in the development lab first and then what remains are problems in the field. These problems can be crashes, performance issues, and / or interoperability issues with other equipment.

24.5. Supporting early adopters

Products may have to be shipped before they are ready. With a ZY1000, early adopters of the product can be effectively supported.

24.6. Adding networking capability to legacy products

Legacy products may need some networking capability. This can be implemented by using semi-hosting where the ZY1000 handles networking and the target has a few small functions to talk to the ZY1000. The application can thus be network-enabled without requiring a re-spin of the PCB and in some cases, without even modifying the application.

24.7. Remote serial port access

Many applications have a serial port. This serial port can be forwarded over TCP/IP using the ZY1000, thus implementing a powerful “serial port extender” when combined with the ZY1000 running Linux.

24.8. Power cycling capability

The ZY1000's relay can be used to power cycle remote equipment. This is not limited to the ZY1000's target, but could be any low-voltage powered hardware. For more information about the relays used in the ZY1000, refer to the following web page:
`https://www.elfaelektronikk.no/elfa3~no_en/elfa/init.do?item=37-209-01&toc=0`

24.9. Semi-hosting

Semi-hosting is a capability where the target can access the file system and network of the ZY1000, this adding networking, storage and Linux capability to a target which does not have this capability itself.

24.10. Logging

The application can store logging information to the ZY1000 and the log then stored on the ZY1000, can then be accessed remotely.

24.11. Adding web interface capability to application

Some applications may need a web interface for early periods of their life or only for a few of the deployed targets. A web interface can be implemented on the ZY1000 running Linux. The web interface and the applications talks via semi-hosting.

24.12. Additional storage for target

If an application needs additional storage, then the ZY1000 and semi-hosting can implement this storage without adding the overhead to the full production series of the application target.

24.13. Adding Linux capability to legacy targets

If a target needs Linux to perform certain tasks, then the ZY1000 can implement the Linux of legacy targets and communication between the ZY1000 and the target is a simple semi-hosting protocols.

25. Translating configuration script syntax to the ZY1000

The ZY1000 configuration script syntax may not match that of a configuration script that you have for your hardware. Although there are many proprietary syntaxes out there, the ZY1000 offers a straightforward capability to write small pieces of script that will greatly simplify the target configuration translation process.

```
# Some helper proc’s to translate BDI syntax to ZY1000 speak
#
# Note that Tcl only supports ’#’ as a prefix to the
# beginning of a statement. If you want to have a comment
# at the end of a line, use ’;’ to start a new statement and then a comment
#
# Change:
#
# WM32 0x53FC0000 0x00000040           ; enable ipu
#
# to:
#
# WM32 0x53FC0000 0x00000040           ;# enable ipu


#Example:
# WM32 0x53FC0000 0x00000040           ; enable ipu
proc WM32 {address val} {
        mww [string tolower $address] [string tolower $val]
}

# Example:
# WM8  0x80000033 0xDA                 ; dummy write only address matter
proc WM8 {address val} {
        mwb [string tolower $address] [string tolower $val]
}

# Example:
# WREG CPSR 0x000000D3
proc WREG {regname val} {
        reg [string tolower $regname] [string tolower $val]
}

# Example:
# WCP15 0x0001 0x00050078              ; CP15 control register
proc WCP15 {regs value} {
        arm mcr 15 [expr ($regs>>12)&0x7] [expr ($regs>>0)&0xf] \
          [expr ($regs>>4)&0xf] [expr ($regs>>8)&0x7] $value
}
```

26. Fast production CFI programming trick

CFI flashes can be quite slow to erase and program. A 16M byte CFI flash could have erase and write performance of a theoretical maximum of approximately 200 to 100K bytes per second. For an erase and write cycle, this yields a best case performance of 1 (1 / (200 + 1) / 100) = 66K bytes per second. For a 16M byte flash file this would yield a best case production time of approximately four (4) minutes. The ZY1000 will regularly be able to reach 50% or better of the maximum theoretical performance.

However, there is a trick that will allow you to reach better performance. The ZY1000 can upload at approximately 1M byte per second to RAM running at 16MHz JTAG frequency in about 20 seconds for 16M byte flash image. This small image could be an application that erase, writes, and verifies the CFI flash.

The following describes the required steps:

1. Connect the ZY1000 to the target hardware.
2. The ZY1000 detects that the hardware is connected, resets the target, and uploads an application to RAM and stars the application.
3. The ZY1000 or the application gives an indication that it is running.
4. Disconnect the ZY1000, but leave the target powered on.
5. Connect the ZY1000 to the next target.
6. Once the target is done, erase, program, verify, and self-test, it flashes a LED or gives some other type of indication. Alternately, the board can be connected to a second ZY1000 that performs the verify.

This yields an effective programming throughput of 1M byte per second, which is a 10x improvement on CFI’s theoretical maximum.