PEEDI

Powerful Embedded Ethernet Debug Interface

User's Manual

Version 22.04.0



RONETIX
DEVELOPMENT TOOLS

February, 2019
Ronetix has made every attempt to ensure that the information in this document is accurate and complete. However, Ronetix assumes no responsibility for any errors, omissions, or for any consequences resulting from the use of the information included herein or the equipment it accompanies. Ronetix reserves the right to make changes in its products and specifications at any time without notice. Any software described in this document is furnished under a license or non-disclosure agreement. It is against the law to copy this software on magnetic tape, disk, or other medium for any purpose other than the licensee's personal use.

Ronetix Development Tools GmbH
Hirschstettner Str.19/Z110
1220 Vienna
Austria
Tel: +43 1 236 1101
Fax: +43 1 236 1101 9
Web: www.ronetix.at
E-Mail: info@ronetix.at

Acknowledgments:
ARM, ARM7, ARM9, ARM11, Cortex-M, Cortex-A and Thumb are trademarks of ARM Ltd.
PowerPC and ColdFire are trademarks of Freescale Ltd.
Blackfin is trademark of Analog Devices Ltd.
Windows, Win32, Windows CE are trademarks of Microsoft Corporation.
Ethernet is a trademark of XEROX.
MIPS is a trademark of MIPS Technologies.
AVR32 is a trademark of Atmel.
All other trademarks are trademarks of their respective companies.

Copyright© 2005-2022 Ronetix Development Tools GmbH

1 Document Revision History

Revision	Date	Notes
3.0	09.12.2020	Initial
3.1	22.03.2021	Add "memory management" commands
3.2	30.04.2021	Add information about Renesas RA Cortex-M33 programming
21.7.0	09.07.2021	Add TCL interpreter Add ‘tcl’ command
21.11.0	02.11.2021	“Specifications”: Add max supported SD card size
22.02.0	14.02.2022	Add “rtt” commands
22.04.0	29.04.2022	Add missing subcommand ‘c’ in ‘amp’ command

Table of Contents

PEEDI....1
Powerful Embedded Ethernet Debug Interface....1
1 Document Revision History....3
2 Introduction....12
2.1 PEEDI in the development process....13
2.1.1 Single developer environment....13
2.1.2 Multiple developers environment....14
2.2 PEEDI in the manufacturing process....14
2.2.1 PEEDI as a standalone FLASH programmer....15
2.2.2 PEEDI as a device tester....16
2.2.3 High productivity with the Multi Program feature....16
3 Installation....16
3.1 Hardware installation....17
3.1.1 Connection instructions....17
3.2 Software installation....19
4 Using PEEDI....19
4.1 PEEDI interface....19
4.2 Setup with RedBoot....20
4.2.1 RedBoot Configuration....21
4.3 Firmware update procedure....22
4.3.1 Update via RS232....23
4.3.2 Update via Ethernet....23
4.4 RedBoot commands used with PEEDI....25
update....25
config....26
memtest....26
4.5 Configure PEEDI....27
4.5.1 Network configuration....27
4.5.2 Target configuration file....27
4.5.3 TCL script language....28
TCL language syntax....29
TCL environment....29
TCL interpreter....29
TCL examples....29
4.5.4 Sections....33
Section LICENSE....33
Section DEBUGGER....33
PROTOCOL....33
REMOTE_PORT....34
FLASHn....34
Section TARGET....34
PLATFORM....34
Section PLATFORM_xxx - parameters for all targets with JTAG interface....35
JTAG_CHAIN....35
JTAG_TDO_DELAY....35
TRST_TYPE....35
RESET_TIME....36
RESET_TYPE....36
WAKEUP_TIME....37
TIME_AFTER_RESET....37

DBGREQ_OUTPUT....37
COREn....37
COREn_STARTUP_MODE....38
COREn_INIT....38
COREn_FLASHm....38
COREn_ENDIAN....39
COREn_BREAKMODE....39
COREn_WORKSPACE....40
COREn_DATASPACE....40
COREn_PATH....40
COREn_FILE....40
COREn_LOCKOUT_RECOVERY....41
COREn_OS....42
Section PLATFORM_ARM and PLATFORM_ARM11....42
COREn....42
COREn_VECTOR_CATCH_MASK....42
COREn_DCC_PORT....43
COREn_USE_FAST_DOWNLOAD....43
COREn_BREAK_PATTERN....43
Section PLATFORM_Cortex & Section PLATFORM_Cortex_SWD....44
COREn....44
COREn_APSEL....44
COREn_DEBUG_ADDR....45
COREn_DAPPC....45
COREn_PERIODIC_TASK....45
COREn_SWO....46
COREn_PROFILING....46
Section PLATFORM_XSCALE....47
COREn....47
COREn_USE_FAST_DOWNLOAD....47
COREn_DEBUG_HANDLER_ADDR....47
COREn_VECTOR/RELOCATED_UNDEF/SWI/PABORT/DABORT/RES/IRQ/FIQ....48
Section PLATFORM_MPC5200....49
COREn....49
COREn_BOOT_ADDR....49
COREn_MEMDELAY....49
Section PLATFORM_MPC5500....50
COREn....50
COREn_NEXUS3_ACCESS....50
MPC5XXX_AUX_TAP_CMD, COREn_AUX_TAP_CMD....50
Section PLATFORM_MPC8300....51
COREn....51
COREn_BOOT_ADDR....51
COREn_RCW....51
COREn_MMU_PTBASE....51
Section PLATFORM_MPC8500....52
COREn....52
COREn_MMU_TRANS....52
COREn_MMU_PTBASE....52
Section PLATFORM_QorIQ_P....53
COREn....53
COREn_REGLIST....53

COREn_MMU_TRANS....54
COREn_MMU_PTBASE....54
COREn_PMEM_BASE....54
Section PLATFORM_PPC400....55
COREn....55
Section PLATFORM_COLDFIRE....55
BDM_CLOCK....55
CORE....55
CORE_MEMMAP....56
Section PLATFORM_BLACKFIN....56
COREn....56
COREn_VMEM....57
COREn_VMEM_WINDOW....57
COREn_VMEM_PINS....57
CORE_MEMMAP....58
Section PLATFORM_MIPS....58
COREn....58
Section PLATFORM_AVR32....59
COREn....59
COREn_BLOCK_ACCESS....59
Section INIT....59
Section FLASH....60
NOR FLASH programming....61
I2C Programming....61
SPI FLASH programming....62
NAND FLASH programming....63
OneNAND FLASH programming....65
MMC/SD card programming....65
Atmel SAM3/SAM4 programming....65
Atmel AVR32UC3 programming....65
Freescale Kinetis programming....66
TI/Luminary LM3S programming....66
NXP LPC2000 programming....66
Nordic Semiconductor nRF51 ans nRF52 programming....68
Freescale MAC7100 programming....69
Freescale ColdFire V2 programming....69
Freescale MPC5000 programming....70
Renesas RA Cortex-M33....70
ST STM32 programming....70
ST STR7 programming....70
ST STR9 programming....71
TI TMS570 programming....71
TI TMS470 programming....72
PIC32, SmartFusion A2F, ADuC, EFM32 programming....73
CHIP....73
CHECK_ID....73
PART_ID....74
PARTITION....74
BANK....74
ACCESS_METHOD....74
CHIP_WIDTH....75
CHIP_COUNT....75

CHIP_SIZE....75
BASE_ADDR....76
FILE....76
SPI_MODE....76
AUTO_ERASE....76
AUTO_LOCK....77
CPU_CLOCK....77
SECURE_FLASH....77
SET_VECTORS_CHECKSUM....78
DATA_BANK....78
BANK_SIZE....78
DATA_FLASH....78
F2F4_PSIZE....79
PROTECTION_KEY0 – PROTECTION_KEY3....79
ALLOW_ZERO_KEYS....79
CPU....80
SPI_DIV....80
nSPI....80
nCS....81
SPI_SPCK, SPI_MISO, SPI_MOSI, SPI_CS....81
CMD_BASE....81
DATA_BASE....82
ADDR_BASE....82
CS_ASSERT/RELEASE....82
ALE_ASSERT/RELEASE....82
CLE_ASSERT/RELEASE....82
BAD_BLOCK_TABLE....83
BAD_BLOCKS....83
ERASE_BAD_BLOCKS....83
SWAP_BI....84
OOB_INFO....84
DAVINCI_UBL_DESCIPTOR_MAGIC....86
DAVINCI_UBL_DESCIPTOR_ENTRY_POINT....86
DAVINCI_UBL_DESCIPTOR_LOAD_ADDR....86
DAVINCI_UBL_MAX_IMAGE_SIZE....86
NUM_ECC....87
HEADER....87
IPS_BASE....87
SPIFI_BASE....88
NCB_DATA....88
LDLB_DATA....88
SERIAL_NUM....88
I2C_ADDR....89
I2C_DELAY....89
SDA_SET/CLR, SDA_IN/OUT, SDA_READ, SCL_SET/CLR....90
CS_ASSERT/RELEASE, SCLK_SET/CLR, MOSI_SET/CLR, MISO_READ....90
Section OS....90
ITEM....90
Section SERIAL....92
BAUD....92
STOP_BITS....92
PARITY....92

TCP_PORT....92
Section TELNET....93
PROMPT....93
BACKSPACE....93
Section DISPLAY....94
VOLUME....94
Section ACTIONS....94
4.6 CPU specific considerations....95
4.6.1 Philips LPC2000 family....95
4.6.2 ST STM32 family....95
4.6.3 Intel XScale family....95
4.6.4 Freescale PowerQUICC II Pro MPC83XX family....96
4.6.5 Analog Devices Blackfin family....97
4.7 Boot sequence....98
4.8 Multiple core support....100
4.9 Script execution using the front panel interface....102
4.10 Serial Interface....104
4.11 ARM DCC Interface....104
4.12 Working with gdb....105
4.13 Debugging Linux kernel....107
4.14 Target OS thread awareness....108
4.15 Working with CLI (Command Line Interface)....111
4.15.1 File path convention....112
4.15.2 CLI commands....114
help....114
transfer....115
type....115
wait....116
core....116
clock....117
run....117
tcl....118
go....118
gm....119
step....120
execute....120
set....121
halt....121
reset....122
reboot....123
echo....123
jtag....124
beep....124
target....124
quit....125
info....125
Info flash....126
info registers....126
info target....127
info config....127
info ice....127
info cp15, info cp14....128

info spr....130
info ctrl....130
info breakpoint....130
memory....131
memory read....131
memory write....132
memory or....133
memory and....134
memory crc....134
memory load....135
memory multi load....136
memory verify....136
memory dump....137
memory management....138
memory test....138
flash....139
flash set....139
flash blank....140
flash erase....140
flash lock....141
flash unlock....141
flash query....142
flash program....142
flash multi erase....143
flash multi blank....144
flash multi program....144
flash verify....145
flash multi verify....146
flash dump....147
flash read....147
flash info....148
flash find....148
flash test....149
flash area....149
flash this....150
flash this hidden....151
flash this markbad....151
flash this nvmbit....152
flash this secure....152
flash this option....152
flash this option....153
flash this write....153
flash this part....154
flash this prot....154
flash this prot read....155
flash this prot program....155
flash this ppb....156
flash this isc_erase....156
flash this isc_conf_write....157
flash this isc_conf_read....157
flash this isc_conf_boot_bank....158
flash this isc_conf_lock....158

breakpoint....158
breakpoint add....159
breakpoint list....160
breakpoint delete....160
card....161
card cd....161
card rd....162
card dir....162
card copy....162
card type....163
card delete....163
card rename....164
eeprom....164
eeprom dir....165
eeprom copy....165
eeprom type....165
eeprom delete....166
eeprom rename....166
eeprom format....167
eeprom alias....167
test....168
amp....168
rtt setup....169
rtt start....169
rtt list....170
rtt server_start....170
rtt server_stop....170
4.16 Real Time Transfer (RTT)....171
4.16.1 Working with the FLASH programmer....171
4.17 Multiple FLASH support....173
4.18 Working with a MMC/SD memory card....173
4.19 JTAG cable adapters....174
4.20 PEEDI licenses....175
5 Specifications....176
5.1 JTAG Target connector signals....177
5.2 RS232 Connector (DB9F, female)....178
5.3 Schematics....179
6 Warranty....179
7 PEEDI Package contents....179
8 FAQ....180
9 Glossary....184

Table of Figures

Figure 2.1: Single developer environment....13
Figure 2.2: Multiple developers environment....14
Figure 2.3: FLASH programmer with PC....15
Figure 2.4: FLASH programmer without PC....15
Figure 2.5: Multi board programming....16
Figure 3.1: PEEDI Front Panel....17
Figure 3.2: PEEDI Rear Panel....17
Figure 3.3: Direct host connection....18
Figure 3.4: LAN connection....18
Figure 3.5: Target connection....18
Figure 4.1: Front panel interface....19
Figure 4.2: Rear panel interface....20
Figure 4.3: MMU_PTBASE....97
Figure 4.4: PEEDI Boot Sequence....99
Figure 4.5: Multi Target connection....100
Figure 4.6: Adapter JTAG cable....174
Figure 4.7: JTAG Adapter PEEDI-4xARM20....175
Figure 5.1: PEEDI JTAG connector....177
Figure 5.2: RS232 connector....178

2 Introduction

PEEDI (Powerful Embedded Ethernet Debug Interface) is an EmbeddedICE solution that enables you to debug software running a wide variety of processor cores via the JTAG port. JTAG is an IEEE standardized protocol that enables full control of the CPU core, giving the opportunity to debug embedded software. The PEEDI will help to reduce Time-To-Market and increase the quality of the end product.

PEEDI is a debugging and development tool that provides the ability to see what is taking place in the target system and control its behavior. PEEDI provides the services needed to perform all debugging operations. It receives command packets over the communication link and translates them into JTAG operations that are needed to provide the specific service. It can control the operation of the target processor and target system, start and stop the processor's execution; it can set breakpoints in a program, examine and store values in the processor’s registers, and examine and store program code or data in the target system’s memory. PEEDI can work in cooperation with a host computer or autonomously using a MMC/SD card.

2.1 PEEDI in the development process

In the development process PEEDI can be used mainly as a debugger JTAG interface and FLASH programmer.

Two major configurations are possible here:

- Single developer environment
- Multiple developers environment

2.1.1 Single developer environment

Using the developer's PC as a host computer - this is suitable for small projects. Here all necessary tools for compiling and debugging the project must be installed on the developers PC, including file server (TFTP, FTP or HTTP) allowing PEEDI to retrieve configuration files or executable images. In this (Figure 1) configuration the developer's PC must be connected to PEEDI in a common LAN using crossover patch cable or by Ethernet via hub/switch.

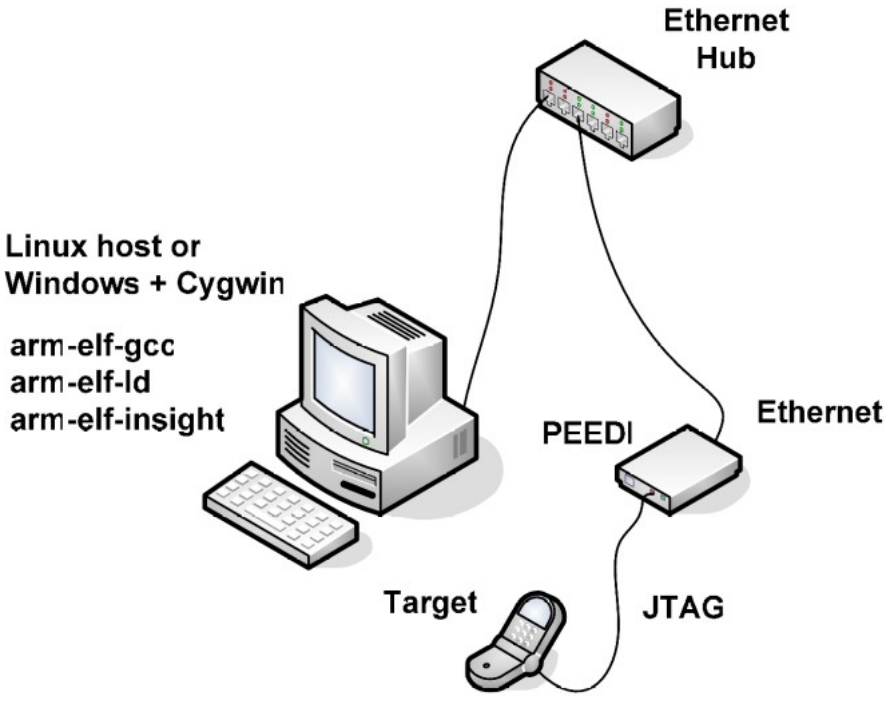


Figure 2.1: Single developer environment

2.1.2 Multiple developers environment

Dedicated server with all the necessary development tools installed is used for a host. The developer uses a PC only as a graphical terminal to logon to the server. No specific software is installed on the developer's PC, so it is very easy to set another working environment for a new developer for the project - just add a new user on the server and make a copy of the project source and make files in the user's home directory. Of course any source control tool, such as CVS or Visual Source Safe, can be used for synchronizing the project files. In this configuration (Figure 2) all devices (the server, the developers' PCs and all PEEDIs) must be connected in a common LAN.

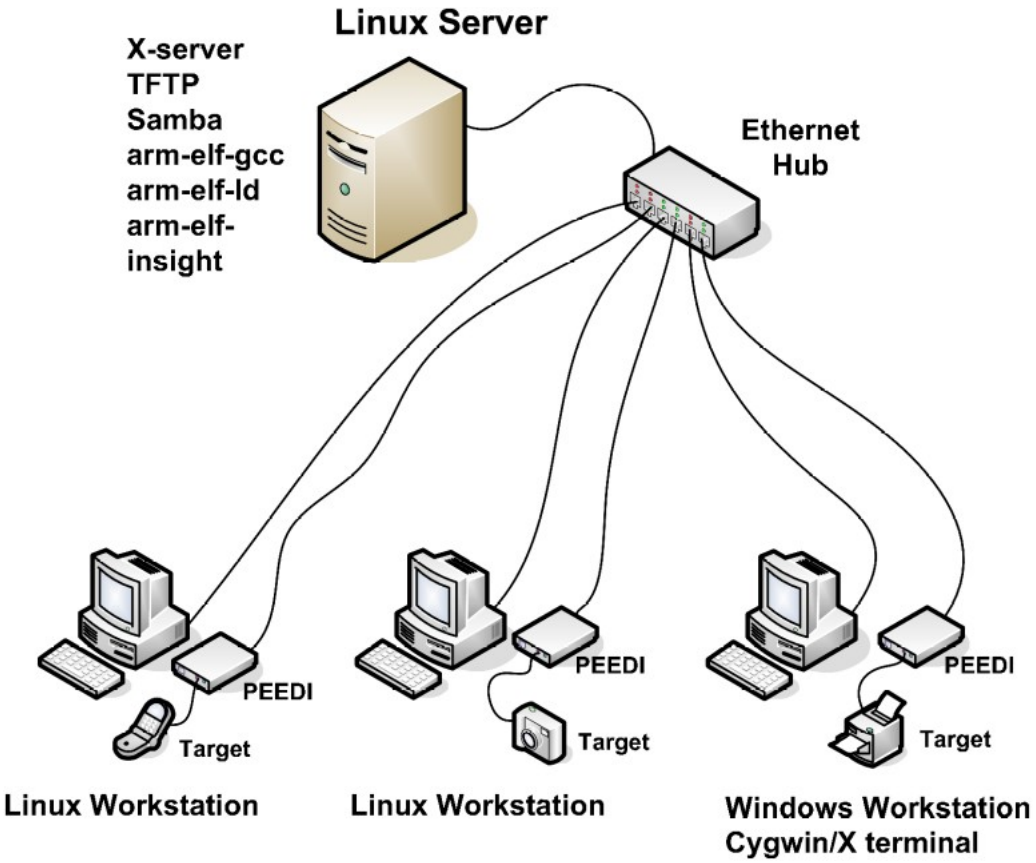


Figure 2.2: Multiple developers environment

2.2 PEEDI in the manufacturing process

PEEDI can be used in the manufacturing process as a tool for testing the device after it is assembled and as a FLASH programmer to program the device firmware. In both scenarios the host computer is not required because all the operations can be formed as script files and executed using the PEEDI's front panel interface. If all the necessary files are stored on the MMC/SD card the Ethernet connection is not required as well.

2.2.1 PEEDI as a standalone FLASH programmer

PEEDI can be used as a FLASH programmer in two ways:

- The first way (Figure 2.3: FLASH programmer with PC) is to connect to PEEDI via telnet and execute FLASH command and script files from the command line interface (CLI). This method enables users to see all the status messages in an easy, understandable format i.e. warnings and errors and therefore, maybe the preferred method.

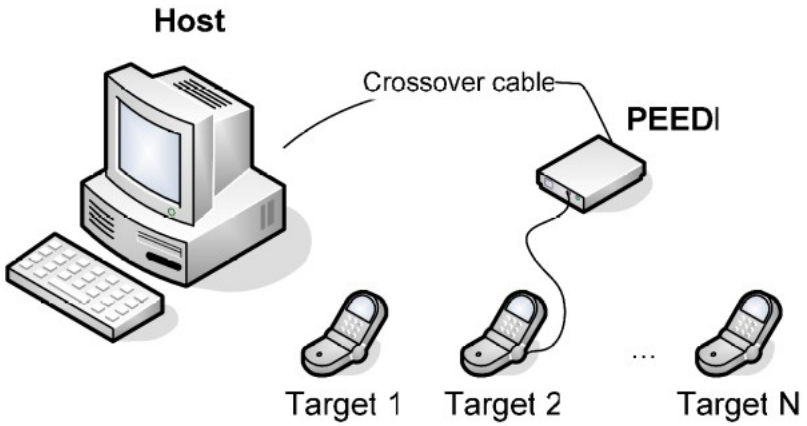


Figure 2.3: FLASH programmer with PC

- The second way (Figure 2.4: FLASH programmer without PC) is to use the front panel interface to choose, start and observe the status of scripts, which invokes the desired FLASH commands. Here you can define an AUTORUN script to be executed every time a target is connected; this way there is no need to start the script manually – very useful and time saving when large volumes of target boards need to be programmed.

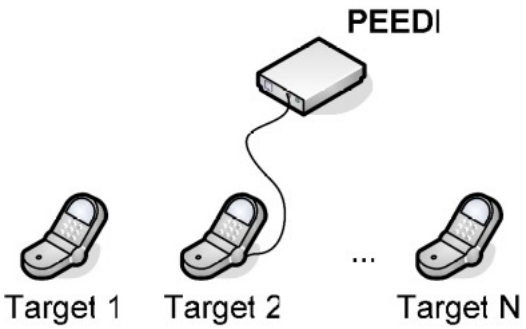


Figure 2.4: FLASH programmer without PC

2.2.2 PEEDI as a device tester

Here the PEEDI can be used in the same manner as in the previous section - making telnet connection or through the front panel interface.

Depending on the specifics of what is to be tested two options can be applied:

- Execute commands that directly make some sort of test i.e. **flash verify , memory test** , etc.
- Download executable code into target, which will perform the desired test and set a CPU register or memory on exit to a value showing the result of the test. This option is often preferred because there are virtually no limits to test examples a user can create.

2.2.3 High productivity with the Multi Program feature

With the “Multi Program” feature users can increase productivity by working on upto four boards simultaneously using a single PEEDI. The boards must be chained using a multi-core adapter (Figure 2.5: Multi board programming) available from Ronetix.

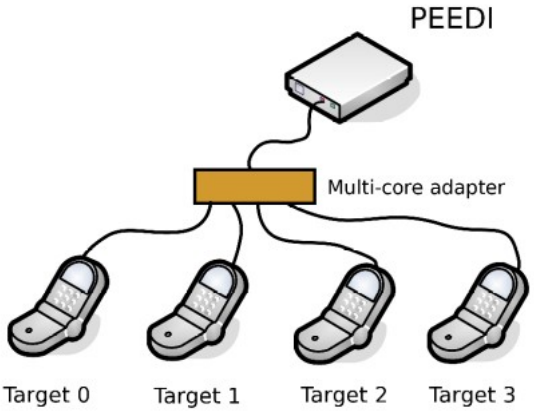


Figure 2.5: Multi board programming

3 Installation

This chapter will explain how to connect PEEDI to the target and how to configure all the tools necessary for development. Two major steps must be followed in order to set up a working PEEDI:

- Connect all required cables, this includes a power cord, target cable and if necessary an Ethernet cable, which will provide connection to a host computer or file server.

  This is explained in subsection 2.1.Hardware installation

- Install and configure insight/gdb debugger.

  This is explained in subsection 2.2.Software installation

3.1 Hardware installation

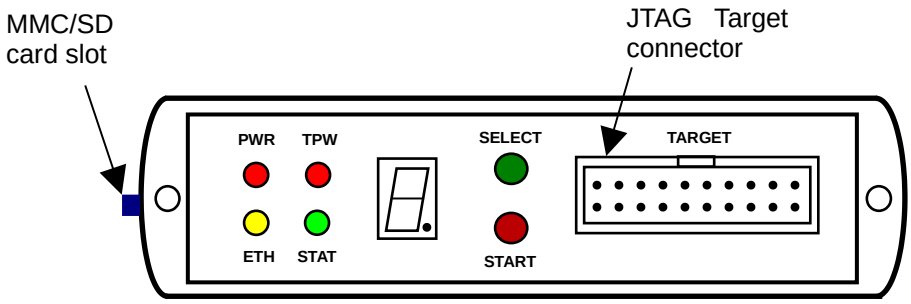


Figure 3.1: PEEDI Front Panel

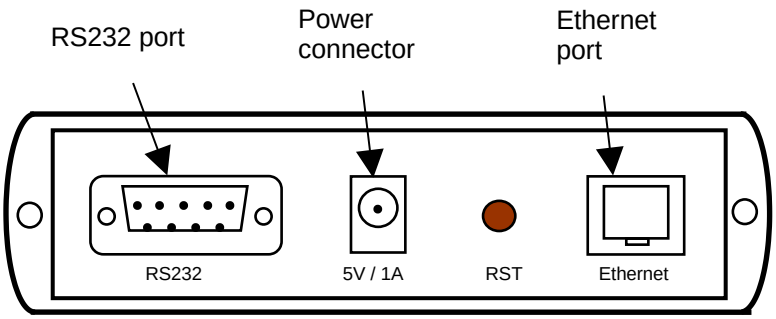


Figure 3.2: PEEDI Rear Panel

3.1.1 Connection instructions

To connect the PEEDI interface unit to your host and to the target hardware:

- Connect the host computer to an Ethernet network or directly to the PEEDI as required:
  - Direct host connection

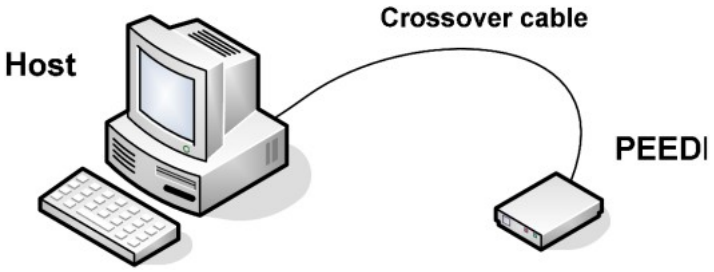


Figure 3.3: Direct host connection

- LAN connection

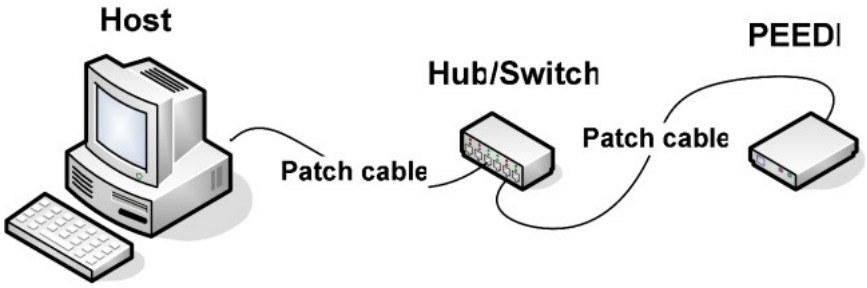


Figure 3.4: LAN connection

- Connect the PEEDI interface unit to the target hardware, using the supplied JTAG adapter and cable. The JTAG adapter must be on the PEEDI side of the JTAG cable. If your target JTAG port pinout is not standard, you may need to make your own target cable considering the PEEDI JTAG connector pinout.

Refer to subsection 4.1 JTAG Target connector signals for the PEEDI JTAG connector pinout.

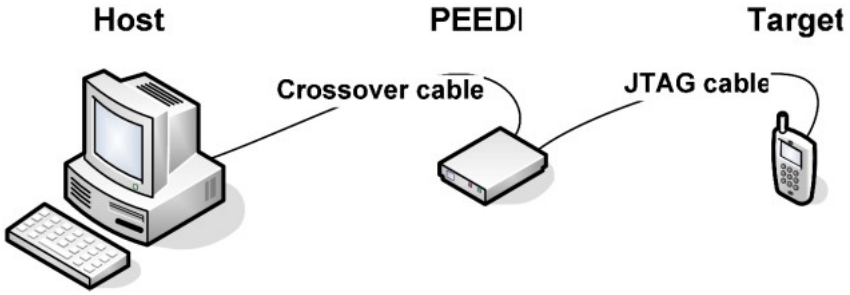


Figure 3.5: Target connection

- Power up the target hardware.
- Connect the external power supply to the PEEDI and apply power.

- When PEEDI boots, if you have a terminal connected to the RS232 port of PEEDI you will see various status messages.

3.2 Software installation

See ’Cross development with GNU toolchain and Eclipse’:

http://download.ronetix.at/toolchains/arm/arm_cross_development_guide.pdf

4 Using PEEDI

This chapter will explain PEEDI’s operating modes, PEEDI’s interface and the basic steps of configuring the software tools for working with PEEDI.

To start using PEEDI you need to:

- configure network settings
- make target configuration file

4.1 PEEDI interface

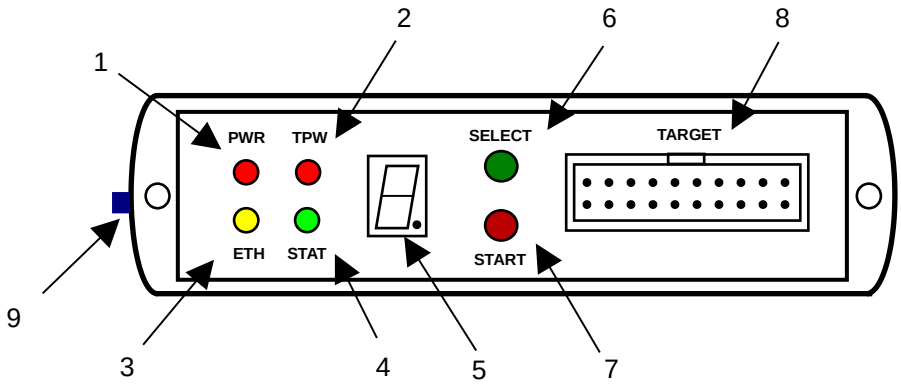


Figure 4.1: Front panel interface

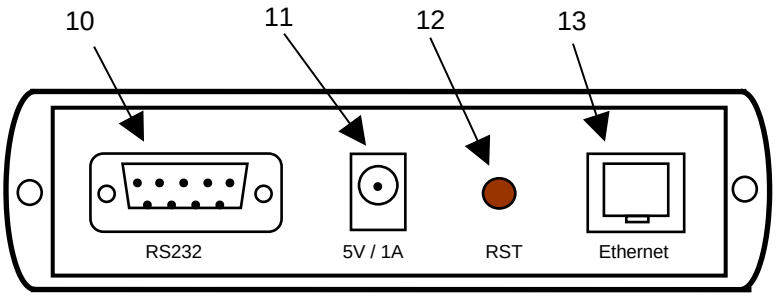


Figure 4.2: Rear panel interface

Table 4.1: PEEDI Interface

Pos.	Description
1	Power LED
2	Target power LED
3	Ethernet connect/activity LED
4	Target connect/activity LED
5	Script number/status LED display
6	Next script button
7	Start script button
8	Target connector
9	MMC/SD card slot
10	RS232 port
11	Power supply
12	Reset button
13	Ethernet port

4.2 Setup with RedBoot

RedBoot is a bootstrap loader, which during normal boot-up is used to load and launch PEEDI's executable image. RedBoot is also used to update PEEDI's firmware and to configure network

settings, which are later used by PEEDI. RedBoot has some useful testing facilities like **ping** and **memtest** .

4.2.1 RedBoot Configuration

RedBoot and PEEDI share the same network settings. To set the network you need to connect a simple terminal application set to 115200, 8, N, 1 (for example HyperTerminal) to the PEEDI's RS232 port using a serial straight-through cable with DB9M (male) and DB9F (female) connectors on each end. Next step is to restart PEEDI by pressing the RESET button while holding both front panel buttons in. This will tell RedBoot not to load and launch the PEEDI executable if available, but to wait for connection on RS232 or Ethernet. While rebooting RedBoot should output some diagnostic information on the serial port which you should see. When RedBoot is ready to accept commands, it will show the command line prompt 'RedBoot> '. Now you can use the **fconfig** command to set and save to FLASH all the parameters. When asked for different parameters please enter the following:



WARNING:
If PEEDI is set to get its network settings from a DHCP server and if the Ethernet cable is unplugged or there is no DHCP server on the Ethernet, it may take some time for PEEDI to boot. To avoid this, make sure PEEDI can reach a DHCP server.

```
Use DHCP for network configuration: yes /[no][ENTER]
Gateway IP address: X.X.X.X
Local IP address: X.X.X.X
Local IP address mask: X.X.X.X
Default server IP address, used by RedBoot and PEEDI: X.X.X.X
```



Note:
Instead of X's enter IP address digits

Next you will be asked for the path of the configuration file:

```
Target config file path:
```

Accepted paths for the different protocols are:

```
tftp://server/sub_directory/filename.cfg
ftp://user:password@server/sub_directory/filename.cfg
http://server/sub_directory/filename.cfg
card://sub_directory/filename.cfg
```



Note:
A server is indicated by its IP address.

Now you may enter DNS server used by RedBoot to resolve hostnames.

```
DNS server IP address:
```

If left blank and PEEDI is set to get the network configuration from DHCP server, the DNS server IP will also be taken from the DHCP.

Next you will be prompted for the RedBoot telnet port:

```
RedBoot telnet port: 23
```

Finally you may enter the update command default file path:

```
Update filepath:
<1> - http://www.ronetix.at/download/firmware/fw_peedi_revA_last.bin
<2> - tftp://192.168.3.1/fw_peedi_rev.A_last.bin
<custom path>
Path: 1
```

If you have changed some of the parameters you will be ask to save them at the end. If you confirm to save them they will be take effect after the next start.

4.3 Firmware update procedure

First of all you need to reset PEEDI by pressing the RESET button on the back while holding both front panel buttons in. This will tell RedBoot not to load and launch the PEEDI executable, but to wait for connection on RS232 or Ethernet.

Entering the RedBoot command line prompt can be done using two different ways: via RS232 port using serial straight-through cable and a simple terminal application set to 115200, 8, N, 1 or if the network is configured you can connect using telnet application. Once in the RedBoot's command line prompt (verify by pressing ENTER, RedBoot's prompt should appear - 'RedBoot> '), you can update the firmware the following ways:

4.3.1 Update via RS232

Firmware update via RS232 is supported by Redboot v15.12.3 or newer.

If you want to update PEEDI via RS232 your terminal application must support XMODEM or YMODEM protocols. Now execute:

```
RedBoot> update xmodem
```

or

```
RedBoot> update ymodem
```

to tell RedBoot to start listening on RS232 port for incoming packets. Next tell your terminal application to start downloading the PEEDI firmware.

4.3.2 Update via Ethernet

Now you may use **update** command to update the PEEDI firmware. You can update using TFTP, HTTP. The syntax of the **update** command is:

```
update [FILEPATH | NUMBER]
```

The command shown below will attempt to download the firmware using the default filepath entered while configuring RedBoot using the fconfig command or the path used when last update command is invoked:

```
update
```

If not changed, the default update path points to the last version of the firmware directly on the RONETIX website. If **update 1** is entered also the last version of the firmware directly from the RONETIX web site will be downloaded.

The following command will attempt to download the firmware using the HTTP protocol from a directory on the server (this syntax can be used with TFTP too):

```
RedBoot> update http://server/subdir/file.bin
```

RedBoot newer than v20.11.4 accept also:

```
RedBoot> update file.bin
```

In this case the command will be expanded to:

```
RedBoot> update tftp://server_ip/file.bin
```

where ‘server_ip’ is defined with ‘fconfig’.

After you enter the command using your specific conditions, if the host is accessible and the file is present you should see this:

```
RedBoot> update
load -r -m tftp -b 0x100000 -h 192.168.1.1 fw_peedi_revA_last.bin
..............................................................
Raw file loaded 0x00100000-0x002ab1bf, assumed entry at 0x00100000
Current Firmware:
-----------------
Hardware Ver. : 1.2
Software Ver. : 1.0

New Firmware:
-------------
Hardware Ver. 1.2
Software Ver. 1.1

Install PEEDI firmware version 1.1 (y/[n])? y

WARNING: The firmware image you are trying to load
exceeds your update license. Continue update (y/[n])? y

fis delete peedi
... Erase from 0x01840000-0x019f0000: ..........................
... Erase from 0x019f0000-0x01a00000: .
... Program from 0x007f0000-0x00800000 at 0x019f0000: .
fis create -b 0x100000 -l 0x1AB1C0 -f 0x1840000 -e 0x600040 -r 0x600000
peedi
... Erase from 0x01840000-0x019f0000: ..........................
... Program from 0x00100000-0x002ab1c0 at 0x01840000:
..........................
... Erase from 0x019f0000-0x01a00000: .
... Program from 0x007f0000-0x00800000 at 0x019f0000: .
RedBoot>
```

4.4 RedBoot commands used with PEEDI

These commands are used to update, configure, test and run PEEDI:

update

Syntax:

`update [FILEPATH | [NUMBER]]`

Description:

Update PEEDI firmware. If no argument is provided last used will be taken. Default first used argument is taken when fconfig command is used. The update 1 command downloads the latest firmware image.

Argument:

FILEPATH	- file path of the file
FILE	- filename which will be expanded to tftp://server_ip/FILE, where '`server_`ip' is defined with '`fconfig`'
NUMBER	- fixed path to latest firmware image. Available argument is 1

Example:

```
update
update http://www.myserver.com/mydir/myfile.bin
update tftp://192.168.1.1/mydir/myfile.bin
update xmodem
update ymodem
update 1
update file.bin  ; expanded to tftp://server_ip/file.bin
```

config

Syntax:

```
config file.cfg
```

Description:

Set target configuration file

Argument:

FILEPATH	- file path of the file
FILE	- filename which will be expanded to tftp://server_ip/FILE, where '`server_ip`' is defined with '`fconfig`'

Example:

```
config tftp://192.168.1.1/file.cfg
config ftp://user:password@server/sub_directory/filename.cfg
config http://server/sub_directory/filename.cfg
config card://sub_directory/filename.cfg
config file.cfg ; expanded to tftp://server_ip/file.cfg
```

memtest

Syntax:

```
memtest [-c]
```

Description:

Test available (not occupied by RedBoot) RAM

Argument:

-c - perform continuous test

Example:

```
memtest
memtest -c
```

4.5 Configure PEEDI

4.5.1 Network configuration

RedBoot and PEEDI share the same network settings. To set up the network look in 'RedBoot Configuration'.



Note:
A new PEEDI is set by the factory to get its network settings from a DHCP server. You can see the PEEDI IP by pressing and holding the green button on the front PEEDI panel. The IP will be shown on the front panel LED indicator. Or connect to PEEDI on the RS232 and the IP is shown during boot-up.

4.5.2 Target configuration file

To operate, PEEDI needs to load a target configuration file, which describes the specifics of the given target, this includes CPU type, FLASH type and metrics, RAM address and size, etc. The target configuration file includes also some settings of PEEDI itself like license keys, baud rates of the serial port, etc.

PEEDI uses **INI file** syntax for configuration and script execution. A configuration file is divided into uniquely named sections. The beginning each of section is denoted by a line of the following form: **[section_name].** Each section is followed by lines up to the next section. If a section contains only commands, it can be executed (with the CLI command **run**) and labeled as script. A semicolon (;) at the beginning of the line indicates a comment. Comment lines are ignored.

The target configuration file can be loaded from TFTP, FTP or HTTP server, MMC/SD card or the internal EEPROM and includes some mandatory sections. Multiple PEEDIs may load single shared target configuration file.

4.5.3 TCL script language

PEEDI implements a simple TCL interpreter which allows executing of scripts. There are 3 ways to execute TCL statements:

- in target configuration file: scripts with extension .tcl

  Example:

```
; TCL script defined in a configuration file
[demo.tcl]
set x 0x12345678
puts $x
```

```
peedi> run $demo.tcl
```

- in text file with extension .tcl

```
; TCL script defined in a text file  example.tcl
set x 0x12345678
puts $x
```

```
peedi> run tftp://192.168.1.0/example.tcl
```

- in the PEEDI console using the command tcl

```
peedi> tcl “x 0x1234; puts $x”
```

The TCL interpreter implements the following commands:

- subst arg
- set var val
- while cond loop

- if {cond} {then} {cond2} {then2} {else}
- proc name args body
- return
- break
- continue
- mon – execute a PEEDI console command
- mrw, mrh, mrb – memory read 32/16/8
- arithmetic operations: +, -, *, /, <, >, <=, >=, ==, !=, &, |
- help – print all supported commands

TCL language syntax

Tcl script is made up of commands separated by semicolons or newline symbols. Commands in their turn are made up of words separated by whitespace. To make whitespace a part of the word one may use double quotes or braces.

Any symbol can be part of the word, except for the following special symbols:

- whitespace, tab - used to delimit words
- \r, \n, semicolon or EOF - used to delimit commands
- Braces, square brackets, dollar sign - used for substitution and grouping

A hash (#) at the beginning of the line indicates a comment. Comment lines are ignored.

TCL environment

The interpreter creates a new environment for each script, file or CLI command.

TCL interpreter

- If argument starts with [- evaluate what's inside the square brackets and return the result.
- If argument is a quoted string (e.g. {foo bar}) - return it as is, just without braces.

TCL examples

Example:

http://download.ronetix.at/peedi/cfg_examples/cortex-m/stm32.cfg

```
; TCL script defined in a configuration file
[demo.tcl]
#-----------------------------------------------------------------------
# IDCODE address of STM32H7
set idcode 0x5C001000
puth $idcode

#-----------------------------------------------------------------------
# set 'dev_id' with the value read from address 'idcode'
set dev_id [mrw $idcode]
puth $dev_id

#-----------------------------------------------------------------------
# memory write to 0x20000000
set addr 0x20000000
set val 0x11223344
mon "memory write $addr $val"

# read back, should print 0x11223344
puth [mrw $addr]

#-----------------------------------------------------------------------
# memory read from 0x8000000
set x 0x08000000
mon "memory read $x 4"

#-----------------------------------------------------------------------
# if (dev_id == 0x10016483)
#    puts "STM32H72x/73x"
# else
#    puts "unknown CPU"
if {== $dev_id 0x10016483} {puts "STM32H72x/73x"} \
   {== $dev_id 0x20036450} {puts "STM32H74x/75x"} \
   {puts "unknown CPU"}

if {< 1 2} {puts "OK: A"}    {puts "FAIL: B"}
if {> 1 2} {puts "FAIL: A"} {puts "OK: B"}
if {> 1 2} {puts "OK: A"}
```

```
#--------------------------------------------------------------------------
# x = 0
# if (x == 0)
#    puts "OK: A"
# else
# if (x == 1)
#    puts "FAIL: B"
# else
#    puts "FAIL: C"
set x 0;
if {== $x 0}                \
     {puts "OK: A"}         \
{== $x 1}                   \
     {puts "FAIL: B"}       \
{puts "FAIL: C"}

set x 1
if {== $x 0} {puts "FAIL: A"} {== $x 1} {puts "OK: B"} {puts "FAIL: C"}

set x 2
if {== $x 0} {puts "FAIL: A"} {== $x 1} {puts "FAIL: B"} {puts "OK: C"}

#--------------------------------------------------------------------------
# should print '5'
# x = 3
# while (x < 5)
#    x = x + 1
set x 3
while {< $x 0x005} {set x [+ $x 1]}
puts $x

#--------------------------------------------------------------------------
# should print '12'
# x = 0
# while (1)
# {
#    x = x + 2
#    if (x > 10)
#          break;
# }
set x 0
while {== 1 1}         \
{                           \
     set x [+ $x 2]; \
     if {> $x 10}      \
          {break}     \
}
puts $x
```

```
#-------------------------------------------------------------------------
# Bitwise INVERT and AND: should print '0x12340000'
# x = x & ~0xFFFF
set x [& 0x12345678 ~0xFFFF]
puth $x

#-------------------------------------------------------------------------
# Bitwise OR: should print '0x1234ABCD'
# x = x | 0xABCD
set x [| $x 0xABCD]
puth $x

#-------------------------------------------------------------------------
# should print '49'
proc square {x} { * $x $x };
puts [square 7]

#-------------------------------------------------------------------------
# should print '16'
set x 4
puts [square $x]

#-------------------------------------------------------------------------
# should print '42'
# (4 * (5 + 7)) - 6 = 42
set a 5; set b 7
puts [- [* 4 [+ $a $b]] 6]

#-------------------------------------------------------------------------
# execute scripts defined in the cfg file
mon {run $script_A}
mon {run $script_B}

; just for demo
[Script_A]
echo "Script A"

; just for demo
[Script_B]
echo "Script B"
```

4.5.4 Sections

Section LICENSE

Listed in this section are all the license keys that are acquired, they will unlock specific features of PEEDI.

Example:

```
[LICENSE]
KEY = ARM7_ARM9, 1111-2222-3333-4
KEY = UPDATE_29AUG2006, 5555-6666-7777-8
```

The licenses can be also stored in a separate file:

```
[LICENSE]
FILE = tftp://192.168.0.1/licenses.txt
```

In this case the file “licenses.txt” should contains:

```
[LICENSE]
KEY = ARM7_ARM9, 1111-2222-3333-4
KEY = UPDATE_29AUG2006, 5555-6666-7777-8
```

Section DEBUGGER

This section describes the protocol used with the host debugger. One debugger protocol is supported: the GDB Remote debug protocol.

PROTOCOL

Synopsis:

```
PROTOCOL = gdb_remote
```

Description:

Describes the debugger protocol. If several protocols need to be enabled, they must be enumerated on the same line, separated by comma.

REMOTE_PORT

Synopsis:

```
REMOTE_PORT = <1024..65535>
```

Description:

TCP port to be used for accepting connections

FLASHn

Synopsis:

```
FLASH<CORE_INDEX> = <FLASH_SECTION>
```

Description:

This enables GDB load command to program code to FLASH

Example:

```
FLASH  = FLASH_NOR  - FLASH section to be used for core 0
FLASH0 = FLASH_NAND - FLASH section to be used for core 0
FLASH1 = FLASH_NAND - FLASH section to be used for core 1
```

Section TARGET

This section describes the target's platform.

PLATFORM

Synopsis:

```
PLATFORM = ARM|ARM11|AVR32|Blackfin|ColdFire|Cortex-A|
Cortex-M|Cortex-M_SWD|JBC_Player|MIPS|MPC5200|MPC5500|
MPC8300|MPC8500|QorIQ_P|XScale
```

Description:

Target's platform

Example:

```
[TARGET]
PLATFORM = ARM
```

Section PLATFORM_xxx - parameters for all targets with JTAG interface

In this section there are parameters specific to all targets with JTAG interface.

JTAG_CHAIN

Synopsis:

```
JTAG_CHAIN = <IR_LEN>
```

Description:

Length of IRs (Instruction Registers) of the devices on the JTAG chain. All IRs must be enumerated; the ones not supported by PEEDI must be skipped when defining COREn parameters (see below). If AUTO X is used first, then PEEDI will try to auto detect the actual number of TAPs connected in the JTAG chain.

Example:

```
JTAG_CHAIN = 4, 4, 6
JTAG_CHAIN = AUTO x 5,4
```

JTAG_TDO_DELAY

Synopsis:

```
JTAG_TDO_DELAY = 0..35 ; the delay in ns.

JTAG_TDO_DELAY = AUTO ; PEEDI tests the CPU and sets the
optimum TDO delay
```

Description:

Delay the sample of the TDO JTAG line. For best performance different CPUs require different TDO sample delay. When this parameter is not preset a 5ns value is set by default.

Example:

```
JTAG_TDO_DELAY = 10
```

TRST_TYPE

Synopsis:

```
TRST_TYPE = OPENDRAIN | PUSHPULL | NONE
```

Description:

Type of TRST output. NONE means no TRST pulse is generated.

RESET_TIME

Synopsis:

```
RESET_TIME = <milliseconds>
```

Description:

If 0 is specified, no reset will be issued, this way PEEDI can be attached to already initialized and running target, so INIT section could also be missing. If the target executes code after reset even CORE_STARTUP_MODE=RESET, this means the TAP is not active during reset.

Example:

```
RESET_TIME = 20
```

RESET_TYPE

Synopsis:

```
RESET_TYPE = ICEPICK-C | ICEPICK-D | U8500 | LS1000 |
ULTRASCALE | BF60x
```

Description:

Configure specific reset/connect procedure.

Example:

```
RESET_TYPE = ICEPICK-C, 3, 1 ; enable TAP3, warm reset
RESET_TYPE = ICEPICK-D, 9, 0 ; enable TAP9, no warm reset
RESET_TYPE = U8500
RESET_TYPE = LS1000
RESET_TYPE = ULTRASCALE
RESET_TYPE = BF60x
```

WAKEUP_TIME

Synopsis:

```
WAKEUP_TIME = <milliseconds>
```

Description:

Time to delay the JTAG operations after target power up is detected.

TIME_AFTER_RESET

Synopsis:

```
TIME_AFTER_RESET = <milliseconds>
```

Description:

Time to delay the JTAG operations after RESET is released.

DBGREQ_OUTPUT

Synopsis:

```
DBGREQ_OUTPUT = HIGH|LOW
```

Description:

Define the state of the JTAG DBGREQ line.

COREn

Synopsis:

COREn = CORE_TYPE, [<tap_num>]

Description:

Type of CORE and a TAP number separated by comma. Every core must be defined with the COREn parameter, where ’n’ is a number; each parameter related to this core must be preceded with the same COREn prefix.

COREn_STARTUP_MODE

Synopsis:

COREn_STARTUP_MODE = RESET | STOP, xx | RUN

Description:

PEEDI behavior when starting the target.

- RESET - Force the target to debug mode immediately out of reset. No code is executed after reset. (default mode)
- STOP, XX - After power-up PEEDI waits XX ms (this gives time to the target to execute its own initialization code) and target is placed in debug mode (halted).
- RUN - After reset, the target executes code until stopped by the Telnet halt command.

COREn_INIT

Synopsis:

COREn_INIT = <init_section>

Description:

Section to be executed in order to initialize the target.

COREn_FLASHm

Synopsis:

COREn_FLASHm = <flash_section>

Description:

This parameter points a section which contains the target FLASH description. If multiple FLASH chips/configurations are present on the target each chip/configurations must be described in different section, where ’m’ should start from 0 (max 15) and increment with each new section. If single FLASH chip/configuration is used the ’m’ integer number may be skipped. When working with the programmer the first FLASH is selected as current by default. To work on another FLASH, use the **flash set** command to select it. The multiple FLASH support, could also be used to describe

different profiles for the same FLASH, for example with different program method type or different image file specified. This way you can easy switch to the desired profile using the **flash set** command

COREn_ENDIAN

Synopsis:

```
COREn_ENDIAN = LITTLE|BIG
```

Description:

Define core endianness

COREn_BREAKMODE

Synopsis:

```
COREn_BREAKMODE = SOFT|HARD
```

COREn_BREAKMODE = HARD, start_addr, end_addr

Description:

Default breakpoint mode. Use to force the usage of hardware break points, when debugging in FLASH, or when working with GDB v5.3, where the hbreak command does not work. If 'start_addr' and 'end_addr' are given, then the hardware breakpoints are used for this address range.



Note:
The ARM EmbeddedICE logic has hardware resources for two break conditions, never mind break or watch points. The use of software breakpoints allows unlimited number of them, but this still requires the hardware resource of one break/watch point. Software breakpoints are possible only if the code is executed from RAM since the desired instruction to be hit is exchanged with special pattern. In brief, you can use up to two watchpoints or hardware breakpoints; or one watchpoint or hardware breakpoint, and unlimited number of software breakpoints. This means that you may use only one watch point and still debug normally in RAM. But if your code is in ROM/FLASH you must use hardware breakpoints, so if you have set one break or watch point you can still do 'single step', 'step in' and 'step out', but if you have set two watch or break points, only 'continue' is possible after the target breaks, since the debugger needs a temporary break point to achieve the 'step' functionality.

COREn_WORKSPACE

Synopsis:

```
COREn_WORKSPACE = <address>, <size>
```

Description:

Base and length in bytes of a region in RAM, used for agent, which allows much faster programming.

COREn_DATASPACE

Synopsis:

```
COREn_DATASPACE = <address>, <size>
```

Description:

If this parameter is present, PEEDI will use the workspace for storing only the agent code and the dataspace for the agent data. This is useful when using internal RAM for agent programming, where the internal RAM is code or data only, for example Blackfin CPUs.

COREn_PATH

Synopsis:

```
COREn_PATH = <path>
```

Description:

This parameter defines the default path to be used if only a file name (without the full path) is provided to a PEEDI command.

COREn_FILE

Synopsis:

```
COREn_FILE = FILE, [FORMAT], [ADDRESS]
```

Description:

This parameter defines the default memory (multi) load command's arguments. This parameter may have two or three arguments. The first argument is the file to be programmed. The second argument is the file type - BIN, SREC, IHEX or ELF. The third argument is mandatory for binary files and optional for all other types of files – it is the address where the file should be loaded.

COREn_LOCKOUT_RECOVERY

Synopsis:

```
Firmware before v20.12.xx:
COREn_LOCKOUT_RECOVERY = LM3S|KINETIS|NRF52
COREn_LOCKOUT_RECOVERY = YES|NO
COREn_LOCKOUT_RECOVERY = <value>

Firmware after v20.12.xx:
COREn_LOCKOUT_RECOVERY = LM3S|KINETIS|NRF52
COREn_LOCKOUT_RECOVERY = MAC7100_4MHz|MAC7100_8MHz
COREn_LOCKOUT_RECOVERY = YES|NO
```

Description:

If this parameter is present, PEEDI automatically executes a 'JTAG Lockout Recovery' procedure during reset processing if the MAC7100, STR9, LM3S, KINETIS or AVR32 flash is secured.

LM3S/KINETIS for Cortex-M devices

YES/NO for STR9 and AVR32 devices

For MAC7100 devices 7-bit value for the CFMCLKD register used during the 'JTAG Lockout Recovery'. Calculate this parameter based on the reset system clock (PLL disabled).

For example (Firmware before v20.12.xx):

19 - CLKD for 8MHz system clock

9 - CLKD for 4MHz system clock

For example (Firmware after v20.12.xx):

COREn_LOCKOUT_RECOVERY = MAC7100_4MHz|MAC7100_8MHz

COREn_OS

Synopsis:

```
COREn_OS = <section>
```

Description:

This parameter points to a section which contains parameters that defines the target Operating System. This guides PEEDI to scan the target OS tasks and pass the list to the host debugger.

Section PLATFORM_ARM and PLATFORM_ARM11

COREn

Synopsis:

```
COREn = ARM7TDMI|ARM9TDMI|ARM920T|ARM940T|ARM926E|ARM946E,
[<tap_num>]

COREn = ARM1136|ARM1156|ARM1176, [tap_num]
```

Description:

Type of CORE and a TAP number separated by comma

COREn_VECTOR_CATCH_MASK

Synopsis:

```
COREn_VECTOR_CATCH_MASK = <mask>
```

Description:

Specifies which interrupts/exceptions to be trapped i.e. break if an interrupt or

exception occurs. ARM9 and XSCALE cores only. Catch vector mask bit meaning:

7	6	5	4	3	2	1	0
FIQ	IRQ	Res	D_Abort	P_Abort	SWI	Undef	Reset

COREn_DCC_PORT

Synopsis:

```
COREn_DCC_PORT = 1024..65535, [0-7|32]
```

Description:

TCP port, the target's DCC channel to be routed to.

Example:

```
COREn_DCC_PORT = 2001 - route 8-bit DCC to TCP port 2001
COREn_DCC_PORT = 2001, 32 - route 32-bit DCC to TCP port 2001
COREn_DCC_PORT = 2001, 4 - route 4 virtual serial ports to TCP
ports 2001-2004
```

COREn_USE_FAST_DOWNLOAD

Synopsis:

```
COREn_USE_FAST_DOWNLOAD = YES|NO
```

Description:

Only by PLATFORM_ARM11.
TCP port, the target's DCC channel to be routed to. In this case the lowest eight bits of the 32 bit DCC word will be transfered to/from single TCP port, forming 8-bit/character channel. If a virtual serial port number (up to 8) is provided after the TCP port, PEEDI will emulate up to 8 virtual serial ports routed to 8 consecutive TCP ports, starting from the given one.

COREn_BREAK_PATTERN

Synopsis:

```
COREn_BREAK_PATTERN = <value>
```

Description:

Software breakpoint pattern.

Since Firmware v20.12.xx this parameter is obsolete.

Section PLATFORM_Cortex & Section PLATFORM_Cortex_SWD

These sections describe the Cortex-A and Cortex-M cores connected to PEEDI via JTAG or SWD (Serial Wire Debug).It has all the parameters described in the PLATFORM_ARM section (except the COREn_VECTOR_CATCH_MASK, and COREn_DCC_PORT. The PLATFORM_Cortex_SWD section has no JTAG_CHAIN parameter, and its clock parameter is named SWD_CLOCK and has the same format as the JTAG_CLOCK parameter. About the CORE parameter:

COREn

Synopsis:

```
COREn = Cortex-M|Cortex-A|Cortex-ARMv8 [tap_num] [tap_id]
COREn = Cortex-A_SMP
COREn = Cortex-A_AMP
```

Description:

The detection of Cortex-A and Cortex-M variants is done automatically. value. Suffix '_SMP' defines cores in a SMP group. Suffix '_AMP' defines cores in a AMP group. All cores in a SMP group start, halt, break and single step simultaneously. All cores in a AMP group can be set to start, halt, break and single step simultaneously or not. This can be set with the 'amp' command.

Example:

```
CORE0 = Cortex-A, 0
CORE0 = Cortex-ARMv8, 0
CORE0 = Cortex-M, 1, 0xBC11477
```

COREn_APSEL

Synopsis:

```
COREn_APSEL = 0 .. 255
```

Description:

Define Access Port

Example:

```
CORE0_APSEL = 1
```

COREn_DEBUG_ADDR

Synopsis:

```
COREn_DEBUG_ADDR = DGB_ADDR [, CROSS_TRIG_ADDR]
```

Description:

Set the address of CoreSight Debug and CrossTrigger components.

Example:

```
COREn_DEBUG_ADDR = 0x80070000, 0x80078000
COREn_DEBUG_ADDR = 0x80070000
```

COREn_DAPPC

Synopsis:

```
COREn_DAPPC = <address>
```

Description:

This parameter is necessary for some TI processors (OMAP3, OMAP4, ...). It defines the address of a special debug control register.

Example:

```
COREn_DAPPC = 0xD401D030
```

COREn_PERIODIC_TASK

Synopsis:

```
COREn_PERIODIC_TASK = <script_name>,
```

```
<time_in_milliseconds>
```

Description:

Execute the given script on specified time interval.

If COREn is omitted, then CORE0 is used.

Currently this parameter is supported only in PLATFORM_CORTEX.

Ask Ronetix for supporting another platforms.

COREn_SWO

Synopsis:

```
COREn_SWO = <stim_chan>, <tcp_port>
COREn_SWO = DWT, <tcp_port>
```

Description:

This parameter is allowed only for the PLATFORM_Cortex-M_SWD section. It tells PEEDI to open a TCP port and listen for incoming telnet connections. PEEDI checks for new incoming telnet connection only when the target CPU is halted. If a telnet session is opened to that TCP port PEEDI will forward all stimulus data for the given stimulus channel. In order for the CPU to transmit stimulus messages, you need to enable this functionality. This can be done by the target application or by PEEDI using the target INIT script - see ST STM32 family .

If DWT is specified instead of stimulus port, PEEDI will forward all enabled DWT messages to the TCP port - PC samples, interrupt entry/exit, timestamps, etc.

COREn_PROFILING

Synopsis:

```
COREn_PROFILING = <start_addr>, <length>, <virtual_addr>
```

Description:

This parameter is allowed only for the PLATFORM_Cortex-M_SWD section.

It tells PEEDI to maintain PC counter array virtually mapped at the target's memory space. When PEEDI receive a PC sample message it will increment the corresponding

PC hit counter. After the target is halted, one may use memory read16 <virtual_address> command to see the counters and thus see where CPU spent too much time. This feature can be use by debuggers too.

In order for the CPU to transmit PC sample messages, you need to enable this functionality. This can be done by the target application or by PEEDI using the target INIT script - see ST STM32 family .

Section PLATFORM_XSCALE

This section describes the XScale cores connected to PEEDI.

COREn

Synopsis:

```
COREn = XScale|PXA320, [tap_num]
```

Description:

Type of CORE and a TAP number separated by comma

COREn_USE_FAST_DOWNLOAD

Synopsis:

```
COREn_USE_FAST_DOWNLOAD = YES|NO
```

Description:

If YES is specified, PEEDI will send data to the target without checking if the target is ready with the previous data, assuming that the target writes the received data faster than PEEDI is sending it. This type of transfer is faster but less reliable. Use it only if you are sure that the target is fast enough i.e. the CPU is running on high frequency.

COREn_DEBUG_HANDLER_ADDR

Synopsis:

```
COREn_DEBUG_HANDLER_ADDR = <addr>
```

Description:

The address where the XScale debug handler should be mapped at.

Choosing address has three limitations:

- Due to the limitation of the ARM branch instruction the address must be within these ranges: 0x00000000 - 0x01FFFC00 or 0xFE000000 - 0xFFFFFC00.
- Must be aligned to a 1KB (0x400) boundary.
- Must not overlap user application code.

COREn_VECTOR/RELOCATED_UNDEF/SWI/PABORT/DABORT/RES/IRQ/FIQ

Synopsis:

```
COREn_VECTOR_UNDEF = AUTO|<instr_code>
COREn_VECTOR_SWI = AUTO|<instr_code>
COREn_VECTOR_PABORT = AUTO|<instr_code>
COREn_VECTOR_DABORT = AUTO|<instr_code>
COREn_VECTOR_RES = AUTO|<instr_code>
COREn_VECTOR_IRQ = AUTO|<instr_code>
COREn_VECTOR_FIQ = AUTO|<instr_code>
COREn_RELOCATED_UNDEF = AUTO|<instr_code>
COREn_RELOCATED_SWI = AUTO|<instr_code>
COREn_RELOCATED_PABORT = AUTO|<instr_code>
COREn_RELOCATED_DABORT = AUTO|<instr_code>
COREn_RELOCATED_RES = AUTO|<instr_code>
COREn_RELOCATED_IRQ = AUTO|<instr_code>
COREn_RELOCATED_FIQ = AUTO|<instr_code>
```

Description:

Because of the XScale debugging specifics, PEEDI must be aware of the exception vectors. Each of these parameters may have value of AUTO or an exact value which represents a hex encoded ARM instruction. In case of AUTO is specified, PEEDI will read the original vector value from the target memory on each debug event (halt, step,

go, etc.). Or you can put a constant value if you exactly know the vector's instruction, for example 0xE59FF018 stands for "ldr pc, [pc, #18]" instruction.

Section PLATFORM_MPC5200

This section describes the MPC5200 cores connected to PEEDI.

COREn

Synopsis:

```
COREn = MPC5200|MPC8200, [tap_num]
```

Description:

Type of CORE and a TAP number separated by comma

COREn_BOOT_ADDR

Synopsis:

```
COREn_BOOT_ADDR = 0x00000100|0xFFF00100
```

Description:

Normally the boot address for PowerPC is 0xFFF00100 or 0x00000100 depending on the Reset Configuration Word (RCW). PEEDI sets a hardware breakpoint at this address to halt the core immediately out of reset.

COREn_MEMDELAY

Synopsis:

```
COREn_MEMDELAY = <NUMBER_OF_CLOCKS>
```

Description:

Additional number of CPU clocks for a memory access.

Section PLATFORM_MPC5500

This section describes the MPC55XX cores connected to PEEDI.

COREn

Synopsis:

```
COREn = MPC5xxx|MPC5xxx_VLE|MPC5xxx_SPE, [tap_num]
```

Description:

Type of CORE and a TAP number separated by comma

COREn_NEXUS3_ACCESS

Synopsis:

```
COREn_NEXUS3_ACCESS = START_ADDRESS, LENGTH
```

Description:

This parameter accepts NO or memory region (start address and length in bytes). If a memory region is supplied (usually this is the RAM of the target), PEEDI will access target memory region using the nexus3 module. This method is about three times faster but it uses physical addresses i.e. bypasses the MMU. You can properly use this method if the MMU is set to be transparent i.e. virtual addresses are equal to physical ones.

MPC5XXX_AUX_TAP_CMD, COREn_AUX_TAP_CMD

Synopsis:

```
MPC5XXX_AUX_TAP_CMD = <TAP_IR_LEN>,<TAP_CMD>
```

```
COREn_AUX_TAP_CMD = <TAP_IR_LEN>,<TAP_CMD>
```

Description:

Set core aux tap select command, if different from the default 0x11 with IR length of 5.

Section PLATFORM_MPC8300

This section describes the MPC83XX cores connected to PEEDI.

COREn

Synopsis:

```
COREn = MPC5121|MPC8306|MPC8308|MPC8313|MPC8315|MPC8321|
MPC8323|MPC8343|MPC8349|MPC8360|MPC8378, [tap_num]
```

Description:

Type of CORE and a TAP number separated by comma

COREn_BOOT_ADDR

Synopsis:

```
COREn_BOOT_ADDR = 0x00000100|0xFFF00100
```

Description:

Normally the boot address for PowerPC is 0xFFF00100 or 0x00000100 depending on the Reset Configuration Word (RCW). PEEDI sets a hardware breakpoint at this address to halt the core immediately out of reset.

COREn_RCW

Synopsis:

```
COREn_RCW = <rcw_high>, <rcw_low>
```

Description:

When this parameter is present, PEEDI overrides the Reset Configuration Words with the values provided.

COREn_MMU_PTBASE

Synopsis:

```
COREn_MMU_PTBASE = <addr>
```

Description:

Address of the of pointer to the two page pointers array This parameter defines the physical memory address, where PEEDI looks for the virtual address of the array with the two page table pointers. If this configuration parameter is present and the MMU translation is enabled, if PEEDI fails to translate the effective address to a physical one using BAT translation, it tries a page translation. For more information see CPU specific considerations.

Section PLATFORM_MPC8500

This section describes the MPC8500 cores connected to PEEDI.

COREn

Synopsis:

```
COREn = MPC8536|MPC8540|MPC8572A/B|P1010|P1011|P1020A/B|
P2020A/B, <tap_num>
```

Description:

Type of CORE and a TAP number separated by comma

COREn_MMU_TRANS

Synopsis:

```
COREn_MMU_TRANS = <addr>
```

Description:

This parameter sets the default MMU translation address. For example the default Linux kernel address is 0xC0000000.

COREn_MMU_PTBASE

Synopsis:

```
COREn_MMU_PTBASE = <addr>
```

Description:

Address of the of pointer to the two page pointers array This parameter defines the physical memory address, where PEEDI looks for the virtual address of the array with the two page table pointers. If this configuration parameter is present and the MMU translation is enabled, if PEEDI fails to translate the effective address to a physical one using BAT translation, it tries a page translation. For more information see CPU specific considerations.

Section PLATFORM_QorIQ_P

This section describes the QorIQ P3/4/5, T1/T2/4 cores connected to PEEDI.

COREn

Synopsis:

```
COREn = P4080A/B/C/D/E/F/G/H|T1040A/B/C/D|T2080/T4080,
<tap_num>
```

Description:

Type of CORE and a TAP number separated by comma

COREn_REGLIST

Synopsis:

```
COREn_REGLIST = 32BIT|64BIT
```

Description:

This parameter sets the type of the register frame sent to GDB, when debugging 64-bit e5500/e6500 cores - 32 or 64 bit registers.

COREn_MMU_TRANS

Synopsis:

```
COREn_MMU_TRANS = <addr>
```

Description:

This parameter sets the default MMU translation address. For example the default Linux kernel address is 0xC0000000.

COREn_MMU_PTBASE

Synopsis:

```
COREn_MMU_PTBASE = <addr>
```

Description:

Address of the of pointer to the two page pointers array This parameter defines the physical memory address, where PEEDI looks for the virtual address of the array with the two page table pointers. If this configuration parameter is present and the MMU translation is enabled, if PEEDI fails to translate the effective address to a physical one using BAT translation, it tries a page translation. For more information see CPU specific considerations.

COREn_PMEM_BASE

Synopsis:

```
COREn_PMEM_BASE = <addr>
```

Description:

Address of the of pointer to the two page pointers array This parameter defines the physical memory address, where PEEDI looks for the virtual address of the array with the two page table pointers. If this configuration parameter is present and the MMU translation is enabled, if PEEDI fails to translate the effective address to a physical one using BAT translation, it tries a page translation. For more information see CPU specific considerations.

Section PLATFORM_PPC400

This section describes the PPC400 cores connected to PEEDI.

COREn

Synopsis:

```
COREn = PPC405|PPC440|PPC464, <tap_num>
```

Description:

Type of CORE and a TAP number separated by comma

Section PLATFORM_COLDFIRE

This section describes the ColdFire cores connected to PEEDI.

BDM_CLOCK

Synopsis:

```
BDM_CLOCK = <INIT>, <NORMAL>
```

```
BDM_CLOCK = ADAPTIVE_n
```

Description:

BDM clock before and after initialization. MAX BDM clock is 33MHz. See your ColdFire CPU user's manual for correct BDM clock. Use ADAPTIVE_n to set the BDM clock to PSTCLK / n.

CORE

Synopsis:

CORE = MCFxxxx

Description:

Type of CORE - MCF5206, MCF5207, MCF5208, MCF5211, MCF5212, MCF5213, MCF5214, MCF5216, MCF521x0, MCF5221x, MCF5222x, MCF5223x, MCF5225x, MCF5227x, MCF523x, MCF5249, MCF525x, MCF5270, MCF5271, MCF5272, MCF5274, MCF5275, MCF528x, MCF5307, MCF532x, MCF537x, MCF5407,

MCF5445x, MCF547x, MCF548x

CORE_MEMMAP

Synopsis:

```
CORE_MEMMAP = <start_addr>, <end_addr>
```

Description:

Defines a valid memory region. Up to 32 regions can be defined in the target configuration file. When even one region is defined, PEEDI begins to check every memory access operation if it falls into a defined memory region. If the memory operation is out of the defined regions, PEEDI interrupts the operation and issues an error. This is made so, because when an access is made to an invalid memory address via the BDM, the ColdFire CPU refuses to respond to any further memory operations until reset.

Section PLATFORM_BLACKFIN

This section describes the Blackfin cores connected to PEEDI.

COREn

Synopsis:

```
COREn = BFXXX, [tap_num]
```

Description:

Type of CORE (BF50X, BF51X, BF522, BF525, BF527, BF531, BF532, BF533, BF534, BF535, BF536, BF537, BF538, BF539, BF542, BF544, BF548, BF549, BF561A, BF561B, BF59X, BF60X_A, BF60X_B, BF70X) and a TAP number separated by comma

The following parameters are not mandatory. They are used to define a ’virtual’ memory region corresponding to an external memory mapped device that is bigger than the visible external asynchronous memory space. The higher address lines of the device that are not connected to the CPU address buss must be driven by the GPIO pins. This way you can use all the PEEDI CLI commands (flash program , memory read , etc.) on the defined virtual region as the whole device is directly visible in the memory space of the target. Actually PEEDI emulates this behavior by

accessing physically the device only through the memory window provided by the CPU external address space and driving the higher address lines of the device depending on the address that is requested to be accessed.

This feature of PEEDI helps programming FLASH chips which are bigger than the visible external asynchronous memory space.

COREn_VMEM

Synopsis:

```
COREn_VMEM = <address>, <length>
```

Description:

Defines a memory region, which is virtually mapped to large external memory mapped device.

COREn_VMEM_WINDOW

Synopsis:

```
COREn_VMEM_WINDOW = <address>, <length>
```

Description:

Defines a memory window to physically access the external memory mapped device, t.e. the memory region where the device is mapped into the CPU memory space.

COREn_VMEM_PINS

Synopsis:

```
COREn_VMEM_PINS = P<A..J><0..15>
```

Description:

Pxy, where ’x’ is the GPIO port A..J and ’y’ is the bit number 0..15. List of the GPIO pins connected to the higher external device address lines that are not connected to the CPU address bus, starting from lowest to highest, separated by comma. The GPIO pins must belong to the same GPIO port.

Imagining that we want to virtually map on address 0x30000000, an 8MB FLASH that is connected to the first chip select of the CPU, so physically accessible at 0x20000000 via 1MB window and its A19, A20 and A21 pins are connected to PF4, PF5 and PF8 CPU pins, the configuration should look like this:

```
; The 8MB FLASH is virtually mapped at 0x30000000
CORE0_VMEM = 0x30000000, 0x800000

; It is physically visible through a 1MB window at 0x20000000
CORE0_VMEM_WINDOW = 0x20000000, 0x100000

; PF4, PF5 and PF8 are used to drive A19, A20 and A21 of the FLASH
CORE0_VMEM_ADDRESS_PINS = PF4, PF5, PF8
```

Now we can erase, program and verify the whole 8MB of FLASH at address 0x30000000 using any PEEDI **flash** command. Keep in mind that when defining [FLASH] section in the target configuration file, you need to specify the virtual address of the FLASH for the BASE_ADDR parameter.

CORE_MEMMAP

Synopsis:

```
CORE_MEMMAP = <start_addr>, <end_addr>
```

Description:

Defines a valid memory region. Up to 32 regions can be defined in the target configuration file. When even one region is defined, PEEDI begins to check every memory access operation if it falls into a defined memory region. If the memory operation is out of the defined regions, PEEDI interrupts the operation and issues an error. This is made so, because when an access is made to an invalid memory address via the JTAG, the Blackfin CPU may stop to respond to any further memory operations until reset.

Section PLATFORM_MIPS

This section describes the MIPS cores connected to PEEDI.

COREn

Synopsis:

```
COREn = MIPS32_24K|MIPS32_4K|MIPS32_M4K|PIC32|RTL8100|
```

```
MIPS64, [tap_num]
```

Description:

Type of CORE and a TAP number separated by comma.

Section PLATFORM_AVR32

This section describes the AVR32 cores connected to PEEDI.

COREn

Synopsis:

```
COREn = AVR32AP7|AVR32UC3, [tap_num]
```

Description:

Type of CORE and a TAP number separated by comma.

COREn_BLOCK_ACCESS

Synopsis:

```
COREn_BLOCK_ACCESS = START_ADDRESS, LENGTH
```

Description:

This parameter accepts NO or memory region (start address and length in bytes). If a memory region is supplied (usually this is the RAM of the target), PEEDI will access target memory region using the MEMORY_WORD_ACCESS TAP command.

Section INIT

This is the section specified by COREn_INIT parameter. It includes commands, which are executed once after every target power detection and target reset. The purpose of this section is to initialize the target (map the memory, init peripherals and so on). Most of these are **memory write** commands.

Example:

```
[INIT_EB55800]
memory write 0xFFFF4020           ; enable main clock
wait 100                          ; wait to stabilize
memory write 0xFFFF4020           ; switch to main clock
memory write 0xFFFF4020           ; enable PLL
wait 100                          ; wait to lock
memory write 0xFFFF4020           ; switch to PLL, pres=4, mul=8

memory write 0xFFE00020 0x00000001 ; cancel reset remapping
memory write 0xFFE00000 0x010020A5 ; csr0 - Flash at 0x1000000, 2 Ws
memory write 0xFFE00004 0x02003029 ; csr1 - RAM at 0x2000000, 2 Ws
```

Sometimes it is impossible to initialize the target only by using the commands in the [INIT] section of the target configuration file. In cases like this to perform the initialization an executable image can be loaded and executed in the target using the memory load and go commands. Before loading the image, the RAM where it will be loaded must be initialized. Follow these steps to make a successful initialization:



Note:
This is working [INIT] section for AT91M55800A CPU. In this case the last instruction of the executable must be SWI, informing that job has finished.

```
[INIT_EB55800]
; First init chip selects
memory write 0xFFE00020 0x00000001 ; cancel reset remapping
memory write 0xFFE00000 0x010020A5 ; csr0 - Flash at 0x1000000, 2 Ws
memory write 0xFFE00004 0x02003029 ; csr1 - RAM at 0x2000000, 2 Ws
memory write 0xFFFFF124 0xFFFFFFFF ; disable all interrupts

; Then load and start the executable image,
; skipping the interrupt table
memory load tftp://192.168.1.1/init.bin bin 0x20
set cpsr 0xD3          ; set supervisor mode, interrupts disabled
set sp 0x200           ; set stack pointer, if program uses stack
breakpoint add 0x8     ; set break at software interruptvector address
go                     ; start executable
wait 50                ; wait to complete
halt                   ; halt if not completed
break del 1            ; del break at software interrupt vector address
```

Section FLASH

This section tells PEEDI what type are the onboard FLASH memory chips and what their configuration is.

NOR FLASH programming

These are all possible variants of connecting NOR FLASH chips:

- One 8-bit chip, forming 8-bit architecture
- Two 8-bit chips, forming 16-bit architecture
- Four 8-bit chips, forming 32-bit architecture
- One 16-bit chip, forming 16-bit architecture
- Two 16-bit chips, forming 32-bit architecture
- One 32-bit chip, forming 32-bit architecture

When describing external NOR FLASH configuration the following parameters have to be specified:

CHIP
CHECK_ID
ACCESS_METHOD
CHIP_WIDTH
CHIP_COUNT
BASE_ADDR
FILE
AUTO_ERASE
AUTO_LOCK

Considering your configuration you must specify CHIP_COUNT, and CHIP_WIDTH parameters, CHIP_WIDTH is the width of a single chip, so system width will be CHIP_COUNT multiplied by CHIP_WIDTH.

Example:

http://download.ronetix.at/peedi/cfg_examples/arm9/mv78100.cfg

I2C Programming

PEEDI supports I2C EEPROM programming, for any CPU that SDA and SCL signals are connected to GPIOs and can be driven using memory operations. Along with the standard flash

commands, you can use **flash this write** command to write up to fourteen bytes to the EEPROM like this:

```
flash this write 0x24 0x36 0x48 - write two bytes at address 0x24
```

The FLASH section for I2C EEPROM programming should include the following parameters:

CHIP
CHIP_SIZE
I2C_ADDR
I2C_DELAY
SDA_SET/CLR, SDA_IN/OUT, SDA_READ, SCL_SET/CLR
FILE
AUTO_ERASE

Example:

http://download.ronetix.at/peedi/cfg_examples/arm11/s3c6410.cfg

SPI FLASH programming

The parameters for SPI NOR FLASH or Atmel DataFlash family, connected to an Atmel AT91 CPU are:

CHIP
SPI_DIV
nSPI
SPI_SPCK, SPI_MISO, SPI_MOSI, SPI_CS
SPI_MODE
FILE
AUTO_ERASE

Example:
http://download.ronetix.at/peedi/cfg_examples/arm9/at91sam9263_pm9263.cfg

Example for SPI memory connected to a Blackfin CPU:

http://download.ronetix.at/peedi/cfg_examples/blackfin/bf532.cfg

Example for SPI memory connected to a NXP LPC2000 CPU:

http://download.ronetix.at/peedi/cfg_examples/arm7/lpc2468.cfg

Example for SPI memory connected to a NXP LPC4000 CPU:

http://download.ronetix.at/peedi/cfg_examples/cortex-m/lpc4300.cfg

PEEDI also supports software emulated SPI interface FLASH programming. In this case the FLASH is connected to CPU GPIOs and PEEDI drives them to emulate SPI interface. Here are the needed configuration parameters:

CHIP
CPU
CS_ASSERT/RELEASE, SCLK_SET/CLR, MOSI_SET/CLR, MISO_READ
FILE
AUTO_ERASE

Example:

http://download.ronetix.at/peedi/cfg_examples/arm9/at91sam9263_soft_spi.cfg

NAND FLASH programming

PEEDI is able to program all NAND chips with 8 and 16 bits data bus.

The INIT section of the config file must include the initialization for the chip select and the GPIOs, because the Flash Programmer doesn't make any initialization. The NAND Flash devices may have blocks that are invalid when they are shipped. An invalid block is one that contains one or more bad bits. Additional bad blocks may develop with use. The factory identifies invalid blocks before shipping by programming data other than FFh (x8) or FFFFh (x16) into the first spare location of the first or second page of each bad block. PEEDI automatically detects the bad blocks and reports them using the **flash info** and **flash query** commands.

Once detected, the bad blocks are protected against erasing and programming.

On demand, PEEDI can be forced to try to erase the existing bad blocks.

It is also possible to force blocks as bad.

To erase all blocks including the bad blocks, set the ERASE_BAD_BLOCKS parameter to YES. After PEEDI restart, the command **flash erase** will erase all blocks.



WARNING:
If you erase blocks factory marked as bad, there is now way to detect which were the bad blocks.

Make sure you have saved the output of the flash query command so you can mark again the bad blocks as bad.

To force marking of blocks 4, 27 and 1002 as bad set BAD_BLOCKS parameter like this:

```
BAD_BLOCKS = 4, 27, 1002
```

After PEEDI restart, the flash info command will mark the given blocks as bad. Once marked as bad, the blocks a not marked anymore.

PEEDI supports direct programming of JFFS2 images to the NAND flash. For this, the OOB_INFO parameter must be set to 'JFFS2'. This way PEEDI will the write the data loading from the image file and will calculate the ECC and program it to the OBB/spare bytes. PEEDI supports only BIN images starting from address 0. When programming the image bad blocks will be just skipped and left un-programmed. They will not affect the block count order.

If you use a custom file system, using PEEDI you can program a bootloader to the NAND chip, that will gain the control of the system after it is rebooted and could handle the programming of the left empty NAND FLASH chip, considering the NAND file system you use and the bad block present in the given target.



WARNING:
The PEEDI TFTP client uses 512 bytes or 2048 bytes (if supported by the TFTP server) transfer block size, which limits the size of the image file to 32MB or 128MB. If your file is bigger, use HTTP/FTP file server or use MMC/SD card to store the file and put it on PEEDI.

Examples:

http://download.ronetix.at/peedi/cfg_examples/arm9/at91sam9263_pm9263.cfg

http://download.ronetix.at/peedi/cfg_examples/cortex-a/atsama5d3.cfg

http://download.ronetix.at/peedi/cfg_examples/arm9/tms320dm365-DM365EVM.cfg

http://download.ronetix.at/peedi/cfg_examples/arm11/mx31.cfg

http://download.ronetix.at/peedi/cfg_examples/powerpc/mpc5121_aria.cfg

http://download.ronetix.at/peedi/cfg_examples/powerpc/mpc5125.cfg

http://download.ronetix.at/peedi/cfg_examples/powerpc/mpc8313.cfg

http://download.ronetix.at/peedi/cfg_examples/blackfin/bf527.cfg

OneNAND FLASH programming

Example:

http://download.ronetix.at/peedi/cfg_examples/xscale/pxa270_onenand.cfg

MMC/SD card programming

The parameters for MMC/SD card are:

CHIP
CPU
PARTITION
FILE

All flash commands on MMC/SD card takes address and length (if application) parameter in blocks, not in bytes.

Example:
http://download.ronetix.at/peedi/cfg_examples/arm11/mx35_eMMC.cfg

Atmel SAM3/SAM4 programming

Example:

http://download.ronetix.at/peedi/cfg_examples/cortex-m/atsam.cfg

Atmel AVR32UC3 programming

Example:

http://download.ronetix.at/peedi/cfg_examples/avr32/avr32uc.cfg

Freescale Kinetis programming

Example:

http://download.ronetix.at/peedi/cfg_examples/cortex-m/kinetis.cfg

TI/Luminary LM3S programming

Example:

http://download.ronetix.at/peedi/cfg_examples/cortex-m/lm3s8962.cfg

NXP LPC2000 programming

To successfully program a LPC2000 device make sure you have specified valid RAM address for the CORE_WORKSPACE parameter in the PLATFORM_ARM section. The internal RAM starts from 0x40000000, so this is a good value for this parameter.

To successfully verify the FLASH contents, first you must set the MEMMAP register to map the flash vectors at address 0x00000000 like this:

```
memory write 0xE01FC040 0x00000001
```

You may issue the previous command every time you need to verify or you may put it in the init section of the core in the target configuration file, this way it will be executed automatically.

To secure the LPC2000 device, your application must set FLASH address location 0x1FC (User flash sector 0) with value 0x87654321 (2271560481 Decimal) when programmed. This will disable the JTAG port and some of the ISP commands on the next reset.

The only way to un-secure the device is to use ISP command to erase the FLASH. This can be made with the Philips LPC2000 FLASH utility.

For more information about the LPC2000 securing (code protection) read the LPC2000 user's manual.

Example:

http://download.ronetix.at/peedi/cfg_examples/arm7/lpc2138.cfg

LPC CPU's may return an incorrect CPU ID as shown in the example below where the LPC1343 CPU is used:

```
lpc> flash info
- error: unknown part ID: 0x3000002b
++ info: some CPUs (like LPC1343) return wrong ID.
In this case put in the configuration file:
'PART_ID = X', where X is the correct device ID from User Manual
- error: unable to start flash programmer
lpc>
```

In such case use the PART_ID parameter as in the following example:

```
PART_ID = 0x3D00002B; Correct LPC1343 CPU ID
```

LPC CPU's consist of six main groups:

- LPC800
- LPC1100
- LPC1700
- LPC2000
- LPC4300
- LPC54100

For example to enable flash programming for LPC1343 do:

```
CHIP = LPC1100
```

Supported devices for CHIP = LPC800:

```
LPC810,LPC811,LPC812,LPC822,LPC824
```

Supported devices for CHIP = LPC1100:

```
LPC1110,LPC1111,LPC11A11,LPC11E11,LPC1311,LPC1112,LPC11A02,
LPC11C12,LPC11C22,LPC11A12,LPC11E12,LPC11U12,LPC11U12,LPC1342,
LPC1113,LPC11A13,LPC11E13,LPC11U13,LPC11U13,LPC11U23,LPC1114,
LPC11A04,LPC11A14,LPC11A14,LPC11C14,LPC11C24,LPC11E14,LPC11U14,
LPC11U14,LPC11U24,LPC1313,LPC1315,LPC1343,LPC1345,LPC11U34,LPC1316,
LPC1346,LPC1115,LPC11U35,LPC1317,LPC1347,LPC11E36,LPC11U36,LPC11E37,
LPC11E37,LPC11U37,LPC11U37H,LPC11U37
```

Supported devices for CHIP = LPC1700:

```
LPC1751,LPC1752,LPC1754,LPC1764,LPC1774,LPC1756,LPC1763,LPC1765,LPC1766,
LPC1776,LPC1785,LPC1786,LPC1758,LPC1759,LPC1767,LPC1768,LPC1769,LPC1777,
LPC1778,LPC1787,LPC1788
```

Supported devices for CHIP = LPC2000:

```
LPC2103,LPC2104,LPC2105,LPC2106,LPC2114,LPC2119,LPC2124,LPC2214,LPC2129,
LPC2194,LPC2292,LPC2294,LPC2131,LPC2141,LPC2141,LPC2142,LPC2134,LPC2144,
LPC2136,LPC2146,LPC2366,LPC2138,LPC2148,LPC2368,LPC2387,
LPC2458,LPC2468,LPC2478
```

Supported devices for CHIP = LPC54100:

```
LPC54101,LPC54102
```

Any flash chips not listed above are specified by there chipname.

Supported devices for CHIP = LPC2900 family:

This family of LPC CPU's is a different group because its flash algorithms for programming vary from the other groups.

Example:
http://download.ronetix.at/peedi/cfg_examples/arm9/lpc2917.cfg

Nordic Semiconductor nRF51 ans nRF52 programming

nRF51xxx and nRF52xxx are supported and automatically detected. User Information Configuration Registers (0 - 255) are mapped as an additional bank and can be programmed with "**flash program**" and erased with "**flash erase**". UICR registers can be also programmed single using:

```
flash this uicr ADDR VAL ; program a UICR register
```

Example (nRF52):

```
flash this uicr 0x208 0xFFFFFF00 ; enable access port protection (enable
flash security)
```

If the Flash security is enabled, the only way to unlock the device is to perform JTAG Lockout Recovery procedure.

PEEDI executes a 'JTAG Lockout Recovery' during reset processing if the nRF5 Flash is secured and if the configuration file contains:

```
CORE_LOCKOUT_RECOVERY = NRF5
```

Example:

http://download.ronetix.at/peedi/cfg_examples/cortex-m/nrf5.cfg

Freescale MAC7100 programming

In program FLASH the address range from 0x0400 to 0x041F is used to set the FLASH security. If there is user code this could secure the FLASH incidentally, so avoid placing code there.

```
flash erase chip ; perform MASS ERASE
flash lock ; write at address 0xFC100414 = 0xFFFFFFFC0 (enable flash
security)
```

If the Flash security is enabled, the only way to unlock the device is to perform JTAG Lockout Recovery procedure.

PEEDI executes a 'JTAG Lockout Recovery' during reset processing if the MAC7100 flash is secured and if the configuration file contains:

```
CORE_LOCKOUT_RECOVERY = MAC7100_4MHZ
CORE_LOCKOUT_RECOVERY = MAC7100_8MHZ
```

Example:

http://download.ronetix.at/peedi/cfg_examples/arm7/mac7100.cfg

Freescale ColdFire V2 programming



WARNING:
Set very carefully the CPU_CLOCK parameter otherwise the FLASH may be damaged.

Example:
http://download.ronetix.at/peedi/cfg_examples/coldfire/mcf5282.cfg

Freescale MPC5000 programming

Example:

http://download.ronetix.at/peedi/cfg_examples/powerpc/mpc5554.cfg

Renesas RA Cortex-M33

Supported devices: RA4M2, RA4M3, RA6M4 and RA6M5.

The internal Code Flash and Data Flash programming is supported. The Data Flash support is enabled with a configuration parameter.

```
DATA_FLASH = YES
```

Example:

http://download.ronetix.at/peedi/cfg_examples/cortex-m/ra4m3.cfg

ST STM32 programming

In the STM32 microcontrollers, the FLASH may be write-protected. The protection is set using the STM32 option bytes. Option bytes are used to configure also other STM32 CPU settings - for more information see the STM32F10xxx Flash programming.

For managing STM32 option bytes, PEEDI has the **flash this option** command. To erase all option

bytes use **flash this option erase**. To write a single option byte, use **flash this option BYTE VALUE** . An option byte can be written only once after it is erased. If you want to change the value of previously written byte you must erase it - this will erase all other option values, so you may need to set them again.

Example:

http://download.ronetix.at/peedi/cfg_examples/cortex-m/stm32.cfg

ST STR7 programming

In the STR7 microcontrollers, several memory regions may be mapped at address 0x00000000, including the internal FLASH. So these are the acceptable values for the BASE_ADDR parameter in the FLASH section:

- 0x00000000 (STR71x and STR73x)
- 0x40000000 (STR71x)
- 0x80000000 (STR73x)

If the SECURE_FLASH target configuration parameter is set to YES. The first time, the device is secured by programming the DBGP bit of the NVAPR0 register. Each time after, the device is secured programming the next un-programmed bit PEN bit of the NVAPR1 register.

Keep in mind that once secured, the device may be temporary or permanently unsecured only by the code that is programmed in the FLASH, so avoid securing the device if the code inside it can not unlock it, because the devise may become unusable.

The device can be permanently secured-unsecured only sixteen times, because after that all NVAPR1 bits are programmed.

Example:

http://download.ronetix.at/peedi/cfg_examples/arm7/str710.cfg

ST STR9 programming

PEEDI has built in commands for managing the STR9 In System Configuration (ISC):

```
flash this isc_erase       - ISC full erase
flash this isc_erase 0x3   - ISC erase sector 0 and 1 of bank 0
flash this isc_conf_write 0x0001000000000000 - set bank 1 as boot
flash this isc_conf_read   - print current configuration
flash this isc_boot_bank 0 - set booting from bank 0
flash this isc_boot_bank 1 - set booting from bank 1
flash this isc_lock        - lock STR9 device
```

Example:

http://download.ronetix.at/peedi/cfg_examples/arm9/str9.cfg

TI TMS570 programming

Example:

http://download.ronetix.at/peedi/cfg_examples/cortex-a/tms570.cfg

TI TMS470 programming

TMS470 devices use four WORD long keys to protect FLASH from unwanted erase/write operations, so be careful not to write them accidentally The keys are reported every time the FLASH is programmed.

Every time PEEDI first tries to unlock FLASH using the default keys (0xFFFFFFFF), if fails it uses the keys pointed out in the target configuration file. This way you can erase and program the FLASH without the need of changing the keys for each operation, because during FLASH erase the keys are automatically set to 0xFFFFFFFF.

In TMS470 devices that have Memory Security Module (MSM), if the currently programmed MSM keys are different from 0xFFFFFFFF, you have to put this unlock sequence in the INIT section:

```
[INIT_TMS470]
; dummy read the four keys
mem read 0x0000ffe0
mem read 0x0000ffe4
mem read 0x0000ffe8
mem read 0x0000ffec
; try to unlock the device using the correct MSM keys
mem write 0xFFFFF700 0xXXXXXXXX
mem write 0xFFFFF704 0xXXXXXXXX
mem write 0xFFFFF708 0xXXXXXXXX
mem write 0xFFFFF70C 0xXXXXXXXX
```

Where 0xXXXXXXXX's are the right MSM keys. The four word passwords location in the internal FLASH for the MSM1 is placed starting from the last eight words of the first flash sector. Please see the datasheet of your TMS470 CPU to check the right addresses of the keys in FLASH memory and the addresses of the registers where the keys have to be entered. These passwords are used to insecure the device in case it has been partly secured.

The ALLOW_ZERO_KEYS FLASH section parameter is used to protect the device from unwanted permanent locking of the device - this may happen if MSM keys all of 0x000000000 are programmed in to the FLASH.

Some TMS470 devices have internal Analog Watch Dog timer (AWD). The AWD must be disabled in order to use PEEDI for debugging or programming. The AWD can be disabled by grounding the AWD pin. Alternatively WDKICK_TIME CFG parameter can be used and PEEDI will kick periodically the AWD.

Example:

http://download.ronetix.at/peedi/cfg_examples/arm7/tms470.cfg

PIC32, SmartFusion A2F, ADuC, EFM32 programming

Examples:

http://download.ronetix.at/peedi/cfg_examples/cortex-m/a2f200.cfg

http://download.ronetix.at/peedi/cfg_examples/arm7/aduc7034.cfg

http://download.ronetix.at/peedi/cfg_examples/mips/pic32.cfg

http://download.ronetix.at/peedi/cfg_examples/cortex-m/efm32.cfg

CHIP

Synopsis:

```
CHIP = <type>
```

Description:

FLASH chip type. To find if your flash is supported and see its exact name, use flash find . If your FLASH chip is not supported by the current FLASH database please contact us and we will provide you with the latest database. This parameter may be present multiple times in a single FLASH section, each time specifying different FLASH chip. This way if the CHECK_ID parameter is YES, PEEDI will read the onboard FLASH ID and will find the right chip among the all chips enumerated using the CHIP parameter.

CHECK_ID

Synopsis:

```
CHECK_ID = YES|NO
```

Description:

When specified YES, if single FLASH chip is described by the CHIP parameter, PEEDI will check if the onboard FLASH chip reports the same as selected by the CHIP parameter. If multiple FLASH chips are enumerated using more than one CHIP parameter, PEEDI will automatically consider the chip which ID matches the reported by the onboard FLASH chip.

PART_ID

Synopsis:

```
PART_ID = <hexadecimal_value>
```

Description:

This parameter is used to override an incorrect CPU ID for LPC processors.

PARTITION

Synopsis:

```
PARTITION = <value>
```

Description:

This parameter is used to select current eMMC partition.

Use ’flash this part’ to manage the eMMC partitions.

BANK

Synopsis:

```
Bank = <number>
```

Description:

Some devices have more than one flash bank. By this parameter it can be specified which flash bank is configured. For example to configure the first bank of a device with flash banks A and B specify the parameter like so:

BANK = 0;

To configure the second bank specify:

BANK = 1;

ACCESS_METHOD

Synopsis:

```
ACCESS_METHOD = AUTO|AGENT|DIRECT
```

Description:

Flash programming method. If AGENT is specified, the FLASH programmer will return an error if the agent failed to start; if AUTO is specified, the programmer will try to start the agent; if failed it will perform direct programming. If DIRECT is specified the programmer will perform direct programming.

CHIP_WIDTH

Synopsis:

```
CHIP_WIDTH = 8|16|32
```

Description:

Chip width, some FLASH chips support several widths.

CHIP_COUNT

Synopsis:

```
CHIP_COUNT = 1|2|4
```

Description:

Number of FLASH chips.

CHIP_SIZE

Synopsis:

```
CHIP_SIZE = <chip_size>, [page_size]
```

Description:

Size of EEPROM chip and optional write page size.

BASE_ADDR

Synopsis:

```
BASE_ADDR = <address>
```

Description:

Start address of FLASH.

FILE

Synopsis:

```
FILE = FILE_NAME, FILE_FORMAT[, FILE_ADDRESS]
```

Description:

This parameter defines the default flash (multi) program command's arguments. This parameter may have two or three arguments. The first argument is the file to be programmed. The second argument is the file type - BIN, SREC, IHEX or ELF. The third argument is mandatory for binary files and optional for all other types of files – it is the address where the file should be loaded.

SPI_MODE

Synopsis:

```
SPI_MODE = <number>
```

Description:

By default, this parameter takes the value of '0'. Available SPI modes are '0' and '3'

AUTO_ERASE

Synopsis:

```
AUTO_ERASE = YES|NO
```

Description:

Do or do not erase affected FLASH sectors before program operation, for more

information, see **flash program** command.

AUTO_LOCK

Synopsis:

```
AUTO_LOCK = YES|NO
```

Description:

Do or do not lock affected FLASH sectors (if supported from FLASH) after program operation, this would prevent FLASH form accidental write or erase operations.

CPU_CLOCK

Synopsis:

```
CPU_CLOCK = <kHz>
```

Description:

The CPU clock after the initialization. Used when describing internal FLASH of Atmel AT91SAM7, Philips LPC2000 and Freescale MAC7100 or MCF5200 series microcontrollers.

SECURE_FLASH

Synopsis:

```
SECURE_FLASH = YES|NO
```

Description:

Do or do not secure FLASH to avoid external reading operations. This disables the JTAG interface until the whole FLASH is erased, so any JTAG operations are impossible after FLASH is programmed and secured.

Used when describing internal FLASH of Atmel AT91SAM7 series microcontrollers.

SET_VECTORS_CHECKSUM

Synopsis:

```
SET_VECTORS_CHECKSUM = YES|NO
```

Description:

Set this parameter to YES, if you want PEEDI to automatically calculate and set the exception vectors checksum at address 0x14 while programming FLASH. This check sum is required by the microcontroller bootloader as evidence that valid user application resist in the FLASH, so the control will be passed to it.

Used when describing internal FLASH of Philips LPC2000 series microcontrollers.

DATA_BANK

Synopsis:

```
DATA_BANK = YES|NO
```

Description:

Set this parameter to YES, if your device has flash data bank (bank1).

Used when describing internal FLASH of ST STR7 series microcontrollers.

BANK_SIZE

Synopsis:

```
BANK_SIZE = <bank0_size>,<bank1_size>
```

Description:

Set this parameter to the sizes of the STR9 FLASH banks.

Used when describing internal FLASH of ST STR9 series microcontrollers.

DATA_FLASH

Synopsis:

```
DATA_FLASH = YES|NO
```

Description:

Set this parameter to YES to enable the DATA Flash programming.

Used when describing internal FLASH of Renesas RA Cortex-M33 series microcontrollers.

F2F4_PSIZE

Synopsis:

```
F2F4_PSIZE = 8|16|32|64
```

Description:

Set this parameter 8, 16, 32 or 64, which selects the program parallelism for fast STM32 FLASH programming. 64-bit parallelism can be used only if external Vpp is provided. If missing, 16-bit parallelism is used by default.

PROTECTION_KEY0 – PROTECTION_KEY3

Synopsis:

```
PROTECTION_KEY0 = <value0>
PROTECTION_KEY1 = <value1>
PROTECTION_KEY2 = <value2>
PROTECTION_KEY3 = <value3>
```

Description:

These four parameters define the four FLASH security keys that are used to unlock the FLASH for erasing and writing.

Used when describing internal FLASH of TI TMS470 series microcontrollers.

ALLOW_ZERO_KEYS

Synopsis:

```
ALLOW_ZERO_KEYS = YES|NO
```

Description:

This parameter is used to prohibit programming of new Memory Security Module keys that are all 0x00000000, because this will permanently lock the TMS against debugging and programming.

Used when describing internal FLASH of TI TMS470 series microcontrollers.

CPU

Synopsis:

CPU = AT91RM9200 | AT91SAM9261 | AT91SAM9263 | AT91SAM7 | ATSAMA5 | iMX21 | iMX23 | iMX25 | iMX27 | iMX31 | iMX35 | iMX51 | iMX53 | BF5XX | BF52X | BF54X | MC1322X | MPC5121 | MPC5125 | MPC83XX | NS92XX | TMS320DM355 | TMS320DM365 | LPC2XXX | LPC318X_MLC | LPC3XXX_SLC | `PXA3XX | GENERIC_SPI | GENERIC_I2C`

Description:

Target CPU.

Used when describing NAND, Card, SPI, I2C or Atmel DataFlash.

SPI_DIV

Synopsis:

```
SPI_DIV = <div>
```

Description:

SPI divider:

AT91RM9200: Fspi = (MCK/2)/SPI_DIV

AT91SAM9261: Fspi = MCK/SPI_DIV

Used when describing SPI Flash.

nSPI

Synopsis:

```
nSPI = 0|1
```

Description:

SPI controller to use.

Used when describing SPI Flash.

nCS

Synopsis:

```
nCS = 0..3
```

Description:

Chip select to use.

Used when describing SPI Flash.

SPI_SPCK, SPI_MISO, SPI_MOSI, SPI_CS

Synopsis:

```
SPI_SPCK = <controller>, <peripheral>, <pin>
```

Description:

This describes which PIOs are dedicated to the SPI SPCK, MISO and MOSI signals. Used when describing Atmel DataFlash. If the CPU parameter is set to BF5XX, only SPI_CS is accepted and expects 1..7, which corresponds to FLG1-FLG7.

See Blackfin's SPI_FLG.

Example:

```
SPI_SPCK = PIOA, A, 2
SPI_MISO = PIOA, A, 0
SPI_MISO = PIOA, A, 1
SPI_CS   = PIOA, A, 3
```

CMD_BASE

Synopsis:

```
CMD_BASE = <address>
```

Description:

Base address, that if written to, the NAND CLE signal will be asserted. On MPC83XX devices with built-in NAND FLASH controller this parameter tells PEEDI the offset of Internal Memory Mapped Registers, i.e. value of IMMRBAR.

DATA_BASE

Synopsis:

```
DATA_BASE = Address
```

Description:

Base address, that if written to, the NAND ALE and CLE signals will be inactive. On MPC83XX devices with built-in NAND FLASH controller this parameter tells PEEDI the address of the data buffer used by NAND FLASH controller.

ADDR_BASE

Synopsis:

```
ADDR_BASE = <address>
```

Description:

Base address, that if written to, the NAND ALE signal will be asserted.

CS_ASSERT/RELEASE

ALE_ASSERT/RELEASE

CLE_ASSERT/RELEASE

Synopsis:

```
CS_ASSERT = <address>, <data>
CS_RELEASE = <address>, <data>
```

```
ALE_ASSERT = <address>, <data>
ALE_RELEASE = <address>, <data>
CLE_ASSERT = <address>, <data>
CLE_RELEASE = <address>, <data>
```

Description:

Describes memory write operation ([address]=data) that will assert/release the NAND chip select, Address Latch Enable and Command Latch Enable connected to a corresponding PIO pin.

BAD_BLOCK_TABLE

Synopsis:

```
BAD_BLOCK_TABLE = YES|NO
```

Description:

If this parameter is set to YES, PEEDI will check for Linux style main and mirror Bad Block Tables and if not found, it will create them on the last two good blocks of the NAND FLASH chip.

BAD_BLOCKS

Synopsis:

BAD_BLOCKS = <bad1>, <bad2>,...

Description:

List of blocks to be marked as bad.

ERASE_BAD_BLOCKS

Synopsis:

```
ERASE_BAD_BLOCKS = YES|NO
```

Description:

If this parameter is set to YES, PEEDI will try to erase even the bad NAND blocks.

SWAP_BI

Synopsis:

```
SWAP_BI = YES|NO
```

Description:

If this parameter is set to YES, PEEDI will swap the bad block marker ECC byte with a spare one.

This option is applicable for iMX21, iMX25, iMX27, iMX31 and iMX35 targets only.

OOB_INFO

Synopsis:

```
OOB_INFO = <oob_type>
```

Description:

How to deal with the Out Of Band (OOB/spare) page. Spare bytes are extra bytes added to the page.

- JFFS2 - data bytes will be read from the image file, spare bytes will be filled with ECC data (6 bytes for 512 bytes page, 24 bytes for 2048 bytes page).
- JFFS2_NO_EM - like JFFS2 but PEEDI does not write erase/clean markers
- RAW - data and spare bytes will be loaded from the image file, default if OOB_INFO parameter is missing
- YAFFS - like RAW, but bad blocks are skipped
- FF - only data bytes will be read from the image file, spare bytes will be set to 0xFF
- ONDIE_ECC - Micron NAND FLASH On-Die ECC
- AT91_PMECC - Atmel AT91SAM9X5 and SAMA5
- BLACKFIN_ECC - ADI Blackfin hardware ECC
- IMX_ECC - Freescale iMX hardware ECC

- IMX23_BCH - Freescale iMX23 hardware ECC
- LPC_ECC - NXP LPC hardware ECC
- OMAP3_ECC - TI Omap3 hardware ECC
- OMAP4_BCH8_ROMCODE - TI Omap4 hardware ECC
- OMAP4_BCH8 - TI Omap4 hardware ECC
- OMAP4_HAMMING - TI Omap4 hardware ECC
- PXA_ECC - Marvell PXA3XX hardware ECC
- ONENAND - OneNAND hardware ECC
- DAVINCI_ECC - TI DaVinci Hardware ECC, 4 bytes per 512 bytes data
- DAVINCI_ECC_HW6_512 - DaVinci hardware ECC, 6 bytes per 512 bytes data
- DAVINCI_ECC_HW6_512_2610 - DaVinci hardware ECC, 6 bytes per 512 bytes data with workaround for a JFFS2 bug in Linux kernel 2.6.10
- DAVINCI_ECC_HW10_512 - DaVinci hardware ECC, 10 bytes per 512 bytes data
- DM355_BOOT - TI TMS320DM355 hardware ECC
- DM355_LINUX - TI TMS320DM355 hardware ECC
- DM355_JFFS2 - TI TMS320DM355 hardware ECC
- DM365_BOOT - TI TMS320DM365 hardware ECC
- DM365_LINUX - TI TMS320DM365 hardware ECC
- DM365_JFFS2 - TI TMS320DM365 hardware ECC
- OMAPL138_ECC - TI OMAP L138 hardware ECC
- S5PC100_1BIT_ECC - Samsung S5PC100 hardware ECC
- S5PC100_8BIT_ECC - Samsung S5PC100 hardware ECC
- MPC5125_LOADER - Freescale MPC5125 hardware ECC
- MPC5125_UBOOT - Freescale MPC5125 hardware ECC
- APM_ECC - AMCC APM83xxx hardware ECC

DAVINCI_UBL_DESCIPTOR_MAGIC

Synopsis:

```
DAVINCI_UBL_DESCIPTOR_MAGIC = <value>
```

Description:

Descriptor magic - the first 32-bit value in the UBL descriptor. It set to non-zero value, programming of the file image is relocated with one NAND Flash page (512 or 2048 bytes). The skipped page is used for the UBL descriptor and it is filled by PEEDI.

Used when describing NAND FLASH for TI DaVinci CPU.

DAVINCI_UBL_DESCIPTOR_ENTRY_POINT

Synopsis:

```
DAVINCI_UBL_DESCIPTOR_ENTRY_POINT = <value>
```

Description:

This value will be programmed at offset 0x4 in the UBL descriptor.

Used when describing NAND FLASH for TI DaVinci CPU.

DAVINCI_UBL_DESCIPTOR_LOAD_ADDR

Synopsis:

```
DAVINCI_UBL_DESCIPTOR_LOAD_ADDR = <value>
```

Description:

Used when describing NAND FLASH for TI DaVinci CPU.

DAVINCI_UBL_MAX_IMAGE_SIZE

Synopsis:

```
DAVINCI_UBL_MAX_IMAGE_SIZE = <Value>
```

Description:

Used by PEEDI to print a warning if the programmed file size exceeds this limit.

Used when describing NAND FLASH for TI DaVinci CPU.

NUM_ECC

Synopsis:

```
NUM_ECC = <Value>
```

Description:

Set the used ECC - 2, 4 or 8-bit.

Used when OOB_INFO = AT91_PMECC.

HEADER

Synopsis:

```
HEADER = YES|NO
```

Description:

If YES, then a 52-byte header will be automatically inserted at the beginning of the image.

Used when OOB_INFO = AT91_PMECC.

IPS_BASE

Synopsis:

```
IPS_BASE = <address>
```

Description:

ColdFire Internal Peripheral System base address.

SPIFI_BASE

Synopsis:

```
SPIFI_BASE = <address>
```

Description:

NXP SPIFI controller base address.

NCB_DATA

Synopsis:

```
NCB_DATA = <value0>, <value1>, ...
```

Description:

Freescale iMX23 NCB data structure to be programmed in NAND.

LDLB_DATA

Synopsis:

```
LDLB_DATA = <value0>, <value1>, ...
```

Description:

Freescale iMX23 LDLB data structure to be programmed in NAND

SERIAL_NUM

Synopsis:

```
SERIAL_NUM = FILE, ADDRESS, WIDTH
```

Description:

If this parameter is present, PEEDI will program a unique serial number on the given FLASH location with each flash program command, this way if PEEDI is used in production, each board programmed will get an unique serial number.

The parameter has tree arguments:

- FILE - path to a text file which contains the last serial number that is

programmed. The file must contain only one line with the number, with no leading trailing spaces or any other characters. For example:

<start of file>

341

<end of file>

Each time PEEDI programs a board, it loads the file, gets the last serial number increments it and stores the new value back in the file, and so if the file resides on a TFTP or a FTP server, the server must allow write access (upload) of the file. File may also resides on a MMC/SD card or in the PEEDI internal EPPROM file system, note that each EEPROM has a limited number of writes.

- ADDRESS - where the serial number will be programmed, must be aligned to the serial number bits width.
- WIDTH - bits width of the serial number value - 16, 32, 64.

Currently the SERIAL_NUM parameter is supported for external NOR FLASH chips, Atmel DataFlash SPI chips and Atmel AT91SAM7 devices.

I2C_ADDR

Synopsis:

```
I2C_ADDR = <address>
```

Description:

The first byte in the I2C communication, which carries the chip address in the bus.

I2C_DELAY

Synopsis:

```
I2C_DELAY = <value>
```

Description:

Number of empty loops, used to achieve the I2C clock period.

SDA_SET/CLR, SDA_IN/OUT, SDA_READ, SCL_SET/CLR

Synopsis:

```
SDA_SET = ADDRESS, AND|OR|EQU, DATA, [x8|x16|x32]
```

Description:

Describes operation that will set/clear, set as input/output and read SDA/SCL pins of the CPU. AND, OR and EQU operations are permitted:

- AND - The value pointed by the address is AND-ed with the given data
- OR - The value pointed by the address is OR-ed with the given data
- EQU The data provided is written at the given address

CS_ASSERT/RELEASE, SCLK_SET/CLR, MOSI_SET/CLR, MISO_READ

Synopsis:

```
MOSI_SET = ADDRESS, AND|OR|EQU, DATA, [x8|x16|x32]
```

Description:

Describes operation that will set/clear or read SPI pins of the CPU. AND, OR and EQU operations are permitted:

- AND - The value pointed by the address is AND-ed with the given data
- OR - The value pointed by the address is OR-ed with the given data
- EQU - The data provided is written at the given address

Section OS

This section contains parameters which help PEEDI scan the target OS task list.

ITEM

Synopsis:

```
ITEM = <type_access>, <name>, <offset> [, offset[,
offset]]
```

Description:

type - type of the field:

- int - item is an integer number
- reg - item is a CPU register

- str - item is a string
-

access - what memory access to be used to read:

- 4x8 - 32-bit value using 8-bit access
- 2x16 - 32-bit value using 16-bit access
- 32 - 32-bit value using 32-bit access
- 8x8 - 64-bit value using 8-bit access
- 4x16 - 64-bit value using 16-bit access
- 2x32 - 64-bit value using 32-bit access
- 64 - 64-bit value using 64-bit access

type_access argument might has a **_abs** suffix, which means that the address calculated of the **offset** parameters is an absolute memory location, not an offset from the base of the task.

name - name of the item, might be a CPU register name or some of there:

- BASE - it tells PEEDI the item is the base address of the task OS list
- NEXT - it is a pointer to the next task in the list
- PID - it is the process ID of the task
- NAME - it is the human readable name of the task

offset - offset of the item in the task list, multiple offsets may be used for a pointer-to-pointer like behavior
As **offset** a valid application symbol might be used, this way, upon GDB connection, PEEDI will ask GDB for the address of the given symbol and use it.

Example:

```
ITEM = int32_abs, BASE, 0x12345678; BASE = 0x12345678
ITEM = int32_abs, BASE, 0x12345678, 0 ; BASE = *0x12345678
ITEM = int32_abs, BASE, 0x12345678, 0x20 ; BASE =
*(0x12345678+0x20)
ITEM = int32_abs, BASE, Cyg_Scheduler_Base::current_thread
ITEM = int32_abs, BASE, Cyg_Scheduler_Base::current_thread,
0x20
ITEM = int32_abs, BASE, 0x20000000, 0x1234, offset2
ITEM = int32, PID, 0xA4 ; PID = *(BASE + 0xA4)
ITEM = string, NAME, 0xEC, offset2, offset3
ITEM = int32, R0, 0xC ; R0 = *(BASE + 0xC)
ITEM = int32, R1, 0x10, 0x20 ;R1 = *(*(BASE + 0x10) + 0x20)
ITEM = reg32_abs, xPSR, 0x20000000, 0 ; xPSR = *0x20000000
```

Section SERIAL

BAUD

Synopsis:

```
BAUD = 1200|2400|4800|9600|19200|38400|57600|115200
```

Description:

Serial Baud rate

STOP_BITS

Synopsis:

```
STOP_BITS = 1|1.5|2
```

Description:

Serial Stop bits

PARITY

Synopsis:

```
PARITY = NONE|EVEN|ODD
```

Description:

Serial Parity

TCP_PORT

Synopsis:

```
TCP_PORT = 0|1024..65535
```

Description:

Port, serial traffic to be routed to. If set to 0, the PEEDI serial port is used for command line interface. 0 - use PEEDI serial for command line interface.

Example:

```
[SERIAL]
BAUD = 115200
STOP_BITS = 1
PARITY = NONE
TCP_PORT = 2023
```

Section TELNET

This section has only two parameters. The first sets the new command prompt string after the configuration file is loaded. The second parameter can be omitted; it tells what ASCII code to be used for backspace action.

PROMPT

Synopsis:

`PROMPT = "<prompt>"`

Description:

This will change the default PEEDI telnet prompt

BACKSPACE

Synopsis:

`BACKSPACE = <code>`

Description:

Telnet backspace character ASCII code.

Example:

```
[TELNET]
PROMPT = "peedi> "
BACKSPACE = 127
```

Section DISPLAY

These sections parameters specify the brightness of the seven segment LED indicator and the volume of the speaker, both accept values in the range 0 - 100.

VOLUME

Synopsis:

```
VOLUME = 0 .. 100
```

Description:

Speaker volume

Example:

```
[DISPLAY]
VOLUME=0 - disable beeper
VOLUME=100 - enable beeper
```

Section ACTIONS

Declares what scripts can be executed using front panel buttons, each declaration must be on a new line. The declaration consists of a number associated with the specified script name. A section with the same name must exist somewhere in the target configuration file. If AUTORUN=N parameter is specified, where N is number of a script, the given script will be executed every time a target is connected to PEEDI. For more information see Script execution using the front panel interface .

Example:

```
[ACTIONS]
AUTORUN = 2
1 = erase_program_verify
2 = prog_http
[erase_program_verify]
flash prog tftp://192.168.1.41/main_romram.bin bin 0x400000
flash verify tftp://192.168.1.41/main_romram.bin bin 0x400000
[prog_http]
flash prog
http://192.168.1.41/main_romram.bin bin 0x400000 erase
```

4.6 CPU specific considerations

4.6.1 Philips LPC2000 family

To successfully connect to a LPC2000 device the pull-down resistor that enables the JTAG interface must not be more than 1k, because PEEDI has internal 10k pull-ups.

Because the JTAG clock is synchronized to the internal CPU clock it is recommended to use adaptive JTAG clock or clock up to 1MHz for normal work (the second argument of the JTAG_CLOCK parameter).

4.6.2 ST STM32 family

Use the following commands in the target INIT script to enable SWO stimulus output:

```
; init SWO
mem wr 0xE0042004 0x20              ; TRACE_IOEN (async mode)
mem wr 0xE0040010 2                 ; TPIU_ACPR = 1 (SWO prescale 2)
mem wr 0xE00400F0 1                 ; TPIU_SPPR = 1 (SWO Manchester
encoding)
mem wr 0xE0040304 0                 ; TPIU: disable formatter
mem wr 0xE0000FB0 0xC5ACCE55        ; ITM: unlock ITM_TCR
mem wr 0xE0000E80 0x10009           ; ITM_TCR: TXENA + ITMENA +
TraceBusID=0x1
mem wr 0xE0000E40 0xF               ; en. all tracing ports
mem wr 0xE0000E00 0xFFFFFFFF        ; en. all stimulus ports
```

PEEDI supports only Manchester SWO encoding up to 66MHz.

PEEDI checks for new incoming telnet connection only when the target CPU is halted.

If the SWO functionality seems unstable, lower the CPU clock or increase the SWO prescaler, both of these will result in lower SWO clock.

4.6.3 Intel XScale family

Debugging XScale core is a little complicated by the fact that the exception vectors must be cached in the mini instruction cache where the debug handler resides. PEEDI provides two ways of defining the vectors see Section PLATFORM_XSCALE . First is to set them fixed - suitable when the vectors are not updated dynamically at runtime. And the second is to tell PEEDI to read them from the target's memory each time a debug event occurs - suitable when vectors are set by the user application at runtime. There are several ways to provoke a debug event:

- Set 32 bit write access watch point at the last modified by the user code vector.

- Set hardware breakpoint to a point of the code where the vectors have been set but not yet enabled.
- In the source code, add a software break ’asm(’bkpt 1’);’, where the vectors have been set but not yet enabled. PEEDI recognizes this special break and immediately starts the target again with refreshed vectors.

Once the target has been stopped by the desired debug event, you can again start it and the exception vectors will be updated. You can use the first way and a wait command to automate this in the INIT section of the target configuration file:

```
[INIT_XSCALE]
break add watch 0xffff001C w 32    ; set watchpoint on FIQ vector
go                                 ; start target
wait 30000 stop                    ; wait to break
go                                 ; start again with updated vectors
```

If the vectors are set during code download, they will be automatically updated if defined AUTO.

If you want the target to be stopped after it is initialized, just remove the last two lines from the previous example of the XScale init section.

Avoid using code that potentially could invalidate the mini instruction cache during debug:

- Don’t use ’MCR p15, 0, rd, c7, c5, 1’, instead use ’MCR p15, 0, rd, c7, c5’
- Disable CONFIG_XSCALE_CACHE_ERRATA (Workaround for XScale cache errata) option when building Linux kernel.

4.6.4 Freescale PowerQUICC II Pro MPC83XX family

PEEDI supports debugging Linux kernel running on MPC8300 devices with MMU enabled. If the instruction or data address translation is enabled (IR and DR bits set in MSR), PEEDI assumes all the addresses used by gdb or the user in the console to be effective and need to be translated to physical ones before the very memory access. First PEEDI ties a BAT translation, if it fails and the COREn_MMU_PTBASE parameter is present in the target configuration file, then PEEDI tries a page translation on the given address.

The COREn_MMU_PTBASE parameter must point to a physical address which contains the virtual address of the two pointers array:

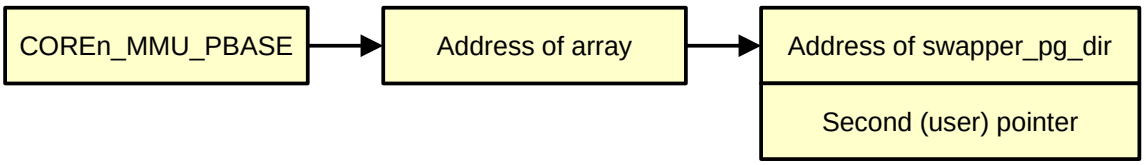


Figure 4.3: MMU_PTBASE

Before debugging the user must manually set all the pointers using mem write commands which can be put in the INIT section for example.

Newer Linux kernels have built-in support for automatic update of the pointers, so no user setup is required, only the COREn_MMU_PTBASE must be set to 0xF0, where the kernel puts the pointer to the array. But this feature must be enabled in order to be used - 'make menuconfig' → kernel hacking → Include BDI-2000 user context switcher, here you can also set the Compile the kernel with debug info option.

Sometimes gdb and insight do not want to load the debug info because of an internal bug, in this case adding '-gstabs+' option in the makefile fixes this.



WARNING:
On some cores (MPC8349) in order the software breakpoints to work, the interrupt vectors must reside in valid memory. So you must either initialize the memory properly or either set the MSR[IP] to a value so the vectors fall on a valid memory.

When the COREn_RCW CFG parameter is present, PEEDI overrides the Reset Configuration Words with the values provided. This is useful when the RCW that is fetched during board reset does not suit the user's needs when debugging or programming the board using PEEDI.

The COREn_BOOT_ADDR CFG must be set considering the RCW that is fetched/set, because RCW sets the reset vector address (the boot address).

If PEEDI reports '++ info: target does not enter debug mode, forcing halt', this might mean that the CPU boots from an address different than the one set by the COREn_BOOT_ADDR parameter. So you should check again both COREn_RCW and COREn_BOOT_ADDR parameters.

4.6.5 Analog Devices Blackfin family

Most of the Blackfin devices can access only up to 4MB of external FLASH memory. Usually this limitation is workarounded by connecting the free higher address pins of FLASH to PIO PFx pins of the CPU. So the software running on the CPU must ensure the PFx pins are driven properly to

access the desired part of the FLASH. PEEDI also supports this kind or FLASH configuration making the work with such kind of configuration just like as the FLASH is entirely visible. For more information see Section PLATFORM_BLACKFIN .

PEEDI supports programming of NAND flash chips connected to the CPU's async bus or the NFC controller. FLASH and Atmel DataFlash chips connected to the CPU's SPI. For all these configurations PEEDI writes only to the specific peripheral controller registers. It is the user's responsibility to set the needed GPIOs in the INIT section of the CFG file, i.e. to set the corresponding PORTx_FAR and PORTx_MUX registers so the FLASH is accessible through async bus, NFC or SPI. For help on this check the sample CFG files from our website.

It is observed that the presence of the ADI USB JTAG debug interface (BF535+Spartan FPGA), interferes the normal PEEDI operation (seen with some EZ-KIT and STAMP boards). Some Blackfin boards do not work reliably with low JTAG clock, so if you experience problems in the INIT section, please use 2MHz init JTAG clock.

4.7 Boot sequence

On power-up if the front panel buttons are pressed, the bootloader is started. If not, PEEDI tries to load the configuration file. After that it checks if the target is powered. Then if the RESET_TIME is bigger than 0 it asserts the target RST and waits the specified time. After that if COREn_STARTUP_MODE is set to RESET, PEEDI sets the target in debug mode, which assures that no instructions are executed when the target RST is released. Next the target RST is released and PEEDI waits the TIME_AFTER_RESET. After that the initial JTAG clock is set. If the COREn_STARTUP_MODE is RUN it switches to the normal JTAG clock and the target is left running. Else it checks if the COREn_STARTUP_MODE is STOP and if yes it waits the specified period and stops the target. Then the init section is processed. After that the JTAG clock is switched to its normal speed.

Note that Different cores may have different COREn_STARTUP_MODE parameter set.

The following diagram shows the boot sequence:

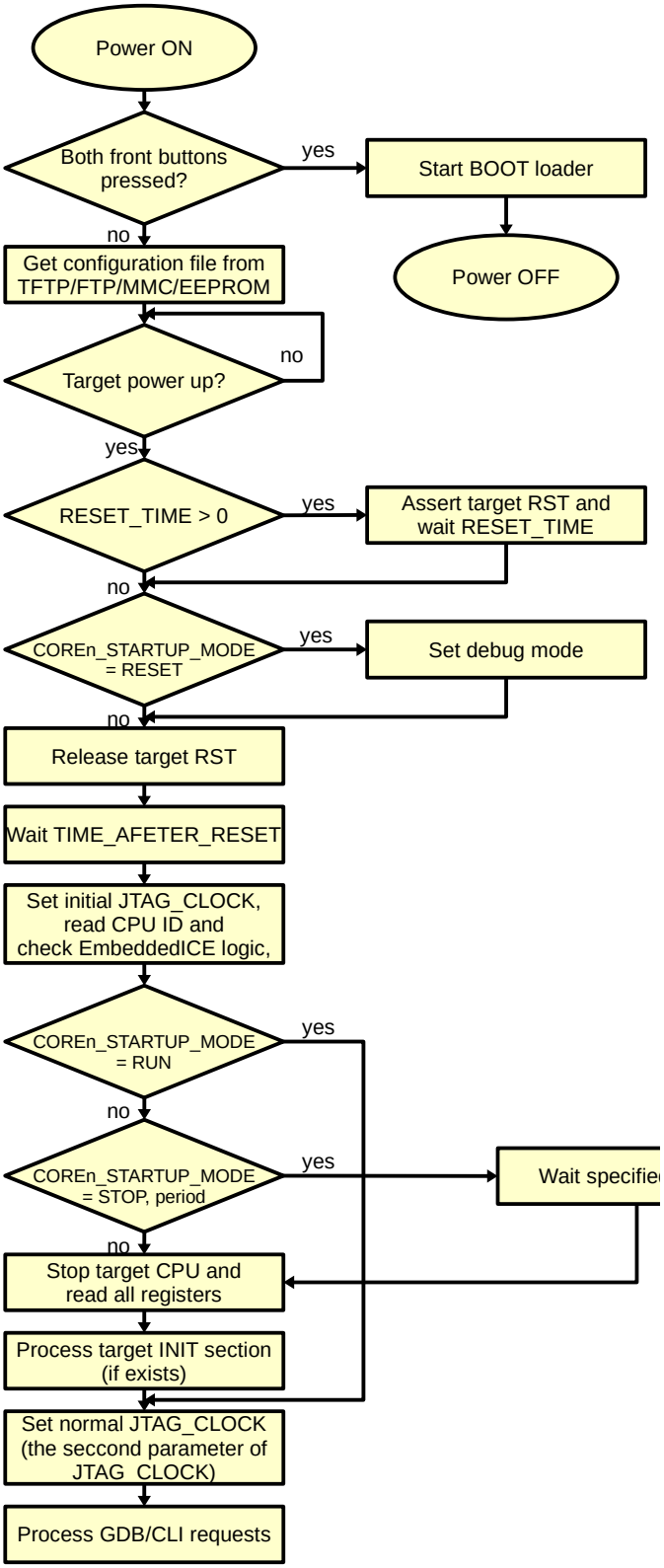


Figure 4.4: PEEDI Boot Sequence

4.8 Multiple core support

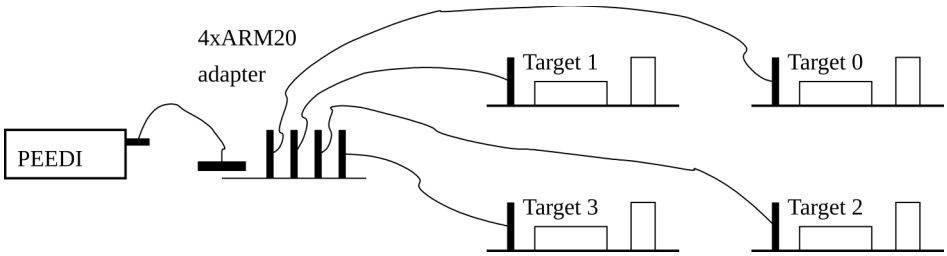


Figure 4.5: Multi Target connection

Extra license SW-MULTIPROG is required to allow you to program multiple targets using only single PEEDI. SW-MULTIPROG is required only when using ‘flash multi‘ sub-commands. The targets must be chained using the multiple core cable adapter available from Ronetix:



WARNING:
All targets must have equal power supply (10% tolerance is permissible). The highest power supply is taken for reference for the PEEDI output schematic, so the JTAG signals will have that value.



WARNING:
As short as possible cables should be used, because the equivalent cable length is the sum of all cables. Even then, you may need to decrease the JTAG clock in the target configuration file.

The JTAG_CHAIN parameter in the PLATFORM_CORTEX section must be correctly set and each core must be described.

Example PLATFORM_CORTEX section describing four chained cores:

```
[PLATFORM_Cortex-M]
JTAG_CHAIN              = AUTO x 5,4   ;  automatically detect up to 4 CPUs
in the JTAG chain
JTAG_CLOCK              = 100, 5000 ; JTAG Clock in [kHz]
TRST_TYPE               = OPENDRAIN ; type of TRST output: OPENDRAIN or
PUSHPULL
RESET_TIME              = 20        ; length of RESET pulse in ms; 0 means
no RESET
CORE0                   = Cortex-M,1; TAP is CortexM3  CPU
CORE0_STARTUP_MODE      = RESET     ; stop the core immediately after reset
CORE0_ENDIAN            = LITTLE    ; core is little endian
CORE0_BREAKMODE         = HARD      ; breakpoint mode
CORE0_INIT              = INIT_STM32    ; init section for STM32
CORE0_FLASH             = FLASH_STM32   ; FLASH section parameters
CORE0_WORKSPACE         = 0x20000000, 0x4000 ; workspace for agent
programming
CORE0_PATH              = "tftp://192.168.3.1"

CORE1                   = Cortex-M,3; TAP is CortexM3  CPU
CORE1_STARTUP_MODE      = RESET     ; stop the core immediately after reset
CORE1_ENDIAN            = LITTLE    ; core is little endian
CORE1_BREAKMODE         = HARD      ; breakpoint mode
CORE1_INIT              = INIT_STM32    ; init section for STM32
CORE1_FLASH             = FLASH_STM32   ; FLASH section parameters
CORE1_WORKSPACE         = 0x20000000, 0x4000 ; workspace for agent
programming
CORE1_PATH              = "tftp://192.168.3.1"

CORE2                   = Cortex-M,5; TAP is CortexM3  CPU
CORE2_STARTUP_MODE      = RESET     ; stop the core immediately after reset
CORE2_ENDIAN            = LITTLE    ; core is little endian
CORE2_BREAKMODE         = HARD      ; breakpoint mode
CORE2_INIT              = INIT_STM32    ; init section for STM32
CORE2_FLASH             = FLASH_STM32   ; FLASH section parameters
CORE2_WORKSPACE         = 0x20000000, 0x4000 ; workspace for agent
programming
CORE2_PATH              = "tftp://192.168.3.1"

CORE3                   = Cortex-M,7; TAP is CortexM3  CPU
CORE3_STARTUP_MODE      = RESET     ; stop the core immediately after reset
CORE3_ENDIAN            = LITTLE    ; core is little endian
CORE3_BREAKMODE         = HARD      ; breakpoint mode
CORE3_INIT              = INIT_STM32    ; init section for STM32
CORE3_FLASH             = FLASH_STM32   ; FLASH section parameters
CORE3_WORKSPACE         = 0x20000000, 0x4000 ; workspace for agent
programming
CORE3_PATH              = "tftp://192.168.3.1"
```

Each core has its own TCP port number to wait for a debug session. The port number is the number specified in the DEBUGGER section of the target configuration file plus the number of the core. For example if the port specified is 2000 and the core number is 2 (starting from 0), then you should connect to PEEDI for a debug session at TCP port 2002:

```
(gdb) target remote 192.168.1.10:2000 // first target
(gdb) target remote 192.168.1.10:2002 // third target
```



Note:
The reset JTAG signal is common for all targets, so if one developer resets his target, all targets will get reset.

When opening a CLI telnet session, the first core (number 0) is selected as default core. To select another core to work with, use the **core** command:

```
peedi> core #1
```

Using the **flash multi program** and f**lash multi erase** CLI commands, you can program up to four targets at once, saving huge amounts of time when many boards need to be programmed:

```
peedi> flash multi erase #0 #1
peedi> flash multi program #0 #1 tftp://192.168.1.1 myfile elf
```

This will program targets 0 and 1 simultaneously.

4.9 Script execution using the front panel interface

You can define various command scripts in the configuration file and execute them using the front panel buttons. Press the green button to choose the script you wish to execute, the LED indicator will show the numbers associated with the available scripts, when ready with the choice press the red button to start the script. While the script is being executed the LED indicator will rotate its segments to show that the execution is in progress. When the script is successfully completed, the led indicator will show the chosen script and the speaker will produce a single beep notifying the end of operation. If an error occurs during execution (some of the commands exited with error code), the execution is terminated, the LED indicator will start to blink with the error code, and the speaker will beep a number of times equal to the error code. Then you can start the script again by pressing the red button. If you press the green button the display will show the current script, pressing it again will show the next available script, so you can chose another script to execute. Here are the available error codes:

- TIMEOUT

- NOT FOUND
- INVALID ARGUMENT
- GENERIC ERROR

More information about the error can be obtained by connecting to PEEDI using telnet and then restarting the script. Status messages are output to every opened telnet connection when a script is executed.

The scripts are defined in the following manner: in the [ACTIONS] section list all scripts that you want to define in this format:

N = script_name

where N a hex number (1-9 and A-F) and is associated with the script_name, then define section named [script_name] and put any number of commands, each command must be on a new line.

These scripts are useful when using PEEDI in autonomous (stand-alone) mode, not connected to a PC. In such mode PEEDI can be used as a stand-alone FLASH programmer. If all needed files are stored on a MMC/SD card no Ethernet cable is necessary and PEEDI will only need a power supply cable. If AUTORUN=N parameter is specified, where N is number of a script, the given script will be executed every time a target is connected to PEEDI. This eliminates the need to manually start the script, very useful and time saving when large volumes of target boards need tobe programmed.

Example:

```
[ACTIONS]
AUTORUN=3
1 = prog_tftp_dump
2 = erase_program
3 = prog_tftp
4 = prog_card
6 = dump_tftp
8 = dump_card

[erase_program]
flash erase 0x400000 0x600000
flash prog tftp://192.168.1.41/main_romram.bin bin 0x400000

[prog_tftp]
flash prog tftp://192.168.1.41/main_romram.bin bin 0x400000 erase

[prog_tftp_dump]
flash prog tftp://192.168.1.41/dump.bin bin 0x400000 erase

[prog_card]
flash prog card://dump.bin bin 0x400000 erase

[dump_tftp]
memory dump 0x400000 0x100000 tftp://192.168.1.41/dump.bin

[dump_card]
memory dump 0x400000 0x100000 card://dump.bin
```

4.10 Serial Interface

PEEDI's RS232 connector is routed to a predefined TCP port (in the configuration file). This way if a telnet connection is opened to that TCP port, the telnet application will receive each byte coming in through the RS232 port. Vice versa, all data that is sent from the telnet application to the TCP port is forwarded to the RS232 port. Note that no flow control is supported. You can use the normal command line telnet connection to PEEDI simultaneously, as they work completely independent.

For information how to set serial port parameters, see section SERIAL in 'Target configuration file' chapter.

4.11 ARM DCC Interface

On ARM targets, PEEDI routes the core's DCC to a TCP port. This way if a telnet connection is opened to that TCP port, the telnet application will receive each byte coming in through the DCC.

The TCP port to connect to, is the value specified for the COREn_DCC_PORT parameter in the target configuration file.

You can use these GNU GCC compatible, simple C functions in your target code to communicate via the DCC:

```
#define DCC_TX_BUSY 2
#define DCC_RX_READY 1

unsigned int dcc_recv_char( void )
{
     unsigned int cc, status;
     do __asm__ volatile ( "mrc p14,0, %0, c0,
     c0\n" : "=r" (status));
     while ( !(status & DCC_RX_READY) );
     __asm__( "mrc p14,0, %0, c1,
     c0\n" : "=r" (cc));
     return cc;
}

void dcc_send_char( unsigned int cc )
{
     unsigned int status;
     do __asm__ volatile ( "mrc p14,0, %0, c0,
     c0\n" : "=r" (status));
     while ( status & DCC_TX_BUSY );
     __asm__( "mcr p14,0, %0, c1,
     c0\n" : : "r" (cc));
}

void dcc_send_string( const char* ss )
{
     while ( *ss ) dcc_send_char( *ss++ );
}
```

Keep in mind that these are blocking functions.

4.12 Working with gdb

Since Firmware version v16.x.x sending of target descriptions to gdb via qXfer:features:read packet is supported and mandatory. PEEDI supports only GDB versions which implement this feature. GDB versions older then v6.8 don’t implement it and are not supported by PEEDI.

To be able to debug an application with gdb the application must be compiled using the ’-g -O0’ options to enable debugging and disable optimizing.

When your application is built and ready to be debugged, start gdb:

```
$ arm-elf-gdb myapp
```

To connect to the target (assuming that your PEEDI is set to use IP 192.168.1.10) type in the console window:

```
(gdb) target remote 192.168.1.10:2000
```

This will tell GDB to connect to PEEDI using remote protocol. Now you can load your application into target's memory like this:

```
(gdb) load
```

This will load required application sections into target memory at addresses specified during the link process. Users can manage these addresses using linker script files. While load command is being executed, gdb sets PC to the entry point of the application. If you want to start execution from another point or just the real entry point is different from the one set by gdb, you can manually set PC to a desired location like this:

```
(gdb) set $pc=0x200040
```

If you want to make sure that your application starts with all interrupts disabled, you can do this:

```
(gdb) set $cpsr=0xD3
```

If your application utilizes stack and the startup code does not initialize the stack pointer you can do this manually like this:

```
(gdb) set $sp=0x201000
```

Now your application is ready to be debugged:

```
gdb) continue    ; start the application
```

or

```
(gdb) si         ; make single step
```

When finished debugging, you can leave gdb/insight in two ways - with or without resetting the target. To exit resetting the target type:

```
(gdb) quit
```

Otherwise type:

```
(gdb) detach
(gdb) quit
```

To make your life easier you may define various commands in a gdb init file and tell gdb to load that file when starting like this:

```
$ arm-elf-insight -command=my_gdb_init
```

Assuming that PEEDI has IP 192.168.1.10, my_gdb_init file may contain something like this:

```
# this will tell gdb to connect to PEEDI using remote protocol
target remote 192.168.1.10:2000
info target

# the following will define a user command
define ll
     set $cpsr=0xD3
     load
end
```

4.13 Debugging Linux kernel

To debug the kernel you bootloader must be set to load and start the kernel successfully.

In the target configuration file set the COREn_STARTUP_MODE to RESET and in the INIT section add this for all targets except Xscale:

```
break add hard 0x90000398    ; addr from 'nm vmlinux | grep start_kernel'
go                           ; start CPU
wait 30000 stop              ; wait 30 seconds to enter debug
break del all                ; remove previously added watchpoint
beep 500 20                  ; beep to signalize ready for debug
```

For Xscale targets use this:

```
break add watch 0xffff001C w 32    ; watch point on setting vectors
go                                 ; start CPU
wait 30000 stop                    ; wait 30 seconds to enter debug
break del all                      ; remove previously added watchpoint
beep 500 20                        ; beep to signalize ready for debug
```

This will set a break/watch in the beginning of the Linux kernel code and will start the kernel. This way after target is powered the kernel will be started and a little later it will enter debug. At this point you can start gdb/insight pointing the kernel ELF image file. Next you can use the target command to connect to PEEDI. Make a si just for the gdb/insight to refresh its source window. Now you can set/remove breakpoints in you source code, step-by-step examine the execution or issue continue to start the kernel after you have set all the breakpoints you desire. If a break point is hit, gdb/insight will highlight the source line where the execution has stopped.

4.14 Target OS thread awareness

PEEDI provides target OS thread awareness for systems that support Context Switching. Such system is eCos for example. In these systems the Process Context is used to store information to be able to stop and re-start the process later. Data structures in the form of Process Control Blocks are used to save the CPU state to perform a process switch. When debugging with GDB, the info threads command provides information for existing threads. PEEDI can be configured to display the existing threads in the project. Before using info threads in the GDB command window, you must first set a section in the target configuration file that tells PEEDI how to find the tasks. This section includes addresses and offsets needed to be filled in order for PEEDI to be able to scan the OS task list. To obtain the correct OS information copy the configuration from the following link to the .gdbinit file of your project.

http://download.ronetix.at/peedi/doc/os_scripts

Keep in mind how the OS thread support in PEEDI works:

- after every CPU halt PEEDI loops all threads using the NEXT pointer.

A loop is successful only if the last NEXT pointer points to the start pointer. In this case PEEDI reports to gdb all found threads.

When looping, if there is a NULL pointer or garbage read, then PEEDI will report only one thread - the current execution.

To prevent garbage reading from uninitialized memory (for example after the first "si" command) it is recommended gdbinit file to clear the first NEXT pointer.

Something like this:

```
set Cyg_Thread::thread_list->next=0
```

Start your project with GDB and observe the console. The script prints information in the form of a configuration file OS section. Use the script only once to obtain the section. For example to print the eCOS section enter ecos in the GDB console.

See an example GDB command window below:

```
source .gdbinit
0xfffffffc in ()
Loading section .rom_vectors, size 0x40 lma 0x600000
Loading section .text, size 0x147a64 lma 0x600040
Loading section .rodata, size 0x5dbb8 lma 0x747aa4
Loading section .data, size 0x6f10 lma 0x7a565c
Start address 0x600040, load size 1754476
Transfer rate: 523 KB\sec, 15949 bytes\write
ecos
[OS_ECOS]
ITEM = int32_abs, BASE, Cyg_Scheduler_Base::current_thread, 0
ITEM = int32, NEXT, 0xA4
ITEM = int32, PID, 0x4C
ITEM = str32, NAME, 0xA0, 0
ITEM = reg32, R0, 0xC, 0xC
ITEM = reg32, R1, 0xC, 0x10
ITEM = reg32, R2, 0xC, 0x14
ITEM = reg32, R3, 0xC, 0x18
ITEM = reg32, R4, 0xC, 0x1C
ITEM = reg32, R5, 0xC, 0x20
ITEM = reg32, R6, 0xC, 0x24
ITEM = reg32, R7, 0xC, 0x28
ITEM = reg32, R8, 0xC, 0x2C
ITEM = reg32, R9, 0xC, 0x30
ITEM = reg32, R10,0xC, 0x34
ITEM = reg32, R11,0xC, 0x38
ITEM = reg32, R12,0xC, 0x3C
ITEM = reg32, sp, 0xC, 0x8
ITEM = reg32, lr, 0xC, 0x40
ITEM = reg32, pc, 0xC, 0x40
ITEM = reg32_abs, xpsr, 0x20010000
```

To enable the OS thread awareness in PEEDI first add a COREn_OS CFG parameter and set it to point to the OS section like this:

```
CORE_OS = OS_ECOS      ; section which contains the OS parameters
```

The following are example configuration files for eCos system.

```
[OS_ECOS]
ITEM = int32_abs, BASE, Cyg_Scheduler_Base::current_thread, 0
ITEM = int32, NEXT, 0xA4
ITEM = int32, PID, 0x4C
ITEM = str32, NAME, 0xA0, 0
ITEM = reg32, R0, 0xC, 0xC
ITEM = reg32, R1, 0xC, 0x10
ITEM = reg32, R2, 0xC, 0x14
ITEM = reg32, R3, 0xC, 0x18
ITEM = reg32, R4, 0xC, 0x1C
ITEM = reg32, R5, 0xC, 0x20
ITEM = reg32, R6, 0xC, 0x24
ITEM = reg32, R7, 0xC, 0x28
ITEM = reg32, R8, 0xC, 0x2C
ITEM = reg32, R9, 0xC, 0x30
ITEM = reg32, R10,0xC, 0x34
ITEM = reg32, R11,0xC, 0x38
ITEM = reg32, R12,0xC, 0x3C
ITEM = reg32, sp, 0xC, 0x8
ITEM = reg32, lr, 0xC, 0x40
ITEM = reg32, pc, 0xC, 0x40
ITEM = reg32_abs, xpsr, 0x20010000
```

4.15 Working with CLI (Command Line Interface)

PEEDI CLI allows you to:

- **Perform simple debugging**

You can load executable image into target RAM, get or set target memory or registers, put break and watch points, start, step or stop the target. For more information, see the description of **core, go, breakpoint, step, halt, reset, info, memory** commands.

- **Program target flash**

Full functional FLASH programmer is available, capable of programming different image file formats. For more information, see the description of **flash** command.

- **Manage files from various sources**

While in CLI, you can copy files from and to local EEPROM and SD/MMC card or FTP, TFTP or HTTP servers. You can create, remove and rename directories and files on the MMC/SD card. For more information, see the description of **transfer, card and eeprom** commands.

4.15.1 File path convention

PEEDI can get files from local EEPROM and MMC/SD card or TFTP, FTP and HTTP server. It can store files on all the previous locations except HTTP server. However the download speed from HTTP and FTP servers is times faster than TFTP servers. FAT12, FAT16 and FA32 formatted MMC/SD cards are supported but there is no support for long file names, so all files should be named using the 8+3 DOS name convention or using names up to twelve characters.

This file path syntax can be used to point the desired location:



Note:
If the file path is skipped the per-core default path will be used. The default core's path is defined in the target configuration file using the COREn_PATH parameter. Full path will be used in the entire manual for clear understanding.



Note:
If the IP address is skipped the default server IP will be used. The default server IP can be set using the fconfig RedBoot command. Full path will be used in the entire manual for clear understanding.

TFTP server 192.168.1.1 will be requested for /subdirectory/file:

```
tftp://192.168.1.1/subdirectory/file
```

TFTP default server IP will be requested for /subdirectory/file

```
tftp:/subdirectory/file
```

FTP server 192.168.1.1 will be requested for subdirectory/file from the current working directory right after the login of user with password:

```
ftp://user:password@192.168.1.1/subdirectory/file
```

FTP server 192.168.1.1 will be requested for subdirectory/file from server root directory using user and password credentials to login:

```
ftp://user:password@192.168.1.1//subdirectory/file
```

FTP server 192.168.1.1 will be requested for subdirectory/file using user **anonymous** and password **guest** to login:

```
ftp://192.168.1.1/subdirectory/file
```

FTP default server IP will be requested for subdirectory/file from the current working directory right after the login of user with password:

```
ftp:user:password/subdirectory/file
```

FTP default server IP will be requested for subdirectory/file from server root directory using user and password credentials to login:

```
ftp:user:password//subdirectory/file
```

FTP default server IP will be requested for subdirectory/file from server root directory using user and password credentials to login:

```
ftp:subdirectory/file
```

FTP default server IP will be requested for subdirectory/file from server root directory right after the login of user **anonymous** with password **guest**:

```
ftp:/subdirectory/file
```

HTTP server 192.168.1.1 will be requested for /subdirectory/file:

```
http://192.168.1.1/subdirectory/file
```

HTTP server 192.168.1.1 at port 8080 will be requested for /subdirectory/file:

```
http://192.168.1.1:8080/subdirectory/file
```

HTTP default server IP will be requested for /subdirectory/file:

```
http:/subdirectory/file
```

Local EEPROM will be searched for file. EEPROM file system if flat, i.e. directories are not supported. Keep in mind it has very limited storage space (tenths of kilobytes):

```
eep://file
```

The default path to the file will be used, got from the COREn_PATH configuration parameter.

```
file
```

4.15.2 CLI commands

PEEDI has full functional telnet command line interface (CLI), which provides many useful commands. It has vary easy to use help system and command auto complete, so instead of **flash program** you could type only **fl pr** , or you could just hit TAB to auto complete the command or sub-command. If you are unsure about some command arguments - hit TAB again and the command help will be printed so you can continue writing your command line.

An expression can also be used instead of a value in a command. The expression can include only the four main math operations: +, -, * and /. And it is interpreted without priority from left to right (e.g. **memory read 4*0x1000-32 32**).

help

Syntax:

```
help [COMMAND [SUBCOMMAND]]
```

Description:

Shows help about command or a sub-command.

Argument:

COMMAND	- command which help will be shown
COMMAND SUBCOMMAD	- sub-command which help will be shown

Example:

```
help
help halt
help flash program
```

transfer

Syntax:

```
transfer SOURCE DESTINATION
```

Description:

Copy file among TFTP, FTP, HTTP, MMC/SD and EEPROM.

Argument:

SOURCE	- the source file to be copied
DESTINATION	- where the file to be saved

Example:

```
; copy file from the mmc/sd card to a TFTP server
transfer card://dump.bin tftp://192.168.1.1/dump.bin

; copy file from the EEPROM card to a FTP server
transfer dump.bin ftp://user:pass@192.168.1.1/dump.bin

; copy file from a HTTP server to a TFTP server
transfer http://192.168.1.1/dump.bin tftp://192.168.1.1/dump
```

type

Syntax:

```
type FILE
```

Description:

Show content of text file.

Argument:

FILE - text file to be shown

Example:

```
type ftp://myuser:mypass@192.168.1.1/target.cfg
```

wait

Syntax:

```
wait MILLISECONDS [stop]
```

Description:

Wait specified time period or wait target to stop with a given timeout. Useful when target needs some delay while executing commands in INIT section of the target configuration or script file.

Argument:

MILLISECONDS - period to be waited in milliseconds. Actual resolution is 10ms

Example:

```
wait 1000
wait 5000 stop
```

core

Syntax:

```
core [#CORE]
```

Description:

Show/set current core.

Argument:

#CORE - core number of desired core to be current

Example:

```
core
core #1
```

clock

Syntax:

```
clock init|normal|kHz
```

Description:

Switch JTAG/BDM target clock - init, normal or any desired frequency. This is useful when the INIT section is too long and takes too much time. Using this command, you can initialize in the beginning the system clock (the PLL) and then switch to normal clock. This will allow much faster execution of the INIT section.

Argument:

init, normal or frequency in kHz - JTAG/BDM clock

Example:

```
core
core #1
```

run

Syntax:

```
run #SCRIPT_NUMBER|$SCRIPT_NAME|SCRIPT_FILE
```

Description:

Execute script from the target configuration file or a file containing CLI commands. If

$SCRIP_NAME or SCRIPT_FILE is with extension **.tcl**, then executed as TCL.

Argument:

#SCRIPT_NUMBER	- number associated with a scrip defined in the configuration file
$SCRIPT_NAME	- name of a scrip defined in the configuration file
SCRIPT_FILE	- file, containing CLI commands to be executed

Example:

```
run $script
run $demo.tcl
run #1
run card://myscript.cmd
```

tcl

Syntax:

```
tcl TCL_STATEMENT
```

Description:

Execute a TCL statement. Every TCL command uses a new environment. A statement can consists of many TCL commands separated be a semicolon ‘;’.

Argument:

STATEMENT	TCL statement

Example:

```
tcl “set x 0x1234; puts $x” ;# print 0x1234

tcl “set x 0x1234”
tcl “puts $x”    ;# print 0 because new environment is used
```

go

Syntax:

```
go [ADDRESS|#CORE|#CORE=ADDRESS|#all]
```

Description:

Start current or specified core(s). If no address is provided the core(s) will start from its current program counter (PC) value. If no core is specified, current core will be started. If argument #all is provided, all cores will be started from their current PC values.

Argument:

ADDRESS	- address to start from
#CORE	- core to be started
#all	- all cores will be started

Example:

```
go 0x100040
go #0
go #0=100040
go #0=100040 #2
go #all
```

gm

Syntax:

```
gm ADDRESS
```

Description:

This command is applicable only for CORTEX-M cores.

Set SP and PC and start CORTEX-M core:

SP = [ADDRESS]

PC = [ADDRESS + 4]

Argument:

ADDRESS	- address to start from
#CORE	- core to be started
#all	- all cores will be started

Example:

```
gm 0x20400000
```

step

Syntax:

```
step [ADDRESS|#CORE|#CORE=ADDRESS|#all]
```

Description:

Step one instruction current or specified core(s). If no address is provided the core(s) will steps from its current PC value. If no core is specified, current core will be stepped. If argument #all is provided, all cores will be stepped from their current PC values.

Argument:

ADDRESS	- address to step from
#CORE	- core to be stepped
#all	- all cores will be stepped

Example:

```
step 0x100040
step #0
step #0=100040
step #0=100040 #2
step #all
```

execute

Syntax:

```
execute OPCODE
```

Description:

Force CPU to execute specified instruction. Supported in MPC5500 targets only.

Argument:

OPCODE - opcode of instruction to be executed

Example:

```
execute 0x7C0007A4
```

set

Syntax:

```
set [coprocessor|spr|ctrl|cp0|tlb] REGISTER VALUE
```

Description:

Set target CPU register. For more information about cp15 see info cp15 command.

Argument:

REGISTER - name of register to set

VALUE - value to set

Example:

```
set r0 0x12345678       ; set general purpose register
set ice8 0x12345678     ; set ICE register 8
set dfsr 0x12345678           ; ARM9: set CP15 instr. Data FSR
register using interpreted access
set cp15 0x51AF 0x123   ; ARM9: set CP15 instr. TTB register using
interpreted access (bit12=1)
set cp15 0x000D 0x678   ; ARM9: set CP15 Process ID register using
physical access (bit12=0)
set MAS0 0x1234               ; PowerPC: set spr register by name
set spr 624 0x1234            ; PowerPC: set spr register by number
set RAMBAR 0x0                ; ColdFire: set control register by name
set ctrl 0xC05 0x0            ; ColdFire: set control register by
address
set cp0 8 0x0                 ; MIPS: set control register by number
set tlb word0 word1 word2     ; PPC4XX: set MMU TLB entry, the first
command used clears all TLB entries
```

halt

Syntax:

```
halt [#CORE|#all]
```

Description:

Stop current or specified core(s). If no core is specified, current will be stopped.

Argument:

#CORE	- core to be stop
#all	- all cores will be stopped

Example:

```
halt
halt #0
halt #all
```

reset

Syntax:

```
reset [detect|reset|run|stop [MILLISECONDS]]
```

Description:

Hardware reset all core on the JTAG chain causing re-initialization of each core.

If no arguments are provided last used will be taken or the reset will be performed considering the CORE_STARTUP_MODE configuration parameter.

Disable/enable target reset detection.

Argument:

reset	- reset and stop the target immediately
run	- reset and leave the target running
stop MILLISECONDS	- reset and stop the target after specified time
detect 0	- disable the target reset detection
detect 1	- enable the target reset detection

Example:

```
reset
reset run
reset stop 1000
```

```
reset detect 0
reset detect 1
```

reboot

Syntax:

```
reboot [redboot|watchdog]
```

Description:

Reboot PEEDI and reload the target configuration file and re-initialize all cores.

Argument:

redboot	- Reboot and enter RedBoot command line.
watchdog	- Enable PEEDI internal watchdog

Example:

```
reboot
reboot redboot
reboot watchdog
```

echo

Syntax:

```
echo TEXT
```

Description:

Display a line of text. Useful for printing info in scripts.

Argument:

redboot	- Reboot and enter RedBoot command line.
watchdog	- Enable PEEDI internal watchdog

Example:

```
echo Initializing SDRAM...
```

jtag

Syntax:

```
jtag
```

Description:

Type jtag help in PEEDI command line for more information

beep

Syntax:

```
beep FREQUENCY DURATION
```

Description:

Beep using given frequency and duration. Useful for signaling end of scripts execution.

Argument:

FREQUENCY	- Frequency in Hz
DURATION	- duration in milliseconds

Example:

```
beep 1000 500
```

target

Syntax:

```
target [detach|attach]
```

Description:

Set PEEDI debug interface in High-Z state.

Argument:

attach

detach

Example:

```
target        ; show current interface state
target detach ; set interface in High-Z
target attach ; set interface to normal mode
```

quit

Syntax:

```
quit
```

Description:

Quit telnet session.

Argument:

Example:

```
quit
```

info

Syntax:

```
info SUBCOMMAND
```

Description:

Show information about specified topic.

Argument:

SUBCOMMAND - sub-command specifying the needed information.

Example:

```
info config
```

Info flash

Syntax:

```
info flash
```

Description:

Show target FLASH configuration information.

Argument:

Example:

```
info flash
```

info registers

Syntax:

```
info registers [#CORE|#all] [all]
```

Description:

Show current CPU registers' values of current, specified all cores.

Argument:

#CORE	- core to show its registers' values
#all	- list all core registers' values
all	- list all modes registers' values

Example:

```
info registers
info registers all
info registers #0
info registers #0 all
info registers #all
```

```
info registers #all all
```

info target

Syntax:

```
info target [#CORE]
```

Description:

Show general core information.

Argument:

Example:

```
info target
info target #0
```

info config

Syntax:

```
info config
```

Description:

Show configuration.

Argument:

Example:

```
info config
```

info ice

Syntax:

```
info ice [REGISTER] [#CORE|#all]
```

Description:

Display ARM ICE Breaker registers.

Argument:

Example:

```
info ice
info ice ice5
info ice 5
```

info cp15, info cp14

Syntax:

```
info cp15 [0xXXXX] [#CORE|#all]
info cp14 [0xXXXX] [#CORE|#all]
```

Description:

List current CP15 registers' values.
The ARM9 control coprocessor, cp15, provides additional registers that are used to configure and control the caches, MMU, protection system, the clocking mode and other system options.

Via JTAG, CP15 registers are accessed either direct (physical access mode) or via interpreted MCR/MRC instructions.

ARM920: Physical and Interpreted access mapping to CP15 registers Register number for physical access mode (bit 12 = 0):

15 13	12	11 9	8	7 5	4	3 0
0 0 0	0	0 0	i	0 0 0	x	CRn

The bit "i" selects the instruction cache (scan chain bit 33)

The bit "x" extends access to register 15 (scan chain bit 38)

Register number for interpreted access mode (bit 12 = 1):

15 13	12	11 8	7 5	4	3 0
opc_2	1	CRm	opc_2	x	CRn

The 16-bit register number is used to build the appropriate MCR/MRC instruction.

- CRm - Specified Coprocessor Action. Determines specific coprocessor action. Its value is dependent on the CP15 register used. For details, refer to CP15 specific register behavior.
- CRn - Determines the destination coprocessor register.
- opc_1 - Defines the coprocessor specific code. Value is c15 for CP15.
- opc_2 - Determines specific coprocessor operation code. By default, set to 0.

ARM926: Physical access mapping to CP15 registers:

13 11	10 8	7 4	3 0
opc_1	opc_2	CRn	CRm

ARM94x: Physical access mapping to CP15 registers:

5	4 1	0
x	CRn	i

The bit "i" selects the instruction cache (scan chain bit 32).

The bit "x" extends access to register 6 (scan chain bit 37).

Argument:

Example:

```
info cp15 - show all CP15 registers

info cp15 0x51AF - ATM920: show inst. TTB register using interpreted
access (bit12=1)

info cp15 0x0109 - ARM920: show inst. cache lockdown register using
physical access (bit12=0)

info cp15 ittb
```

info spr

Syntax:

```
info spr [NAME|NUMBER] [#CORE|#all]
```

Description:

List current SPR registers' values. PowerPC targets only.

Argument:

Example:

```
info spr
info spr PID
info spr 48
```

info ctrl

Syntax:

```
info ctrl [NAME|ADDRESS]
```

Description:

List current control registers' values. ColdFire targets only.

Argument:

Example:

```
info ctrl
info ctrl RAMBAR
info ctrl 0xC05
```

info breakpoint

Syntax:

```
info breakpoint [#CORE]
```

Description:

List all set break and watch points of current or a specified core.

Argument:

#CORE - core's break and watch points to be listed

Example:

```
info breakpoint
info breakpoint #1
```

memory

Syntax:

```
memory SUBCOMMAND
```

Description:

Manage target memory. Sub-command must be provided.

Argument:

SUBCOMMAND - sub-command specifying the memory operation.

Example:

```
memory read
```

memory read

Syntax:

```
memory read[TYPE ADDRESS [COUNT]]
```

Description:

Read and show target memory contents. If no arguments are provided last used will be taken, ignoring ADDRESS and starting the listing with the next address to be listed after the previous execution of memory read. Default first used arguments are 8 32-bit values at address 0x00000000.

Argument:

TYPE	- memory access • 8 - value is 8-bits (byte long) • 16 - value is 16-bits (half word long) • 32 - value is 32-bits (word long) • 64 - value is 64-bits (double word long) • $ - value is string
ADDRESS	- where the value resides
COUNT	- how many consecutive values to be listed, if not provided count 1 is assumed

Example:

```
memory read 0x1000
memory read8 0x1000
memory read16 0x1000
memory read32 0x1000
memory read64 0x1000
memory read$ 0x1000 8
memory read
```

memory write

Syntax:

```
memory write[TYPE ADDRESS VALUE [COUNT]]
```

Description:

Write target memory with specified value. If no arguments are provided last used will be taken.

Argument:

TYPE	- memory access • 8 - value is 8-bits (byte long) • 16 - value is 16-bits (half word long) • 32 - value is 32-bits (word long) • 64 - value is 64-bits (double word long) • $ - value is string
ADDRESS	- where the value is to be written
COUNT	- how many consecutive values to be written,, if not provided

count 1 is assumed

Example:

```
memory write 0x1000 0x5555AAAA
memory write8 0x1000 0x5A
memory write16 0x1000 0x55AA
memory write32 0x1000 0x5555AAAA
memory write64 0x1000 0x1234567811223344
memory write$ 0x1000 „hi there“
memory write write 0x1000 0x5555AAAA
```

memory or

Syntax:

```
memory or[TYPE] ADDRESS MASK
```

Description:

Make logical OR with, and apply to target memory.

Argument:

TYPE	- memory access • 8 - value is 8-bits (byte long) • 16 - value is 16-bits (half word long) • 32 - value is 32-bits (word long) • 64 - value is 64-bits (double word long) • $ - value is string
ADDRESS	- where the value is to be written
COUNT	- how many consecutive values to be written,, if not provided count 1 is assumed

Example:

```
memory or 0x1000 0x5555AAAA
memory or8 0x1000 0x5A
memory or16 0x1000 0x55AA
memory or32 0x1000 0x5555AAAA
memory or64 0x1000 0x1234567811223344
```

memory and

Syntax:

```
memory and[TYPE] ADDRESS MASK
```

Description:

Make logical AND with, and apply to target memory.

Argument:

TYPE	- memory access • 8 - value is 8-bits (byte long) • 16 - value is 16-bits (half word long) • 32 - value is 32-bits (word long) • 64 - value is 64-bits (double word long) • $ - value is string
ADDRESS	- where the value is to be written
COUNT	- how many consecutive values to be written,, if not provided count 1 is assumed

Example:

```
memory and 0x1000 0x5555AAAA
memory and8 0x1000 0x5A
memory and16 0x1000 0x55AA
memory and32 0x1000 0x5555AAAA
memory and64 0x1000 0x1234567811223344
```

memory crc

Syntax:

```
memory crc ADDRESS LENGTH [CRC]
```

Description:

Calculate or check CRC32 on a given memory region.

Argument:

ADDRESS	- beginning of region

LENGTH	- length of region
CRC	- crc to check

Example:

```
memory crc 0x100000 1024
memory crc 0x100000 1024 0x1DF37A8C
```

memory load

Syntax:

```
memory load [FILE [FORMAT [OFFSET]]]
```

Description:

Load image file into target memory. If no arguments are provided last used will be taken. Default first used arguments are taken from COREn_FILE of target configuration file. While file is loaded PC will be set at start of the image or at entry point if provided by the image file.

Argument:

FILE	- the image file to be loaded
FORMAT	- format of image file: • bin - binary file • ihex - Intel HEX format • srec - Motorola S-record format • elf - ELF format
OFFSET	- Must be provided for binary files because they don't have any address information. If provided with ihex, srec or elf formats, all the code will be shifted regarding the specified offset.

Example:

```
memory load tftp://192.168.1.1/image.bin bin 0x1000
memory load tftp://192.168.1.1/image.elf elf
```

memory multi load

Syntax:

```
memory multi load #CORE0 ... #COREn FILE FORMAT [OFFSET]
```

Description:

Load image file into several targets simultaneously. If no arguments are provided last used will be taken. Default first used arguments are taken from COREn_FILE of target configuration file. While file is loaded PC will be set at start of the image or at entry point if provided by the image file.

Argument:

#CORE0 .. #COREn or #all	- cores to load the file to
FILE	- the image file to be loaded
FORMAT	- format of image file: • bin - binary file • ihex - Intel HEX format • srec - Motorola S-record format • elf - ELF format
OFFSET	- Must be provided for binary files because they don't have any address information. If provided with ihex, srec or elf formats, all the code will be shifted regarding the specified offset.

Example:

```
memory multi load #all tftp://192.168.1.1/image.bin bin 0x1000
memory multi load #0 #2 tftp://192.168.1.1/image.elf elf
```

memory verify

Syntax:

```
memory verify [FILE [FORMAT [OFFSET]]]
```

Description:

Verify target RAM with image file. If no arguments are provided last used with the memory load command will be taken.

Argument:

FILE	- the image file to be verified
FORMAT	- format of image file: • bin - binary file • ihex - Intel HEX format • srec - Motorola S-record format • elf - ELF format
OFFSET	- Must be provided for binary files because they don't have any address information. If provided with ihex, srec or elf formats, all the code will be shifted regarding the specified offset.

Example:

```
flash verify tftp://192.168.1.1/image.elf elf
flash verify tftp://192.168.1.1/image.bin bin 0x1000
```

memory dump

Syntax:

```
memory dump ADDRESS LENGTH FILE
```

Description:

Dump target memory to a file. If no arguments are provided last used will be taken.

Argument:

ADDRESS	- beginning of memory region
LENGTH	- length of memory region
FILE	- file to store the image. All path except HTTP server are accepted

Example:

```
memory dump 0 1024 tftp://192.168.1.1/ram.bin
```

memory management

Syntax:

```
memory management tlbr
memory management tlbw
memory management tlbc
```

Description:

tlbr – read TLB entries

tlbw – write TLB entries

tlbc – clear TLB entries

Example:

```
Example PPC440:
  memory management tlbr START_TLB_INDEX END_TLB_INDEX
  memory management tlbw TLB_INDEX WS0_WORD WS1_WORD WS2_WORD
  memory management tlbw 0 0xFF000270 0xFF000004 0x0000053F
  memory management tlbc 0 63 ; clear all TLB entries

Example T1040:
  memory management tlbr 0 0 127 ; list all TLB0 entries
  memory management tlbr 1 0 127 ; list all TLB1 entries
  memory management tlbw 0 TLB_INDEX MAS1 MAS2 MAS3 MAS7 ; set a TLB0
entry
  memory management tlbw 1 TLB_INDEX MAS1 MAS2 MAS3 MAS7 ; set a TLB1
entry

  memory management tlbc 0 0 127 ; clear all TLB0 entries
  memory management tlbc 1 0 127 ; clear all TLB1 entries
```

memory test

Syntax:

```
memory test ADDRESS LENGTH | [STEP COUNT]
```

Description:

Test target RAM region.
Test 32-bytes target RAM with given ’step’ and ’count’

Argument:

ADDRESS	- beginning of memory region
LENGTH	- length of memory region
STEP	- incrementing address step
COUNT	- number of incrementing steps

Example:

```
memory test 0x100000 1024

; Test first 32 bytes of every 8KB from 512MB total memory
memory test 0x20000000 8192 1024*1024*512/8
```

flash

Syntax:

```
flash SUBCOMMAND
```

Description:

Manage target FLASH. Subcommand must be provided.

Argument:

SUBCOMMAND	- subcommand specifying the FLASH operation

Example:

```
flash erase
```

flash set

Syntax:

```
flash set [FLASHn]
```

Description:

Show/set current FLASH target section.

Argument:

FLASHn	- FLASH section number desired to be current

Example:

```
flash set
flash set 1
```

flash blank

Syntax:

```
flash blank ADDRESS [LENGTH]
```

Description:

Check FLASH region if it is blank, i.e. filled with 0xFF. If no arguments are provided last used will be taken. Default first used region is whole FLASH.

Argument:

ADDRESS	- beginning of FLASH region
LENGTH	- length of FLASH region, default is 1, if not supplied

Example:

```
flash blank 0x400000 0x1000
```

flash erase

Syntax:

```
flash erase ADDRESS [LENGTH]
flash erase chip
```

Description:

Erase all FLASH sectors that belong or overlap to the specified region. If no arguments are provided last used will be taken. Default first used region is whole FLASH.

flash erase chip - erase using the CHIP ERASE command if supported from the flash device.

Argument:

ADDRESS	- beginning of FLASH region

LENGTH	- length of FLASH region, default is 1, if not supplied

Example:

```
flash erase 0x400000 0x1000
flash erase chip
```

flash lock

Syntax:

```
flash lock ADDRESS [LENGTH]
```

Description:

If supported by FLASH, lock (protect against write/erase) all FLASH sectors that belong or overlap to the specified region. If no arguments are provided last used will be taken. Default first used region is whole FLASH.

Argument:

ADDRESS	- beginning of FLASH region
LENGTH	- length of FLASH region, default is 1, if not supplied

Example:

```
flash lock 0x400000 0x1000
```

flash unlock

Syntax:

```
flash unlock ADDRESS [LENGTH]
```

Description:

If supported by FLASH, unlock (unprotect against write/erase) all FLASH sectors that belong or overlap to the specified region. If no arguments are provided last used will be taken. Default first used region is whole FLASH.

Argument:

ADDRESS	- beginning of FLASH region

LENGTH	- length of FLASH region, default is 1, if not supplied

Example:

```
flash unlock 0x400000 0x1000
```

flash query

Syntax:

```
flash query ADDRESS [LENGTH]
```

Description:

If supported by FLASH, show the lock status of all FLASH sectors that belong or overlap to the specified region. On NAND devices, show the bad block list. If no arguments are provided last used will be taken. Default first used region is whole FLASH.

Argument:

ADDRESS	- beginning of FLASH region
LENGTH	- length of FLASH region, default is 1, if not supplied

Example:

```
flash query 0x400000 0x1000
```

flash program

Syntax:

```
flash program [FILE [FORMAT [OFFSET] [erase]]]
```

Description:

Program image file into target FLASH. If no arguments are provided last used will be taken. Default first used arguments are taken from FILE parameter of the currently selected FLASH section in target configuration file.

Argument:

FILE	- the image file to be programmed
FORMAT	- format of image file: • bin - binary file • ihex - Intel HEX format • srec - Motorola S-record format • elf - ELF format
OFFSET	- Must be provided for binary files because they don't have any address information. If provided with ihex, srec or elf formats, all the code will be shifted regarding the specified offset
erase	- if this argument is provided, all affected FLASH sectors will be pre-erased upon programming

Example:

```
flash program tftp://192.168.1.1/image.elf elf erase
flash program tftp://192.168.1.1/image.elf elf 0x1000
flash program tftp://192.168.1.1/image.bin bin 0x1000
```

flash multi erase

Syntax:

```
flash multi erase #CORE0 ... #COREn [ADDRESS LENGTH|chip]
```

Description:

Erase all FLASH sectors that belong or overlap to the specified region on into several targets simultaneously. If no arguments are provided last used will be taken. Default first used region is whole FLASH.

Argument:

#CORE0 .. #COREn or #all	- cores to erase
ADDRESS	- beginning of FLASH region
LENGTH	- length of FLASH region, default is 1, if not supplied

Example:

```
flash multi erase #all 0x400000 0x1000
flash multi erase #1 #2 0x400000 0x1000
flash multi erase #all chip
```

flash multi blank

Syntax:

```
flash multi blank #CORE0 ... #COREn [ADDRESS LENGTH]
```

Description:

Check if blank all FLASH sectors that belong or overlap to the specified region on into several targets simultaneously. If no arguments are provided last used will be taken. Default first used region is whole FLASH.

Argument:

#CORE0 .. #COREn or #all	- cores to check
ADDRESS	- beginning of FLASH region
LENGTH	- length of FLASH region, default is 1, if not supplied

Example:

```
flash multi blank #all 0x400000 0x1000
flash multi blank #1 #2 0x400000 0x1000
```

flash multi program

Syntax:

```
flash multi program #CORE0 .. #COREn FILE [ADDRESS LENGTH]
```

Description:

Program image file into several targets simultaneously. If no arguments are provided last used will be taken. Default first used arguments are taken from FILE parameter of

the currently selected FLASH section in target configuration file.

Argument:

#CORE0 .. #COREn or #all	- cores to program
FILE	- the image file to be programmed
FORMAT	- format of image file: • bin - binary file • ihex - Intel HEX format • srec - Motorola S-record format • elf - ELF format
OFFSET	- Must be provided for binary files because they don't have any address information. If provided with ihex, srec or elf formats, all the code will be shifted regarding the specified offset

Example:

```
flash multi program #0 #2 tftp://192.168.1.1/image.elf elf
flash multi program #all ftp://192.168.1.1/imge.bin bin 0x100
```

flash verify

Syntax:

```
flash verify #CORE0 .. #COREn FILE [ADDRESS LENGTH]
```

Description:

Verify target FLASH with image file. If no arguments are provided last used with the flash program command will be taken.

Argument:

FILE	- the image file to be verified
FORMAT	- format of image file:

- bin - binary file
- ihex - Intel HEX format
- srec - Motorola S-record format
- elf - ELF format

OFFSET - Must be provided for binary files because they don't have any address information. If provided with ihex, srec or elf formats, all the code will be shifted regarding the specified offset

Example:

```
flash verify tftp://192.168.1.1/image.elf elf
flash verify tftp://192.168.1.1/image.bin bin 0x1000
```

flash multi verify

Syntax:

```
flash multi verify #CORE0 .. #COREn FILE [ADDRESS LENGTH]
```

Description:

Check image file onto several targets simultaneously. If no arguments are provided last used will be taken. Default first used arguments are taken from FILE parameter of the currently selected FLASH section in target configuration file.

Argument:

#CORE0 .. #COREn or #all - cores to verify

FILE - the image file to be programmed

FORMAT - format of image file:

- bin - binary file
- ihex - Intel HEX format
- srec - Motorola S-record format
- elf - ELF format

OFFSET	- Must be provided for binary files because they don't have any address information. If provided with ihex, srec or elf formats, all the code will be shifted regarding the specified offset

Example:

```
flash multi verify #all tftp://192.168.1.1/image.elf elf
flash multi verify #0 #2 ftp://192.168.1.1/image.bin bin 0x100
```

flash dump

Syntax:

```
flash dump ADDRESS LENGTH FILE
```

Description:

Dump target FLASH to a file. If no arguments are provided last used will be taken.

Argument:

ADDRESS	- beginning of memory region
LENGTH	- length of memory region
FILE	- file to store the image. All path except HTTP server are accepted

Example:

```
flash dump 0 1024 tftp://192.168.1.1/ram.bin
```

flash read

Syntax:

```
flash read [ADDRESS [COUNT]]
```

Description:

Read and show FLASH memory contents. Useful for NAND, SPI and DataFlash FLASH types, i.e. chips which are not visible through the CPU memory map.

For NOR chips the **memory read** command may be used.

Argument:

ADDRESS	- start address
COUNT	- count in bytes

Example:

```
flash read 0x1000 8
flash read
```

flash info

Syntax:

```
flash info
```

Description:

Show target FLASH configuration information.

Argument:

Example:

```
flash info
```

flash find

Syntax:

```
flash find [SEARCHCRITERIA]
```

Description:

List specified or all chips in FLASH data base.

Keep in mind that the following automatically detected devices are not listed:

- CFI compliant NOR Flash devices
- NAND Flash devices

- eMMC devices

Argument:

Example:

```
flash find
flash find AT49BV160
flash find *29F*
```

flash test

Syntax:

```
flash test ADDR LENGTH
```

Description:

On NAND Flash devices: the whole NAND Flash is erased and then the given region is programmed and verified the with two patterns. At the end the whole device is erased. The already existing bad blocks will be skipped, but the tested area will be not expanded. The new detected bad blocks are listed, but not marked.

On all other flash devices: the given region is erased, programmed with random values and verified. At the end the region is not erased.

Argument:

ADDRESS	- start address of the region to be tested
COUNT	- length in bytes including the spare bytes

Example:

```
flash test 0 2112*64*4
```

flash area

Syntax:

```
flash area add ADDR LENGTH
flash area delete
```

```
flash area list
flash area test [markbad]
```

Description:

Currently implemented only for NAND Flash devices. One or more test regions can be added and then tested at once. The whole NAND Flash is erased and then the given region is programmed and verified the with two patterns. At the end the whole device is erased. The already existing bad blocks will be skipped, but the tested area will be not expanded. The new detected bad blocks can be marked if the argument 'markbad' is applied.

Argument:

ADDRESS	- start address of the region to be tested
COUNT	- length in bytes including the spare bytes

Example:

```
flash area add 0x00000000 2112*64*4
flash area add 0x00010000 2112*64*6
flash area delete
flash area list
flash area test
flash area test markbad
```

flash this

Syntax:

```
flash this SUBCOMMAND
```

Description:

The **flash this** command is used to execute FLASH specific subcommand available only for the given FLASH. See below for the available commands.

Argument:

SUBCOMMAND	- FLASH specific subcommand to be executed

Example:

```
flash this hidden enter|exit
flash this nvmbit BIT VALUE
flash this secure
```

```
flash this option BYTE VALUE
flash this write ADDRESS VALUE1 .. VALUE14
```

flash this hidden

Syntax:

```
flash this hidden enter|exit
```

Description:

Enter/exit hidden ROM mode on some FLASH devices. Once the hidden ROM mode is entered, the flash erase and flash program commands can be used on the hidden FLASH sector.

Argument:

enter	- enter hidden ROM mode
exit	- exit hidden ROM mode

Example:

```
flash this hidden enter
flash this hidden exit
```

flash this markbad

Syntax:

```
flash this markbad NUMBLOCK NUMBLOCK NUMBLOCK . . .
```

Description:

NAND Flash - mark one or more blocks as bad.

Argument:

NUMBLOCK	- number of block to be marked as bad

Example:

```
flash this markbad 5
flash this markbad 13 123 1365
```

flash this nvmbit

Syntax:

```
flash this nvmbit BIT VALUE
```

Description:

Set/clear Atmel AT91SAM7 general purpose NVM bit.

Argument:

BIT	- bit number
VALUE	- 0 to clear or 1 to set the specified bit

Example:

```
flash this nvmbit 2 0
flash this nvmbit 2 1
```

flash this secure

Syntax:

```
flash this secure
```

Description:

Secure AT91SAM7 CPU.

Argument:

Example:

```
flash this secure
```

flash this option

Syntax:

```
flash this option erase
```

```
flash this option BYTE VALUE
```

Description:

Manage ST STM32F1 CPU option bytes.

Argument:

BYTE	- byte number 0-7
VALUE	- value to be written to the option byte

Example:

```
flash this option 0x0FFFAAEC
```

flash this option

Syntax:

```
flash this OPTCR_VALUE
```

Description:

Manage ST STM32F2 CPU option bits.

Argument:

OPTCR_VALUE	- option bits value

Example:

```
flash this option 0x0FFFAAEC
```

flash this write

Syntax:

```
flash this write ADDRESS BYTE1 .. BYTE14
```

Description:

Write 1 to 14 bytes at given EEPROM address.

Argument:

ADDRESS	- address to write bytes to
BYTE1..14	- bytes to be written

Example:

```
flash this write
```

flash this part

Syntax:

```
flash this part VALUE
```

Description:

Used to program eMMC register PARTITION_CONFIG [179].

Argument:

VALUE	- register value

Example:

```
flash this part 0x48 - program partition configuration register with
value 0x48

PARTITION_CONFIG [179]:
bit 6 - BOOT_ACK (R/W/non-volatile)
--- 0x0 - No boot acknowledge sent (default)
--- 0x1 - Boot acknowledge sent during boot operation
bit 5:3 - BOOT_PARTITION_ENABLE (R/W/non-volatile)
--- 0x0 - Device not boot enabled (default)
--- 0x1 - Boot partition 1 enabled for boot
--- 0x2 - Boot partition 2 enabled for boot
--- 0x3 - 0x6 - Reserved
--- 0x7 - User area enabled for boot
bit 2:0 - PARTITION_ACCESS: Automatically set by PEEDI depending on
parameter 'PARTITION'

Use 'flash info' to see the current value of register [179]
```

flash this prot

Syntax:

```
flash this prot SUBCOMMAND
```

Description:

The **flash this prot** command is used to execute FLASH reading or writing of protected registers only for the Intel Strata NOR flash. See below for the available commands.

Argument:

SUBCOMMAND	- FLASH specific subcommand to be executed

Example:

```
flash this prot read addr
flash this prot read addr count
flash this prot prog addr value
flash this prot help
```

flash this prot read

Syntax:

```
flash this prot read [ADDRESS] [COUNT]
```

Description:

The **flash this prot read** command is used to execute FLASH reading of protected registers only for the Intel Strata NOR flash. See below for the available commands.

Argument:

ADDRESS	- Flash address to be read
COUNT	- The number of registers to be read

Example:

```
flash this prot read 0x85 - read one protection register at addr 0x85
flash this prot read 0x80 8 - read 8 protection registers at addr 0x80
```

flash this prot program

Syntax:

```
flash this prot program [ADDRESS] [VALUE]
```

Description:

The **flash this prot program** command is used to execute FLASH writing of protected registers only for the Intel Strata NOR flash. See below for the available commands.

Argument:

ADDRESS	- Flash address to be programmed
COUNT	- The value to be programmed

Example:

```
flash this prot prog 0x85 0xA5 - program one register at address 0x85
```

flash this ppb

Syntax:

```
flash this ppb SUBCOMMAND
```

Description:

The **flash this ppb** command is used to execute FLASH locking and unlocking using PPB (Persistent Protection Block) for Spansion NOR flash devices. See below for the available commands.

Argument:

SUBCOMMAND	- FLASH specific subcommand to be executed

Example:

```
flash this ppb query
flash this ppb unlock
flash this ppb lock addr
flash this ppb lock
```

flash this isc_erase

Syntax:

```
flash this isc_erase [SECTOR_BITMASK]
```

Description:

The **flash this isc_erase** command is used to erase STR9 ISC.

Argument:

SECTOR_BITMASK

Example:

```
flash this isc_erase - ISC full erase
flash this isc_erase 0x3 - ISC erase sector 0 and 1 of bank 0
```

flash this isc_conf_write

Syntax:

```
flash this isc_conf_write <VALUE>
```

Description:

The f**lash this isc_conf_write** command is used to write STR9 ISC.

Argument:

VALUE - value to be written

Example:

```
flash this isc_conf_write 0x0001000000000000 ; set bank 1 as boot
```

flash this isc_conf_read

Syntax:

```
flash this isc_conf_read
```

Description:

The f**lash this isc_conf_read** command is used to read current STR9 ISC.

Argument:

Example:

```
flash this isc_conf_read
```

flash this isc_conf_boot_bank

Syntax:

```
flash this isc_conf_boot_bank <BANK>
```

Description:

The flash this isc_conf_boot_bank command is used to set the STR9 device boot bank .

Argument:

BANK - bank to boot from

Example:

```
flash this isc_boot_bank 0 - set booting from bank 0
flash this isc_boot_bank 1 - set booting from bank 1
```

flash this isc_conf_lock

Syntax:

```
flash this isc_conf_lock
```

Description:

The **flash this isc_conf_lock** command is used to lock the STR9 device.

Argument:

Example:

```
flash this isc_conf_lock
```

breakpoint

Syntax:

```
breakpoint SUBCOMMAND
```

Description:

Manage target break and watch points. Subcommand must be provided.

Argument:

SUBCOMMAND	- subcommand specifying the operation

Example:

```
breakpoint list
```

breakpoint add

Syntax:

```
breakpoint add [hard | watch] ADDRESS [ACCESS TYPE]
```

Description:

Set software break point. Unlimited number of software break points can be set.

If address -1 or 0xFFFFFFFF is specified, only the ARM ICE registers will be set for software breakpoints, but not actual breakpoint will be set. In this case the CPU will break (enter debug) if the breakpoint pattern is met anywhere during the code execution. Suitable to embed breaks in the source of the debugged application.

Argument:

ADDRESS	- address of the breakpoint
hard	- set hardware break point.
watch	- set watch point
ACCESS	- access type to break on:

- r - read access
- w - write access
- rw - any access

TYPE - type of watched value:

- 8 - value is 8-bit (byte)
- 16 - value is 16-bit (half word)
- 32 - value is 32-bit (word)

Example:

```
breakpoint add watch 0x400040 32 r
```

breakpoint list

Syntax:

```
breakpoint list [#CORE]
```

Description:

List all set break and watch points for the current or specified core.

Argument:

#CORE - core's break and watch points to be listed

Example:

```
breakpoint list
breakpoint list #1
```

breakpoint delete

Syntax:

```
breakpoint delete ID|all
```

Description:

Delete break or watch point.

Argument:

ID - id number of break or watch point desired to be removed, taken using breakpoint list command

all - if provided all break and watch points will be deleted

Example:

```
breakpoint delete 7
breakpoint delete all
```

card

Syntax:

```
card SUBCOMMAND
```

Description:

Manage MMC/SC card files. Subcommand must be provided.

Argument:

SUBCOMMAND - subcommand specifying the operation

Example:

```
card dir
```

card cd

Syntax:

```
card cd DIRECTORY
```

Description:

Change current directory

Argument:

DIRECTORY - directory to make current

Example:

```
card cd mydir
```

card rd

Syntax:

```
card rd DIRECTORY
```

Description:

Remove directory. The directory must be empty.

Argument:

DIRECTORY	- directory to be removed

Example:

```
card rd mydir
```

card dir

Syntax:

```
card dir [SEARCHCRITERIA|DIRECTORY]
```

Description:

Displays a list of files and subdirectories in a directory.

Argument:

SEARCHCRITERIA	- string to filter printed output
DIRECTORY	- directory which content to be listed

Example:

```
card dir
card dir *.bin
card dir mydir
```

card copy

Syntax:

```
card copy SOURCE DESTINATION
```

Description:

Copy file.

Argument:

SOURCE	- the source file to be copied
DESTINATION	- file to be saved

Example:

```
card copy image.bin mydir/backup.bin
```

card type

Syntax:

```
card type FILE
```

Description:

Show content of text file.

Argument:

FILE	- text file to be shown

Example:

```
card type target.cfg
```

card delete

Syntax:

```
card delete FILE
```

Description:

Delete file.

Argument:

FILE	- file to be deleted

Example:

```
card delete target.cfg
```

card rename

Syntax:

```
card rename FILE NEWNAME
```

Description:

Rename file.

Argument:

FILE	- file to be renamed
NEWNAME	- new file name

Example:

```
card rename image.bin backup.bin
```

eeprom

Syntax:

```
eeprom SUBCOMMAND
```

Description:

Manage EEPROM files. EEPROM file system if flat, i.e. directories are not supported. Keep in mind it has very limited storage space (tenths of kilobytes). Subcommand must be provided.

Argument:

SUBCOMMAND	- subcommand specifying the operation

Example:

```
eeprom dir
```

eeprom dir

Syntax:

```
eeprom dir [SEARCHCRITERIA]
```

Description:

Displays a list of files

Argument:

SEARCHCRITERIA	- string to filter printed output

Example:

```
eeprom dir
eeprom dir *.txt
```

eeprom copy

Syntax:

```
eeprom copy SOURCE DESTINATION
```

Description:

Copy file.

Argument:

SOURCE	- the source file to be copied
DESTINATION	- file to be saved

Example:

```
eeprom copy target.cfg backup.cfg
```

eeprom type

Syntax:

```
eeprom type FILE
```

Description:

Show content of text file.

Argument:

FILE - text file to be shown

Example:

```
eeprom type target.cfg
```

eeprom delete

Syntax:

```
eeprom delete FILE
```

Description:

Delete file.

Argument:

FILE - file to be deleted

Example:

```
eeprom delete target.cfg
```

eeprom rename

Syntax:

```
eeprom rename FILE NEWNAME
```

Description:

Rename file.

Argument:

FILE	- file to be renamed
NEWNAME	- new file name

Example:

```
eeprom rename image.bin backup.bin
```

eeprom format

Syntax:

```
eeprom format
```

Description:

Format EEPROM file system erasing all files.

Argument:

Example:

```
eeprom format
```

eeprom alias

Syntax:

```
eeprom alias [ALIAS [MEANING]]
```

Description:

List or (un)define an alias.

Argument:

ALIAS	- alias to be (un)defined
MEANING	- alias meaning to be defined

Example:

```
eeprom alias
eeprom alias cl 'card dir'
eeprom alias cl "
```

test

Syntax:

```
test FILE ADDR CRC32 COUNT
```

Description:

Load a file info the target memory, calculate the crc32 checksum and compare it with the given CRC32. After every test loop the number of the loops and errors are printed.

Argument:

FILE	- file to be loaded, only BIN format is supported
ADDR	- address in the target memory where to be loaded the file
CRC32	- crc32 of the given file. In Linux this can be done with “crc32 FILE”
COUNT	- number of test loops.

Example:

```
test abcd.bin 0x20000000 0x54327865 5
```

amp

Syntax:

```
amp b | c | h | r | s
```

Description:

Set AMP mode for CORTEX cores with Cross Trigger Interface

Argument:

b	- break
c	- copy all breakpoints to all cores in the group
h	- halt
r	- resume
g	- resume

s - single step

Example:

```
amp     ; print current mode
amp b   ; all cores in the AMP group halt if one core hit a break
amp r   ; all cores in the AMP group resume if one core resume
amp h   ; all cores in the AMP group halt if one core stop
amp rbh ; all cores in the AMP group break, halt and resume
```

rtt setup

Syntax:

```
rtt setup ADDR LEN [ID]
```

Description:

Configure RTT for the currently selected core.

Argument:

ADDR	- address in the target memory from where to start searching of a control block
LEN	- memory length in bytes where to search for a control block
ID	- control block identifier. If omitted, “SEGGER” is used.

Example:

```
rtt setup 0x20000000 4096
```

rtt start

Syntax:

```
rtt start
```

Description:

Start searching of a RTT control block

Example:

```
rtt start
```

rtt list

Syntax:

```
rtt list
```

Description:

List all RTT channels

Example:

```
rtt list
```

rtt server_start

Syntax:

```
rtt server_start PORT CHANNEL
```

Description:

Start a telnet server on PORT for CHANNEL

Argument:

port - TCP port

channel - RTT channel number

Example:

```
rtt server_start 2001 0
```

rtt server_stop

Syntax:

```
rtt server_stop PORT
```

Description:

Stop telnet server on PORT

Argument:

port - TCP port

Example:

```
rtt server_stop 2001
```

4.16 Real Time Transfer (RTT)

Real Time Transfer (RTT) is an interface specified by SEGGER based on basic memory reads and writes to transfer data bidirectionally between target and host. The specification is independent of the target architecture. Every target that supports so called "background memory access", which means that the target memory can be accessed by the debugger while the target is running, can be used. This interface is especially of interest for targets without Serial Wire Output (SWO).

Example of typical usage:

```
rtt setup 0x20000000 4096    ; configure start address for searching
rtt start                    ; start searching of control block
rtt server_start 2001 0      ; start telnet server on port 2001, channel 0
```

4.16.1 Working with the FLASH programmer

PEEDI has built-in universal FLASH programmer. The programmer is used through the flash CLI commands.

The programmer can program FLASH chips in two ways:

1. Directly - completely non intrusive, no target memory is used, but very slow.
2. Using small agent program which is downloaded to the target RAM (1KB) and uses configurable data buffer (0.5-64KB).

So you can choose which is best suitable for your needs, but keep in mind that the "agent" method is much faster. The programming method is set in the FLASH section of the target configuration file, where the configurations are available - DIRECT, AGENT and AUTO - where first agent is tried, if failed the direct method is used.

The image to program is not buffered to the PEEDI's RAM, but it is downloaded from a TFTP/FTP/HTTP server or a MMC/SD card and programmed in configurable data blocks (0.5-64KB), which means that there is no theoretical maximum size limit of the image to be programmed.

Using the programmer you can:

- program the FLASH chip
- verify the FLASH chip
- erase part or entire FLASH chip
- blank check the FLASH chip
- lock/unlock (if supported)

To erase the FLASH, type:

```
peedi> flash erase
```

this will erase the whole FLASH, to erase all sectors within specified FLASH region, type:

```
peedi> flash erase 0x200000 0x1000
```

To program the FLASH using the default arguments from the target configuration file, type:

```
peedi> flash program
```

To program the FLASH using specific file type in a given format to an exact address issue:

```
peedi> flash program tftp://192.168.1.1/mydir/myimage.bin bin 0x100
```

The address to program the image at must be aligned to the FLASH access width, i.e. if the FLASH is 16 bits (2 bytes) accessible the address must be aligned by 2. If the FLASH is an Intel Strata the alignment must be 32 bytes. If the internal FLASH of an Atmel AT91SAM7 series microcontroller is programmed, the alignment must be equal to the FLASH page size (128 or 256 bytes). If the internal FLASH of a Philips LPC2000 series microcontroller is programmed, the alignment must 256 bytes.

After the flash is programmed, you can verify it by:

```
peedi> flash verify
```

or

```
peedi> flash verify tftp://192.168.1.1/mydir/myimage.bin bin 0x100
```



Note:
Most of the flash commands if executed without arguments, will take the last used arguments. If executed for first time they will take their default arguments. For more information on how to use the FLASH programmer, please see the flash CLI commands.

4.17 Multiple FLASH support

The PEEDI FLASH programmer supports targets with multiple FLASH chips mapped at different addresses. Every FLASH must be described in separate section in the target configuration file. If multiple FLASH chips/configurations are present on the target each chip/configurations must be described in different section (see section PLATFORM_ARM). If single FLASH chip/configuration is used the 'm' integer number may be skipped. When working with the programmer the first FLASH is selected as current by default. To work on another FLASH, use the **flash set** command to select it. The multiple FLASH support could also be used to describe different profiles for the same FLASH, for example with different program method type or different image file specified. This way you can easy switch to the desired profile using the **flash set** command.

4.18 Working with a MMC/SD memory card

As mentioned before PEEDI can operate autonomously i.e. without an Ethernet and a host computer. This is achieved by storing all necessary files (target configuration, image, script and other files) into a MMC or SD memory card.



WARNING:
If PEEDI is set to get its network settings from a DHCP server and if the Ethernet cable is unplugged or there is no DHCP server on the Ethernet, it may take some minutes for PEEDI to boot. To avoid this, make sure PEEDI can reach a DHCP server or set it to use a static IP address.

PEEDI can not format a MMC/SD card. The card must be FAT file system formatted in order to use it with PEEDI. There are two ways to copy the necessary files to the memory card. First is to use a MMC/SD card reader and a PC to copy the files. The second way, when no card reader is available is to copy the needed files using the PEEDI CLI transfer command, for this purpose you will also need a FTP, TFTP or HTTP server to copy the files from.

```
peedi> transfer tftp://192.168.1.1/mydir/MyFile.txt card://myfile.txt
```

The **transfer** command can also be used to copy files from the memory card to any file server on the Ethernet.
Before actually copy the files, you may need to create some directories, delete old files or something else. To do this use the PEEDI CLI **card** sub-commands.

4.19 JTAG cable adapters

PEEDI is packed with one suitable JTAG adapter for connecting to a target system.

There are several target adapters available upon request:

- 10-pin - for Cortex targets
- 14-pin - for MIPS 32, TI OMAP, Freescale PowerPC MPC5500 and Analog Devices Blackfin targets
- 16-pin - for Freescale Power QUICK II Pro MPC83xx, Freescale Power QUICC III - MPC85xx targets
- 20-pin - for ARM targets
- 26-pin - for Freescale ColdFire MCF52xx,MCF53xx, MCF54xx targets

For additional information refer to:

https://ronetix.at/product/peedi-jtag-swd-bdm-emulator-and-flash-programmer/#tab-target-adapters

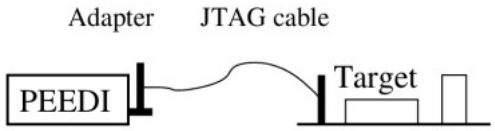


Figure 4.6: Adapter JTAG cable

All adapters are mounted on the PEEDI JTAG connector and next the target cable is connected to the given adapter:

- The ARM20 adapter has standard ARM pinout and may be used with almost all ARM evaluation boards.
- The ARM10 adapter has no standard pinout, but it is useful when the target JTAG cable connector has to be small.
- The ARM14 adapter is used for some old ARM evaluation boards.

If your target JTAG connector pinout is not standard, you may need to make your own target cable considering the PEEDI JTAG connector pinout.

The 4xARM20 adapter is used when you want to take advantage of the multiple core support. The adapter automatically shorts the unused JTAG connector pins to chain the available targets, so there is no need to set jumpers manually.

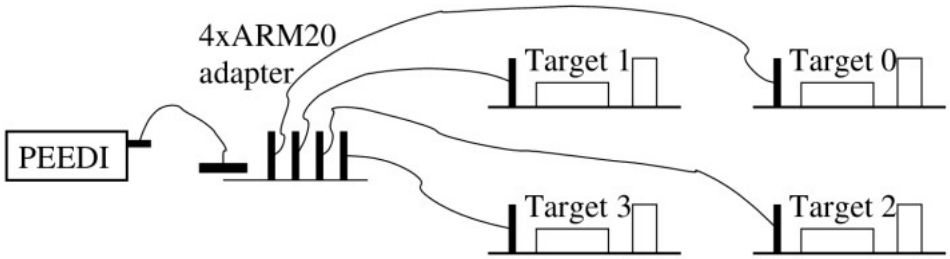


Figure 4.7: JTAG Adapter PEEDI-4xARM20

4.20 PEEDI licenses

PEEDI needs some licenses to operate. Each license unlocks specific feature of PEEDI. Licenses are kept in and loaded from the [LICENSE] section of the target configuration file. You must load the licenses you have acquired before you start using PEEDI. The minimum required licenses are provided when PEEDI is purchased and are printed on the bottom side of PEEDI. Also new units are set to load the target configuration file from the EEPROM, where we have put the file and the licenses.

The UPDATE_DDMMMYYYY license allows you to update PEEDI firmware to version signed to DDMM-MYYYY date. This is the date when your ’firmware warranty’ expires (see Warranty). If you update your PEEDI with firmware released after that date, your PEEDI will refuse to work. You can recover from this situation either loading older firmware or acquire a new update license, so please contact your distributor if the UPDATE license has expired and you need to update PEEDI firmware.

To acquire a license, we need your PEEDI serial number, which is sent over the RS232 port when PEEDI boots or printed when you connect to PEEDI telnet CLI. The PEEDI serial number should look like this – '**SN: PD-1234-5678-90AB**'. After we receive it, we will send you the license, which should look like this - '**KEY = DESCRIPTION, 1234-5678-90AB-C**'. You have to insert that string in a new line in the [LICENSE] section of you target configuration file and reboot PEEDI. If the license is not meant for this PEEDI, it will be simply skipped, this means that multiple PEEDIs may load single shared target configuration file, just fill in all PEEDIs' licenses.

5 Specifications

JTAG Clock	5kHz – 33MHz Adaptive Clocking
Target Voltage	1.2V – 5.0V
Network Interface	Ethernet 10/100 BaseT
Serial Interface	RS232
Power supply	5V / 1A reverse polarity protection over-voltage protection up to 100V 6.9V over-voltage shutdown
Robust Aluminum case:	
Dimensions	115x105x35mm
Weight	270 gram
LEDs:	
Power	Red
Target Power	Red
Ethernet Status	Orange
JTAG Status	Green
Buttons:	
On front panel	Two: red and green
On back panel	One: red
SD card:	Max 32GB, 8+3 DOS name convention
I/O Ports:	
JTAG Header 2x10 2.54mm pitch	Standard ESD Human Body Model IEC 1000-4-2, Direct Discharge > 4kV
RJ45	Dielectric Withstand Voltage: 1500 VAC
RS232	ESD Protection Exceeds ±15 kV Using Human-Body Model

Power Jack 2.1mm 	2.5-kV Human-Body-Model, 500-V CDM Electrostatic Discharge Protection
Operating temperature	+5ºC … +60ºC
Storage temperature	-20ºC …+80ºC
Relative humidity, non condensing	< 90%

5.1 JTAG Target connector signals

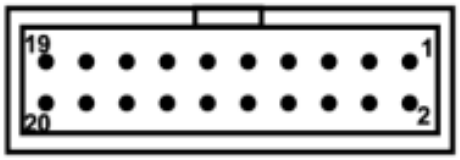


Figure 5.1: PEEDI JTAG connector

Pin	Name	Type	Description
1	**Vcc Target**	Input	**1.2V – 5.0V** Target reference voltage: used to create the logic-level reference for the input comparators. It also controls the output logic levels to the target. It is normally fed from Vcc I/O on the target board.
2	**GND**		
3	**TCK**	Output	**JTAG Clock** Connects to the target TCK line
4	**GND**		
5	**TDI**	Output	**JTAG TDI** Test Data In signal from PEEDI to the target JTAG port. Connects to the target TDI line.
6	**GND**		
7	**TMS**	Output	**JTAG TMS** Connects to the target TMS line.
8	**GND**		

9	**GDBRQ**	Output	Controlled from a parameter in config file
10	**GND**		
11	**TDO**	Input	**JTAG TDO** Test Data Out signal from the target to PEEDI Connects to the target TDO line
12	**GND**		
13	**RTCK**	Input	**Returned JTAG Clock** Connects to the target RTCK line
14	**GND**		
15	**GDBACK**	Input	Not Used
16	**GND**		
17	**Reserved**		
18	**GND**		
19	**TRST**	Push-Pull or Open Drain output	**JTAG Reset** Resets the JTAG TAP controller on the target. Driver type is specified in config file
20	**RST**	Open Drain	**RESET** Resets the target system

Note:
Each signal JTAG pin has a 10k pull-up.

5.2 RS232 Connector (DB9F, female)

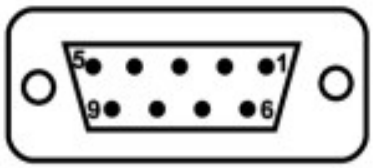


Figure 5.2: RS232 connector

RS232 connector pin configuration	
Pin	Description
1	Not connected

2	Tx
3	Rx
4	Not connected
5	Ground
6	Not connected
7	CTS
8	RTS
9	Not connected

5.3 Schematics

JTAG cable adapter schematics can be found here:

http://download.ronetix.at/peedi/doc/schematics/

6 Warranty

RONETIX warrants PEEDI to be free of defects in materials and workmanship for a period of 24 months following the date of purchase when used under normal conditions.

In the event of notification within the warranty period of defects in material or workmanship, RONETIX will replace defective PEEDI. The remedy for breach of this warranty shall be limited to replacement and shall not encompass any other damages, including but not limited loss of profit, special, incidental, consequential, or other similar claims. RONETIX specifically disclaims all other warranties - expressed or implied, including but not limited to implied warranties of merchantability and fitness for particular purposes - with respect to defects in PEEDI, and the program license granted herein, including without limitation the operation of the program with respect to any particular application, use, or purposes. In no event shall RONETIX be liable for any loss of profit or any other commercial damage, including but not limited to special, incidental, consequential, or other damages. Failure in handling which leads to defects are not covered under this warranty. The warranty is void under any self-made repair operation except exchanging the fuse.

RONETIX warrants PEEDI firmware for a period of 12 months following the date of purchase, i.e. every reported bug will be fixed and an update will be made available.

7 PEEDI Package contents

Make sure all the items listed below are present, when opening the PEEDI package:

- PEEDI
- Power adapter 5V / 1A
- JTAG or BDM cable and adapter
- Patch cable CAT5, 2m
- Serial cable, 1:1, 2m

8 FAQ

Q: What is JTAG?

A: This is a standardized high-speed serial interface, IEEE 1149, widely used for programming and debugging programmable logic and processors. It is non-intrusive, runs regardless of the state of the processor, and gives access to processor registers, memory, and other resources.

Q: What is TAP Controller?

A: The TAP controller provides access to many of the test support functions built into the JTAG compliant device. The TAP is a state machine. The state machine controls all operations for one JTAG compliant device. Each JTAG compliant device has its own TAP controller. You can sequence through the state machine functions via the TCK and TMS inputs.

Q: What is EmbeddedICE?

A: EmbeddedICE is an extension to the core architecture and provides the ability to do in-circuit-emulation with deeply embedded cores. The EmbeddedICE macrocell, adds a JTAG TAP controller and breakpoint/watchpoint logic to the ARM microcontroller which can be accessed externally through a JTAG port. Hence, software debug is facilitated by interfacing these JTAG pins of the micro to the host development system containing the ARM software development tools through a JTAG interface device such as PEEDI.

Q: What is PEEDI?

A: The PEEDI (Powerful Embedded Ethernet Debug Interface) is a debugging and development tool that provides the user the ability to see what is taking place in the target system, and control its behavior. The PEEDI probe provides the debug services that the debugger uses to perform debug operations. It receives command packets over the communication link, and translates them into the JTAG operations that are needed to provide the specific service. First, it can control the operation of

the target processor and target system. What does it mean to ’control’ the target? In most cases it means to start and stop the processor's execution of instructions at arbitrary points in a program, examine and store values in the processor’s registers, and examine and store program code or data in the target system’s memory.

Q: What is debugging?

A: Debugging is the process of removing bugs from computer programs. On one end of the spectrum, debugging means staring at your source code until you see the bug. An infinitely more effective method is to use a special program called a "debugger".

Q: What is a debugger?

A: A debugger is a program that runs other programs. A debugger lets the user (programmer) stop running the program at any time and poke around internally. You can examine and change memory contents, call functions, and look at system registers. Besides all these fun things, a debugger can be used to fix your programs.

Q: How to set gdb to work with PEEDI?

A: First compiled your application with the ’-g -O0’ option to enable debugging. Next start gdb pointing your application:

```
$ arm-elf-gdb myapp
```

To connect to the target (assuming that your PEEDI is set to use IP 192.168.1.10) type in the console window:

```
(gdb) target remote 192.168.1.10:2000
```

This will tell GDB to connect to PEEDI using remote protocol. Now you can load your application into targets memory like this:

```
(gdb) load
```

And your application is ready for debugging:

```
(gdb) continue          ; start the application
```

or

```
(gdb) si                ; make single step
```

Q: What is Eclipse?

A: The Eclipse IDE is a complete integrated development platform similar to Microsoft's Visual Studio. Originally developed by IBM, it has been donated to the Open-Source community and is now a massive world-wide Open-Source development project.

Q: What is Cygwin?

A: Cygwin is a free Linux-like environment for Windows. It works on all Windows 32-bit OS versions since Windows 95 except Windows CE. Cygwin is not a way to run native Linux apps on Windows. Applications must be rebuilt from source code to get it running on Windows.

Q: What is Cygwin/X?

A: Cygwin/X is a port of the X Window System to the Microsoft Windows family of operating systems. Cygwin/X runs on all recent consumer and business versions of Windows; as of 2003-12-27 those versions are specifically Windows 95, Windows 98, Windows Me, Windows NT 4.0, Windows 2000, Windows XP, and Windows Server 2003. For more information see http://x.cygwin.com .

Q: What are GNU cross-development tools?

A: A toolchain is a collection of software tools used for the development and building of software for a particular target architecture. The GNU toolkit consists of the following software utilities:

- gcc - an ANSI C compiler
- g++ - an ANSI tracking C++ compiler
- gdb - source and assembly language command line debugger
- as - GNU assembler

- ld - GNU linker
- Insight - a graphical user interface for gdb

For more information see http://www.gnu.org .

Q: How to enter RedBoot command line?

A: First restart PEEDI holding front panel buttons pressed, this way RedBoot will not execute its boot script and the main PEEDI application will not be loaded. Then you can access the command line via the RS232 port using suitable terminal application capable of opening the serial PC RS232 port or via telnet connecting to the port specified by the fconfig command.

Q: How to update PEEDI firmware?

A: See ’Firmware update procedure’.

Q: How to set target configuration file path?

A: Enter RedBoot command line and use either fconfig or config commands.

Example:

```
config new_target_cfg_file_path
```

Q: How to set the network configuration of PEEDI?

A: Enter RedBoot command line and use fconfig command.

Q: Why PEEDI has a display and two buttons on the front panel?

A: These are used to select, start and observe the execution of user defined scripts which contain PEEDI commands. Those scripts are defined in the target configuration file, for more information see ’Using scripts’.

Q: How big image can PEEDI program?

A: The image to program is not buffered to the PEEDI's RAM, but it is downloaded from a TFTP/FTP/HTTP server or a MMC/SD card and programmed in configurable data blocks (0.5-64KB). Which means, there is no theoretical maximum size limit of the image to be programmed.

Q: PEEDI does not connect to a Philips LPC2XXX device. What should I do?

A: First make sure the pull-down resistor that enables the JTAG interface is not more than 1k and second verify that the CORE_STARTUP_MODE parameter gives the device at least 100ms to run.

Q: When debuging mixed ARM/Thumb code using gdb/insight the debugger can not step in from ARM to Thumb function. What to do?

A: Use the si (step one instruction) command in the gdb/insight several times to step-in to the desired Thumb function.

Q: Is there some examples of target configuration files?

A: Yes

http://download.ronetix.at/peedi/cfg_examples

9 Glossary

A

Alias	- User defined alias of a command including its arguments.
Agent	- Small program downloaded into the target, which is used for faster operations.

B

Breakpoint	- A user-defined point where execution stops so that a debugger can examine the state of memory and registers.
Big-Endian	- Memory organization where the least significant byte of a word is at the highest address and the most significant byte is at the lowest address in the word.

C

CLI	- Command Line Interface.
Cygwin	- Linux-like run-time environment for Windows.
Current core	- The core which is set to be current, i.e. default when no core is specified.

D

Default server	- Default server address used when no server is specified.
DCC	- Debug Communication Channel, communication channel over the JTAG.

G

gdb	- The GNU Debugger.

H

Host	- A computer that provides data and other services to another computer. Especially, a computer providing debugging services to a target being debugged.
Hardware breakpoint	

I

Insight	- Graphical User Interface of GDB.
Image	- An executable file that has been loaded onto a processor for execution.

J

JTAG	- Type of interface which enables direct access to most CPU resources.

M

MMC/SD card	- Multi Media or Secure Digital memory card, used to store files.

P

PC - Program Counter, CPU register that holds the address of the next instruction to be executed.

R

RedBoot - The Red Hat boot loader used for update, setting some configuration parameters or to load and launch the PEEDI executable image.

S

Script - List of CLI commands executed one by one until the last or until an error is returned.

T

Target configuration file - File used to describe target specifics loaded at boot-up.